

# AVR-LibC

2.2.0

Generated by Doxygen 1.9.6

---

<b>1 AVR-LibC</b>	<b>1</b>
1.1 Introduction	1
1.2 General Information about this Library	2
1.3 Supported Devices	2
1.4 AVR-LibC License	3
<b>2 Toolchain Overview</b>	<b>5</b>
2.1 Introduction	5
2.2 FSF and GNU	5
2.3 GCC	5
2.4 GNU Binutils	6
2.5 AVR-LibC	7
2.6 Building Software	8
2.7 AVRDUDE	8
2.8 GDB / Insight / DDD	8
2.9 AVaRICE	8
2.10 SimulAVR	8
2.11 Utilities	8
2.12 Toolchain Distributions (Distros)	9
2.13 Open Source	9
<b>3 Memory Areas and Using malloc()</b>	<b>9</b>
3.1 Introduction	9
3.2 Internal vs. external RAM	10
3.3 Tunables for malloc()	10
3.4 Implementation details	12
<b>4 Memory Sections</b>	<b>13</b>
4.1 Concepts	14
4.1.1 Named Sections	14
4.1.2 Orphan Sections	15
4.1.3 LMA: Load Memory Address	15
4.1.4 VMA: Virtual Memory Address	15
4.2 The Linker Script: Building Blocks	15
4.2.1 Input Sections and Output Sections	16
4.2.2 Memory Regions	16
4.3 Output Sections of the Default Linker Script	17
4.3.1 The .text Output Section	17
4.3.2 The .data Output Section	20
4.3.3 The .bss Output Section	20
4.3.4 The .noinit Output Section	20
4.3.5 The .rodata Output Section	20
4.3.6 The .eeprom Output Section	21

---

4.3.7 The .fuse, .lock and .signature Output Sections . . . . .	21
4.3.8 The .note.gnu.avr.deviceinfo Section . . . . .	21
4.4 Symbols in the Default Linker Script . . . . .	22
4.5 Output Sections and Code Size . . . . .	22
4.6 Using Sections . . . . .	23
4.6.1 In C/C++ Code . . . . .	23
4.6.2 In Assembly Code . . . . .	23
<b>5 Data in Program Space</b> . . . . .	<b>24</b>
5.1 Introduction . . . . .	24
5.2 A Note On const . . . . .	24
5.3 Storing and Retrieving Data in the Program Space . . . . .	25
5.4 Storing and Retrieving Strings in the Program Space . . . . .	26
5.5 Caveats . . . . .	27
<b>6 AVR-LibC and Assembler Programs</b> . . . . .	<b>27</b>
6.1 Introduction . . . . .	27
6.2 Invoking the Compiler . . . . .	28
6.3 Example Program . . . . .	28
6.4 Assembler Directives . . . . .	30
6.5 Operand Modifiers . . . . .	31
<b>7 Inline Assembler Cookbook</b> . . . . .	<b>32</b>
7.1 About this Document . . . . .	33
7.2 The Anatomy of a GCC asm Statement . . . . .	33
7.3 Special Sequences . . . . .	35
7.4 Constraints . . . . .	35
7.5 Print Modifiers . . . . .	37
7.6 Operand Modifiers . . . . .	38
7.7 Examples . . . . .	39
7.7.1 Swapping Nibbles . . . . .	39
7.7.2 Swapping Bytes . . . . .	39
7.7.3 Accessing Memory . . . . .	40
7.7.4 Accessing Bytes of wider Expressions . . . . .	41
7.7.5 Jumping and Branching . . . . .	42
7.8 Binding local Variables to Registers . . . . .	43
7.8.1 Interfacing non-ABI Functions . . . . .	43
7.9 Specifying the Assembly Name of Static Objects . . . . .	44
7.10 What won't work . . . . .	44
7.10.1 Setting a Register on one asm and using it in a different one . . . . .	45
7.10.2 Letting an Operand cross the Boundaries of the Y Register . . . . .	45
7.10.3 Using Matching Constraints "=0"... "=9" with Output Operands . . . . .	45
<b>8 How to Build a Library</b> . . . . .	<b>45</b>

---

8.1 Introduction	45
8.2 How the Linker Works	46
8.3 How to Design a Library	46
8.4 Creating a Library	46
8.5 Using a Library	47
<b>9 Benchmarks</b>	<b>48</b>
9.1 A few of libc functions.	48
9.2 Math functions.	49
<b>10 Porting From IAR to AVR GCC</b>	<b>50</b>
10.1 Introduction	50
10.2 Registers	50
10.3 Interrupt Service Routines (ISRs)	51
10.4 Intrinsic Routines	51
10.5 Flash Variables	51
10.6 Non-Returning main()	52
10.7 Locking Registers	53
<b>11 Frequently Asked Questions</b>	<b>53</b>
11.1 FAQ Index	53
11.2 Why doesn't my program recognize a variable updated in an interrupt routine?	54
11.3 How to permanently bind a variable to a register?	55
11.4 How to modify MCUCR or WDTCR early?	55
11.5 What is all this _BV() stuff about?	56
11.6 Can I use C++ on the AVR?	56
11.7 Shouldn't I initialize all my variables?	57
11.8 Why do some 16-bit timer registers sometimes get trashed?	57
11.9 How do I use a #define'd constant in an asm statement?	58
11.10 Why does the PC randomly jump around when single-stepping through my program in avr-gdb?	58
11.11 How do I trace an assembler file in avr-gdb?	59
11.12 How do I pass an IO port as a parameter to a function?	60
11.13 What registers are used by the C compiler?	61
11.14 How do I put an array of strings completely in ROM?	62
11.14.1 Using named address-spaces	63
11.15 How to use external RAM?	63
11.16 Which -O flag to use?	64
11.17 How do I relocate code to a fixed address?	64
11.18 My UART is generating nonsense! My ATmega128 keeps crashing! Port F is completely broken!	65
11.19 Why do all my "foo...bar" strings eat up the SRAM?	66
11.20 How to detect RAM memory and variable overlap problems?	66
11.21 Is it really impossible to program the ATtinyXX in C?	67
11.22 What is this "clock skew detected" message?	67

---

11.23 Why are (many) interrupt flags cleared by writing a logical 1? . . . . .	67
11.24 Why have "programmed" fuses the bit value 0? . . . . .	68
11.25 Which AVR-specific assembler operators are available? . . . . .	68
11.26 Why are interrupts re-enabled in the middle of writing the stack pointer? . . . . .	68
11.27 Why are there five different linker scripts? . . . . .	69
11.28 How to add a raw binary image to linker output? . . . . .	69
11.29 How do I perform a software reset of the AVR? . . . . .	70
11.30 What pitfalls exist when writing reentrant code? . . . . .	70
11.31 Why are some addresses of the EEPROM corrupted (usually address zero)? . . . . .	72
11.32 Why is my baud rate wrong? . . . . .	73
11.33 On a device with more than 128 KiB of flash, how to make function pointers work? . . . . .	73
11.34 Why is assigning ports in a "chain" a bad idea? . . . . .	73
11.35 Which header files are included in my program? . . . . .	74
11.36 Which macros are defined in my program? Where are they defined, and to what value? . . . . .	74
11.37 What ISR names are available for my device? . . . . .	75
<b>12 Building and Installing the GNU Tool Chain . . . . .</b>	<b>76</b>
12.1 Required AVR Tools . . . . .	76
12.2 Optional AVR Tools . . . . .	77
12.3 Building and Installing under Linux, FreeBSD, and Others . . . . .	77
12.3.1 Preparations . . . . .	77
12.3.2 GNU Binutils for the AVR target . . . . .	78
12.3.3 GCC for the AVR target . . . . .	79
12.3.4 AVR-LibC . . . . .	80
12.3.5 AVRDUDE . . . . .	80
12.3.6 SimuAVR . . . . .	81
12.3.7 AVaRICE . . . . .	81
12.4 Building and Installing under Windows . . . . .	81
12.4.1 Tools Required for Building the Toolchain for Windows . . . . .	82
12.4.2 Building the Toolchain for Windows . . . . .	83
12.5 Canadian Cross Builds . . . . .	87
12.6 Using Git . . . . .	88
<b>13 Using the GNU tools . . . . .</b>	<b>89</b>
13.1 Options for the C compiler <code>avr-gcc</code> . . . . .	89
13.1.1 Machine-specific options for the AVR . . . . .	89
13.1.2 Selected general compiler options . . . . .	90
13.2 Options for the assembler <code>avr-as</code> . . . . .	92
13.2.1 Machine-specific assembler options . . . . .	93
13.2.2 Examples for assembler options passed through the C compiler . . . . .	93
13.3 Controlling the linker <code>avr-ld</code> . . . . .	94
13.3.1 Selected linker options . . . . .	94
13.3.2 Passing linker options from the C compiler . . . . .	94

---

<b>14 Compiler optimization</b>	<b>95</b>
14.1 Problems with reordering code . . . . .	95
<b>15 Using the avrdude program</b>	<b>97</b>
<b>16 Acknowledgments</b>	<b>98</b>
<b>17 Deprecated List</b>	<b>99</b>
<b>18 Module Index</b>	<b>99</b>
18.1 Modules . . . . .	99
<b>19 Data Structure Index</b>	<b>101</b>
19.1 Data Structures . . . . .	101
<b>20 File Index</b>	<b>101</b>
20.1 File List . . . . .	101
<b>21 Module Documentation</b>	<b>103</b>
21.1 <alloca.h>: Allocate space in the stack . . . . .	103
21.1.1 Detailed Description . . . . .	103
21.1.2 Function Documentation . . . . .	103
21.2 <assert.h>: Diagnostics . . . . .	104
21.2.1 Detailed Description . . . . .	104
21.2.2 Macro Definition Documentation . . . . .	104
21.3 <ctype.h>: Character Operations . . . . .	105
21.3.1 Detailed Description . . . . .	105
21.3.2 Function Documentation . . . . .	105
21.4 <errno.h>: System Errors . . . . .	107
21.4.1 Detailed Description . . . . .	108
21.4.2 Macro Definition Documentation . . . . .	108
21.4.3 Variable Documentation . . . . .	108
21.5 <inttypes.h>: Integer Type conversions . . . . .	108
21.5.1 Detailed Description . . . . .	111
21.5.2 Macro Definition Documentation . . . . .	111
21.5.3 Typedef Documentation . . . . .	121
21.6 <math.h>: Mathematics . . . . .	122
21.6.1 Detailed Description . . . . .	124
21.6.2 Macro Definition Documentation . . . . .	125
21.6.3 Function Documentation . . . . .	127
21.7 <setjmp.h>: Non-local goto . . . . .	142
21.7.1 Detailed Description . . . . .	143
21.7.2 Function Documentation . . . . .	143
21.8 <stdint.h>: Standard Integer Types . . . . .	144
21.8.1 Detailed Description . . . . .	147

---

21.8.2 Macro Definition Documentation	147
21.8.3 Typedef Documentation	152
21.9 <stdio.h>: Standard IO facilities	156
21.9.1 Detailed Description	157
21.9.2 Macro Definition Documentation	159
21.9.3 Typedef Documentation	161
21.9.4 Function Documentation	162
21.10 <stdlib.h>: General utilities	172
21.10.1 Detailed Description	173
21.10.2 Macro Definition Documentation	173
21.10.3 Typedef Documentation	174
21.10.4 Function Documentation	174
21.10.5 Variable Documentation	183
21.11 <string.h>: Strings	184
21.11.1 Detailed Description	185
21.11.2 Macro Definition Documentation	185
21.11.3 Function Documentation	185
21.12 <time.h>: Time	197
21.12.1 Detailed Description	198
21.12.2 Macro Definition Documentation	199
21.12.3 Typedef Documentation	200
21.12.4 Enumeration Type Documentation	200
21.12.5 Function Documentation	201
21.13 <avr/boot.h>: Bootloader Support Utilities	207
21.13.1 Detailed Description	207
21.13.2 Macro Definition Documentation	208
21.14 <avr/cpufunc.h>: Special AVR CPU functions	212
21.14.1 Detailed Description	213
21.14.2 Macro Definition Documentation	213
21.14.3 Function Documentation	213
21.15 <avr/eeprom.h>: EEPROM handling	214
21.15.1 Detailed Description	215
21.15.2 Macro Definition Documentation	215
21.15.3 Function Documentation	216
21.16 <avr/fuse.h>: Fuse Support	219
21.17 <avr/interrupt.h>: Interrupts	222
21.17.1 Detailed Description	222
21.17.2 Macro Definition Documentation	225
21.18 <avr/io.h>: AVR device-specific IO definitions	228
21.18.1 Detailed Description	229
21.18.2 Macro Definition Documentation	229
21.19 <avr/lock.h>: Lockbit Support	230

---

21.20 <avr/pgmspace.h>: Program Space Utilities	232
21.20.1 Detailed Description	234
21.20.2 Macro Definition Documentation	234
21.20.3 Function Documentation	237
21.21 <avr/power.h>: Power Reduction Management	259
21.21.1 Detailed Description	259
21.21.2 Macro Definition Documentation	261
21.21.3 Function Documentation	262
21.22 Additional notes from <avr/sfr_defs.h>	263
21.23 <avr/sfr_defs.h>: Special function registers	263
21.23.1 Detailed Description	264
21.23.2 Macro Definition Documentation	264
21.24 <avr/signature.h>: Signature Support	265
21.25 <avr/sleep.h>: Power Management and Sleep Modes	266
21.25.1 Detailed Description	266
21.25.2 Function Documentation	266
21.26 <avr/version.h>: avr-libc version macros	268
21.26.1 Detailed Description	268
21.26.2 Macro Definition Documentation	268
21.27 <avr/builtins.h>: avr-gcc builtins documentation	269
21.27.1 Detailed Description	269
21.27.2 Function Documentation	269
21.28 <avr/wdt.h>: Watchdog timer handling	271
21.28.1 Detailed Description	271
21.28.2 Macro Definition Documentation	271
21.29 <util/delay.h>: Convenience functions for busy-wait delay loops	273
21.29.1 Detailed Description	274
21.29.2 Macro Definition Documentation	274
21.29.3 Function Documentation	274
21.30 <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks	276
21.30.1 Detailed Description	276
21.30.2 Macro Definition Documentation	277
21.31 <util/crc16.h>: CRC Computations	278
21.31.1 Detailed Description	279
21.31.2 Function Documentation	279
21.32 <util/delay_basic.h>: Basic busy-wait delay loops	282
21.32.1 Detailed Description	282
21.32.2 Function Documentation	282
21.33 <util/eu_dst.h>: Daylight Saving function for the European Union.	282
21.33.1 Detailed Description	283
21.33.2 Function Documentation	283
21.34 <util/parity.h>: Parity bit generation	283



---

21.34.1 Detailed Description	283
21.34.2 Function Documentation	283
21.35 <util/setbaud.h>: Helper macros for baud rate calculations	284
21.35.1 Detailed Description	284
21.35.2 Macro Definition Documentation	285
21.36 <util/twi.h>: TWI bit mask definitions	286
21.36.1 Detailed Description	286
21.36.2 Macro Definition Documentation	287
21.37 <util/usa_dst.h>: Daylight Saving function for the USA.	290
21.37.1 Detailed Description	290
21.37.2 Function Documentation	290
21.38 <compat/deprecated.h>: Deprecated items	291
21.38.1 Detailed Description	291
21.38.2 Macro Definition Documentation	292
21.38.3 Function Documentation	293
21.39 <compat/ina90.h>: Compatibility with IAR EWB 3.x	293
21.40 Demo projects	294
21.40.1 Detailed Description	294
21.41 Combining C and assembly source files	295
21.41.1 Hardware setup	295
21.41.2 A code walkthrough	295
21.41.3 The source code	297
21.42 A simple project	297
21.42.1 The Project	297
21.42.2 The Source Code	299
21.42.3 Compiling and Linking	300
21.42.4 Examining the Object File	300
21.42.5 Linker Map Files	304
21.42.6 Generating Intel Hex Files	305
21.42.7 Letting Make Build the Project	306
21.42.8 Reference to the source code	307
21.43 A more sophisticated project	308
21.43.1 Hardware setup	308
21.43.2 Functional overview	310
21.43.3 A code walkthrough	310
21.43.4 The source code	312
21.44 Using the standard IO facilities	312
21.44.1 Hardware setup	312
21.44.2 Functional overview	313
21.44.3 A code walkthrough	314
21.44.4 The source code	317
21.45 Example using the two-wire interface (TWI)	317

---

21.45.1 Introduction into TWI . . . . .	318
21.45.2 The TWI example project . . . . .	318
21.45.3 The Source Code . . . . .	318
<b>22 Data Structure Documentation</b>	<b>321</b>
22.1 div_t Struct Reference . . . . .	321
22.1.1 Detailed Description . . . . .	321
22.1.2 Field Documentation . . . . .	322
22.2 ldiv_t Struct Reference . . . . .	322
22.2.1 Detailed Description . . . . .	322
22.2.2 Field Documentation . . . . .	322
22.3 tm Struct Reference . . . . .	323
22.3.1 Detailed Description . . . . .	323
22.3.2 Field Documentation . . . . .	323
22.4 week_date Struct Reference . . . . .	324
22.4.1 Detailed Description . . . . .	324
22.4.2 Field Documentation . . . . .	324
<b>23 File Documentation</b>	<b>325</b>
23.1 project.h . . . . .	325
23.2 iocompat.h . . . . .	325
23.3 defines.h . . . . .	327
23.4 hd44780.h . . . . .	328
23.5 lcd.h . . . . .	329
23.6 uart.h . . . . .	329
23.7 alloca.h . . . . .	330
23.8 assert.h File Reference . . . . .	331
23.9 assert.h . . . . .	331
23.10 boot.h File Reference . . . . .	332
23.11 boot.h . . . . .	333
23.12 builtins.h File Reference . . . . .	341
23.13 builtins.h . . . . .	341
23.14 cpufunc.h File Reference . . . . .	343
23.15 cpufunc.h . . . . .	343
23.16 eeprom.h . . . . .	344
23.17 fuse.h File Reference . . . . .	348
23.18 fuse.h . . . . .	348
23.19 interrupt.h File Reference . . . . .	352
23.20 interrupt.h . . . . .	352
23.21 io.h File Reference . . . . .	357
23.22 io.h . . . . .	357
23.23 lock.h File Reference . . . . .	366
23.24 lock.h . . . . .	366

---

23.25 pgmspace.h File Reference . . . . .	369
23.26 pgmspace.h . . . . .	372
23.27 portpins.h . . . . .	396
23.28 power.h File Reference . . . . .	402
23.29 power.h . . . . .	403
23.30 sfr_defs.h . . . . .	424
23.31 signal.h . . . . .	427
23.32 signature.h File Reference . . . . .	428
23.33 signature.h . . . . .	428
23.34 sleep.h File Reference . . . . .	429
23.35 sleep.h . . . . .	429
23.36 version.h . . . . .	433
23.37 wdt.h File Reference . . . . .	434
23.38 wdt.h . . . . .	435
23.39 xmega.h . . . . .	442
23.40 deprecated.h . . . . .	443
23.41 ina90.h . . . . .	446
23.42 ctype.h File Reference . . . . .	447
23.43 ctype.h . . . . .	448
23.44 errno.h File Reference . . . . .	450
23.45 errno.h . . . . .	450
23.46 inttypes.h File Reference . . . . .	452
23.47 inttypes.h . . . . .	454
23.48 math.h File Reference . . . . .	461
23.49 math.h . . . . .	464
23.50 setjmp.h File Reference . . . . .	472
23.51 setjmp.h . . . . .	472
23.52 stdint.h File Reference . . . . .	474
23.53 stdint.h . . . . .	477
23.54 stdio.h File Reference . . . . .	485
23.55 stdio.h . . . . .	486
23.56 stdlib.h File Reference . . . . .	498
23.57 stdlib.h . . . . .	500
23.58 string.h File Reference . . . . .	509
23.59 string.h . . . . .	510
23.60 time.h File Reference . . . . .	517
23.61 time.h . . . . .	518
23.62 atomic.h File Reference . . . . .	525
23.63 atomic.h . . . . .	525
23.64 crc16.h File Reference . . . . .	529
23.65 crc16.h . . . . .	529
23.66 delay.h File Reference . . . . .	533

23.67 delay.h . . . . .	534
23.68 delay_basic.h File Reference . . . . .	537
23.69 delay_basic.h . . . . .	537
23.70 eu_dst.h File Reference . . . . .	539
23.71 eu_dst.h . . . . .	539
23.72 parity.h File Reference . . . . .	540
23.73 parity.h . . . . .	540
23.74 setbaud.h File Reference . . . . .	541
23.75 setbaud.h . . . . .	541
23.76 compat/twi.h . . . . .	544
23.77 twi.h File Reference . . . . .	544
23.78 util/twi.h . . . . .	545
23.79 usa_dst.h File Reference . . . . .	548
23.80 usa_dst.h . . . . .	548
23.81 eedef.h . . . . .	549
23.82 fdevopen.c File Reference . . . . .	551
23.83 stdio_private.h . . . . .	551
23.84 xtoa_fast.h . . . . .	552
23.85 dtoa_conv.h . . . . .	552
23.86 stdlib_private.h . . . . .	553
23.87 ephemer_common.h . . . . .	554
<b>Index</b>	<b>557</b>

## 1 AVR-LibC

### 1.1 Introduction

The latest version of this document is always available from <https://avrdudes.github.io/avr-libc/>

This documentation is distributed under the same licensing conditions as the entire library itself, see [License](#) below.

The AVR-LibC package provides a subset of the standard C library for [Microchip \(formerly Atmel\) AVR 8-bit RISC microcontrollers](#). In addition, the library provides the basic startup code needed by most applications.

There is a wealth of information in this document which goes beyond simply describing the interfaces and routines provided by the library. We hope that this document provides enough information to get a new AVR developer up to speed quickly using the freely available development tools: binutils, gcc, AVR-LibC and many others.

If you find yourself stuck on a problem which this document doesn't quite address, you may wish to post a message to the avr-gcc mailing list. Most of the developers of the AVR Binutils and GCC ports in addition to the developers of AVR-LibC subscribe to the list, so you will usually be able to get your problem resolved. You can subscribe to the list at <http://lists.nongnu.org/mailman/listinfo/avr-gcc-list>. Before posting to the list, you might want to try reading the [Frequently Asked Questions](#) chapter of this document.

#### Note

If you think you've found a bug, or have a suggestion for an improvement, either in this documentation or in the library itself, please use the [bug tracker](#) to ensure the issue won't be forgotten.

## 1.2 General Information about this Library

In general, it has been the goal to stick as best as possible to established standards while implementing this library. Commonly, this refers to the C library as described by the ANSI X3.159-1989 and ISO/IEC 9899:1990 ("ANSI-C") standard, as well as parts of their successor ISO/IEC 9899:1999 ("C99"). Some additions have been inspired by other standards like IEEE Std 1003.1-1988 ("POSIX.1"), while other extensions are purely AVR-specific (like the entire program-space string interface).

Unless otherwise noted, functions of this library are *not* guaranteed to be reentrant. In particular, any functions that store local state are known to be non-reentrant, as well as functions that manipulate I/O registers like the [EEPROM](#) access routines. If these functions are used within both standard and interrupt contexts undefined behaviour will result. See the [FAQ](#) for a more detailed discussion.

## 1.3 Supported Devices

The following is a list of AVR devices currently supported by the library. Note that actual support for some newer devices depends on the ability of the compiler to support these devices at library compile-time.

**megaAVR Devices:** ATmega103, ATmega128, ATmega128A, ATmega1280, ATmega1281, ATmega1284, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega164A, ATmega164P, ATmega164PA, ATmega165, ATmega165A, ATmega165P, ATmega165PA, ATmega168, ATmega168A, ATmega168P, ATmega168PA, ATmega168PB, ATmega16A, ATmega2560, ATmega2561, ATmega32, ATmega32A, ATmega323, ATmega324A, ATmega324P, ATmega324PA, ATmega324PB, ATmega325, ATmega325A, ATmega325P, ATmega325PA, ATmega3250, ATmega3250A, ATmega3250P, ATmega3250PA, ATmega328, ATmega328P, ATmega328PB, ATmega48, ATmega48A, ATmega48PA, ATmega48PB, ATmega48P, ATmega64, ATmega64A, ATmega640, ATmega644, ATmega644A, ATmega644P, ATmega644PA, ATmega645, ATmega645A, ATmega645P, ATmega6450, ATmega6450A, ATmega6450P, ATmega8, ATmega8A, ATmega88, ATmega88A, ATmega88P, ATmega88PA, ATmega88PB, ATmega8515, ATmega8535

**megaAVR 0-Series Devices:** ATmega808, ATmega809, ATmega1608, ATmega1609, ATmega3208, ATmega3209, ATmega4808, ATmega4809

**tinyAVR Devices:** ATtiny11 [1], ATtiny12 [1], ATtiny13, ATtiny13A, ATtiny15 [1], ATtiny22, ATtiny24, ATtiny24A, ATtiny25, ATtiny26, ATtiny261, ATtiny261A, ATtiny28 [1], ATtiny2313, ATtiny2313A, ATtiny4313, ATtiny43U, ATtiny44, ATtiny44A, ATtiny441, ATtiny45, ATtiny461, ATtiny461A, ATtiny48, ATtiny828, ATtiny84, ATtiny84A, ATtiny841, ATtiny85, ATtiny861, ATtiny861A, ATtiny88, ATtiny1634

**tinyAVR 0-Series Devices:** ATtiny202, ATtiny204, ATtiny402, ATtiny404, ATtiny406, ATtiny804, ATtiny806, ATtiny807, ATtiny1604, ATtiny1606, ATtiny1607

**tinyAVR 1-Series Devices:** ATtiny212, ATtiny214, ATtiny412, ATtiny414, ATtiny416, ATtiny417, ATtiny814, ATtiny816, ATtiny817, ATtiny1614, ATtiny1616, ATtiny1617, ATtiny3214, ATtiny3216, ATtiny3217

**tinyAVR 2-Series Devices:** ATtiny424, ATtiny426, ATtiny427, ATtiny824, ATtiny826, ATtiny827, ATtiny1624, ATtiny1626, ATtiny1627, ATtiny3224, ATtiny3226, ATtiny3227

**Reduced tinyAVR Devices with only 16 GPRs:** ATtiny4, ATtiny5, ATtiny9, ATtiny10, ATtiny102, ATtiny104, ATtiny20, ATtiny40

**Automotive AVR Devices:** ATtiny87, ATtiny167, ATA5505, ATA5272, ATA5702M322, ATA5782, ATA5790, ATA5790N, ATA5831, ATA5795, ATA6285, ATA6286, ATA6289, ATA6612C, ATA6613C, ATA6614Q, ATA6616C, ATA6617C, ATA664251

**Automotive CAN AVR Devices:** ATmega16M1, ATmega32C1, ATmega32M1, ATmega64C1, ATmega64M1

**CAN AVR Devices:** AT90CAN32, AT90CAN64, AT90CAN128

**LCD AVR Devices:** ATmega169, ATmega169A, ATmega169P, ATmega169PA, ATmega329, ATmega329A, ATmega329P, ATmega329PA, ATmega3290, ATmega3290A, ATmega3290P, ATmega3290PA, ATmega649, ATmega649A, ATmega6490, ATmega6490A, ATmega6490P, ATmega649P

**Lighting AVR Devices:** AT90PWM1, AT90PWM2, AT90PWM2B, AT90PWM216, AT90PWM3, AT90PWM3B, AT90PWM316, AT90PWM161, AT90PWM81

**Smart Battery AVR Devices:** ATmega8HVA, ATmega16HVA, ATmega16HVA2, ATmega16HVB, ATmega16HVBREVB, ATmega32HVB, ATmega32HVBREVB, ATmega64HVE, ATmega64HVE2, ATmega406

**USB AVR Devices:** AT76C711 [3], AT90USB82, AT90USB162, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287, ATmega8U2, ATmega16U2, ATmega16U4, ATmega32U2, ATmega32U4, ATmega32U6, AT43USB320, AT43USB355

**XMEGA Devices:** ATxmega8E5, ATxmega16A4, ATxmega16D4, ATxmega32A4, ATxmega32D3, ATxmega32D4, ATxmega32E5, ATxmega64A1, ATxmega64A3, ATxmega64D3, ATxmega64D4, ATxmega128A1, ATxmega128A3, ATxmega128D3, ATxmega128D4, ATxmega192A3, ATxmega192D3, ATxmega256A3, ATxmega256A3B, ATxmega256D3

**USB XMEGA Devices:** ATxmega16A4U, ATxmega16C4, ATxmega32A4U, ATxmega32C3, ATxmega32C4, ATxmega64A1U, ATxmega64A3U, ATxmega64A4U, ATxmega64B1, ATxmega64B3, ATxmega64C3, ATxmega128A1U, ATxmega128A3U, ATxmega128A4U, ATxmega128B1, ATxmega128B3, ATxmega128C3, ATxmega192A3U, ATxmega192C3, ATxmega256A3U, ATxmega256A3BU, ATxmega256C3, ATxmega384C3, ATxmega384D3

**AVR-Dx Devices:** AVR16DD14, AVR16DD20, AVR16DD28, AVR16DD32, AVR32DA28, AVR32DA32, AVR32DA48, AVR32DB28, AVR32DB32, AVR32DB48, AVR32DD14, AVR32DD20, AVR32DD28, AVR32DD32, AVR64DA28, AVR64DA32, AVR64DA48, AVR64DA64, AVR64DB28, AVR64DB32, AVR64DB48, AVR64DB64, AVR64DD14, AVR64DD20, AVR64DD28, AVR64DD32, AVR128DA28, AVR128DA32, AVR128DA48, AVR128DA64, AVR128DB28, AVR128DB32, AVR128DB48, AVR128DB64

**USB AVR-Dx Devices:** AVR64DU28, AVR64DU32

**AVR-Ex Devices:** AVR16EA28, AVR16EA32, AVR16EA48, AVR16EB14, AVR16EB20, AVR16EB28, AVR16EB32, AVR32EA28, AVR32EA32, AVR32EA48, AVR64EA28, AVR64EA32, AVR64EA48

**Wireless AVR Devices:** ATmega644RFR2, ATmega64RFR2, ATmega128RFA1, ATmega1284RFR2, ATmega128RFR2, ATmega2564RFR2, ATmega256RFR2

**Miscellaneous Devices:** AT94K [2], AT86RF401, AT90SCR100, M3000 [4]

**Classic AVR Devices:** AT90S1200 [1], AT90S2313, AT90S2323, AT90S2333, AT90S2343, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90C8534, AT90S8535

**Note [1]** Assembly only. There is no direct support for these devices to be programmed in C since they do not have a RAM based stack. Still, it could be possible to program them in C, see the [FAQ](#) for an option.

**Note [2]** The AT94K devices are a combination of FPGA and AVR microcontroller. [TRoth-2002/11/12: Not sure of the level of support for these. More information would be welcomed.]

**Note [3]** The AT76C711 is a USB to fast serial interface bridge chip using an AVR core.

**Note [4]** The M3000 is a motor controller AVR ASIC from Intelligent Motion Systems (IMS) / Schneider Electric.

## 1.4 AVR-LibC License

AVR-LibC can be freely used and redistributed, provided the following license conditions are met.

The contents of AVR-LibC are licensed with a Modified BSD License.

All of this is supposed to be Free Software, Open Source, DFSG-free, GPL-compatible, and OK to use in both free and proprietary applications.

See the license information in the individual source files for details.

Additions and corrections to this file are welcome.

```
*****  
Portions of avr-libc are Copyright (c) 1999-2024  
Werner Boellmann,  
Dean Camera,  
Pieter Conradie,  
Brian Dean,  
Keith Gudger,  
Wouter van Gulik,  
Bjoern Haase,  
Steinar Haugen,  
Peter Jansen,  
Reinhard Jessich,  
Magnus Johansson,  
Georg Johann Lay,  
Harald Kipp,  
Carlos Lamas,  
Cliff Lawson,  
Artur Lipowski,  
Marek Michalkiewicz,  
Todd C. Miller,  
Rich Neswold,  
Colin O'Flynn,  
Bob Paddock,  
Andrey Pashchenko,  
Reiner Patommel,  
Florin-Viorel Petrov,  
Alexander Popov,  
Michael Rickman,  
Theodore A. Roth,  
Juergen Schilling,  
Philip Soeberg,  
Anatoly Sokolov,  
Nils Kristian Strom,  
Michael Stumpf,  
Stefan Swanepoel,  
Helmut Wallner,  
Eric B. Weddington,  
Joerg Wunsch,  
Dmitry Xmelkov,  
egnite Software GmbH,  
The Regents of the University of California.  
All rights reserved.
```

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* Neither the name of the copyright holders nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF

SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*\*\*\*\*

## 2 Toolchain Overview

### 2.1 Introduction

Welcome to the open source software development toolset for the Microchip (formerly Atmel) AVR!

There is not a single tool that provides everything needed to develop software for the AVR. It takes many tools working together. Collectively, the group of tools are called a toolset, or commonly a toolchain, as the tools are chained together to produce the final executable application for the AVR microcontroller.

The following sections provide an overview of all of these tools. You may be used to cross-compilers that provide everything with a GUI front-end, and not know what goes on "underneath the hood". You may be coming from a desktop or server computer background and not used to embedded systems. Or you may be just learning about the most common software development toolchain available on Unix and Linux systems. Hopefully the following overview will be helpful in putting everything in perspective.

### 2.2 FSF and GNU

According to its website, "the Free Software Foundation (FSF), established in 1985, is dedicated to promoting computer users' rights to use, study, copy, modify, and redistribute computer programs. The FSF promotes the development and use of free software, particularly the GNU operating system, used widely in its GNU/Linux variant." The FSF remains the primary sponsor of the GNU project.

The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. GNU is a recursive acronym for "GNU's Not Unix"; it is pronounced guh-noo, approximately like canoe.

One of the main projects of the GNU system is the GNU Compiler Collection, or GCC, and its sister project, GNU Binutils. These two open source projects provide a foundation for a software development toolchain. Note that these projects were designed to originally run on Unix-like systems.

### 2.3 GCC

GCC stands for GNU Compiler Collection. GCC is highly flexible compiler system. It has different compiler front-ends for different languages. It has many back-ends that generate assembly code for many different processors and host operating systems. All share a common "middle-end", containing the generic parts of the compiler, including a lot of optimizations.

In GCC, a *host* system is the system (processor/OS) that the compiler runs on. A *target* system is the system that the compiler compiles code for. And, a *build* system is the system that the compiler is built (from source code) on. If a compiler has the same system for *host* and for *target*, it is known as a *native* compiler. If a compiler has different systems for *host* and *target*, it is known as a cross-compiler. (And if all three, *build*, *host*, and *target* systems are different, it is known as a Canadian cross compiler, but we won't discuss that here.) When GCC is built to execute on a *host* system such as FreeBSD, Linux, or Windows, and it is built to generate code for the AVR microcontroller



*target*, then it is a cross compiler, and this version of GCC is commonly known as "AVR GCC". In documentation, or discussion, AVR GCC is used when referring to GCC targeting specifically the AVR, or something that is AVR specific about GCC. The term "GCC" is usually used to refer to something generic about GCC, or about GCC as a whole.

GCC is different from most other compilers. GCC focuses on translating a high-level language to the target assembly only. AVR GCC has three available compilers for the AVR: C language, C++, and Ada. The compiler itself does not assemble or link the final code.

GCC is also known as a "driver" program, in that it knows about, and drives other programs seamlessly to create the final output. The assembler, and the linker are part of another open source project called GNU Binutils. GCC knows how to drive the GNU assembler (gas) to assemble the output of the compiler. GCC knows how to drive the GNU linker (ld) to link all of the object modules into a final executable.

The two projects, GCC and Binutils, are very much interrelated and many of the same volunteers work on both open source projects.

When GCC is built for the AVR target, the actual program names are prefixed with "avr-". So the actual executable name for AVR GCC is: avr-gcc. The name "avr-gcc" is used in documentation and discussion when referring to the program itself and not just the whole AVR GCC system.

See the GCC Web Site and GCC User Manual for more information about GCC.

## 2.4 GNU Binutils

The name GNU Binutils stands for "Binary Utilities". It contains the GNU assembler (gas), and the GNU linker (ld), but also contains many other utilities that work with binary files that are created as part of the software development toolchain.

Again, when these tools are built for the AVR target, the actual program names are prefixed with "avr-". For example, the assembler program name, for a native assembler is "as" (even though in documentation the GNU assembler is commonly referred to as "gas"). But when built for an AVR target, it becomes "avr-as". Below is a list of the programs that are included in Binutils:

**avr-as**

The Assembler.

**avr-ld**

The Linker.

**avr-ar**

Create, modify, and extract from libraries (archives).

**avr-ranlib**

Generate index to library (archive) contents.

**avr-objcopy**

Copy and translate object files to different formats.

**avr-objdump**

Display information from object files including disassembly.

**avr-size**

List section sizes and total size.

**avr-nm**

List symbols from object files.

**avr-strings**

List printable strings from files.

**avr-strip**

Discard symbols from files.

**avr-readelf**

Display the contents of ELF format files.

**avr-addr2line**

Convert addresses to file and line.

**avr-c++filt**

Filter to demangle encoded C++ symbols.

## 2.5 AVR-LibC

GCC and Binutils provides a lot of the tools to develop software, but there is one critical component that they do not provide: a Standard C Library.

There are different open source projects that provide a Standard C Library depending upon your system time, whether for a native compiler (GNU Libc), for some other embedded system (Newlib), or for some versions of Linux (uCLibc). The open source AVR toolchain has its own Standard C Library project: AVR-LibC.

AVR-LibC provides many of the same functions found in a regular Standard C Library and many additional library functions that is specific to an AVR. Some of the Standard C Library functions that are commonly used on a PC environment have limitations or additional issues that a user needs to be aware of when used on an embedded system.

AVR-LibC also contains the most documentation about the whole AVR toolchain.

## 2.6 Building Software

Even though GCC, Binutils, and AVR-LibC are the core projects that are used to build software for the AVR, there is another piece of software that ties it all together: Make. GNU Make is a program that makes things, and mainly software. Make interprets and executes a Makefile that is written for a project. A Makefile contains dependency rules, showing which output files are dependent upon which input files, and instructions on how to build output files from input files.

Some distributions of the toolchains, and other AVR tools such as MFile, contain a Makefile template written for the AVR toolchain and AVR applications that you can copy and modify for your application.

See the GNU Make User Manual for more information.

## 2.7 AVRDUDE

After creating your software, you'll want to program your device. You can do this by using the program AVRDUDE which can interface with various hardware devices to program your processor.

AVRDUDE is a very flexible package. All the information about AVR processors and various hardware programmers is stored in a text database. This database can be modified by any user to add new hardware or to add an AVR processor if it is not already listed.

## 2.8 GDB / Insight / DDD

The GNU Debugger (GDB) is a command-line debugger that can be used with the rest of the AVR toolchain. Insight is GDB plus a GUI written in Tcl/Tk. Both GDB and Insight are configured for the AVR and the main executables are prefixed with the target name: `avr-gdb`, and `avr-insight`. There is also a "text mode" GUI for GDB: `avr-gdbtui`. DDD (Data Display Debugger) is another popular GUI front end to GDB, available on Unix and Linux systems.

## 2.9 AVaRICE

AVaRICE is a back-end program to AVR GDB and interfaces to the AVR JTAG In-Circuit Emulator (ICE), to provide emulation capabilities.

## 2.10 SimulAVR

SimulAVR is an AVR simulator used as a back-end with AVR GDB.

## 2.11 Utilities

There are also other optional utilities available that may be useful to add to your toolset.

`SRecord` is a collection of powerful tools for manipulating EPROM load files. It reads and writes numerous EPROM file formats, and can perform many different manipulations.

`MFile` is a simple Makefile generator is meant as an aid to quickly customize a Makefile to use for your AVR application.

## 2.12 Toolchain Distributions (Distros)

All of the various open source projects that comprise the entire toolchain are normally distributed as source code. It is left up to the user to build the tool application from its source code. This can be a very daunting task to any potential user of these tools.

Luckily there are people who help out in this area. Volunteers take the time to build the application from source code on particular host platforms and sometimes packaging the tools for convenient installation by the end user. These packages contain the binary executables of the tools, pre-made and ready to use. These packages are known as "distributions" of the AVR toolchain, or by a more shortened name, "distros".

AVR toolchain distros are available on FreeBSD, Windows, Mac OS X, and certain flavors of Linux.

## 2.13 Open Source

All of these tools, from the original source code in the multitude of projects, to the various distros, are put together by many, many volunteers. All of these projects could always use more help from other people who are willing to volunteer some of their time. There are many different ways to help, for people with varying skill levels, abilities, and available time.

You can help to answer questions in mailing lists such as the `avr-gcc-list`, or on forums at the AVR Freaks website. This helps many people new to the open source AVR tools.

If you think you found a bug in any of the tools, it is always a big help to submit a good bug report to the proper project. A good bug report always helps other volunteers to analyze the problem and to get it fixed for future versions of the software.

You can also help to fix bugs in various software projects, or to add desirable new features.

Volunteers are always welcome! :-)

# 3 Memory Areas and Using `malloc()`

## 3.1 Introduction

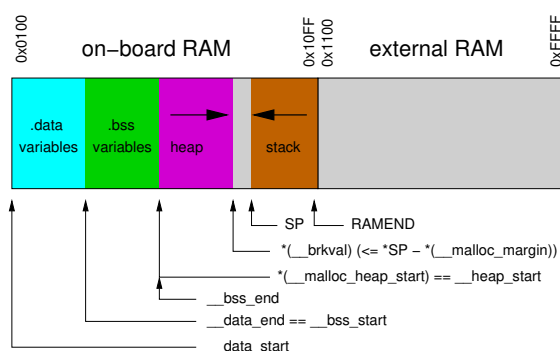
Many of the devices that are possible targets of AVR-LibC have a minimal amount of RAM. The smallest parts supported by the C environment come with 128 bytes of RAM. This needs to be shared between initialized and uninitialized variables ([sections](#) `.data` and `.bss`), the dynamic memory allocator, and the stack that is used for calling subroutines and storing local (automatic) variables.

Also, unlike larger architectures, there is no hardware-supported memory management which could help in separating the mentioned RAM regions from being overwritten by each other.

The standard RAM layout is to place `.data` variables first, from the beginning of the internal RAM, followed by `.bss`. The stack is started from the top of internal RAM, growing downwards. The so-called "heap" available for the dynamic memory allocator will be placed beyond the end of `.bss`. Thus, there's no risk that dynamic memory will ever collide with the RAM variables (unless there were bugs in the implementation of the allocator). There is still a risk that the heap and stack could collide if there are large requirements for either dynamic memory or stack space. The former can even happen if the allocations aren't all that large but dynamic memory allocations get fragmented over time such that new requests don't quite fit into the "holes" of previously freed regions. Large stack space requirements can arise in a C function containing large and/or numerous local variables or when recursively calling function.

## Note

The pictures shown in this document represent typical situations where the RAM locations refer to an ATmega128. The memory addresses used are not displayed in a linear scale.



**Figure 1** RAM map of a device with internal RAM

On a simple device like a microcontroller it is a challenge to implement a dynamic memory allocator that is simple enough so the code size requirements will remain low, yet powerful enough to avoid unnecessary memory fragmentation and to get it all done with reasonably few CPU cycles. Microcontrollers are often low on space and also run at much lower speeds than the typical PC these days.

The memory allocator implemented in AVR-LibC tries to cope with all of these constraints, and offers some tuning options that can be used if there are more resources available than in the default configuration.

## 3.2 Internal vs. external RAM

Obviously, the constraints are much harder to satisfy in the default configuration where only internal RAM is available. Extreme care must be taken to avoid a stack-heap collision, both by making sure functions aren't nesting too deeply, and don't require too much stack space for local variables, as well as by being cautious with allocating too much dynamic memory.

If external RAM is available, it is strongly recommended to move the heap into the external RAM, regardless of whether or not the variables from the .data and .bss sections are also going to be located there. The stack should always be kept in internal RAM. Some devices even require this, and in general, internal RAM can be accessed faster since no extra wait states are required. When using dynamic memory allocation and stack and heap are separated in distinct memory areas, this is the safest way to avoid a stack-heap collision.

## 3.3 Tunables for malloc()

There are a number of variables that can be tuned to adapt the behavior of malloc() to the expected requirements and constraints of the application. Any changes to these tunables should be made before the very first call to malloc(). Note that some library functions might also use dynamic memory (notably those from the [<stdio.h>: Standard IO facilities](#)), so make sure the changes will be done early enough in the startup sequence.

The variables `__malloc_heap_start` and `__malloc_heap_end` can be used to restrict the malloc() function to a certain memory region. These variables are statically initialized to point to `__heap_start` and `__heap_end`, respectively, where `__heap_start` is filled in by the linker to point just beyond .bss, and `__heap_end` is set to 0 which makes malloc() assume the heap is below the stack.

If the heap is going to be moved to external RAM, `__malloc_heap_end` *must* be adjusted accordingly. This can either be done at run-time, by writing directly to this variable, or it can be done automatically at link-time, by adjusting the value of the symbol `__heap_end`.

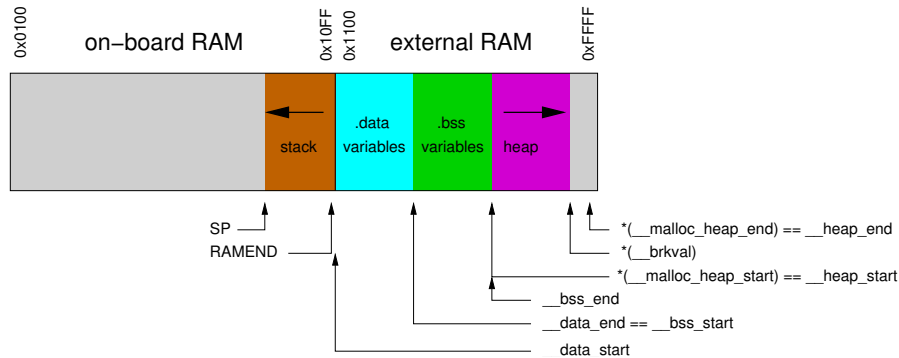
The following example shows a linker command to relocate the entire .data and .bss segments, and the heap to location 0x1100 in external RAM. The heap will extend up to address 0xffff.

```
avr-gcc ... -Wl,--section-start,.data=0x801100,--defsym=__heap_end=0x80ffff ...
```

## Note

See [explanation](#) for offset 0x800000. See the chapter about [using gcc](#) for the `-Wl` options.

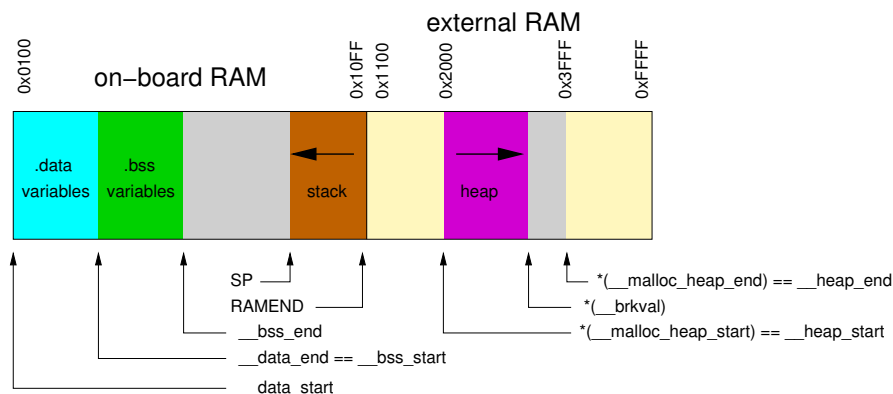
The ld (linker) user manual states that using `-Tdata=<x>` is equivalent to using `--section-start,.data=<x>`. However, you have to use `--section-start` as above because the GCC frontend also sets the `-Tdata` option for all MCU types where the SRAM doesn't start at 0x800060. Thus, the linker is being faced with two `-Tdata` options. Starting with binutils 2.16, the linker changed the preference, and picks the "wrong" option in this situation.



**Figure 2 Internal RAM: stack only, external RAM: variables and heap**

If dynamic memory should be placed in external RAM, while keeping the variables in internal RAM, something like the following could be used. Note that for demonstration purposes, the assignment of the various regions has not been made adjacent in this example, so there are "holes" below and above the heap in external RAM that remain completely inaccessible by regular variables or dynamic memory allocations (shown in light bisque color in the picture below).

```
avr-gcc ... -Wl,--defsym=__heap_start=0x802000,--defsym=__heap_end=0x803fff ...
```



**Figure 3 Internal RAM: variables and stack, external RAM: heap**

If `__malloc_heap_end` is 0, the allocator attempts to detect the bottom of stack in order to prevent a stack-heap collision when extending the actual size of the heap to gain more space for dynamic memory. It will not try to go beyond the current stack limit, decreased by `__malloc_margin` bytes. Thus, all possible stack frames of interrupt routines that could interrupt the current function, plus all further nested function calls must not require more stack space, or they will risk colliding with the data segment.

The default value of `__malloc_margin` is set to 32.

### 3.4 Implementation details

Dynamic memory allocation requests will be returned with a two-byte header prepended that records the size of the allocation. This is later used by `free()`. The returned address points just beyond that header. Thus, if the application accidentally writes before the returned memory region, the internal consistency of the memory allocator is compromised.

The implementation maintains a simple freelist that accounts for memory blocks that have been returned in previous calls to `free()`. Note that all of this memory is considered to be successfully added to the heap already, so no further checks against stack-heap collisions are done when recycling memory from the freelist.

The freelist itself is not maintained as a separate data structure, but rather by modifying the contents of the freed memory to contain pointers chaining the pieces together. That way, no additional memory is required to maintain this list except for a variable that keeps track of the lowest memory segment available for reallocation. Since both, a chain pointer and the size of the chunk need to be recorded in each chunk, the minimum chunk size on the freelist is four bytes.

When allocating memory, first the freelist is walked to see if it could satisfy the request. If there's a chunk available on the freelist that will fit the request exactly, it will be taken, disconnected from the freelist, and returned to the caller. If no exact match could be found, the closest match that would just satisfy the request will be used. The chunk will normally be split up into one to be returned to the caller, and another (smaller) one that will remain on the freelist. In case this chunk was only up to two bytes larger than the request, the request will simply be altered internally to also account for these additional bytes since no separate freelist entry could be split off in that case.

If nothing could be found on the freelist, heap extension is attempted. This is where `__malloc_margin` will be considered if the heap is operating below the stack, or where `__malloc_heap_end` will be verified otherwise.

If the remaining memory is insufficient to satisfy the request, `NULL` will eventually be returned to the caller.

When calling `free()`, a new freelist entry will be prepared. An attempt is then made to aggregate the new entry with possible adjacent entries, yielding a single larger entry available for further allocations. That way, the potential for heap fragmentation is hopefully reduced. When deallocating the topmost chunk of memory, the size of the heap is reduced.

A call to `realloc()` first determines whether the operation is about to grow or shrink the current allocation. When shrinking, the case is easy: the existing chunk is split, and the tail of the region that is no longer to be used is passed to the standard `free()` function for insertion into the freelist. Checks are first made whether the tail chunk is large enough to hold a chunk of its own at all, otherwise `realloc()` will simply do nothing, and return the original region.

When growing the region, it is first checked whether the existing allocation can be extended in-place. If so, this is done, and the original pointer is returned without copying any data contents. As a side-effect, this check will also record the size of the largest chunk on the freelist.

If the region cannot be extended in-place, but the old chunk is at the top of heap, and the above freelist walk did not reveal a large enough chunk on the freelist to satisfy the new request, an attempt is made to quickly extend this topmost chunk (and thus the heap), so no need arises to copy over the existing data. If there's no more space available in the heap (same check is done as in `malloc()`), the entire request will fail.

Otherwise, `malloc()` will be called with the new request size, the existing data will be copied over, and `free()` will be called on the old region.

## 4 Memory Sections

Sections are used to organize code and data of a program on the binary level.

The (compiler-generated) assembly code assigns code, data and other entities like debug information to so called input sections. These sections serve as input to the linker, which bundles similar sections together to output sections like `.text` and `.data` according to rules defined in the linker description file.

The final ELF binary is then used by programming tools like `avrdude`, simulators, debuggers and other programs, for example programs from the GNU Binutils family like `avr-size`, `avr-objdump` and `avr-readelf`.

Sections may have extra properties like [section alignment](#), [section flags](#), [section type](#) and rules to locate them or to assign them to [memory regions](#).

- [Concepts](#)
  - [Named Sections](#)
    - \* [Section Flags](#)
    - \* [Section Type](#)
    - \* [Section Alignment](#)
    - \* [Subsections](#)
  - [Orphan Sections](#)
  - [LMA: Load Memory Address](#)
  - [VMA: Virtual Memory Address](#)
- [The Linker Script: Building Blocks](#)
  - [Input Sections and Output Sections](#)
  - [Memory Regions](#)
- [Output Sections of the Default Linker Script](#)
  - [.text](#)
  - [.data](#)
  - [.bss](#)
  - [.noinit](#)
  - [.rodata](#)
  - [.eeprom](#)
  - [.fuse](#), [.lock](#) and [.signature](#)
  - [.note.gnu.avr.deviceinfo](#)
- [Symbols in the Default Linker Script](#)
- [Output Sections and Code Size](#)
- [Using Sections](#)
  - [In C/C++ Code](#)
  - [In Assembly Code](#)



## 4.1 Concepts

### 4.1.1 Named Sections

*Named sections* are sections that can be referred to by their name. The name and other properties can be provided with the `.section` directive like in

```
.section name, "flags", @type
```

or with the `.pushsection` directive, which directs the assembler to assemble the following code into the named section.

An example of a section that is not referred to by its name is the COMMON section. In order to put an object in that section, special directives like `.comm name, size` or `.lcomm name, size` have to be used.

Directives like `.text` are basically the same like `.section .text`, where the assembler assumes appropriate section flags and type; same for directives `.data` and `.bss`.

**4.1.1.1 Section Flags** The *section flags* can be specified with the `.section` and `.pushsection` directives, see [section type](#) for an example. Section flags of output sections can be specified in the linker description file, and the linker implements heuristics to determine the section flags of output sections from the various input section that go into it.

**Table 1 Section Flags**

Flag	Meaning
a	The section will be <b>allocated</b> , i.e. it occupies space on the target hardware
w	The section contains data that can be <b>written</b> at run-time. Sections that only contain read-only entities don't have the <code>w</code> flag set
x	The section contains <b>executable</b> code, though the section may also contain non-executable objects
M	A <b>mergeable</b> section
S	A <b>string</b> section
G	A section <b>group</b> , like used with <code>comdat</code> objects

The last three flags are listed for completeness. They are used by the compiler, for example for header-only C++ modules and to ensure that multiple instantiations of the same template in different compilation units does occur at most once in the executable file.

**4.1.1.2 Section Type** The *section type* can be specified with the `.section` and `.pushsection` directives, like in

```
.section .text.myfunc,"ax",@progbits
.pushsection ".data.myvar", "a", "progbits"
```

On ELF level, the section type is stored in the section header like `Elf32_Shdr.sh_type = SHT_PROGBITS`.

**Table 2 Section Types**

Type	Meaning
@progbits	The section contains data that will be loaded to the target, like objects in the <code>.text</code> and <code>.data</code> sections.
@nobits	The section does not contain data that needs to be transferred to the target device, like data in the <code>.bss</code> and <code>.noinit</code> sections. The section still occupies space on the target.
@note	The section is a note, like for example the <code>.note.gnu.avr.deviceinfo</code> section.

**4.1.1.3 Section Alignment** The *alignment* of a section is the maximum over the alignments of the objects in the section.

**4.1.1.4 Subsections** *Subsections* are compartments of named sections and are introduced with the `.subsection` directive. Subsections are located in order of increasing index in their input section. The default subsection after switching to a new section is subsection 0.

#### Note

A common misconception is that a section like `.text.module.func` were a subsection of `.text.module`. This is not the case. These two sections are independent, and there is no subset relation. The sections may have different flags and type, and they may be assigned to different output sections.

## 4.1.2 Orphan Sections

*Orphan sections* are sections that are not mentioned in the linker description file. When an input section is orphan, then the GNU linker implicitly generates an output section of the same name. The linker implements various heuristics to determine sections flags, section type and location of orphaned sections. One use of orphan sections is to [locate code to a fixed address](#).

Like for any other output section, the start address can be specified by means of linking with `-Wl, --section-start, secname=`

## 4.1.3 LMA: Load Memory Address

The LMA of an object is the address where a loader like `avrdude` puts the object when the binary is being uploaded to the target device.

## 4.1.4 VMA: Virtual Memory Address

The VMA is the address of an object as used by the running program.

VMA and LMA may be different: Suppose a small ATmega8 program with executable code that extends from byte address 0x0 to 0x20f, and one variable `my_var` in static storage. The default linker script puts the content of the `.data` output section after the `.text` output section and into the `text` [segment](#). The startup code then copies `my_data` from its LMA location beginning at 0x210 to its VMA location beginning at 0x800060, because C/C++ requires that all data in static storage must have been initialized when `main` is entered.

The internal SRAM of ATmega8 starts at RAM address 0x60, which is offset by 0x800000 in order to linearize the address space (VMA 0x60 is a flash address). The AVR program only ever uses the lower 16 bits of VMAs in static storage so that the offset of 0x800000 is masked out. But code like `"LDI r24, hh8(my_data)"` actually sets R24 to 0x80 and reveals that `my_data` is an object located in RAM.

## 4.2 The Linker Script: Building Blocks

The linker description file is the central hub to channel functions and static storage objects of a program to the various memory spaces and address ranges of a device.

## 4.2.1 Input Sections and Output Sections

*Input sections* are sections that are inputs to the linker. Functions and static variables but also additional notes and debug information are assigned to different input sections by means of [assembler directives](#) like `.section` or `.text`. The linker takes all these sections and assigns them to output sections as specified in the linker script.

*Output sections* are defined in the linker description file. Contrary to the unlimited number of input sections a program can come up with, there is only a handful of output sections like `.text` and `.data`, that roughly correspond to the memory spaces of the target device.

One step in the final link is to *locate* the sections, that is the linker/locator determines at which memory location to put the output sections, and how to arrange the many input sections within their assigned output section. *Locating* means that the linker assigns [Load Memory Addresses](#) — addresses as used by a loader like `avrdude` — and [Virtual Memory Addresses](#), which are the addresses as used by the running program.

While it is possible to directly assign LMAs and VMAs to output sections in the linker script, the default linker scripts provided by Binutils assign *memory regions* (aka. *memory segments*) to the output sections. This has some advantages like a linker script that is easier to maintain. An output section can be assigned to more than one memory region. For example, non-zero data in static storage (`.data`) goes to

1. the `data` region (VMA), because such variables occupy RAM which has to be allocated
2. the `text` region (LMA), because the initializers for such data has to be kept in some non-volatile memory (program ROM), so that the startup code can initialize that data so that the variables have their expected initial values when `main()` is entered.

The `SECTIONS{ }` portion of a linker script models the input and output section, and it assigns the output section to the memory regions defined in the `MEMORY{ }` part.

## 4.2.2 Memory Regions

The *memory regions* defined in the default linker script model and correspond to the different kinds of memories of a device.

**Table 3 Memory Regions of the Default Linker Script**

Region	Virtual Address <sup>1</sup>	Flags	Purpose
<code>text</code>	0 <sup>2</sup>	<code>rx</code>	Executable code, vector table, data in <code>PROGMEM</code> , <code>__flash</code> and <code>__memx</code> , startup code, linker stubs, initializers for <code>.data</code>
<code>data</code>	0x800000 <sup>2</sup>	<code>rw</code>	Data in static storage
<code>rodata</code> <sup>3</sup>	0xa00000 <sup>2</sup>	<code>r</code>	Read-only data in static storage
<code>eeprom</code>	0x810000	<code>rw</code>	EEPROM data
<code>fuse</code>	0x820000	<code>rw</code>	Fuse bytes
<code>lock</code>	0x830000	<code>rw</code>	Lock bytes
<code>signature</code>	0x840000	<code>rw</code>	Device signature
<code>user_signatures</code>	0x850000	<code>rw</code>	User signature

### Notes

1. The VMAs for regions other than `text` are offset in order to linearize the non-linear memory address space

of the AVR Harvard architecture. The target code only ever uses the lower 16 bits of the VMA to access objects in non-`text` regions.

- The addresses for regions `text`, `data` and `rodata` are actually defined as symbols like `__TEXT__↔REGION_ORIGIN__`, so that they can be adjusted by means of, say `-Wl,--defsym,__DATA__↔REGION_ORIGIN__=0x800060`. Same applies for the lengths of all the regions, which is `__NAME__↔_REGION_LENGTH__` for region `name`.
- The `rodata` region is only present in the `avrxcmega2_flmap` and `avrxcmega4_flmap` emulations, which is the case for Binutils since v2.42 for the AVR64 and AVR128 devices without `-mrodata-in-ram`.

### 4.3 Output Sections of the Default Linker Script

This section describes the various [output sections](#) defined in the default linker description files.

**Table 4 Output Sections and Memory Regions**

Output Section	Purpose	Memory Region	
		LMA	VMA
<code>.text</code>	Executable code, data in progmem	text	text
<code>.data</code>	Non-zero data in static storage	text	data
<code>.bss</code>	Zero data in static storage	—	data
<code>.noinit</code>	Non-initialized data in static storage	—	data
<code>.rodata</code> <sup>1</sup>	Read-only data in static storage	text	LMA + offset <sup>3</sup>
<code>.rodata</code> <sup>2</sup>	Read-only data in static storage	0x8000 * <code>__flmap</code> <sup>4</sup>	rodata
<code>.eeprom</code>	Data in EEPROM	Note <sup>5</sup>	eeprom
<code>.fuse</code>	Fuse bytes		fuse
<code>.lock</code>	Lock bytes		lock
<code>.signature</code>	Signature bytes		signature
	User signature bytes		user_signatures

#### Notes

- On `avrxcmega3` and `avrtiny` devices.
- On `AVR64` and `AVR128` devices without `-mrodata-in-ram`.
- With an offset `__RODATA_PM_OFFSET__` of 0x4000 or 0x8000 depending on the device.
- The value of symbol `__flmap` defaults to the last 32 KiB block of program memory, see the GCC [v14 release notes](#).
- The **LMA** actually equals the **VMA**, but is unused. The flash loader like `avrdude` knows where to put the data,

#### 4.3.1 The `.text` Output Section

The `.text` output section contains the actual machine instructions which make up the program, but also additional code like jump tables and lookup tables placed in program memory with the **PROGMEM** attribute.

The `.text` output section contains the input sections described below. Input sections that are not used by the tools are omitted. A `*` wildcard stands for any sequence of characters, including empty ones, that are valid in a section name.

**.vectors** The `.vectors` sections contains the interrupt vector table which consists of jumps to [weakly](#) defined labels: To `__init` for the first entry at index 0, and to `__vector_N` for the entry at index  $N \geq 1$ . The default value for `__vector_N` is `__bad_interrupt`, which jumps to weakly defined `__vector_default`, which jumps to `__vectors`, which is the start of the `.vectors` section.

Implementing an interrupt service routine (ISR) is performed with the help of the [ISR](#) macro in C/C++ code.

**.progmem.data**

**.progmem.data.\***

**.progmem.gcc.\*** This section is used for read-only data declared with attribute [PROGMEM](#), and for data in address-space `__flash`.

The compiler assumes that the `.progmem` sections are located in the lower 64 KiB of program memory. When it does not fit in the lower 64 KiB block, then the program reads garbage except `pgm_read*_far` is used. In that case however, code can be located in the `.progmemx` section which does not require to be located in the lower program memory.

**.trampolines** Linker stubs for indirect jumps and calls on devices with more than 128 KiB of program memory. This section must be located in the same 128 KiB block like the interrupt vector table. For some background on linker stubs, see the GCC documentation on [EIND](#).

**.text**

**.text.\*** Executable code. This is where almost all of the executable code of an application will go.

**.ctors**

**.dtors** Tables with addresses of static constructors and destructors, like C++ static constructors and functions declared with attribute `constructor`.

**The .initN Sections** These sections are used to hold the startup code from reset up through the start of `main()`.

The `.initN` sections are executed in order from 0 to 9: The code from one init section falls through to the next higher init section. This is the reason for why code in these sections must be naked (more precisely, it must not contain return instructions), and why code in these sections must never be called explicitly.

When several modules put code in the same init section, the order of execution is not specified.

**Table 5 The .initN Sections**

Section	Performs	Hosted By	Symbol <sup>1</sup>
<code>.init0</code>	Weakly defines the <code>__init</code> label which is the jump target of the first vector in the interrupt vector table. When the user defines the <code>__init()</code> function, it will be jumped to instead.	AVR-LibC <sup>2</sup>	
<code>.init1</code>	Unused	—	
<code>.init2</code>	<ul style="list-style-type: none"> <li>Clears <code>__zero_reg__</code></li> <li>Initializes the stack pointer to the value of weak symbol <code>__stack</code>, which has a default value of <code>RAMEND</code> as defined in <a href="#">avr/io.h</a></li> <li>Initializes <code>EIND</code> to <code>hh8(pgm(__↔vectors))</code> on devices that have it</li> <li>Initializes <code>RAMPX</code>, <code>RAMPY</code>, <code>RAMPZ</code> and <code>RAMPD</code> on devices that have all of them</li> </ul>	AVR-LibC	
<code>.init3</code>	Initializes the <code>NVMCTRLB.FLMAP</code> bit-field on devices that have it, except when <code>-mrodata-in-ram</code> is specified	AVR-LibC	<code>__do_flmap_init</code>

Section	Performs	Hosted By	Symbol <sup>1</sup>
.init4	Initializes data in static storage: Initializes <code>.data</code> and clears <code>.bss</code>	libgcc	<code>__do_copy_data</code> <code>__do_clear_bss</code>
.init5	Unused	—	
.init6	Run static C++ constructors and functions defined with <code>__attribute__((constructor))</code> .	libgcc	<code>__do_global_ctors</code>
.init7	Unused	—	
.init8	Unused	—	
.init9	Calls <code>main</code> and then jumps to <code>exit</code>	AVR-LibC	

## Notes

- Code in the `.init3`, `.init4` and `.init6` sections is optional; it will only be present when there is something to do. This will be tracked by the compiler — or has to be tracked by the assembly programmer — which pulls in the code from the respective library by means of the mentioned symbols, e.g. by linking with `-Wl, -u, __do_flmap_init` or by means of `.global __do_copy_data`.  
Conversely, when the respective code is not desired for some reason, the symbol can be satisfied by defining it with, say, `-Wl, --defsym, __do_copy_data=0` so that the code is not pulled in any more.
- The code is provided by `gcrt1.S`.

**The `.finiN` Sections** Shutdown code. These sections are used to hold the exit code executed after return from `main()` or a call to `exit()`.

The `.finiN` sections are executed in descending order from 9 to 0 in a fallthrough manner.

Table 6 The `.finiN` Sections

Section	Performs	Hosted By	Symbol
.fini9	Defines <code>_exit</code> and weakly defines the <code>exit</code> label	libgcc	
.fini8	Run functions registered with <code>atexit()</code>	AVR-LibC	
.fini7	Unused	—	
.fini6	Run static C++ destructors and functions defined with <code>__attribute__((destructor))</code>	libgcc	<code>__do_global_dtors</code>
.fini5...1	Unused	—	
.fini0	Globally disables interrupts and enters an infinite loop to label <code>__stop_program</code>	libgcc	

It is unlikely that ordinary code uses the `fini` sections. When there are no static destructors and `atexit()` is not used, then the respective code is not pulled in from the libraries, and the `fini` code just consumes four bytes: a `CLI` and a `RJMP` to itself. Common use cases of `fini` code is when running the GCC test suite where it reduces fallout, and in simulators to determine (un)orderly termination of a simulated program.

**`.progmemx.*`** Read-only data in program memory without the requirement that it must reside in the lower 64 KiB. The compiler uses this section for data in the named address-space `__memx`. Data can be accessed with `pgm_read_*_far` when it is not in a named address-space:

```
#include <avr/pgmspace.h>

const __memx int array1[] = { 1, 4, 9, 16, 25, 36 };

PROGMEM_FAR
const int array2[] = { 2, 3, 5, 7, 11, 13, 17 };

int add (uint8_t id1, uint8_t id2)
{
    uint_farptr_t p_array2 = pgm_get_far_address (array2);
```

```

    int val2 = pgm_read_int_far (p_array2 + sizeof(int) * id2);
    return val2 + array1[id1];
}

```

**.jumpables\*** Used to place jump tables in some cases.

### 4.3.2 The .data Output Section

This section contains data in static storage which has an initializer that is not all zeroes. This includes the following input sections:

**.data\*** Read-write data

**.rodata\*** Read-only data. These input sections are only included on devices that host read-only data in RAM.

It is possible to tell the linker the SRAM address of the beginning of the `.data` section. This is accomplished by linking with

```
avr-gcc ... -Tdata addr -Wl,--defsym,__DATA_REGION_START__=addr
```

Note that `addr` must be [offset](#) by adding 0x800000 to the real SRAM address so that the linker knows that the address is in the SRAM memory segment. Thus, if you want the `.data` section to start at 0x1100, pass 0x801100 as the address to the linker.

#### Note

When using `malloc()` in the application (which could even happen inside library calls), [additional adjustments](#) are required.

### 4.3.3 The .bss Output Section

Data in static storage that will be zeroed by the startup code. This are data objects without explicit initializer, and data objects with initializers that are all zeroes.

Input sections are `.bss*` and `COMMON`. Common symbols are defined with directives `.comm` or `.lcomm`.

### 4.3.4 The .noinit Output Section

Data objects in static storage that should not be initialized by the startup code. As the C/C++ standard requires that *all* data in static storage is initialized — which includes data without explicit initializer, which will be initialized to all zeroes — such objects have to be put into section `.noinit` by hand:

```
__attribute__((section(".noinit")))
int foo;
```

The only input section in this output section is `.noinit`. Only data without initializer can be put in this section.

### 4.3.5 The .rodata Output Section

This section contains read-only data in static storage from `.rodata*` input sections. This output section is only present for devices where read-only data remains in program memory, which are the devices where (parts of) the program memory are visible in the RAM address space. This is currently the case for the emulations `avrtiny`, `avrxcmega3`, `avrxcmega2_flmap` and `avrxcmega4_flmap`.

### 4.3.6 The .eeprom Output Section

This is where EEPROM variables are stored, for example variables declared with the `EEMEM` attribute. The only input section (pattern) is `.eeprom*`.

### 4.3.7 The .fuse, .lock and .signature Output Sections

These sections contain fuse bytes, lock bytes and device signature bytes, respectively. The respective input section patterns are `.fuse*`, `.lock*` and `.signature*`.

### 4.3.8 The .note.gnu.avr.deviceinfo Section

This section is actually *not mentioned* in the default linker script, which means it is an [orphan section](#) and hence the respective output section is implicit.

The startup code from AVR-LibC puts device information in that section to be picked up by simulators or tools like `avr-size`, `avr-objdump`, `avr-readelf`, etc,

The section is contained in the ELF file but not loaded onto the target. Source of the device specific information are the device header file and compiler builtin macros. The layout conforms to the standard [ELF note section layout](#) and is laid out as follows.

```
#include <elf.h>

typedef struct
{
    Elf32_Word n_namesz;      /* AVR_NOTE_NAME_LEN */
    Elf32_Word n_descsz;     /* size of avr_desc */
    Elf32_Word n_type;       /* 1 - the only AVR note type */
} Elf32_Nhdr;

#define AVR_NOTE_NAME_LEN 4

struct note_gnu_avr_deviceinfo
{
    Elf32_Nhdr nhdr;
    char note_name[AVR_NOTE_NAME_LEN]; /* = "AVR\0" */

    struct
    {
        Elf32_Word flash_start;
        Elf32_Word flash_size;
        Elf32_Word sram_start;
        Elf32_Word sram_size;
        Elf32_Word eeprom_start;
        Elf32_Word eeprom_size;
        Elf32_Word offset_table_size;
        /* Offset table containing byte offsets into
         string table that immediately follows it.
         index 0: Device name byte offset */
        Elf32_Off offset_table[1];
        /* Standard ELF string table.
         index 0 : NULL
         index 1 : Device name
         index 2 : NULL */
        char strtabs[2 + strlen(__AVR_DEVICE_NAME__)];
    } avr_desc;
};
```

The contents of this section can be displayed with

- `avr-objdump -P avr-deviceinfo file`, which is supported since Binutils v2.43.
- `avr-readelf -n file`, which displays all notes.



## 4.4 Symbols in the Default Linker Script

Most of the symbols like `main` are defined in the code of the application, but some symbols are defined in the default linker script:

**\_\_name\_REGION\_ORIGIN\_\_** Describes the physical properties of memory region *name*, where *name* is one of `TEXT` or `DATA`. The address is a VMA and offset as explained above.

The linker script only supplies a default for the symbol values when they have not been defined by other means, like for example in the startup code or by `--defsym`. For example, to let the code start at address `0x100`, one can link with

```
avr-gcc ... -Ttext=0x100 -Wl,--defsym,__TEXT_REGION_ORIGIN_=0x100
```

**\_\_name\_REGION\_LENGTH\_\_** Describes the physical properties of memory region *name*, where *name* is one of: `TEXT`, `DATA`, `EEPROM`, `LOCK`, `FUSE`, `SIGNATURE` or `USER_SIGNATURE`.

Only a default is supplied when the symbol is not yet defined by other means. Most of these symbols are **weakly** defined in the startup code.

**\_\_data\_start**

**\_\_data\_end** Start and (one past the) end VMA address of the `.data` section in RAM.

**\_\_data\_load\_start**

**\_\_data\_load\_end** Start and (one past the) end LMA address of the `.data` section initializers located in program memory. Used together with the VMA addresses above by the **startup code** to copy data initializers from program memory to RAM.

**\_\_bss\_start**

**\_\_bss\_end** Start and (one past the) end VMA address of the `.bss` section. The startup code clears this part of the RAM.

**\_\_rodata\_start**

**\_\_rodata\_end**

**\_\_rodata\_load\_start**

**\_\_rodata\_load\_end** Start and (one past the) end VMA resp. LMA address of the `.rodata` output section. These symbols are only defined when `.rodata` is not output to the `text` region, which is the case for emulations `avrxcmega2_flmap` and `avrxcmega4_flmap`.

**\_\_heap\_start** One past the last object located in static storage. Immediately follows the `.noinit` section (which immediately follows `.bss`, which immediately follows `.data`). Used by `malloc()` and friends.

Code that computes a checksum over all relevant code and data in program memory has to consider:

- The range from the beginning of the `.text` section (address `0x0` in the default layout) up to `__data_load_start`.
- For emulations that have the `rodata` **memory region**, the range from `__rodata_load_start` to `__rodata_load_end` has also to be taken into account.

## 4.5 Output Sections and Code Size

The `avr-size` program (part of Binutils), coming from a Unix background, doesn't account for the `.data` initialization space added to the `.text` section, so in order to know how much flash the final program will consume, one needs to add the values for both, `.text` and `.data` (but not `.bss`), while the amount of pre-allocated SRAM is the sum of `.data` and `.bss`.

Memory usage and free memory can also be displayed with

```
avr-objdump -P mem-usage code.elf
```

## 4.6 Using Sections

### 4.6.1 In C/C++ Code

The following example shows how to read and reset the MCUCR special function register on ATmega328. This SFR holds to reset source like "watchdog reset" or "external reset", and should be read early, prior to the initialization of RAM and execution of static constructors which may take some time. This means the code has to be placed prior to `.init4` which initializes static storage, but after `.init2` which initializes `__zero_reg__`. As the code runs prior to the initialization of static storage, variable `mcucr` must be placed in section `.noinit` so that it won't be overridden by that part of the startup code:

```
#include <avr/io.h>

__attribute__((section(".noinit")))
uint8_t mcucr;

__attribute__((used, unused, naked, section(".init3")))
static void read_MCUCR (void)
{
    mcucr = MCUCR;
    MCUCR = 0;
}
```

- The `used` attribute tells the compiler that the function is used although it is never called.
- The `unused` attribute tells the compiler that it is fine that the function is unused, and silences respective diagnostics about the seemingly unused functions.
- The `naked` attribute is required because the code is located in an `init` section. The function *must not have a RET statement* because the function is never called. According to the GCC documentation, the only code supported in naked functions is inline assembly, but the code above is simple enough so that GCC can deal with it.

### 4.6.2 In Assembly Code

Example:

```
#include <avr/io.h>

.section .init3,"ax",@progbits
    lds    r0, MCUCR

.pushsection .noinit,"a",@nobits
mcucr:
    .type  mcucr, @object
    .size  mcucr, 1
    .space 1
.popsection                ; Proceed with .init3

    sts    mcucr, r0
    sts    MCUCR, __zero_reg__ ; Initialized in .init2

.text
.global main
.type    main, @function
lds     r24, mcucr
clr     r25
rjmp    putchar
.size   main, .-main
```

- The `"ax"` [flags](#) tells that the sections is [allocatable](#) (consumes space on the target hardware) and is [executable](#).
- The `@progbits` [type](#) tells that the section contains bits that have to be uploaded to the target hardware.

For more details, see the see the gas user manual on the `.section` directive.

## 5 Data in Program Space

### 5.1 Introduction

So you have some constant data and you're running out of room to store it? Many AVR's have limited amount of RAM in which to store data, but may have more Flash space available. The AVR is a Harvard architecture processor, where Flash is used for the program, RAM is used for data, and they each have separate address spaces. It is a challenge to get constant data to be stored in the Program Space, and to retrieve that data to use it in the AVR application.

The problem is exacerbated by the fact that the C Language was not designed for Harvard architectures, it was designed for Von Neumann architectures where code and data exist in the same address space. This means that any compiler for a Harvard architecture processor, like the AVR, has to use other means to operate with separate address spaces.

Some compilers use non-standard C language keywords, or they extend the standard syntax in ways that are non-standard. The AVR toolset takes a different approach.

GCC has a special keyword, `__attribute__` that is used to attach different attributes to things such as function declarations, variables, and types. This keyword is followed by an attribute specification in double parentheses. In AVR GCC, there is a special attribute called `progmem`. This attribute is use on data declarations, and tells the compiler to place the data in the Program Memory (Flash).

AVR-LibC provides a simple macro `PROGMEM` that is defined as the attribute syntax of GCC with the `progmem` attribute. This macro was created as a convenience to the end user, as we will see below. The `PROGMEM` macro is defined in the `<avr/pgmspace.h>` system header file.

It is difficult to modify GCC to create new extensions to the C language syntax, so instead, AVR-LibC has created macros to retrieve the data from the Program Space. These macros are also found in the `<avr/pgmspace.h>` system header file.

### 5.2 A Note On `const`

Many users bring up the idea of using C's keyword `const` as a means of declaring data to be in Program Space. Doing this would be an abuse of the intended meaning of the `const` keyword.

`const` is used to tell the compiler that the data is to be "read-only". It is used to help make it easier for the compiler to make certain transformations, or to help the compiler check for incorrect usage of those variables.

For example, the `const` keyword is commonly used in many functions as a modifier on the parameter type. This tells the compiler that the function will only use the parameter as read-only and will not modify the contents of the parameter variable.

`const` was intended for uses such as this, not as a means to identify where the data should be stored. If it were used as a means to define data storage, then it loses its correct meaning (changes its semantics) in other situations such as in the function parameter example.

## 5.3 Storing and Retrieving Data in the Program Space

Let's say you have some global data:

```
unsigned char mydata[11][10] =
{
    {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09},
    {0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13},
    {0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D},
    {0x1E, 0x1F, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27},
    {0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31},
    {0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B},
    {0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x42, 0x43, 0x44, 0x45},
    {0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F},
    {0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59},
    {0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F, 0x60, 0x61, 0x62, 0x63},
    {0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D}
};
```

and later in your code you access this data in a function and store a single byte into a variable like so:

```
byte = mydata[i][j];
```

Now you want to store your data in Program Memory. Use the `PROGMEM` macro found in `<avr/pgmspace.h>` and put it after the declaration of the variable, but before the initializer, like so:

```
#include <avr/pgmspace.h>
.
.
.
const unsigned char mydata[11][10] PROGMEM =
{
    {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09},
    {0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13},
    {0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D},
    {0x1E, 0x1F, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27},
    {0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31},
    {0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B},
    {0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x42, 0x43, 0x44, 0x45},
    {0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F},
    {0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59},
    {0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F, 0x60, 0x61, 0x62, 0x63},
    {0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D}
};
```

That's it! Now your data is in the Program Space. You can compile, link, and check the map file to verify that `mydata` is placed in the correct section.

Now that your data resides in the Program Space, your code to access (read) the data will no longer work. The code that gets generated will retrieve the data that is located at the address of the `mydata` array, plus offsets indexed by the `i` and `j` variables. However, the final address that is calculated where to retrieve the data points to the Data Space! Not the Program Space where the data is actually located. It is likely that you will be retrieving some garbage. The problem is that AVR GCC does not intrinsically know that the data resides in the Program Space.

The solution is fairly simple. The "rule of thumb" for accessing data stored in the Program Space is to access the data as you normally would (as if the variable is stored in Data Space), like so:

```
byte = mydata[i][j];
```

then take the address of the data:

```
byte = &(mydata[i][j]);
```

then use the appropriate `pgm_read_*` macro, and the address of your data becomes the parameter to that macro:

```
byte = pgm_read_byte(&(mydata[i][j]));
```

The `pgm_read_*` macros take an address that points to the Program Space, and retrieves the data that is stored at that address. This is why you take the address of the offset into the array. This address becomes the parameter to the macro so it can generate the correct code to retrieve the data from the Program Space. There are different `pgm_read_*` macros to read different sizes of data at the address given.

## 5.4 Storing and Retrieving Strings in the Program Space

Now that you can successfully store and retrieve simple data from Program Space you want to store and retrieve strings from Program Space. And specifically you want to store an array of strings to Program Space. So you start off with your array, like so:

```
const char* const string_table[] =
{
    "String 1",
    "String 2",
    "String 3",
    "String 4",
    "String 5"
};
```

and then you add your `PROGMEM` macro to the end of the declaration:

```
const char* const string_table[] PROGMEM =
{
    "String 1",
    "String 2",
    "String 3",
    "String 4",
    "String 5"
};
```

Right? WRONG!

Unfortunately, with GCC attributes, they affect only the declaration that they are attached to. So in this case, we successfully put the `string_table` variable, the array itself, in the Program Space. This DOES NOT put the actual strings themselves into Program Space. At this point, the strings are still in the Data Space, which is probably not what you want.

In order to put the strings in Program Space, you have to have explicit declarations for each string, and put each string in Program Space:

```
const char string_1[] PROGMEM = "String 1";
const char string_2[] PROGMEM = "String 2";
const char string_3[] PROGMEM = "String 3";
const char string_4[] PROGMEM = "String 4";
const char string_5[] PROGMEM = "String 5";
```

Then use the new symbols in your table, like so:

```
const char* const string_table[] PROGMEM =
{
    string_1,
    string_2,
    string_3,
    string_4,
    string_5
};
```

Now this has the effect of putting `string_table` in Program Space, where `string_table` is an array of pointers to characters (strings), where each pointer is a pointer to the Program Space, where each string is also stored.

Retrieving the strings are a different matter. You probably don't want to pull the string out of Program Space, byte by byte, using the `pgm_read_byte()` macro. There are other functions declared in the `<avr/pgmspace.h>` header file that work with strings that are stored in the Program Space.

For example if you want to copy the string from Program Space to a buffer in RAM (like an automatic variable inside a function, that is allocated on the stack), you can do this:

```
void foo(void)
{
    char buffer[10];

    for (uint8_t i = 0; i < 5; i++)
    {
        strcpy_P(buffer, (const char*) pgm_read_ptr(&(string_table[i])));

        // Display buffer on LCD.
    }
    return;
}
```

Here, the `string_table` array is stored in Program Space, so we access it normally, as if were stored in Data Space, then take the address of the location we want to access, and use the address as a parameter to `pgm_read_ptr`. We use the `pgm_read_ptr` macro to read the string pointer out of the `string_table` array. Remember that a pointer is 16-bits, or word size. The `pgm_read_ptr` macro will return a `void*`. This pointer is an address in Program Space pointing to the string that we want to copy. This pointer is then used as a parameter to the function `strcpy_P`. The function `strcpy_P` is just like the regular `strcpy` function, except that it copies a string from Program Space (the second parameter) to a buffer in the Data Space (the first parameter).

There are many string functions available that work with strings located in Program Space. All of these special string functions have a suffix of `_P` in the function name, and are declared in the `<avr/pgmspace.h>` header file.

## 5.5 Caveats

The macros and functions used to retrieve data from the Program Space have to generate some extra code in order to actually load the data from the Program Space. This incurs some extra overhead in terms of code space (extra opcodes) and execution time. Usually, both the space and time overhead is minimal compared to the space savings of putting data in Program Space. But you should be aware of this so you can minimize the number of calls within a single function that gets the same piece of data from Program Space. It is always instructive to look at the resulting disassembly from the compiler.

## 6 AVR-LibC and Assembler Programs

- [Introduction](#)
- [Invoking the Compiler](#)
- [Example Program](#)
- [Assembler Directives](#)
  - [Sections](#)
  - [Symbols](#)
  - [Data and Alignment](#)
- [Operand Modifiers](#)

### 6.1 Introduction

There might be several reasons to write code for AVR microcontrollers using plain assembler source code. Among them are:

- Code for devices that do not have RAM and are thus not supported by the C compiler.
- Code for very time-critical applications.
- Special tweaks that cannot be done in C.

Usually, all but the first could probably be done easily using the [inline assembler](#) facility of the compiler.

Although AVR-LibC is primarily targeted to support programming AVR microcontrollers using the C (and C++) language, there's limited support for direct assembler usage as well. The benefits of it are:

- Use of the C preprocessor and thus the ability to use the same symbolic constants that are available to C programs, as well as a flexible macro concept that can use any valid C identifier as a macro (whereas the assembler's macro concept is basically targeted to use a macro in place of an assembler instruction).
- Use of the runtime framework like automatically assigning interrupt vectors. For devices that have RAM, [initializing the RAM variables](#) can also be utilized.

## 6.2 Invoking the Compiler

For the purpose described in this document, the assembler and linker are usually not invoked manually, but rather using the C compiler frontend (`avr-gcc`) that in turn will call the assembler and linker as required.

This approach has the following advantages:

- There is basically only one program to be called directly, `avr-gcc`, regardless of the actual source language used.
- The invocation of the C preprocessor will be automatic, and will include the appropriate options to locate required include files in the filesystem.
- The invocation of the linker will be automatic, and will include the appropriate options to locate additional libraries as well as the application start-up code (`crtXXX.o`) and linker script.

Note that the invocation of the C preprocessor will be automatic when the filename provided for the assembler file ends in `.S` (the capital letter "s"). This would even apply to operating systems that use case-insensitive filesystems since the actual decision is made based on the case of the filename suffix given on the command-line, not based on the actual filename from the file system.

As an alternative to using `.S`, the suffix `.sx` is recognized for this purpose (starting with GCC v4.3). This is primarily meant to be compatible with other compiler environments that have been providing this variant before in order to cope with operating systems where filenames are case-insensitive (and, with some versions of `make` that could not distinguish between `.s` and `.S` on such systems).

Alternatively, the language can explicitly be specified using the `-x assembler-with-cpp` option.

## 6.3 Example Program

The following annotated example features a simple 100 kHz square wave generator using an AT90S1200 clocked with a 10.7 MHz crystal. Pin PD6 will be used for the square wave output.

```
#include <avr/io.h>           // Note [1]

work    = 16                 // Note [2]
tmp     = 17

inttmp  = 19
intsav  = 0

SQUARE  = PD6               // Note [3]

#define IO(x) _SFR_IO_ADDR(x)

// Note [4]:
// 100 kHz => 200000 edges/s
tmconst = 10700000 / 200000

// # clocks in ISR until TCNT0 is set
fuzz = 8

.text

.global main                 // Note [5]
main:
    rcall    ioinit
1:  rjmp     1b               // Note [6]

.global TIMER0_OVF_vect     // Note [7]
TIMER0_OVF_vect:
    ldi     inttmp, 256 - tmconst + fuzz
    out    IO(TCNT0), inttmp // Note [8]

    in     intsav, IO(SREG)  // Note [9]

    sbic   IO(PORTD), SQUARE
    rjmp  1f
```

```

    sbi      IO(PORTD), SQUARE
    rjmp    2f
1:  cbi      IO(PORTD), SQUARE
2:
    out     IO(SREG), intsav
    reti

ioinit:
    sbi      IO(DDRD), SQUARE

    ldi     work, _BV(TOIE0)
    out     IO(TIMSK), work

    ldi     work, _BV(CS00)      // tmr0: CK/1
    out     IO(TCCR0), work

    ldi     work, 256 - tmconst
    out     IO(TCNT0), work

    sei

    ret

.global __vector_default      // Note [10]
__vector_default:
    reti

```

**Note [1]** As in C programs, this includes the central processor-specific file containing the IO port definitions for the device. Note that not all include files can be included into assembler sources.

**Note [2]** Assignment of registers to symbolic names used locally. Another option would be to use a C preprocessor macro instead:

```
#define work 16
```

**Note [3]** Our bit number for the square wave output. Note that the right-hand side consists of a CPP macro which will be substituted by its value (6 in this case) before actually being passed to the assembler.

**Note [4]** The assembler uses integer operations in the host-defined integer size (32 bits or longer) when evaluating expressions. This is in contrast to the C compiler that uses the C type `int` by default in order to calculate constant integer expressions.

In order to get a 100 kHz output, we need to toggle the PD6 line 200000 times per second. Since we use timer 0 without any prescaling options in order to get the desired frequency and accuracy, we already run into serious timing considerations: while accepting and processing the timer overflow interrupt, the timer already continues to count. When pre-loading the `TCCNT0` register, we therefore have to account for the number of clock cycles required for interrupt acknowledge and for the instructions to reload `TCCNT0` (4 clock cycles for interrupt acknowledge, 2 cycles for the jump from the interrupt vector, 2 cycles for the 2 instructions that reload `TCCNT0`). This is what the constant `fuzz` is for.

**Note [5]** External functions need to be declared to be `.global`. `main` is the application entry point that will be jumped to from the initialization routine in `crt0.o`.

**Note [6]** The main loop is just a single jump back to itself. Square wave generation itself is completely handled by the timer 0 overflow interrupt service. A `sleep` instruction (using idle mode) could be used as well, but probably would not conserve much energy anyway since the interrupt service is executed quite frequently.

**Note [7]** Interrupt functions can get the [usual names](#) that are also available to C programs. The linker will then put them into the appropriate interrupt vector slots. Note that they must be declared `.global` in order to be acceptable for this purpose. This will only work if `<avr/io.h>` has been included. Note that the assembler or linker have no chance to check the correct spelling of an interrupt function, so it should be double-checked. (When analyzing the resulting object file using `avr-objdump` or `avr-nm`, a name like `__vector_N` should appear, with `N` being a small integer number.)

**Note [8]** As explained in the section about [special function registers](#), the actual IO port address should be obtained using the macro `__SFR_IO_ADDR`. (The AT90S1200 does not have RAM thus the memory-mapped approach to access the IO registers is not available. It would be slower than using `in/out` instructions anyway.)

Since the operation to reload `TCCNT0` is time-critical, it is even performed before saving `SREG`. Obviously, this requires that the instructions involved would not change any of the flag bits in `SREG`.



**Note [9]** Interrupt routines must not clobber the global CPU state. Thus, it is usually necessary to save at least the state of the flag bits in `SREG`. (Note that this serves as an example here only since actually, all the following instructions would not modify `SREG` either, but that's not commonly the case.)

Also, it must be made sure that registers used inside the interrupt routine do not conflict with those used outside. In the case of a RAM-less device like the AT90S1200, this can only be done by agreeing on a set of registers to be used exclusively inside the interrupt routine; there would not be any other chance to "save" a register anywhere.

If the interrupt routine is to be linked together with C modules, care must be taken to follow the [register usage guidelines](#) imposed by the C compiler. Also, any register modified inside the interrupt service needs to be saved, usually on the stack.

**Note [10]** As explained in [Interrupts](#), a global "catch-all" interrupt handler that gets all unassigned interrupt vectors can be installed using the name `__vector_default`. This must be `.global`, and obviously, should end in a `reti` instruction. (By default, a jump to location 0 would be implied instead.)

## 6.4 Assembler Directives

The directives available in the assembler are described in the GNU assembler (gas) manual at [Assembler Directives](#).

As gas comes from a Unix origin, its directives and overall assembler syntax is slightly different than the one being used by other assemblers. Numeric constants follow the C notation (prefix `0x` for hexadecimal constants, `0b` for binary constants), expressions use a C-like syntax.

Some common directives include:

**Table 7 Assembler Directives: Sections**

Section Ops	Description
<code>.section name, "flags", @typ</code>	Put the following objects into <a href="#">named section name</a> . Set <a href="#">section flags</a> and <a href="#">section type</a> to <code>typ</code>
<code>.pushsection ...</code> <code>.popsection</code>	Like <code>.section</code> , but also pushes the current section and subsection onto the section stack. The current section and subsection can be restored with <code>.popsection</code> .
<code>.subsection int</code>	Put the following code into subsection number <code>int</code> which is some integer. Subsections are located in order of increasing index within their input section. The default after switching to a new section by means of <code>.section</code> or <code>.pushsection</code> is subsection 0.
<code>.text</code> <code>.data</code> <code>.bss</code>	Put the following code into the <code>.text</code> section, <code>.data</code> section or <code>.bss</code> section, respectively. The assembler knows the right section flags and section type, for example the <code>.text</code> directive is basically the same like <code>.section .text, "ax", @progbits</code> . The directives support an optional subsection argument, see <code>.subsection</code> above.

**Table 8 Assembler Directives: Symbols**

Symbol Ops	Description
<code>.global sym</code> <code>.globl sym</code>	Globalize symbol <code>sym</code> so that it can be referred to in other modules. When a symbol is used without prior declaration or definition, the symbol is implicitly global. The <code>.global</code> directive can also be used to refer to that symbol, so that the linker pulls in code that defines the symbol (provided such a symbol definition exists). For example, code that puts objects in the <code>.data</code> section and that assumes that the <a href="#">startup code</a> initializes that area, would use <code>.global __do_copy_data</code> .

Symbol Ops	Description
<code>.weak syms</code>	Declare symbols <i>syms</i> as weak symbols, where <i>syms</i> is a comma-separated list of symbols. This applies only when the symbols are also defined in the same module. When the linker encounters a weak symbol that is also defined as <code>.global</code> in a different module, then the linker will use the latter without raising a diagnostic about multiple symbol definitions.
<code>.type sym, @kind</code>	Set the type of symbol <i>sym</i> to <i>kind</i> . Commonly used symbol types are <code>@function</code> for function symbols like <code>main</code> and <code>@object</code> for data symbols. This has an affect for disassemblers, debuggers and tools that show function / object properties.
<code>.size sym, size</code>	Set the size associated with symbol <i>sym</i> to expression <i>size</i> . The linker works on the level of sections, it does not even know what functions are. This directive serves book-keeping, and may be useful for debuggers, disassemblers or tools that show which function / object consumes how much memory.
<code>.set sym, expr</code> <code>.equ sym, expr</code> <code>sym = expr</code>	Set the value of symbol <i>sym</i> to the value of expression <i>expr</i> . When a global symbol is set multiple times, the value stored in the object file is the last value stored into the symbol.
<code>.extern</code>	Ignored for compatibility with other assemblers.
<code>.org</code>	Advance the location pointer to a specified offset <i>relative</i> to the beginning of the <a href="#">input section</a> . The location counter cannot be moved backwards. This is a fairly pointless directive in an assembler environment that uses relocatable object files. The linker determines the final location of the objects. See the <a href="#">FAQ</a> on how to relocate code to a fixed address.

Table 9 Assembler Directives: Data and Alignment

Data Ops	Description	Alias
<code>.byte list</code>	Allocate bytes specified by a list of comma-separated expressions.	
<code>.2byte list</code>	Similar to <code>.byte</code> , but for 16-bit values.	<code>.word</code>
<code>.4byte list</code>	Similar to <code>.byte</code> , but for 32-bit values.	<code>.long</code>
<code>.8byte list</code>	Similar to <code>.byte</code> , but for 64-bit values.	<code>.qword</code>
<code>.ascii "string"</code>	Allocate a string of characters without <code>\0</code> termination.	
<code>.asciz "string"</code>	Allocate a <code>\0</code> terminated string.	
<code>.float list</code>	Allocate IEEE-754 single 32-bit floating-point values specified in the comma-separated <i>list</i> .	
<code>.double list</code>	Same, but for IEEE-754 double 64-bit floats.	
<code>.space num[, val]</code>	Allocate <i>num</i> bytes with value <i>val</i> where <i>val</i> is optional and defaults to zero.	<code>.skip</code>
<code>.zero num</code>	Insert <i>num</i> zero bytes.	
Alignment Ops	Description	Alias
<code>.balign val</code>	Align the following code to <i>val</i> bytes, where <i>val</i> is an absolute expression that evaluates to a power of 2.	<code>.align</code>
<code>.p2align expo</code>	Align the following code to $2^{expo}$ bytes.	

Moreover, there is the `.macro` directive, which starts an assembler macro. The GNU assembler implements a powerful macro processor which even supports recursive macro definitions. For an example, see the gas documentation for `.macro`. A gas `.macro` can further be combined with C preprocessor directives. For some real-world examples, see the AVR-LibC sources `macros.inc` and `asmdef.h`.

## 6.5 Operand Modifiers

There are some AVR-specific operators available like `lo8()`, `hi8()`, `pm()`, `gs()` etc. For an overview see the documentation of the [operand modifiers](#) in the inline assembly Cookbook.

Example:

```
ldi r24, lo8 (gs (somefunc))
ldi r25, hi8 (gs (somefunc))
call something
subi r24, lo8 (- (my_var))
sbci r25, hi8 (- (my_var))
```

This passes the address of function `somefunc` as the first parameter to function `something`, and *adds* the address of variable `my_var` to the 16-bit return value of `something`.

## 7 Inline Assembler Cookbook

AVR-GCC

Inline Assembler Cookbook

- [About this Document](#)
- [Building Blocks](#)
  - [The Anatomy of a GCC asm Statement](#)
  - [Special Sequences](#)
  - [Constraints](#)
    - \* [Constraint Modifiers](#)
    - \* [Instructions and Constraints](#)
  - [Print Modifiers](#)
  - [Operand Modifiers](#)
- [Examples](#)
  - [Swapping Nibbles](#)
  - [Swapping Bytes](#)
  - [Accessing Memory](#)
  - [Accessing Bytes of wider Expressions](#)
  - [Inline Functions and `\_\_builtin\_constant\_p`](#)
  - [Jumping and Branching](#)
- [Binding local Variables to Registers](#)
  - [Interfacing non-ABI Functions](#)
- [Specifying the Assembly Name of Static Objects](#)
- [What won't work](#)

## 7.1 About this Document

The GNU C/C++ compiler for AVR RISC processors offers to embed assembly language code into C/C++ programs. This cool feature may be used for manually optimizing time critical parts of the software, or to use specific processor instructions which are not available in the C language.

It's assumed that you are familiar with writing AVR assembler programs, because this is not an AVR assembler programming tutorial. It's not a C/C++ tutorial either.

Note that this document does not cover files written completely in assembly language, refer to [AVR-LibC and Assembler Programs](#) for this.

Copyright (C) 2001-2002 by egnite Software GmbH

Permission is granted to copy and distribute verbatim copies of this manual provided that the copyright notice and this permission notice are preserved on all copies. Permission is granted to copy and distribute modified versions of this manual provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

This document describes version 4.7 of the compiler or newer.

Herne, 17th of May 2002 Harald Kipp harald.kipp-at-egnite.de

## 7.2 The Anatomy of a GCC asm Statement

A GCC inline assembly statement starts with the keyword `asm`, `__asm` or `__asm__`, where the first one is not available in strict ANSI mode.

In its simplest form, the inline assembly statement has no operands and injects just one instruction into the code stream, like in

```
__asm ("nop");
```

In its generic form, an asm statements can have one of the following three forms:

**A simple asm without operands** `__asm (code-string);`

`code-string` is a string literal that will be added as is into the generated assembly code. This even applies to the `%` character. The only replacement is that `\n` and `\t` are interpreted as newline resp. TAB character.

This type of asm statement may occur at top level, outside any function as global asm. When its placement relative to functions is important, consider `-fno-toplevel-reorder`.

**An asm with operands** `__asm volatile (code-string : output-operands : input-operands : clobbers);`

This is the most widely used form of an asm statement. It must be located in a function.

`output-operands`, `input-operands` and `clobbers` are comma-separated lists of operands resp. clobber specifications. Any of them may be empty, for example when the asm has no outputs. At least one `:` (colon) must be present, otherwise it will be a simple asm without operands and without `%` replacements.

**An asm goto statement** `__asm goto (code-string : : input-operands : clobbers : labels);`

Like the asm above, but `labels` is a comma-separated list of C/C++ code labels which would be valid in a `goto` statement. And `output-operands` must be empty, because it is impossible to generate output reloads after the code has transferred control to one of the labels.

As there are no output operands, `asm goto` is implicitly volatile. When `volatile` is specified explicitly, the `goto` keyword may be placed after or before the `volatile`.

Notes on the various parts:

**Volatility** Keyword `volatile` is optional and means that the asm statement has side effects that are not expressed in terms of the operands or clobbers. The asm statement must not be optimized away or reordered with respect to other volatile statements like volatile memory accesses or other volatile asm.

Any asm statement without `output-operands` is implicitly volatile.

A non-volatile asm statement with output operands that are all unused may be optimized away when all output operands are unused.

Instead of `volatile`, `__volatile` or `__volatile__` can be used.

**code-string** A string literal that contains the code that is to be injected in the assembly code generated by the compiler. `%-expressions` are replaced by the string representations of the operands, and the number of lines is determined to estimate the code size of the asm.

Apart from that, **the compiler does not analyze the code provided in the code template.**

This means that the code appears to the compiler *as if it was executed in one parallel chunk, all at once*. It is important to keep that in mind, in particular for cases where input and output operands may overlap.

### output-operands

**input-operands** A comma-separated list of operands, which may take the following forms. In any case, the first operand can be referred to as `"%0"` in `code-string`, the second one as `"%1"` etc.

**"constraints" (expr)** `expr` is a C expression that's an input or output (or both) to the asm statement. An output expression must be an lvalue, i.e. it must be valid to assign a value to it.

`"constraints"` is a string literal with [constraints](#) and [constraint modifiers](#). For example, constraint

`"r"` stands for *general-purpose register*. A simple input operand would be `"r" (value + 1)`

The compiler computes `value + 1` and supplies it in some general-purpose register R2...R31. In many cases, an upper d-register R16...R31 is required for instructions like `LDI` or `ANDI`. A respective output operand specification is

`"=d" (result)`

**Notice that this operand may overlap with input operands!**

When an operand is written before all input operands are consumed, then in almost all cases the output operand requires an early-clobber modifier `&` so that it won't overlap with any input operand:

`"=&d" (result)`

An operand that's both an output and an input can be expressed with the `+` constraint modifier:

`"+d" (result)`

Such an operand is both output and input, and hence it won't overlap with other operands.

**[name] "constraints" (expr)** Like above. In addition, a named operand can be referred to as `%[name]` in `code-string`. This is useful in long asm statements with many operands.

**clobbers** A comma-separated list of string literals like `"16"`, `"r16"` or `"memory"`.

The first two clobbers mean that the asm destroys register R16. Only the lower-case form is allowed, and register names like `Z` are not recognized.

`"memory"` means that the asm touches memory in some way. When the asm writes to some RAM location for example, the compiler must not optimize RAM accesses across the asm because the memory may change.

Clobbering `__tmp_reg__` by means of `"r0"` has no effect, but such a clobber may be added to indicate to the reader that the asm clobbers R0.

Clobbering `__zero_reg__` by means of `"r1"` has no effect. When the asm destroys the zero register, for example by means of a `MUL` instruction, then the code must restore the register at the end by means of `"clr __zero_reg__"`

**The size of an asm** The code size of an asm statement is the number of lines multiplied by 4 bytes, the maximal possible AVR instruction length. The length is needed when (conditional) jumps cross the asm statement in order to compute (upper bounds for) jump offsets of PC-relative jumps.

The number of lines is one plus the number of line breaks in `code-string`. These may be physical line breaks from `\n` characters and logical line breaks from `$` characters.

Before we start with the first examples, we list all the bells and whistles that can be used to compose an inline assembly statement: [special sequences](#), [constraints](#), [constraint modifiers](#), [print modifiers](#) and [operand modifiers](#).

## 7.3 Special Sequences

There are special sequences that can be used in the assembly template.

**Table 10 Inline asm Special Sequences**

Sequence	Description
<code>__SREG__</code>	The I/O address of the status register SREG at 0x3F
<code>__tmp_reg__</code>	The <a href="#">temporary register</a> R0 (R16 on reduced Tiny)
<code>__zero_reg__</code>	The <a href="#">zero register</a> R1, always zero (R17 on reduced Tiny)
<code>\$</code>	A logical line separator, used to separate multiple instruction in one physical line
<code>\n</code>	A physical newline, used to separate multiple instructions
<code>\t</code>	A TAB character, can be used for better legibility of the generated asm
<code>\"</code>	A " character (double quote)
<code>\\</code>	A \ character (backslash)
<code>%%</code>	A % character (percent)
<code>%~</code>	"r" or "r", used to construct <code>call</code> or <code>rcall</code> by means of " <code>%~call</code> ", depending on the architecture
<code>%!</code>	"e" or "e", used to construct indirect calls like <code>icall</code> or <code>ecall</code> by means of " <code>%!icall</code> ", depending on the architecture
<code>%=</code>	A number that's unique for the compilation unit and the respective inline asm code, used to construct unique labels
Comment	Description
<code>; text</code>	A single-line assembly comment that extends to the end of the physical line
<code>/* text */</code>	A multi-line C comment

- Moreover, the following I/O addresses are defined provided the device supports the respective SFR: `__SP_L__`, `__SP_H__`, `__CCP__`, `__RAMPX__`, `__RAMPY__`, `__RAMPZ__`, `__RAMPD__`.
- Register `__tmp_reg__` may be freely used by inline assembly code and need not be restored at the end of the code.
- Register `__zero_reg__` contains a value of zero. When that value is destroyed, for example by a `MUL` instruction, its value has to be restored at the end of the code by means of `clr __zero_reg__`
- In inline asm without operands (i.e without a single colon), a `%` will always insert a single `%`. No `%-`codes are available.

Sequences like `__SREG__` are not evaluated as part of the inline asm, they are just copied to the asm code as they are. At the top of each assembly file, the compiler prints definitions like `__SREG__ = 0x3f`

so that they can also be used in inline assembly.

## 7.4 Constraints

The most up-to-date and detailed information on constraints for the AVR can be found in the [avr-gcc Wiki](#).

Table 11 Inline asm Operand Constraints

Constraint	Registers	Range
a	Simple upper registers that support FMUL	R16 ... R23
b	Base pointer registers that support LDD, STD	Y, Z (R28 ... R31)
d	Upper registers	R16 ... R31
e	Pointer registers that support LD, ST	X, Y, Z (R26 ... R31)
l	Lower registers	R2 ... R15
r	Any register	R2 ... R31
w	Upper registers that support ADIW	R24 ... R31
x	X pointer registers	R26, R27
y	Y pointer registers	R28, R29
z	Z pointer registers	R30, R31
Constraint	Constant	Range
I	6-bit unsigned integer constant	0 to 63
J	6-bit negative integer constant	-63 to 0
M	8-bit unsigned integer constant	0 to 255
n	Integer constant	
i	Immediate value known at link-time, like the address of a variable in static storage	
EF	Floating-point constant	
Ynn	Fixed-point or integer constant	
Constraint	Explanation	Notes
m	A memory location	
X	Any valid operand	
0 ... 9	Matches the respective operand number	

- Constraints without a modifier specify input operands.
- Constraints with a modifier specify output operands.
- More than one constraint like in "rn" specifies the union of the specified constraints; "r" and "n" in this case.
- All constraints listed above are single-letter constraints, except Ynn which is a 3-letter constraint.

Constraint modifiers are:

Table 12 Constraint Modifiers

Modifier	Meaning
=	Output-only operand. Without & it may overlap with input operands
+	Output operand that's also an input
=&	"Early-clobber". Register should be used for output only and won't overlap with any input operand(s)

The selection of the proper constraint depends on the range of the constants or registers, which must be acceptable to the AVR instruction they are used with. The C compiler doesn't check any line of your assembler code. But it is able to check the constraint against your C expression. However, if you specify the wrong constraints, then the compiler may silently pass wrong code to the assembler. And, of course, the assembler will fail with some cryptic output or internal errors, or in the worst case wrong code may be the result.

For example, if you specify the constraint "r" and you are using this register with an ORI instruction, then the compiler may select any register. This will fail if the compiler chooses R2 to R15. (It will never choose R0 or R1,

because these are used for special purposes.) That's why the correct constraint in that case is "d". On the other hand, if you use the constraint "M", the compiler will make sure that you don't pass anything else but an 8-bit unsigned integer value known at compile-time.

The following table shows all AVR assembler mnemonics which require operands, and the related constraints.

**Table 13 AVR Instructions and Constraints**

Mnemonic	Constraints	Mnemonic	Constraints
adc	r, r	add	r, r
adiw	w, I	and	r, r
andi	d, M	asr	r
bclr	I	bld	r, I
brbc	I, label	brbs	I, label
bset	I	bst	r, I
call	i	cbi	I, I
cbr	d, I	clr	r
com	r	cp	r, r
cpc	r, r	cpi	d, M
cpse	r, r	dec	r
elpm	r, z	eor	r, r
fmul	a, a	fmuls	a, a
fmulsu	a, a	in	r, I
inc	r	jmp	i
lac	z, r	las	z, r
lat	z, r	ld	r, e
ldd	r, b	ldi	d, M
lds	r, i	lpm	r, z
lsl	r	lsr	r
mov	r, r	movw	r, r
mul	r, r	muls	r, r
mulsu	a, a	neg	r
or	r, r	ori	d, M
out	I, r	pop	r
push	r	rcall	i
rjmp	i	rol	r
ror	r	sbc	r, r
sbc	d, M	sbi	I, I
sbic	I, I	sbiw	w, I
sbr	d, M	sbrc	r, I
sbrs	r, I	ser	d
st	e, r	std	b, r
sts	i, r	sub	r, r
subi	d, M	swap	r
tst	r	xch	z, r

## 7.5 Print Modifiers

The %-operands in the inline assembly template can be adjusted by special print-modify characters. The one-letter modifier follows the % and precedes the operand number like in "%a0", or precedes the name in named operands like in "%a[address]".



Table 14 Inline asm Print Modifiers

Modifier	Number of Arguments	Explanation	Suitable Constraints
%a0	1	Print pointer register as address X, Y or Z, like in "LD r0, %a0+"	x, y, z, b, e
%i0	1	Print compile-time RAM address as I/O address, like in "OUT %i0, r0" with argument "n" (&SREG)	n
%n0	1	Print the negative of a compile-time integer constant	n
%r0	1	Print the register number of a register, like in "CLR %r0+7" for the MSB of a 64-bit register	reg
%x0	1	Print a function name without <code>gs()</code> modifier, like in "%~CALL %x0" with argument "s" (main)	s
%A0	1	Add 0 to the register number (no effect)	reg
%B0	1	Add 1 to the register number	reg
%C0	1	Add 2 to the register number	reg
%D0	1	Add 3 to the register number	reg
%T0%t1	2	Print the register that holds bit number %1 of register %0	reg + n
%T0%T1	2	Print operands suitable for BLD/BST, like in "BST %T0%T1", including the required ,	reg + n

- Register constraints are: r, d, w, x, y, z, b, e, a, l.

## 7.6 Operand Modifiers

Table 15 Assembly Code Operand Modifiers

Modifier	Explanation	Purpose
l08()	1 <sup>st</sup> Byte of a link-time constant, bits 0...7	Getting parts of a byte-address
hi8()	2 <sup>nd</sup> Byte of a link-time constant, bits 8...15	
hl08()	3 <sup>rd</sup> Byte of a link-time constant, bits 16...23	
hhi8()	4 <sup>th</sup> Byte of a link-time constant, bits 24...31	
hh8()	Same like hl08	
pm_l08()	1 <sup>st</sup> Byte of a link-time constant divided by 2, bits 1...8	Getting parts of a word-address
pm_hi8()	2 <sup>nd</sup> Byte of a link-time constant divided by 2, bits 9...16	
pm_hh8()	3 <sup>rd</sup> Byte of a link-time constant divided by 2, bits 17...24	
pm()	Link-time constant divided by 2 in order to get a program memory (word) addresses, like in <code>l08(pm(main))</code>	Word-address
gs()	Function address divided by 2 in order to get a (word) addresses, like in <code>l08(gs(main))</code> . Generate <code>stub</code> (trampoline) as needed. This is required to calculate the address of a code label on devices with more than 128 KiB of program memory that's supposed to be used in <code>EICALL</code> . For rationale, see the <a href="#">GCC documentation</a> . On devices with less program memory, <code>gs()</code> behaves like <code>pm()</code>	Function address for [E] ICALL

When the argument of a modifier is not computable at assembler-time, then the assembler has to encode the expression in an abstract form using `RELOCs`. Consequence is that only a very limited number of argument expressions is supported when they are not computable at assembler-time.

## 7.7 Examples

Some examples show the assembly code as generated by the compiler. It's the code from the `.s` files as generated with option `-save-temps`. Adding the high-level source to the generated assembly can be turned on with `-fverbose-asm` since GCC v8.

### 7.7.1 Swapping Nibbles

The first example uses the `swap` instruction to swap the nibbles of a byte. Input and output of `swap` are located in the same general purpose register. This means the input operand, operand 1 below, must be located in the same register(s) like operand 0, so that the right [constraint](#) for operand 1 is `"0"`:

```
asm ("swap" : "=r" (value) : "0" (value));
```

All side effects of the code are described by the constraints and the clobbers, so that there is no need for this asm to be volatile. In particular, this asm may be optimized out when the output value is unused.

A shorter pattern to state that `value` is both input and output is by means of [constraint modifier](#) +

```
asm ("swap" : "+r" (value));
```

### 7.7.2 Swapping Bytes

Swapping nibbles was a piece of cake, so let's swap the bytes of a 16-bit value. In order to access the constituent bytes of the 16-bit input and output values, we use the [print modifiers](#) `%A` and `%B`.

The asm is placed in a small C test case so that we can inspect the resulting assembly code as generated by the compiler with `-save-temps`.

```
void callee (int, int);

void func (int param)
{
    int swapped;

    asm ("mov %A0, %B1" "\n\t"
        "mov %B0, %A1"
        : "=r" (swapped) : "r" (param));

    callee (param, swapped);
}
```

The `"\n\t"` [sequence](#) adds a line feed that is required between the two instructions, and a TAB to align the two instructions in the generated assembly. There is no `"\n\t"` after the last instruction because that would just increase the [size of the asm](#).

The generated assembly works as expected. The compiler wraps it in `#APP / #NOAPP` annotations:

```
func:
/* #APP */
    mov r22, r25    ; swapped, param
    mov r23, r24    ; swapped, param
/* #NOAPP */
    jmp callee
```

**Wrong! While the generated code above is correct, the inline asm itself is not!**

We see this with a slightly adjusted test case where the arguments of `callee` have been swapped, but that uses the same inline asm:

```
void func (int param)
{
    int swapped;

    asm ("mov %A0, %B1" "\n\t"
        "mov %B0, %A1"
        : "=r" (swapped) : "r" (param));

    callee (swapped, param);
}
```

The result is the following assembly:

```
func:
```

```

    movw r22,r24
/* #APP */
    mov r24, r25    ; swapped, param
    mov r25, r24    ; swapped, param
/* #NOAPP */
    jmp callee

```

which is obviously wrong, because after the code from the inline asm, the low byte of `swapped` and the high byte will always have the same value of `r25`.

The reason is that the output operand overlaps the input, *and* the output is changed before all of the input operands are consumed. This is a so-called *early-clobber* situation. There are two possible solutions to this predicament:

- Mark the output operand with the early-clobber [constraint modifier](#):

```

asm ("mov %A0, %B1" "\n\t"
     "mov %B0, %A1"
     : "=&r" (swapped) : "r" (param));

```

- Use constraints and a code sequence that expect input and output in the same registers:

```

asm ("eor %A0, %B0" "\n\t"
     "eor %B0, %A0" "\n\t"
     "eor %A0, %B0"
     : "=r" (swapped) : "0" (param));

```

### 7.7.3 Accessing Memory

Accessing memory requires that the AVR instructions that perform the memory access are provided with the appropriate memory address.

1. The address can be provided directly, like `__SREG__`, `0x3f`, as a symbol, or as a symbol plus a constant offset.
2. Provide the address by means of an inline asm operand.

Approach 1 is simpler as it does not require an asm operand, while approach 2 is in many cases more powerful because macros defined per, say, `#include <avr/io.h>` can be used as operands, whereas such headers are not included in the assembly code as generated by the compiler.

Reading a SFR like `PORTB` can be performed by

```
asm volatile ("in %0, %1" : "=r" (result) : "I" _SFR_IO_ADDR (PORTB));
```

Macro `_SFR_IO_ADDR` is provided by `avr/sfr_defs.h` which is included by `avr/io.h`.

Since GCC v4.7, [print modifier](#) `%i` is supported, which prints RAM addresses like `& PORTB` as an I/O address:

```
asm volatile ("in %0, %i1" : "=r" (result) : "I" (& PORTB));
```

When the address is not an I/O address, then `LDS` or `LD` must be used, depending on whether the address is known at link-time or only at run-time. For example, the following macro provides the functionality to clear an SFR. The code discriminates between the possibilities that

- The SFR address is known at compile-time and is an I/O address.
- The SFR address is known at compile-time but is not in the I/O range.
- The SFR address is not known at compile-time.

```
#include <avr/io.h>

#define CLEAR_REG(sfr) \
do { \
    if (__builtin_constant_p (& (sfr)) \
        && _SFR_IO_REG_P (sfr)) \
        asm volatile ("out %i0, __zero_reg__" \
            :: "I" (& (sfr)) : "memory"); \
    else if (__builtin_constant_p (& (sfr))) \
        asm volatile ("sts %0, __zero_reg__" \
            :: "n" (& (sfr)) : "memory"); \
    else \
        asm volatile ("st %a0, __zero_reg__" \
            :: "e" (& (sfr)) : "memory"); \
} while (0)
```

The last case with constraint "e" works because `&sfr` is a 16-bit value, and 16-bit values (and larger) start in even registers. Therefore, the address will be located in R27:R26, R29:R28 or in R31:R30, which print modifier %a will print as X, Y or Z, respectively. The address will never end up in, say, R30:R29.

#### The test case

```
void clear_3_regs (uint8_t volatile *psfr)
{
    CLEAR_REG (PORTB);
    CLEAR_REG (UDR0);
    CLEAR_REG (*psfr);
}
```

compiles for ATmega328 and with optimization turned on to

```
clear_3_regs:
    movw r30,r24
/* #APP */
    out 0x5, __zero_reg__
    sts 198, __zero_reg__
    st Z, __zero_reg__ ; psfr
/* #NOAPP */
    ret
```

As `__builtin_constant_p` is used to infer whether the address of the SFR is known at compile-time, extra care must be taken when the functionality is implemented as an inline function:

```
static inline __attribute__((__always_inline__))
void clear_reg (uint8_t volatile *psfr)
{
    // !!! The following cast is required to make __builtin_constant_p
    // !!! work as expected in the inline function.
    uintptr_t addr = (uintptr_t) psfr;

    if (__builtin_constant_p (addr)
        && _SFR_IO_REG_P (* psfr))
        asm volatile ("out %i0, __zero_reg__"
            :: "I" (addr) : "memory");
    else if (__builtin_constant_p (addr))
        asm volatile ("sts %0, __zero_reg__"
            :: "n" (addr) : "memory");
    else
        asm volatile ("st %a0, __zero_reg__"
            :: "e" (addr) : "memory");
}

void clear_3_pregs (uint8_t volatile *psfr)
{
    clear_reg (& PORTB);
    clear_reg (& UDR0);
    clear_reg (psfr);
}
```

**Casting the address `psfr` to an integer type in the inline function is required** so that the compiler will recognize constant addresses.

Also notice that we have to pass the *address of the SFR* to the inline function. Passing the SFR directly like in the marco approach won't work for obvious reasons.

#### 7.7.4 Accessing Bytes of wider Expressions

Finally, an example that atomically increments a 16-bit integer. The code is wrapped in `IN SREG / CLI / OUT SREG` to make it atomic. It reads the 16-bit value `data` from its absolute address, increments it and then writes it back:

```

uint16_t volatile data;

void inc_data (void)
{
    uint16_t tmp;
    asm volatile ("in __tmp_reg__, __SREG__" "\n\t"
                 "cli" "\n\t"
                 "lds %A[tmp], %[addr]" "\n\t"
                 "lds %B[tmp], %[addr]+1" "\n\t"
#ifdef __AVR_TINY__
                 // Reduced Tiny does not have ADIW.
                 "subi %A[tmp], lo8(-1)" "\n\t"
                 "sbci %B[tmp], hi8(-1)" "\n\t"
#else
                 "adiw %[tmp], 1" "\n\t"
#endif
                 "sts %[addr]+1, %B[tmp]" "\n\t"
                 "sts %[addr], %A[tmp]" "\n\t"
                 "out __SREG__, __tmp_reg__"
#ifdef __AVR_TINY__
                 // No need to restrict tmp to a "w" register. And on
                 // avr-gcc v13.2 and older, "w" contains no regs.
                 : [tmp] "=d" (tmp), "+m" (data)
#else
                 : [tmp] "=w" (tmp), "+m" (data)
#endif
#ifdef __AVR_TINY__
                 : [addr] "i" (& data));
}

```

Notice there are three different ways required to access the different bytes of the involved 16-bit entities:

- For the 16-bit general purpose register `%[tmp]`, [print modifiers](#) `%A` and `%B` are used.
- For the 16-bit value `data` in static storage, `%[addr]+1` is used to access the high byte. The resulting expression `data+1` is computable at link-time and evaluated by the linker.
- In the compilation variant for Reduced Tiny, the bytes of the 16-bit subtrahend `-1` are accessed with the [operand modifiers](#) `lo8` and `hi8` that are evaluated by the assembler because `-1` is known at assembler-time.

`data` is located in static storage, hence its address is known to the linker and fits [constraint](#) `"i"`.

The sole purpose of operand `" +m" (data)` is to describe the effect of the asm on data memory: It changes `data`. Notice that there is no "memory" clobber, because that operand already describes all memory side effects, and it does this in a less intrusive way than a catch-all "memory". The operand is not used in the asm template; but in principle it would be possible to use it as operand with `LDS` and `STS` instead of operand `[addr] "i" (& data)`. However, there are many situations where a memory operand constrained by "m" takes a form that cannot be used with AVR instructions because there are no matching print modifiers, or because it is not known a priori what specific form the memory operand takes. In such cases, one would take the address of the operand and supply it as address in a pointer register to the inline asm. The compiler generates the required instructions for address computation, and the inline asm knows that it can use `LD` and `ST`.

## 7.7.5 Jumping and Branching

When an inline asm contains jumps, then it also requires labels. When the label is inside the asm, then care must be taken that the label is unique in the compilation unit even when the inline asm is used multiple times, e.g. when the code is located in an unrolled loop or a function has multiple incarnations due to cloning, or simply because a macro or inline function that contains an asm statement is used more than once.

There are two kinds of labels that can be used:

- Local labels of the form `n`: where `n` is some (small, non-negative) number. They can be targeted by means of `nb` or `nf`, depending on whether the jump direction is backwards or forwards. Such a numeric labels may be present more than once. The taken label is the first one with the specified number in the respective direction:

```

// Loop until bit PORTB.7 is set.
asm volatile ("l: sbrs %i[sfr], %[bitno]" "\n\t"
             "rjmp lb"
             :: [sfr] "I" (& PORTB), [bitno] "n" (PB7));

```

- Local labels that contain the `sequence` `%=` which yields some number that's unique amongst all asm incarnations in the respective compilation unit:

```
// Loop until bit PORTB.7 is set.
asm volatile (".Loop.%=: sbrs %[sfr], %[bitno]" "\n\t"
             "rjmp .Loop.%="
             :: [sfr] "I" (& PORTB), [bitno] "n" (PB7));
```

Which form is used is a matter of taste. In practice, the first variant is often preferred in short sequences, whereas the second form is usually seen in longer algorithms.

For labels that are defined in the surrounding C/C++ code, `asm goto` has to be used. The `print modifier` `%x0` prints `panic` as a raw label, not as `gs(panic)` like it would be the case with `%0`.

```
int main (void)
{
    asm goto ("tst __zero_reg__" "\n\t"
            "brne %x0"
            ::: panic);
    /* ...Application code here... */
    return 0;
panic:
    // __zero_reg__ is supposed to contain 0, but doesn't.
    return 1;
}
```

This assumes that the jump offset can be encoded in the `brne` instruction in all situations. When static analysis cannot prove that the jump offset fits, then a jumpity jump has to be used:

```
asm goto ("tst __zero_reg__" "\n\t"
         "breq lf" "\n\t"
         "%~jmp %x0" "\n\t"
         "l: ;; all fine"
         ::: panic);
```

Sequence `"%~jmp"` yields `"rjmp"` or `"jmp"` depending on the architecture. Notice that a `jmp` can be relaxed to an `rjmp` with option `-mrelax` provided the jump offset fits.

## 7.8 Binding local Variables to Registers

One use of GCC's `asm` keyword is to bind local register variables to hardware registers.

**Such bindings of local variables to registers are only guaranteed during inline asm which has these variables as operands.**

### 7.8.1 Interfacing non-ABI Functions

Suppose we want to interface a non-ABI assembly function `mul_8_16` that multiplies R24 with R27:R26, clobbers R0, R1 and R25, and returns the 24-bit result in R20:R19:R18. One way to implement such an interface would be to provide an assembly function that performs the required copying and call to `mul_8_16`. Such a function would destroy some of the performance gain obtained by using assembly for `mul_8_16`: Additional copying back and forth and extra `CALL` and `RET` instructions.

The compiler comes to the rescue. We can bind local variables to the required registers:

```
extern void mul_8_16 (void); // Non-ABI function. Don't call in C/C++!

static inline __attribute__((__always_inline__))
__uint24 mul_8_16_gccabi (uint8_t val8, uint16_t val16)
{
    register uint8_t r24 __asm("r24") = val8;
    register __uint24 r18 __asm("r18");

    asm ("%~call %[func]" "\n\t"
        "clr __zero_reg__"
        : "=r" (r18)
        : "r" (r24), "x" (val16), [func] "i" (mul_8_16)
        : "r25", "r0");

    return r18;
}
```

- The 8-bit parameter is bound to R24, and the 24-bit return value is bound to R18...R20.
- The `register` keyword is mandatory.
- The hard register is specified as a string literal for the lower case register name or register number, like `"r18"` or `"r18"`. Specifications like `"R18"`, `18` or `"Z"` are not supported.
- The 16-bit parameter of `mul_8_16` happens to be required in R27:R26, which is the X register for which there is [register constraint "x"](#). Therefore, no register binding is required for `val16`.
- As `mul_8_16` clobbers the zero register R1, it has to be restored by means of `clr __zero_reg__`
- The asm is pure arithmetic and hence not volatile. (It might be advisable to make it volatile anyway, so that it won't be reordered across `sei()` or `cli()` instructions.)

Let's have a look at how this performs in a test case:

```
void use_mul_8_16_gccabi (uint8_t val, uint8_t a, uint8_t b)
{
    if (mul_8_16_gccabi (val, a * b) >= 0x2010)
        __builtin_abort();
}
```

For ATmega8 we get the following assembly:

```
use_mul_8_16_gccabi:
    mul    r22,r20
    movw  r26,r0
    clr   __zero_reg__
/* #APP */
    rcall mul_8_16
    clr   __zero_reg__
/* #NOAPP */
    cpi   r18,16
    sbci  r19,32
    cpc   r20,__zero_reg__
    brlo .L1
    rcall abort
.L1:
    ret
```

No superfluous register moves. Great!

## 7.9 Specifying the Assembly Name of Static Objects

Sometimes, it is desirable to use a different name for an object or function rather than the (mangled) name from the C/C++ implementation. Just add an asm specifier with the desired name as a string literal at the end of the declaration.

For example, this is how `avr/eeprom.h` implements the `eeprom_read_double()` function:

```
#if __SIZEOF_DOUBLE__ == 4
double eeprom_read_double (const double*) __asm("eeprom_read_dword");
#elif __SIZEOF_DOUBLE__ == 8
double eeprom_read_double (const double*) __asm("eeprom_read_qword");
#endif
```

- It uses the implementation of `eeprom_read_dword` for `eeprom_read_double`, provided `double` is a 32-bit type.
- It uses the implementation of `eeprom_read_qword` for 64-bit doubles.

## 7.10 What won't work

GCC inline asm has some limitations.

### 7.10.1 Setting a Register on one asm and using it in a different one

Sequences like the following are not supposed to work:

```
char var;

void set_var (char c)
{
    __asm ("inc r24");
    __asm ("sts var, r24");
}
```

- There is no guarantee whatsoever that the value in R24 will survive from one asm to the next. Such code might work in many situations, but it is still wrong and the compiler may very well put instructions between the asm statements that change R24 prior to the first asm and also between the asm statements.
- R24 is changed without noticing the compiler. When R24 contains other data, then that data will be trashed.

A correct code would be

```
__asm ("inc %0" "\n\t"
      "sts var, %0"
      :: "r" (c) : "memory");
```

Or

```
__asm ("inc %1" "\n\t"
      "sts %0, %1"
      : "=m" (var) : "r" (c));
```

### 7.10.2 Letting an Operand cross the Boundaries of the Y Register

It is not possible to bind a value to a local register variable that crosses the boundaries of the Y register. For example, trying to bind a 32-bit value to R31:R28 by means of

```
register uint32_t r28 __asm ("r28");
```

will result in an error message like

```
error: register specified for 'r28' isn't suitable for data type
```

Similarly, an operand described by a constraint will be located either completely below the Y register, as part of Y register, or above it.

### 7.10.3 Using Matching Constraints "=0"... "=9" with Output Operands

Suppose we want an inline asm that returns the low byte of a 16-bit value `val16`:

```
asm (" : "=l" (lo8) : "r" (val16));
```

The diagnostic will be:

```
error: matching constraint not valid in output operand
```

## 8 How to Build a Library

### 8.1 Introduction

So you keep reusing the same functions that you created over and over? Tired of cut and paste going from one project to the next? Would you like to reduce your maintenance overhead? Then you're ready to create your own library! Code reuse is a very laudable goal. With some upfront investment, you can save time and energy on future projects by having ready-to-go libraries. This chapter describes some background information, design considerations, and practical knowledge that you will need to create and use your own libraries.



## 8.2 How the Linker Works

The compiler compiles a single high-level language file (C language, for example) into a single object module file. The linker (`ld`) can only work with object modules to link them together. Object modules are the smallest unit that the linker works with.

Typically, on the linker command line, you will specify a set of object modules (that has been previously compiled) and then a list of libraries, including the Standard C Library. The linker takes the set of object modules that you specify on the command line and links them together. Afterwards there will probably be a set of "undefined references". A reference is essentially a function call. An undefined reference is a function call, with no defined function to match the call.

The linker will then go through the libraries, in order, to match the undefined references with function definitions that are found in the libraries. If it finds the function that matches the call, the linker will then link in the object module in which the function is located. This part is important: the linker links in THE ENTIRE OBJECT MODULE in which the function is located. Remember, the linker knows nothing about the functions internal to an object module, other than symbol names (such as function names). The smallest unit the linker works with is object modules.

When there are no more undefined references, the linker has linked everything and is done and outputs the final application.

## 8.3 How to Design a Library

How the linker behaves is very important in designing a library. Ideally, you want to design a library where only the functions that are called are the only functions to be linked into the final application. This helps keep the code size to a minimum. In order to do this, the way the linker works, is to only write one function per code module. This will compile to one function per object module. This is usually a very different way of doing things than writing an application!

There are always exceptions to the rule. There are generally two cases where you would want to have more than one function per object module.

The first is when you have very complementary functions that it doesn't make much sense to split them up. For example, `malloc()` and `free()`. If someone is going to use `malloc()`, they will very likely be using `free()` (or at least should be using `free()`). In this case, it makes more sense to aggregate those two functions in the same object module.

The second case is when you want to have an Interrupt Service Routine (ISR) in your library that you want to link in. The problem in this case is that the linker looks for unresolved references and tries to resolve them with code in libraries. A reference is the same as a function call. But with ISRs, there is no function call to initiate the ISR. The ISR is placed in the Interrupt Vector Table (IVT), hence no call, no reference, and no linking in of the ISR. In order to do this, you have to trick the linker in a way. Aggregate the ISR, with another function in the same object module, but have the other function be something that is required for the user to call in order to use the ISR, like perhaps an initialization function for the subsystem, or perhaps a function that enables the ISR in the first place.

## 8.4 Creating a Library

The librarian program is called `ar` (for "archiver") and is found in the GNU Binutils project. This program will have been built for the AVR target and will therefore be named `avr-ar`.

The job of the librarian program is simple: aggregate a list of object modules into a single library (archive) and create an index for the linker to use. The name that you create for the library filename must follow a specific pattern: `libname.a`. The *name* part is the unique part of the filename that you create. It makes it easier if the *name* part relates to what the library is about. This *name* part must be prefixed by "lib", and it must have a file extension of `.a`, for "archive". The reason for the special form of the filename is for how the library gets used by the toolchain, as we will see later on.

**Note**

The filename is case-sensitive. Use a lowercase "lib" prefix, and a lowercase ".a" as the file extension.

The command line is fairly simple:

```
avr-ar rcs <library name> <list of object modules>
```

The `r` command switch tells the program to insert the object modules into the archive with replacement. The `c` command line switch tells the program to create the archive. And the `s` command line switch tells the program to write an object-file index into the archive, or update an existing one. This last switch is very important as it helps the linker to find what it needs to do its job.

**Note**

The command line switches are case sensitive! There are uppercase switches that have completely different actions.

MFile and the WinAVR distribution contain a Makefile Template that includes the necessary command lines to build a library. You will have to manually modify the template to switch it over to build a library instead of an application.

See the GNU Binutils manual for more information on the `ar` program.

## 8.5 Using a Library

To use a library, use the `-l` switch on your linker command line. The string immediately following the `-l` is the unique part of the library filename that the linker will link in. For example, if you use:

```
-lm
```

this will expand to the library filename:

```
libm.a
```

which happens to be the math library included in AVR-LibC.

If you use this on your linker command line:

```
-lprintf_float
```

then the linker will look for a library called:

```
libprintf_float.a
```

This is why naming your library is so important when you create it!

The linker will search libraries in the order that they appear on the command line. Whichever function is found first that matches the undefined reference, it will be linked in.

There are also command line switches that tell GCC which directory to look in (`-L`) for the libraries that are specified to be linked in with `-l`.

See the GNU Binutils manual for more information on the GNU linker (`ld`) program.

## 9 Benchmarks

The results below can only give a rough estimate of the resources necessary for using certain library functions. There is a number of factors which can both increase or reduce the effort required:

- Expenses for preparation of operands and their stack are not considered.
- In the table, the size includes all additional functions (for example, function to multiply two integers) but they are only linked from the library.
- Expenses of time of performance of some functions essentially depend on parameters of a call, for example, `qsort()` is recursive, and `sprintf()` receives parameters in a stack.
- Different versions of the compiler can give a significant difference in code size and execution time. For example, the `dtostrf()` function, compiled with `avr-gcc 3.4.6`, requires 930 bytes. After transition to `avr-gcc 4.2.3`, the size become 1088 bytes.

### 9.1 A few of libc functions.

Avr-gcc version is 4.7.1

The size of function is given in view of all picked up functions. By default AVR-LibC is compiled with `-mcall-prologues` option. In brackets the size without taking into account modules of a prologue and an epilogue is resulted. Both of the size can coincide, if function does not cause a prologue/epilogue.

Function	Units	Avr2	Avr25	Avr4
atoi ("12345")	Flash bytes	82 (82)	78 (78)	74 (74)
	Stack bytes	2	2	2
	MCU clocks	155	149	149
atol ("12345")	Flash bytes	122 (122)	118 (118)	118 (118)
	Stack bytes	2	2	2
	MCU clocks	221	219	219
dtostrf (1.2345, s, 6, 0)	Flash bytes	1116 (1004)	1048 (938)	1048 (938)
	Stack bytes	17	17	17
	MCU clocks	1247	1105	1105
dtostrf (1.2345, 15, 6, s)	Flash bytes	1616 (1616)	1508 (1508)	1508 (1508)
	Stack bytes	38	38	38
	MCU clocks	1634	1462	1462
itoa (12345, s, 10)	Flash bytes	110 (110)	102 (102)	102 (102)
	Stack bytes	2	2	2
	MCU clocks	879	875	875
ltoa (12345L, s, 10)	Flash bytes	134 (134)	126 (126)	126 (126)
	Stack bytes	2	2	2
	MCU clocks	1597	1593	1593
malloc (1)	Flash bytes	768 (712)	714 (660)	714 (660)
	Stack bytes	6	6	6
	MCU clocks	215	201	201
realloc ((void *)0, 1)	Flash bytes	1284 (1172)	1174 (1064)	1174 (1064)
	Stack bytes	18	18	18
	MCU clocks	305	286	286
qsort (s, sizeof(s), 1, cmp)	Flash bytes	1252 (1140)	1022 (912)	1028 (918)
	Stack bytes	42	42	42
	MCU clocks	21996	19905	17541
sprintf_min (s, "%d", 12345)	Flash bytes	1224 (1112)	1092 (982)	1088 (978)
	Stack bytes	53	53	53
	MCU clocks	1841	1694	1689

sprintf (s, "%d", 12345)	Flash bytes Stack bytes MCU clocks	1614 (1502) 58 1647	1476 (1366) 58 1552	1454 (1344) 58 1547
sprintf_fit (s, "%e", 1.2345)	Flash bytes Stack bytes MCU clocks	3228 (3116) 67 2573	2990 (2880) 67 2311	2968 (2858) 67 2311
sscanf_min ("12345", "%d", &i)	Flash bytes Stack bytes MCU clocks	1532 (1420) 55 1607	1328 (1218) 55 1446	1328 (1218) 55 1446
sscanf ("12345", "%d", &i)	Flash bytes Stack bytes MCU clocks	2008 (1896) 55 1610	1748 (1638) 55 1449	1748 (1638) 55 1449
sscanf ("point,color", "[%a-z]", s)	Flash bytes Stack bytes MCU clocks	2008 (1896) 86 3067	1748 (1638) 86 2806	1748 (1638) 86 2806
sscanf_fit ("1.2345", "%e", &x)	Flash bytes Stack bytes MCU clocks	3464 (3352) 71 2497	3086 (2976) 71 2281	3070 (2960) 71 2078
strtod ("1.2345", &p)	Flash bytes Stack bytes MCU clocks	1632 (1520) 20 1235	1536 (1426) 20 1177	1480 (1480) 21 1124
strtol ("12345", &p, 0)	Flash bytes Stack bytes MCU clocks	918 (806) 22 956	834 (724) 22 891	792 (792) 28 794

## 9.2 Math functions.

The table contains the number of MCU clocks to calculate a function with a given argument(s). The main reason of a big difference between Avr2 and Avr4 is a hardware multiplication.

Function	Avr2	Avr4
__addsf3 (1.234, 5.678)	113	108
__mulsf3 (1.234, 5.678)	375	138
__divsf3 (1.234, 5.678)	466	465
acos (0.54321)	4411	2455
asin (0.54321)	4517	2556
atan (0.54321)	4710	2271
atan2 (1.234, 5.678)	5270	2857
cbrt (1.2345)	2684	2555
ceil (1.2345)	177	177
cos (1.2345)	3387	1671
cosh (1.2345)	4922	2979
exp (1.2345)	4708	2765
fdim (5.678, 1.234)	111	111
floor (1.2345)	180	180
fmax (1.234, 5.678)	39	37
fmin (1.234, 5.678)	35	35
fmod (5.678, 1.234)	131	131
frexp (1.2345, 0)	42	41
hypot (1.234, 5.678)	1341	866
ldexp (1.2345, 6)	42	42
log (1.2345)	4142	2134
log10 (1.2345)	4498	2260
modf (1.2345, 0)	433	429
pow (1.234, 5.678)	9293	5047
round (1.2345)	150	150

sin (1.2345)	3353	1653
sinh (1.2345)	4946	3003
sqrt (1.2345)	494	492
tan (1.2345)	4381	2426
tanh (1.2345)	5126	3173
trunc (1.2345)	178	178

## 10 Porting From IAR to AVR GCC

### 10.1 Introduction

C language was designed to be a portable language. There two main types of porting activities: porting an application to a different platform (OS and/or processor), and porting to a different compiler. Porting to a different compiler can be exacerbated when the application is an embedded system. For example, the C language Standard, strangely, does not specify a standard for declaring and defining Interrupt Service Routines (ISRs). Different compilers have different ways of defining registers, some of which use non-standard language constructs.

This chapter describes some methods and pointers on porting an AVR application built with the IAR compiler to the GNU toolchain (AVR GCC). Note that this may not be an exhaustive list.

### 10.2 Registers

IO header files contain identifiers for all the register names and bit names for a particular processor. IAR has individual header files for each processor and they must be included when registers are being used in the code. For example:

```
#include <iom169.h>
```

#### Note

IAR does not always use the same register names or bit names that are used in the AVR datasheet.

AVR GCC also has individual IO header files for each processor. However, the actual processor type is specified as a command line flag to the compiler. (Using the `-mmcu=processor` flag.) This is usually done in the Makefile. This allows you to specify only a single header file for any processor type:

```
#include <avr/io.h>
```

#### Note

The forward slash in the `<avr/io.h>` file name that is used to separate subdirectories can be used on Windows distributions of the toolchain and is the recommended method of including this file.

The compiler knows the processor type and through the single header file above, it can pull in and include the correct individual IO header file. This has the advantage that you only have to specify one generic header file, and you can easily port your application to another processor type without having to change every file to include the new IO header file.

The AVR toolchain tries to adhere to the exact names of the registers and names of the bits found in the AVR datasheet. There may be some discrepancies between the register names found in the IAR IO header files and the AVR GCC IO header files.

## 10.3 Interrupt Service Routines (ISRs)

As mentioned above, the C language Standard, strangely, does not specify a standard way of declaring and defining an ISR. Hence, every compiler seems to have their own special way of doing so.

IAR declares an ISR like so:

```
#pragma vector=TIMER0_OVF_vect
__interrupt void MotorPWMBottom()
{
    // code
}
```

In AVR GCC, you declare an ISR like so:

```
ISR(PCINT1_vect)
{
    //code
}
```

AVR GCC uses the `ISR` macro to define an ISR. This macro requires the header file:

```
#include <avr/interrupt.h>
```

The names of the various interrupt vectors are found in the individual processor IO header files that you must include with `<avr/io.h>`.

### Note

The names of the interrupt vectors in AVR GCC has been changed to match the names of the vectors in IAR. This significantly helps in porting applications from IAR to AVR GCC.

## 10.4 Intrinsic Routines

IAR has a number of intrinsic routine such as

```
__enable_interrupts() __disable_interrupts() __watchdog_reset()
```

These intrinsic functions compile to specific AVR opcodes (SEI, CLI, WDR).

There are equivalent macros that are used in AVR GCC, however they are not located in a single include file.

AVR GCC has `sei()` for `__enable_interrupts()`, and `cli()` for `__disable_interrupts()`. Both of these macros are located in `<avr/interrupt.h>`.

AVR GCC has the macro `wdt_reset()` in place of `__watchdog_reset()`. However, there is a whole Watchdog Timer API available in AVR GCC that can be found in `<avr/wdt.h>`.

## 10.5 Flash Variables

The C language was not designed for Harvard architecture processors with separate memory spaces. This means that there are various non-standard ways to define a variable whose data resides in the Program Memory (Flash).

IAR uses a non-standard keyword to declare a variable in Program Memory:

```
__flash int mydata[] = ...
```

AVR GCC uses Variable Attributes to achieve the same effect:

```
int mydata[] __attribute__((progmem))
```

**Note**

See the GCC User Manual for more information about Variable Attributes.

AVR-LibC provides a convenience macro for the Variable Attribute:

```
#include <avr/pgmspace.h>
.
.
.
int mydata[] PROGMEM = ....
```

**Note**

The `PROGMEM` macro expands to the Variable Attribute of `progmem`. This macro requires that you include `<avr/pgmspace.h>`. This is the canonical method for defining a variable in Program Space.

To read back flash data, use the `pgm_read_*`() macros defined in `<avr/pgmspace.h>`. All Program Memory handling macros are defined there.

There is also a way to create a method to define variables in Program Memory that is common between the two compilers (IAR and AVR GCC). Create a header file that has these definitions:

```
#if defined(__ICCAVR__) // IAR C Compiler
#define FLASH_DECLARE(x) __flash x
#endif
#if defined(__GNUC__) // GNU Compiler
#define FLASH_DECLARE(x) x __attribute__((__progmem__))
#endif
```

This code snippet checks for the IAR compiler or for the GCC compiler and defines a macro `FLASH_DECLARE(x)` that will declare a variable in Program Memory using the appropriate method based on the compiler that is being used. Then you would use it like so:

```
FLASH_DECLARE(int mydata[] = ...);
```

## 10.6 Non-Returning main()

To declare `main()` to be a non-returning function in IAR, it is done like this:

```
__C_task void main(void)
{
    // code
}
```

To do the equivalent in AVR GCC, do this:

```
void main(void) __attribute__((noreturn));
```

```
void main(void)
{
    //...
}
```

**Note**

See the GCC User Manual for more information on Function Attributes.

In AVR GCC, a prototype for `main()` is required so you can declare the function attribute to specify that the `main()` function is of type "noreturn". Then, define `main()` as normal. Note that the return type for `main()` is now `void`.

## 10.7 Locking Registers

The IAR compiler allows a user to lock general registers from r15 and down by using compiler options and this keyword syntax:

```
__regvar __no_init volatile unsigned int filteredTimeSinceCommutation @14;
```

This line locks r14 for use only when explicitly referenced in your code through the var name "filteredTimeSinceCommutation". This means that the compiler cannot dispose of it at its own will.

To do this in AVR GCC, do this:

```
register unsigned char counter asm("r3");
```

Typically, it should be possible to use r2 through r15 that way.

### Note

Do not reserve r0 or r1 as these are used internally by the compiler for a temporary register and for a zero value.

Locking registers is not recommended in AVR GCC as it removes this register from the control of the compiler, which may make code generation worse. Use at your own risk.

## 11 Frequently Asked Questions

### 11.1 FAQ Index

#### • Interrupts

- [Why doesn't my program recognize a variable updated in an interrupt routine?](#)
- [Why do some 16-bit timer registers sometimes get trashed?](#)
- [What ISR names are available for my device?](#)
- [What pitfalls exist when writing reentrant code?](#)
- [Why are interrupts re-enabled in the middle of writing the stack pointer?](#)
- [Why are \(many\) interrupt flags cleared by writing a logical 1?](#)

#### • C/C++

- [Can I use C++ on the AVR?](#)
- [Which -O flag to use?](#)
- [Shouldn't I initialize all my variables?](#)
- [How do I pass an IO port as a parameter to a function?](#)
- [Why do all my "foo...bar" strings eat up the SRAM?](#)
- [How do I put an array of strings completely in ROM?](#)
- [How to modify MCUCR or WDTCSR early?](#)
- [How do I perform a software reset of the AVR?](#)
- [On a device with more than 128 KiB of flash, how to make function pointers work?](#)
- [What registers are used by the C compiler?](#)
- [How to permanently bind a variable to a register?](#)
- [Why is assigning ports in a "chain" a bad idea?](#)
- [What is all this \\_BV\(\) stuff about?](#)
- [Is it really impossible to program the ATtinyXX in C?](#)



- **(Inline) Assembly**
  - How do I use a `#define`'d constant in an asm statement?
  - Which AVR-specific assembler operators are available?
- **Linking and Binaries**
  - How do I relocate code to a fixed address?
  - How to add a raw binary image to linker output?
  - Why are there five different linker scripts?
- **Static Analysis**
  - Which header files are included in my program?
  - Which macros are defined in my program? Where are they defined, and to what value?
  - How to detect RAM memory and variable overlap problems?
- **Debugging**
  - Why does the PC randomly jump around when single-stepping through my program in `avr-gdb`?
  - How do I trace an assembler file in `avr-gdb`?
- **Hardware**
  - Why are some addresses of the EEPROM corrupted (usually address zero)?
  - My UART is generating nonsense! My ATmega128 keeps crashing! Port F is completely broken!
  - Why have "programmed" fuses the bit value 0?
  - How to use external RAM?
- **Other**
  - Why is my baud rate wrong?
  - What is this "clock skew detected" message?

## 11.2 Why doesn't my program recognize a variable updated in an interrupt routine?

When using the optimizer, in a loop like the following one:

```
uint8_t flag;
...
ISR(SOME_vect) {
    flag = 1;
}
...

while (flag == 0) {
    ...
}
```

the compiler will typically access `flag` only once, and optimize further accesses completely away, since its code path analysis shows that nothing inside the loop could change the value of `flag` anyway. To tell the compiler that this variable could be changed outside the scope of its code path analysis (e. g. from within an interrupt routine), the variable needs to be declared like:

```
volatile uint8_t flag;
```

Back to [FAQ Index](#).

## 11.3 How to permanently bind a variable to a register?

This can be done with

```
register uint8_t counter __asm("r3");
```

Typically, it should be safe to use r2 through r7 that way.

Registers r8 through r25 can be used for argument passing by the compiler in case many or long arguments are being passed to callees. If this is not the case throughout the entire application, these registers could be used for register variables as well.

Extreme care should be taken that the entire application is compiled with a consistent set of register-allocated variables including possibly used library functions. This can be achieved by compiling each module with `-ffixed-r3` or `-ffixed-3`. Notice that when you are using library functions from `libgcc` (the `avr-gcc` runtime library) or `AVR-LibC`, then these libraries were generated *without* the requirement to avoid specific registers. Hence when you are using libraries from the distribution, you must make sure that none of the reserved registers is used in the generated binary.

Also notice that global register variables can't be volatile, because only variables in memory can be volatile, and register variables are not located in memory.

Back to [FAQ Index](#).

## 11.4 How to modify MCUCR or WDTCR early?

Basically, write a small function which looks like this:

```
#include <avr/io.h>

static __attribute__((used, unused, naked, section(".init3")))
void init_MCUCR (void);

void init_MCUCR (void)
{
    MCUCR = _BV(SRE) | _BV(SRW);
}
```

Do not call this function by hand! This piece of code will be inserted in [startup code](#), which is run right after reset. For the meaning of the attributes, see [How do I perform a software reset of the AVR?](#)

The advantage of this method is that you can insert any initialization code you want (just remember that this is very early startup – no stack and no `__zero_reg__` yet), and no program memory space is wasted if this feature is not used.

There should be no need to modify linker scripts anymore, except for some very special cases. It is best to leave `__stack` at its default value (end of internal SRAM – faster, and required on some devices like ATmega161 because of errata), and add `-Wl,-Tdata,0x801100` to start the data section above the stack.

For more information on using sections, see [Memory Sections](#). There is also an example for [In C/C++ Code](#). Note that in C code, any such function would preferably be placed into section `.init3` as the code in `.init2` ensures the internal register `__zero_reg__` is already cleared.

Back to [FAQ Index](#).

## 11.5 What is all this `_BV()` stuff about?

When performing low-level output work, which is a very central point in microcontroller programming, it is quite common that a particular bit needs to be set or cleared in some IO register. While the device documentation provides mnemonic names for the various bits in the IO registers, and the [AVR device-specific IO definitions](#) reflect these names in definitions for numerical constants, a way is needed to convert a bit number (usually within a byte register) into a byte value that can be assigned directly to the register. However, sometimes the direct bit numbers are needed as well (e. g. in an `SBI()` instruction), so the definitions cannot usefully be made as byte values in the first place.

So in order to access a particular bit number as a byte value, use the `_BV()` macro. Of course, the implementation of this macro is just the usual bit shift (which is done by the compiler anyway, thus doesn't impose any run-time penalty), so the following applies:

```
_BV(3) => 1 << 3 => 0x08
```

However, using the macro often makes the program better readable.

"BV" stands for "bit value", in case someone might ask you. :-)

**Example:** clock timer 2 with full IO clock (`CS2x = 0b001`), toggle OC2 output on compare match (`COM2x = 0b01`), and clear timer on compare match (`CTC2 = 1`). Make OC2 (`PD7`) an output.

```
TCCR2 = _BV(COM20) | _BV(CTC2) | _BV(CS20);
DDRD = _BV(PD7);
```

Back to [FAQ Index](#).

## 11.6 Can I use C++ on the AVR?

Basically yes, C++ is supported (assuming your compiler has been configured and compiled to support it, of course). Source files ending in `.cc`, `.cpp` or `.C` will automatically cause the compiler frontend to invoke the C++ compiler. Alternatively, the C++ compiler could be explicitly called by the name `avr-c++`.

However, there's currently no support for `libstdc++`, the standard support library needed for a complete C++ implementation. This imposes a number of restrictions on the C++ programs that can be compiled. Among them are:

- Obviously, none of the C++ related standard functions, classes, and template classes are available.
- The operators `new` and `delete` are not implemented, attempting to use them will cause the linker to complain about undefined external references. (This could perhaps be fixed.)
- Some of the supplied include files are not C++ safe, i. e. they need to be wrapped into

```
extern "C" { ... }
```

 (This could certainly be fixed, too.)
- Exceptions are not supported. Since exceptions are enabled by default in the C++ frontend, they explicitly need to be turned off using `-fno-exceptions` in the compiler options. Failing this, the linker will complain about an undefined external reference to `__gxx_personality_sj0`.

Constructors and destructors *are* supported though, including global ones.

When programming C++ in space- and runtime-sensitive environments like microcontrollers, extra care should be taken to avoid unwanted side effects of the C++ calling conventions like implied copy constructors that could be called upon function invocation etc. These things could easily add up into a considerable amount of time and program memory wasted. Thus, casual inspection of the generated assembler code (using the `-S` compiler option) seems to be warranted.

Back to [FAQ Index](#).

## 11.7 Shouldn't I initialize all my variables?

Variables in static storage are guaranteed to be initialized by the C standard. This includes global and static variables without explicit initializer, which are initialized to 0. `avr-gcc` does this by placing the appropriate code into section `.init4`. With respect to the standard, this sentence is somewhat simplified (because the standard allows for machines where the actual bit pattern used differs from all bits being 0), but for the AVR target, in general, all integer-type variables are set to 0, all pointers to a NULL pointer, and all floating-point variables to 0.0.

As long as these variables are not explicitly initialized, or their initializer is all zeros, they go into the `.bss` output section. This section simply records the size of the variable, but otherwise doesn't consume space, neither within the object file nor within flash memory. (Of course, being a variable, it will consume space in the target's SRAM.)

In contrast, global and static variables that have a non-zero initializer go into the `.data` output section of the file. This will cause them to consume space in the object file (in order to record the initializing value), *and* in the flash ROM of the target device. The latter is needed to initialize the variables in RAM from the initializers kept in ROM during the startup code, so that all variables will have their expected initial values when `main()` is entered.

Back to [FAQ Index](#).

## 11.8 Why do some 16-bit timer registers sometimes get trashed?

Some of the timer-related 16-bit IO registers use a temporary register (called TEMP in the AVR datasheet) to guarantee an atomic access to the register despite the fact that two separate 8-bit IO transfers are required to actually move the data. Typically, this includes access to the current timer/counter value register (`TCNTn`), the input capture register (`ICRn`), and write access to the output compare registers (`OCRnM`). Refer to the actual datasheet for each device's set of registers that involves the TEMP register.

When accessing one of the registers that use TEMP from the main application, and possibly any other one from within an interrupt routine, care must be taken that no access from within an interrupt context could clobber the TEMP register data of an in-progress transaction that has just started elsewhere.

To protect interrupt routines against other interrupt routines, it's usually best to use the `ISR()` macro when declaring the interrupt function, and to ensure that interrupts are still disabled when accessing those 16-bit timer registers.

Within the main program, access to those registers could be encapsulated in calls to the `cli()` and `sei()` macros. If the status of the global interrupt flag before accessing one of those registers is uncertain, something like the following example code can be used.

```
uint16_t
read_timer1(void)
{
    uint8_t sreg;
    uint16_t val;

    sreg = SREG;
    cli();
    val = TCNT1;
    SREG = sreg;

    return val;
}
```

Back to [FAQ Index](#).

## 11.9 How do I use a #define'd constant in an asm statement?

So you tried this:

```
asm volatile ("sbi 0x18, 7");
```

Which works. When you do the same thing but replace the address of the port by its macro name, like this:

```
asm volatile ("sbi PORTB, 7");
```

you get a syntax error from the assembler: "Error: constant value required".

PORTB is a precompiler definition included in the processor specific file included in `avr/io.h`. As you may know, the precompiler will not touch strings, and PORTB gets passed to the assembler instead of 0x18. One way to avoid this problem is:

```
asm volatile ("sbi %0, 7" :: "I" (_SFR_IO_ADDR(PORTB)));
```

### Note

For C programs, rather use the standard C bit operators instead, so the above would be expressed as `PORTB |= (1 << 7)`. The optimizer will take care to transform this into a single SBI instruction, assuming the operands allow for this.

There are situation though where the address of a special function register (SFR) is required in inline assembly. When the register can be accessed by LDS and STS, one can use the RAM address of the SFR:

```
asm volatile ("sts %0, __zero_reg__" :: "n" (&PORTB));
```

When the I/O address of the register is required, one way is to use `_SFR_IO_ADDR` to get the I/O address like in the example above. A different approach is to use inline asm [print modifier i](#) supported since `avr-gcc v4.7`:

```
asm volatile ("out %i0, __zero_reg__" :: "n" (&PORTB));
```

The `i0` will print the address of `PORTB` as an I/O address.

Back to [FAQ Index](#).

## 11.10 Why does the PC randomly jump around when single-stepping through my program in avr-gdb?

When compiling a program with both optimization (`-O`) and debug information (`-g`) which is fortunately possible in `avr-gcc`, the code watched in the debugger is optimized code. It is guaranteed that the code runs with the exact same optimizations as it would run without the `-g` switch.

Since the compiler is free to reorder code execution as long as the semantics do not change, code is often rearranged in order to make it possible to use a single branch instruction for conditional operations. Branch instructions can only cover a short range for the target PC (-63 through +64 words from the current PC). If a branch instruction cannot be used directly, the compiler needs to work around it by combining a skip instruction together with a relative jump (`rjmp`) instruction, which will need one additional word of ROM.

Another side effect of optimization is that variable usage is restricted to the area of code where it is actually used. So if a variable was placed in a register at the beginning of some function, this same register can be re-used later on if the compiler notices that the first variable is no longer used inside that function, even though the variable is still in lexical scope. When trying to examine the variable in `avr-gdb`, the displayed result will then look garbled.

So in order to avoid these side effects, optimization can be turned off while debugging, or at least optimization level `-Og` can be used which was introduced to improve good debugging experience while it still provides a reasonable amount of optimization.

However, some of these optimizations might also have the side effect of uncovering bugs that would otherwise not be obvious, so it must be noted that turning off optimization can easily change the bug pattern. In most cases, you are better off leaving optimizations enabled while debugging.

Back to [FAQ Index](#).

## 11.11 How do I trace an assembler file in avr-gdb?

When using the `-g` compiler option, `avr-gcc` only generates line number and other debug information for C (and C++) files that pass the compiler. Functions that don't have line number information will be completely skipped by a single `step` command in `gdb`. This includes functions linked from a standard library, but by default also functions defined in an assembler source file, since the `-g` compiler switch does not apply to the assembler.

So in order to debug an assembler input file (possibly one that has to be passed through the C preprocessor), it's the assembler that needs to be told to include line-number information into the output file. (Other debug information like data types and variable allocation cannot be generated, since unlike a compiler, the assembler basically doesn't know about this.) This is done using the (GNU) assembler option `--gstabs`.

Example:

```
$ avr-as -mmcu=atmega128 --gstabs -o foo.o foo.s
```

When the assembler is not called directly but through the C compiler frontend (either implicitly by passing a source file ending in `.S`, or explicitly using `-x assembler-with-cpp`), the compiler frontend needs to be told to pass the `--gstabs` option down to the assembler. This is done using `-Wa,--gstabs`. Please take care to *only* pass this option when compiling an assembler input file. Otherwise, the assembler code that results from the C compilation stage will also get line number information, which confuses the debugger.

Note

You can also use `-Wa, -gstabs` since the compiler will add the extra `'-'` for you.

Example:

```
$ EXTRA_OPTS="-Wall -mmcu=atmega128 -x assembler-with-cpp"
$ avr-gcc -Wa,--gstabs ${EXTRA_OPTS} -c -o foo.o foo.S
```

Also note that the debugger might get confused when entering a piece of code that has a non-local label before, since it then takes this label as the name of a new function that appears to have been entered. Thus, the best practice to avoid this confusion is to only use non-local labels when declaring a new function, and restrict anything else to local labels. Local labels consist just of a number only. References to these labels consist of the number, followed by the letter **b** for a backward reference, or **f** for a forward reference. These local labels may be re-used within the source file, references will pick the closest label with the same number and given direction.

Example:

```
myfunc:
    push    r16
    push    r17
    push    r18
    push    YL
    push    YH
    ...
    clr     r16                ; start loop
    ldi     YL, lo8(sometable)
    ldi     YH, hi8(sometable)
    rjmp   2f                ; jump to loop test at end
1:  ld     r17, Y+            ; loop continues here
    ...
    breq   3f                ; return from myfunc prematurely
    ...
    inc    r16
2:  cmp    r16, r18
    brlo   1b                ; jump back to top of loop
3:  pop    YH
    pop    YL
    pop    r18
    pop    r17
    pop    r16
    ret
```

Back to [FAQ Index](#).

## 11.12 How do I pass an IO port as a parameter to a function?

Consider this example code:

```
#include <inttypes.h>
#include <avr/io.h>

void
set_bits_func_wrong (volatile uint8_t port, uint8_t mask)
{
    port |= mask;
}

void
set_bits_func_correct (volatile uint8_t *port, uint8_t mask)
{
    *port |= mask;
}

#define set_bits_macro(port,mask) ((port) |= (mask))

int main (void)
{
    set_bits_func_wrong (PORTB, 0xaa);
    set_bits_func_correct (&PORTB, 0x55);
    set_bits_macro (PORTB, 0xf0);

    return (0);
}
```

The first function will generate object code which is not even close to what is intended. The major problem arises when the function is called. When the compiler sees this call, it will actually pass the value of the PORTB register (using an IN instruction), instead of passing the address of PORTB (e.g. memory mapped io addr of 0x38, io port 0x18 for the mega128). This is seen clearly when looking at the disassembly of the call:

```
    set_bits_func_wrong (PORTB, 0xaa);
10a:  6a ea          ldi    r22, 0xAA
10c:  88 b3          in     r24, 0x18
10e:  0e 94 65 00    call   0xca
```

So, the function, once called, only sees the value of the port register and knows nothing about which port it came from. At this point, whatever object code is generated for the function by the compiler is irrelevant. The interested reader can examine the full disassembly to see that the function's body is completely fubar.

The second function shows how to pass (by reference) the memory mapped address of the io port to the function so that you can read and write to it in the function. Here's the object code generated for the function call:

```
    set_bits_func_correct (&PORTB, 0x55);
112:  65 e5          ldi    r22, 0x55
114:  88 e3          ldi    r24, 0x38
116:  90 e0          ldi    r25, 0x00
118:  0e 94 7c 00    call   0xf8
```

You can clearly see that 0x0038 is correctly passed for the address of the io port. Looking at the disassembled object code for the body of the function, we can see that the function is indeed performing the operation we intended:

```
void
set_bits_func_correct (volatile uint8_t *port, uint8_t mask)
{
    f8:  fc 01          movw   r30, r24
        *port |= mask;
    fa:  80 81          ld     r24, Z
    fc:  86 2b          or    r24, r22
    fe:  80 83          st    Z, r24
}
100:  08 95          ret
```

Notice that we are accessing the io port via the LD and ST instructions.

The port parameter must be volatile to avoid a compiler warning.

**Note**

Because of the nature of the `IN` and `OUT` assembly instructions, they can not be used inside the function when passing the port in this way. Readers interested in the details should consult the *Instruction Set* datasheet.

Finally we come to the macro version of the operation. In this contrived example, the macro is the most efficient method with respect to both execution speed and code size:

```
set_bits_macro (PORTB, 0xf0);
11c:  88 b3          in     r24, 0x18
11e:  80 6f          ori   r24, 0xF0
120:  88 bb          out   0x18, r24
```

Of course, in a real application, you might be doing a lot more in your function which uses a passed by reference io port address and thus the use of a function over a macro could save you some code space, but still at a cost of execution speed.

Care should be taken when such an indirect port access is going to one of the 16-bit IO registers where the order of write access is critical (like some timer registers). All versions of `avr-gcc` up to 3.3 will generate instructions that use the wrong access order in this situation (since with normal memory operands where the order doesn't matter, this sometimes yields shorter code).

See <http://mail.gnu.org/archive/html/avr-libc-dev/2003-01/msg00044.html> for a possible workaround.

`avr-gcc` versions after 3.3 have been fixed in a way where this optimization will be disabled if the respective pointer variable is declared to be `volatile`, so the correct behaviour for 16-bit IO ports can be forced that way.

Back to [FAQ Index](#).

**11.13 What registers are used by the C compiler?**

See also the [Type Layout](#), [Register Layout](#) and [Calling Convention](#) sections in the `avr-gcc` Wiki.

**Data types** `char` is 8 bits, `int` and `short` are 16 bits, `long` is 32 bits, `long long` is 64 bits, `float` is 32 bits, `double` and `long double` are 32 bits or 64 bits, pointers are 16 bits (function pointers are word addresses to allow addressing up to 128K program memory space).

- There is a `-mint8` option (see [Options for the C compiler avr-gcc](#)) to make `int` and `short` 8 bits, `long` 16 bits and `long long` 32 bits. But that is not supported by AVR-LibC (except for `stdint.h` and `avr/pgmspace.h`, but no 64-bit integer types are available) and violates C standards (`int` must be at least 16 bits).

**Call-used registers (r18-r27, r30-r31)** May be allocated by `gcc` for local data. You *may* use them freely in assembly subroutines. Calling C subroutines can clobber any of them - the caller is responsible for saving and restoring.

For the AVR\_TINY architecture (ATtiny10 and relatives), r20-r27 and r30-31 are call-clobbered.

**Call-saved registers (r2-r17, r28-r29)** May be allocated by `gcc` for local data. Calling C subroutines leaves them unchanged. Assembly subroutines are responsible for saving and restoring these registers, if changed. r29↔:r28 (Y pointer) is used as a frame pointer (points to local data on stack) if necessary. The requirement for the callee to save/preserve the contents of these registers even applies in situations where the compiler assigns them for argument passing.

For the AVR\_TINY architecture (ATtiny10 etc.), r18-r19 and r28-r29 are call-saved. Registers r0 through r15 do not exist.



**Fixed registers (r0, r1)** Never allocated by gcc for local data, but often used for fixed purposes:

- **r0** (`__tmp_reg__`) — temporary register, can be clobbered by any code (except interrupt handlers which save it), *may* be used to remember something for a while within one piece of assembly code
- **r1** (`__zero_reg__`) — assumed to be always zero in any C code, *may* be used to remember something for a while within one piece of assembler code, but *must* then be cleared after use (`clr __zero_reg__`). This includes any use of the `[f]mul[s[u]]` instructions, which return their result in r1:r0. Interrupt handlers save and clear `__zero_reg__` on entry, and restore it on exit (in case it was non-zero).
- **T flag** — the T flag in the status register (SREG) can be used the same way like `__tmp_reg__`.

For the AVR\_TINY architecture (ATtiny10 etc.), `__tmp_reg__` is r16, and `__zero_reg__` is r17.

**Function call conventions** Arguments - allocated left to right, r25 to r8. All arguments are aligned to start in even-numbered registers (odd-sized arguments, including `char`, have one free register above them). This allows making better use of the `movw` instruction on the enhanced core.

If too many, those that don't fit are passed on the stack.

On AVR\_TINY, r25 to r20 are used to pass values.

- Return values: 8-bit in r24, 16-bit in r25:r24, up to 32 bits in r22-r25, up to 64 bits in r18-r25.
- Arguments to functions with a variable number of lists like `printf` get all their values on the stack. `char` is extended to `int`, and `float` is extended to `double`.
- When an argument is passed on the stack, all subsequent arguments are also passed on the stack.
- An argument is either passed completely in registers or completely on the stack.
- Arguments with a size of zero or with a size larger than 8 bytes (4 bytes on AVR\_TINY) are returned in memory. The caller provides the memory location as implicit first argument to the callee.
- When an argument is returned in registers, its size is padded to the next power of 2.

Back to [FAQ Index](#).

## 11.14 How do I put an array of strings completely in ROM?

There are times when you may need an array of strings which will never be modified. In this case, you don't want to waste ram storing the constant strings. The most obvious (and incorrect) thing to do is this:

```
#include <avr/pgmspace.h>

const char* const array[2] PROGMEM = {
    "Foo",
    "Bar"
};

int main (void)
{
    char buf[32];
    strcpy_P (buf, array[1]);
    return 0;
}
```

The result is not what you want though. What you end up with is the array stored in ROM, while the individual strings end up in RAM (in some `.rodata` input section).

To work around this, you need to do something like this:

```
#include <avr/pgmspace.h>

static const char foo[] PROGMEM = "Foo";
static const char bar[] PROGMEM = "Bar";

const char* const array[2] PROGMEM = {
    foo,
    bar
}
```

```
};

void func (uint8_t i)
{
    char buf[32];

    const char *p = pgm_read_ptr (&array[i]);
    strcpy_P (buf, p);
}

```

Looking at the disassembly of the resulting object file we see that array is in flash as such:

```
00000026 <array>:
 26:  2e 00 2a 00

0000002a <bar>:
 2a:  42 61 72 00          Bar.

0000002e <foo>:
 2e:  46 6f 6f 00          Foo.

```

foo is at address 0x002e.

bar is at address 0x002a.

array is at address 0x0026.

### 11.14.1 Using named address-spaces

An alternative is to use the named address-space `__flash`, which is supported since `avr-gcc v4.7` and in GNU-C99 and up:

```
#include <avr/pgmspace.h>

#define F(X) ((const __flash char[]) { X })

const __flash char* const __flash array[] =
{
    F("Foo"), F("Bar")
};

int compare (const char *str, uint8_t i)
{
    return strcmp_P (str, array[i]);
}

```

Moreover, there is no more need for `pgm_read_xxx()`: The (addresses of) the string literals can be read directly by means of `array[i]`. The header is only needed for the `strcmp_P` prototype.

Back to [FAQ Index](#).

## 11.15 How to use external RAM?

Well, there is no universal answer to this question; it depends on what the external RAM is going to be used for.

Basically, the bit `SRE` (SRAM enable) in the `MCUCR` register needs to be set in order to enable the external memory interface. Depending on the device to be used, and the application details, further registers affecting the external memory operation like `XMCRA` and `XMCRB`, and/or further bits in `MCUCR` might be configured. Refer to the datasheet for details.

If the external RAM is going to be used to store the variables from the C program (i. e., the `.data` and/or `.bss` segment) in that memory area, it is essential to set up the external memory interface early during the [device initialization](#) so the initialization of these variable will take place. Refer to [How to modify MCUCR or WDTCR early?](#) for a description how to do this using few lines of assembler code, or to the chapter about memory sections for an [example written in C](#).

The explanation of `malloc()` contains a [discussion](#) about the use of internal RAM vs. external RAM in particular with respect to the various possible locations of the *heap* (area reserved for `malloc()`). It also explains the linker

command-line options that are required to move the memory regions away from their respective standard locations in internal RAM.

Finally, if the application simply wants to use the additional RAM for private data storage kept outside the domain of the C compiler (e. g. through a `char *` variable initialized directly to a particular address), it would be sufficient to defer the initialization of the external RAM interface to the beginning of `main()`, so no tweaking of the `.init3` section is necessary. The same applies if only the heap is going to be located there, since the application start-up code does not affect the heap.

It is not recommended to locate the stack in external RAM. In general, accessing external RAM is slower than internal RAM, and errata of some AVR devices even prevent this configuration from working properly at all.

Back to [FAQ Index](#).

## 11.16 Which -O flag to use?

There's a common misconception that larger numbers behind the `-O` option might automatically cause "better" optimization. First, there's no universal definition for "better", with optimization often being a speed vs. code size trade off. See the [detailed discussion](#) for which option affects which part of the code generation.

A test case was run on an ATmega128 to judge the effect of compiling the library itself using different optimization levels. The following table lists the results. The test case consisted of around 2 KB of strings to sort. Test #1 used `qsort()` using the standard library `strcmp()`, test #2 used a function that sorted the strings by their size (thus had two calls to `strlen()` per invocation).

When comparing the resulting code size, it should be noted that a floating point version of `fprintf()` was linked into the binary (in order to print out the time elapsed) which is entirely not affected by the different optimization levels, and added about 2.5 KB to the code.

Optimization Flags	Size of .text	Time for Test #1	Time for Test #2
<code>-O3</code>	6898	903 $\mu$ s	19.7 ms
<code>-O2</code>	6666	972 $\mu$ s	20.1 ms
<code>-Os</code>	6618	955 $\mu$ s	20.1 ms
<code>-Os -mcall-prologues</code>	6474	972 $\mu$ s	20.1 ms

(The difference between 955  $\mu$ s and 972  $\mu$ s was just a single timer-tick, so take this with a grain of salt.)

So generally, it seems `-Os -mcall-prologues` is the most universal "best" optimization level. Only applications that need to get the last few percent of speed benefit from using `-O3`.

Back to [FAQ Index](#).

## 11.17 How do I relocate code to a fixed address?

First, put the function into a new, orphan [named section](#). This is done with a `section` attribute that specifies the name of the [input section](#) with the prototype of the function:

```
__attribute__((noinline, noclone, section(".bootloader")))
void boot(void);
```

The `noinline` and `noclone` attributes are required to make sure that the function is not (partially) inlined into the caller, which does not have a respective section attribute.

Second, locate the section to the desired fixed address by means of linking with, say

## 11.18 My UART is generating nonsense! My ATmega128 keeps crashing! Port F is completely broken! 65

```
-Wl,--section-start,.bootloader=0x1E000
```

see the [-Wl compiler option](#). The name after `--section-start` is the name of the section to be located, and the number specifies the beginning address of the named section.

This will only work when the section is an [orphan section](#), i.e. a section that is not mentioned in the linker script. For sections that *are* mentioned in the linker script, like for example `.text.bootloader`, this will not work because `--section-start` refers to an output section, but the output section for input section `.text.bootloader` is the `.text` section.

To verify that everything went as expected, generate the disassembly with `avr-objdump ... -j .bootloader`. The top of the list file will show

```
1 .bootloader 00000004 00002000 00002000 00000204 2**0
             CONTENTS, ALLOC, LOAD, READONLY, CODE
```

Or display section properties with `avr-readelf --section-details`

```
$ avr-readelf -t main.elf
Section Headers:
 [Nr] Name
      Type           Addr      Off      Size    ES   Lk Inf Al
      Flags
 [ 2] .bootloader
      PROGBITS       00002000 000204 000004 00    0  0  1
 [00000006]: ALLOC, EXEC
```

A different way to locate the section is by means of a custom linker script. The [avr-gcc Wiki](#) has an example that locates the `.progmem2.data` section which is used by the compiler for variables in address-space `__flash2`.

Back to [FAQ Index](#).

## 11.18 My UART is generating nonsense! My ATmega128 keeps crashing! Port F is completely broken!

Well, certain odd problems arise out of the situation that the AVR devices as shipped by Atmel often come with a default fuse bit configuration that doesn't match the user's expectations. Here is a list of things to care for:

- All devices that have an internal RC oscillator ship with the fuse enabled that causes the device to run off this oscillator, instead of an external crystal. This often remains unnoticed until the first attempt is made to use something critical in timing, like UART communication.
- The ATmega128 ships with the fuse enabled that turns this device into ATmega103 compatibility mode. This means that some ports are not fully usable, and in particular that the internal SRAM is located at lower addresses. Since by default, the stack is located at the top of internal SRAM, a program compiled for an ATmega128 running on such a device will immediately crash upon the first function call (or rather, upon the first function return).
- Devices with a JTAG interface have the `JTAGEN` fuse programmed by default. This will make the respective port pins that are used for the JTAG interface unavailable for regular IO.

Back to [FAQ Index](#).

## 11.19 Why do all my "foo...bar" strings eat up the SRAM?

By default, all strings are handled as all other initialized variables: they occupy RAM (even though the compiler might warn you when it detects write attempts to these RAM locations), and occupy the same amount of flash ROM so they can be initialized to the actual string by startup code.

That way, any string literal will be a valid argument to any C function that expects a `const char*` argument.

Of course, this is going to waste a lot of SRAM. In [Program Space String Utilities](#), a method is described how such constant data can be moved out to flash ROM. However, a constant string located in flash ROM is no longer a valid argument to pass to a function that expects a `const char*`-type string, since the AVR processor needs the special instruction `LPM` to access these strings. Thus, separate functions are needed that take this into account. Many of the standard C library functions have equivalents available where one of the string arguments can be located in flash ROM. Private functions in the applications need to handle this, too. For example, the following can be used to implement simple debugging messages that will be sent through a UART:

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>

int uart_putchar(char c)
{
    if (c == '\n')
        uart_putchar('\r');
    loop_until_bit_is_set(USR, UDRE);
    UDR = c;
    return 0; /* so it could be used for fdevopen(), too */
}

void debug_P(const char *addr)
{
    char c;

    while ((c = pgm_read_byte(addr++)))
        uart_putchar(c);
}

int main(void)
{
    ioinit(); /* initialize UART, ... */
    debug_P(PSTR("foo was here\n"));
    return 0;
}
```

### Note

By convention, the suffix `_P` to the function name is used as an indication that this function is going to accept a "program-space string". Note also the use of the `PSTR()` macro.

Back to [FAQ Index](#).

## 11.20 How to detect RAM memory and variable overlap problems?

You can simply run `avr-nm` on your output (ELF) file. Run it with the `-n` option, and it will sort the symbols numerically (by default, they are sorted alphabetically).

Look for the symbol `_end`, that's the first address in RAM that is not allocated by a variable. (`avr-gcc` internally adds `0x800000` to all `data/bss` variable addresses, so please ignore this offset.) Then, the run-time initialization code initializes the stack pointer (by default) to point to the last available address in (internal) SRAM. Thus, the region between `_end` and the end of SRAM is what is available for stack. (If your application uses `malloc()`, which e. g. also can happen inside `printf()`, the heap for dynamic memory is also located there. See [Memory Areas and Using malloc\(\)](#).)

The amount of stack required for your application cannot be determined that easily. For example, if you recursively call a function and forget to break that recursion, the amount of stack required is infinite. :-) You can look at the generated assembler code (`avr-gcc . . . -S`), there's a comment in each generated assembler file that tells you the frame size for each generated function. That's the amount of stack required for this function, you have to add up that for all functions where you know that the calls could be nested.

Back to [FAQ Index](#).

## 11.21 Is it really impossible to program the ATtinyXX in C?

While some small AVRs are not directly supported by the C compiler since they do not have a RAM-based stack (and some do not even have RAM at all), it is possible anyway to use the general-purpose registers as a RAM replacement since they are mapped into the data memory region.

Bruce D. Lightner wrote an excellent description of how to do this, and offers this together with a toolkit on his web page:

<http://lightner.net/avr/ATtinyAvrGcc.html>

Back to [FAQ Index](#).

## 11.22 What is this "clock skew detected" message?

It's a known problem of the MS-DOS FAT file system. Since the FAT file system has only a granularity of 2 seconds for maintaining a file's timestamp, and it seems that some MS-DOS derivative (Win9x) perhaps rounds up the current time to the next second when calculating the timestamp of an updated file in case the current time cannot be represented in FAT's terms, this causes a situation where `make` sees a "file coming from the future".

Since all `make` decisions are based on file timestamps, and their dependencies, `make` warns about this situation.

Solution: don't use inferior file systems / operating systems. Neither Unix file systems nor HPFS (aka NTFS) do experience that problem.

Workaround: after saving the file, wait a second before starting `make`. Or simply ignore the warning. If you are paranoid, execute `make clean all` to make sure everything gets rebuilt.

In networked environments where the files are accessed from a file server, this message can also happen if the file server's clock differs too much from the network client's clock. In this case, the solution is to use a proper time keeping protocol on both systems, like NTP. As a workaround, synchronize the client's clock frequently with the server's clock.

Back to [FAQ Index](#).

## 11.23 Why are (many) interrupt flags cleared by writing a logical 1?

Usually, each interrupt has its own interrupt flag bit in some control register, indicating the specified interrupt condition has been met by representing a logical 1 in the respective bit position. When working with interrupt handlers, this interrupt flag bit usually gets cleared automatically in the course of processing the interrupt, sometimes by just calling the handler at all, sometimes (e. g. for the U[S]ART) by reading a particular hardware register that will normally happen anyway when processing the interrupt.

From the hardware's point of view, an interrupt is asserted as long as the respective bit is set, while global interrupts are enabled. Thus, it is essential to have the bit cleared before interrupts get re-enabled again (which usually happens when returning from an interrupt handler).

Only few subsystems require an explicit action to clear the interrupt request when using interrupt handlers. (The notable exception is the TWI interface, where clearing the interrupt indicates to proceed with the TWI bus hardware handshake, so it's never done automatically.)

However, if no normal interrupt handlers are to be used, or in order to make extra sure any pending interrupt gets cleared before re-activating global interrupts (e. g. an external edge-triggered one), it can be necessary to explicitly clear the respective hardware interrupt bit by software. This is usually done by writing a logical 1 into this bit position.

This seems to be illogical at first, the bit position already carries a logical 1 when reading it, so why does writing a logical 1 to it *clear* the interrupt bit?

The solution is simple: writing a logical 1 to it requires only a single `OUT` instruction, and it is clear that only this single interrupt request bit will be cleared. There is no need to perform a read-modify-write cycle (like, an `SBI` instruction), since all bits in these control registers are interrupt bits, and writing a logical 0 to the remaining bits (as it is done by the simple `OUT` instruction) will not alter them, so there is no risk of any race condition that might accidentally clear another interrupt request bit. So instead of writing

```
TIFR |= _BV(TOV0); /* wrong! */
```

simply use

```
TIFR = _BV(TOV0);
```

Back to [FAQ Index](#).

## 11.24 Why have "programmed" fuses the bit value 0?

Basically, fuses are just a bit in a special EEPROM area. For technical reasons, erased E[EEP]ROM cells have all bits set to the value 1, so unprogrammed fuses also have a logical 1. Conversely, programmed fuse cells read out as bit value 0.

Back to [FAQ Index](#).

## 11.25 Which AVR-specific assembler operators are available?

See [Pseudo-Ops](#) and [Operand Modifiers](#).

Back to [FAQ Index](#).

## 11.26 Why are interrupts re-enabled in the middle of writing the stack pointer?

When setting up space for local variables on the stack, the compiler generates code like this:

```
/* prologue: frame size=20 */
  push r28
  push r29
  in r28, __SP_L__
  in r29, __SP_H__
  sbiw r28, 20
  in __tmp_reg__, __SREG__
  cli
  out __SP_H__, r29
  out __SREG__, __tmp_reg__
  out __SP_L__, r28
/* prologue end (size=10) */
```

It reads the current stack pointer value, decrements it by the required amount of bytes, then disables interrupts, writes back the high part of the stack pointer, writes back the saved `SREG` (which will eventually re-enable interrupts if they have been enabled before), and finally writes the low part of the stack pointer.

At the first glance, there's a race between restoring `SREG`, and writing `SPL`. However, after enabling interrupts (either explicitly by setting the `I` flag, or by restoring it as part of the entire `SREG`), the AVR hardware executes (at least) the next instruction still with interrupts disabled, so the write to `SPL` is guaranteed to be executed with interrupts disabled still. Thus, the emitted sequence ensures interrupts will be disabled only for the minimum time required to guarantee the integrity of this operation.

Back to [FAQ Index](#).

## 11.27 Why are there five different linker scripts?

From a comment in the source code:

Which one of the five linker script files is actually used depends on command line options given to ld.

A .x script file is the default script A .xr script is for linking without relocation (-r flag) A .xu script is like .xr but \*do\* create constructors (-Ur flag) A .xn script is for linking with -n flag (mix text and data on same page). A .xnb script is for linking with -N flag (mix text and data on same page).

Back to [FAQ Index](#).

## 11.28 How to add a raw binary image to linker output?

The GNU linker `avr-ld` cannot handle binary data directly. However, there's a companion tool called `avr-objcopy`. This is already known from the output side: it's used to extract the contents of the linked ELF file into an Intel Hex load file.

`avr-objcopy` can create a relocatable object file from arbitrary binary input, like

```
avr-objcopy -I binary -O elf32-avr foo.bin foo.o
```

This will create a file named `foo.o`, with the contents of `foo.bin`. The contents will default to section `.data`, and two symbols will be created named `_binary_foo_bin_start` and `_binary_foo_bin_end`. These symbols can be referred to inside a C source to access these data.

If the goal is to have those data go to flash ROM (similar to having used the `PROGMEM` attribute in C source code), the sections have to be renamed while copying, and it's also useful to set the section flags:

```
avr-objcopy --rename-section .data=.progmem.data,contents,alloc,load,readonly,data -I binary -O elf32-avr
foo.bin foo.o
```

Note that all this could be conveniently wired into a Makefile, so whenever `foo.bin` changes, it will trigger the recreation of `foo.o`, and a subsequent relink of the final ELF file.

Below are two Makefile fragments that provide rules to convert a .txt file to an object file, and to convert a .bin file to an object file:

```
$(OBJDIR)/%.o : %.txt
    @echo Converting $<
    @cp $(<) $(*).tmp
    @echo -n 0 | tr 0 '\000' >> $(*).tmp
    @$ (OBJCOPY) -I binary -O elf32-avr \
    --rename-section .data=.progmem.data,contents,alloc,load,readonly,data \
    --redefine-sym _binary_$(*)_tmp_start=$(*) \
    --redefine-sym _binary_$(*)_tmp_end=$(*)_end \
    --redefine-sym _binary_$(*)_tmp_size=$(*)_size_sym \
    $(*).tmp $(@)
    @echo "extern const char" $(*)"[]" PROGMEM;" > $(*).h
    @echo "extern const char" $(*)_end"[]" PROGMEM;" >> $(*).h
    @echo "extern const char" $(*)_size_sym"[]" >> $(*).h
    @echo "#define $(*)_size ((int)$(*)_size_sym)" >> $(*).h
    @rm $(*).tmp

$(OBJDIR)/%.o : %.bin
    @echo Converting $<
    @$ (OBJCOPY) -I binary -O elf32-avr \
    --rename-section .data=.progmem.data,contents,alloc,load,readonly,data \
    --redefine-sym _binary_$(*)_bin_start=$(*) \
    --redefine-sym _binary_$(*)_bin_end=$(*)_end \
    --redefine-sym _binary_$(*)_bin_size=$(*)_size_sym \
    $(<) $(@)
    @echo "extern const char" $(*)"[]" PROGMEM;" > $(*).h
    @echo "extern const char" $(*)_end"[]" PROGMEM;" >> $(*).h
    @echo "extern const char" $(*)_size_sym"[]" >> $(*).h
    @echo "#define $(*)_size ((int)$(*)_size_sym)" >> $(*).h
```

Back to [FAQ Index](#).



## 11.29 How do I perform a software reset of the AVR?

The canonical way to perform a software reset of non-XMega AVR's is to use the watchdog timer. Enable the watchdog timer to the shortest timeout setting, then go into an infinite, do-nothing loop. The watchdog will then reset the processor.

XMega parts have a specific bit `RST_SWRST_bm` in the `RST_CTRL` register, that generates a hardware reset. `RST_SWRST_bm` is protected by the XMega Configuration Change Protection system.

The reason why using the watchdog timer or `RST_SWRST_bm` is preferable over jumping to the reset vector, is that when the watchdog or `RST_SWRST_bm` resets the AVR, the registers will be reset to their known, default settings. Whereas jumping to the reset vector will leave the registers in their previous state, which is generally not a good idea.

**CAUTION!** Older AVR's will have the watchdog timer disabled on a reset. For these older AVR's, doing a soft reset by enabling the watchdog is easy, as the watchdog will then be disabled after the reset. On newer AVR's, once the watchdog is enabled, then it **stays enabled, even after a reset!** For these newer AVR's a function needs to be added to the `.init3` section (i.e. during the startup code, before `main()`) to disable the watchdog early enough so it does not continually reset the AVR.

Here is some example code that creates a macro that can be called to perform a soft reset:

```
#include <avr/wdt.h>

static inline __attribute__((__always_inline__))
void soft_reset (void)
{
    wdt_enable (WDTO_15MS);
    for(;;) {}
}
```

For newer AVR's (such as the ATmega1281) also add this function to your code to then disable the watchdog after a reset (e.g., after a soft reset):

```
#include <avr/wdt.h>

// Function Prototype
static __attribute__((used, unused, naked, section(".init3")))
void wdt_init (void);

// Function Implementation
void wdt_init (void)
{
    MCUSR = 0;
    wdt_disable();
}
```

The code is placed in section `.init3` so that it is executed as part of the normal startup procedure. The `naked` attribute is required so that the code does not `return` (Code in `init` sections is executed as it is located; the code is not called, and code from one `init` section falls through to the code in the next one). The `used` attribute makes sure that the compiler does not throw the seemingly unused function away. The `unused` attributes avoids warnings about unused code.

Back to [FAQ Index](#).

## 11.30 What pitfalls exist when writing reentrant code?

Reentrant code means the ability for a piece of code to be called simultaneously from two or more threads. Attention to re-entrability is needed when using a multi-tasking operating system, or when using interrupts since an interrupt is really a temporary thread.

The code generated natively by `gcc` is reentrant. But, only some of the libraries in AVR-LibC are explicitly reentrant, and some are known not to be reentrant. In general, any library call that reads and writes global variables (including I/O registers) is not reentrant. This is because more than one thread could read or write the same storage at the same time, unaware that other threads are doing the same, and create inconsistent and/or erroneous results.

A library call that is known not to be reentrant will work if it is used only within one thread *and* no other thread makes use of a library call that shares common storage with it.

Below is a table of library calls with known issues.

Library Call	Reentrant Issue	Workaround / Alternative
<code>rand</code> , <code>random</code>	Uses global variables to keep state information.	Use special reentrant versions↔ : <code>rand_r</code> , <code>random_r</code> .
<code>strtof</code> , <code>strtod</code> , <code>strtol</code> , <code>strtoul</code>	Uses the global variable <code>errno</code> to return success/failure.	Ignore <code>errno</code> , or protect calls with <code>cli/sei</code> or <code>ATOMIC_BLOCK</code> if the application can tolerate it. Or use <code>scanf</code> or <code>scanf_P</code> if possible.
<code>malloc</code> , <code>realloc</code> , <code>calloc</code> , <code>free</code>	Uses the stack pointer and global variables to allocate and free memory.	Protect calls with <code>cli/sei</code> or <code>ATOMIC_BLOCK</code> if the application can tolerate it. If using an OS, use the OS provided memory allocator since the OS is likely modifying the stack pointer anyway.
<code>fdevopen</code> , <code>fclose</code>	Uses <code>calloc</code> and <code>free</code> .	Protect calls with <code>cli/sei</code> or <code>ATOMIC_BLOCK</code> if the application can tolerate it. Or use <code>fdev_setup_stream</code> or <code>FDEV_SETUP_STREAM</code> . Note: <code>fclose</code> will only call <code>free</code> if the stream has been opened with <code>fdevopen</code> .
<code>eeeprom*</code> , <code>boot*</code>	Accesses I/O registers.	Protect calls with <code>cli/sei</code> , <code>ATOMIC_BLOCK</code> , or use OS locking.
<code>pgm*_far</code>	Accesses I/O register RAMPZ.	Starting with GCC 4.3, RAMPZ is automatically saved for <code>ISRs</code> , so nothing further is needed if only using interrupts. Some Oses may automatically preserve RAMPZ during context switching. Check the OS documentation before assuming it does. Otherwise, protect calls with <code>cli/sei</code> , <code>ATOMIC_BLOCK</code> , or use explicit OS locking.
<code>printf</code> , <code>printf_P</code> , <code>vprintf</code> , <code>puts</code> , <code>puts_P</code>	Alters flags and character count in global FILE <code>stdout</code> .	Use only in one thread. Or if returned character count is unimportant, do not use the <code>*_P</code> versions. Note: Formatting to a string output, e.g. <code>sprintf</code> , <code>sprintf_P</code> , <code>snprintf</code> , <code>snprintf_P</code> , <code>vsprintf</code> , <code>vsprintf_P</code> , <code>vsnprintf</code> , <code>vsnprintf_P</code> , is thread safe. The formatted string could then be followed by an <code>fwrite</code> which simply calls the lower layer to send the string.
<code>fprintf</code> , <code>fprintf_P</code> , <code>vfprintf</code> , <code>vfprintf_P</code> , <code>fputs</code> , <code>fputs_P</code>	Alters flags and character count in the FILE argument. Problems can occur if a global FILE is used from multiple threads.	Assign each thread its own FILE for output. Or if returned character count is unimportant, do not use the <code>*_P</code> versions.
<code>assert</code>	Contains an embedded <code>fprintf</code> . See above for <code>fprintf</code> .	See above for <code>fprintf</code> .

Library Call	Reentrant Issue	Workaround / Alternative
<code>clearerr</code>	Alters flags in the FILE argument.	Assign each thread its own FILE for output.
<code>getchar</code> , <code>gets</code>	Alters flags, character count, and unget buffer in global FILE <code>stdin</code> .	Use only in one thread. ***
<code>fgetc</code> , <code>ungetc</code> , <code>fgets</code> , <code>scanf</code> , <code>scanf_P</code> , <code>fscanf</code> , <code>fscanf_P</code> , <code>vscanf</code> , <code>vfscanf</code> , <code>vfscanf_P</code> , <code>fread</code>	Alters flags, character count, and unget buffer in the FILE argument.	Assign each thread its own FILE for input. *** Note: Scanning from a string, e.g. <code>sscanf</code> and <code>sscanf_P</code> , are thread safe.

#### Note

It's not clear one would ever want to do character input simultaneously from more than one thread anyway, but these entries are included for completeness.

An effort will be made to keep this table up to date if any new issues are discovered or introduced.

Back to [FAQ Index](#).

### 11.31 Why are some addresses of the EEPROM corrupted (usually address zero)?

The two most common reason for EEPROM corruption is either writing to the EEPROM beyond the datasheet endurance specification, or resetting the AVR while an EEPROM write is in progress.

EEPROM writes can take up to tens of milliseconds to complete. So that the CPU is not tied up for that long of time, an internal state-machine handles EEPROM write requests. The EEPROM state-machine expects to have all of the EEPROM registers setup, then an EEPROM write request to start the process. Once the EEPROM state-machine has started, changing EEPROM related registers during an EEPROM write is guaranteed to corrupt the EEPROM write process. The datasheet always shows the proper way to tell when a write is in progress, so that the registers are not changed by the user's program. The EEPROM state-machine will **always** complete the write in progress unless power is removed from the device.

As with all EEPROM technology, if power fails during an EEPROM write the state of the byte being written is undefined.

In older generation AVRs the EEPROM Address Register (EEAR) is initialized to zero on reset, be it from Brown Out Detect, Watchdog or the Reset Pin. If an EEPROM write has just started at the time of the reset, the write will be completed, but now at address zero instead of the requested address. If the reset occurs later in the write process both the requested address and address zero may be corrupted.

To distinguish which AVRs may exhibit the corrupt of address zero while a write is in process during a reset, look at the "initial value" section for the EEPROM Address Register. If EEAR shows the initial value as 0x00 or 0x0000, then address zero and possibly the one being written will be corrupted. Newer parts show the initial value as "undefined", these will not corrupt address zero during a reset (unless it was address zero that was being written).

EEPROMs have limited write endurance. The datasheet specifies the number of EEPROM writes that are guaranteed to function across the full temperature specification of the AVR, for a given byte. A read should always be performed before a write, to see if the value in the EEPROM actually needs to be written, so not to cause unnecessary EEPROM wear.

The failure mechanism for an overwritten byte is generally one of "stuck" bits, i. e. a bit will stay at a one or zero state regardless of the byte written. Also a write followed by a read may return the correct data, but the data will change with the passage of time, due the EEPROM's inability to hold a charge from the excessive write wear.

Back to [FAQ Index](#).

## 11.32 Why is my baud rate wrong?

Some AVR datasheets give the following formula for calculating baud rates:

```
(F_CPU / (UART_BAUD_RATE * 16L) - 1)
```

Unfortunately that formula does not work with all combinations of clock speeds and baud rates due to integer truncation during the division operator.

When doing integer division it is usually better to round to the nearest integer, rather than to the lowest. To do this add 0.5 (i. e. half the value of the denominator) to the numerator before the division, resulting in the formula:

```
((F_CPU + UART_BAUD_RATE * 8L) / (UART_BAUD_RATE * 16L) - 1)
```

This is also the way it is implemented in [<util/setbaud.h>](#): [Helper macros for baud rate calculations](#).

Back to [FAQ Index](#).

## 11.33 On a device with more than 128 KiB of flash, how to make function pointers work?

Function pointers beyond the "magical" 128 KiB barrier(s) on larger devices are supposed to be resolved through so-called *trampolines* by the linker, so the actual pointers used in the code can remain 16 bits wide.

In order for this to work, the option `-mrelax` must be given on the compiler command-line that is used to link the final ELF file. (Older compilers did not implement this option for the AVR, use `-Wl,--relax` instead.)

See also the `avr-gcc` online documentation on the [EIND special function register](#) and indirect calls.

Back to [FAQ Index](#).

## 11.34 Why is assigning ports in a "chain" a bad idea?

Suppose a number of IO port registers should get the value `0xff` assigned. Conveniently, it is implemented like this:

```
DDRB = DDRD = 0xff;
```

According to the rules of the C language, this causes `0xff` to be assigned to `DDRD`, then `DDRD` is read back, and the value is assigned to `DDRB`. The compiler stands no chance to optimize the readback away, as an IO port register is declared "volatile". Thus, chaining that kind of IO port assignments would better be avoided, using explicit assignments instead:

```
DDRB = 0xff;
DDRD = 0xff;
```

Even worse ist this, e. g. on an ATmega1281:

```
DDRA = DDRB = DDRC = DDRD = DDRE = DDRF = DDRG = 0xff;
```

The same happens as outlined above. However, when reading back register `DDRG`, this register only implements 6 out of the 8 bits, so the two topmost (unimplemented) bits read back as 0! Consequently, all remaining `DDRx` registers get assigned the value `0x3f`, which does not match the intention of the developer in any way.

## 11.35 Which header files are included in my program?

Suppose we have a simple program like

```
#include <avr/pgmspace.h>

int main (void)
{
    return 0;
}
```

and we want to know which files this `#include` triggers. Just add option `-H` to the compiler options and check what is printed on standard output:

```
$ avr-gcc -H -S main.c -mmcu=atmega8
. <install>/avr/include/avr/pgmspace.h
.. <install>/avr/include/inttypes.h
... <install>/lib/gcc/avr/<version>/include/stdint.h
.... <install>/avr/include/stdint.h
.. <install>/lib/gcc/avr/<version>/include/stddef.h
.. <install>/avr/include/avr/io.h
... <install>/avr/include/avr/sfr_defs.h
... <install>/avr/include/avr/iom8.h
... <install>/avr/include/avr/portpins.h
...
```

where `<install>` denotes the installation path, `<version>` denotes the GCC version, and the number of dots indicates the include level, e.g. `inttypes.h` is included by `pgmspace.h`.

When `-v` is added to the compiler options, then the search paths are also displayed (amongst other stuff):

```
#include ".." search starts here:
#include <...> search starts here:
<install>/bin/./lib/gcc/avr/<version>/include
<install>/bin/./lib/gcc/avr/<version>/include-fixed
<install>/bin/./lib/gcc/avr/<version>/../../../../avr/include
End of search list.
```

After resolving the `..`'s for "*parent directory*", the last directory becomes `<install>/avr/include`.

Back to [FAQ Index](#).

## 11.36 Which macros are defined in my program? Where are they defined, and to what value?

One way is to add `-save-temps` and `-g3` to the compiler options. This saves the temporary files like the pre-processed source code in an `.i` file (for C sources), an `.ii` (for C++), or a `.s` (for assembly). A debug level of DWARF3 or higher is required to include the macro definitions in the file, with lower debug levels, only the preprocessed source will be present.

For a module with a simple `#include <avr/pgmspace.h>`, the saved intermediate file might look something like:

```
# 0 "<built-in>"
#define __STDC__ 1
```

The `__STDC__` macro is defined built-in in the compiler.

```
# 0 "<command-line>"
#define __AVR_DEVICE_NAME__ atmega8
```

The `__AVR_DEVICE_NAME__` macro is defined on the command line by means of `-D __AVR_DEVICE_NAME__=atmega8`. In this special case, the `-D` option is added by the specs file `specs-atmega8`.

```
# 81 "<install>/avr/include/avr/pgmspace.h" 3
#define __PGMSPACE_H_ 1
```

```
#define __need_size_t
```

The `__PGMSPACE_H_` macro is defined in line 81 of that header file. When there is no line note directly above the definition, go up until you find a line note. For example, the `__need_size_t` macro is defined in line 84 of that file.

Back to [FAQ Index](#).

## 11.37 What ISR names are available for my device?

One way to find the possible [ISR](#) names is to pre-process a small file, and to grep for possible names, like in:

```
$ echo "#include <avr/io.h>" | avr-gcc -xc - -mmcu=atmega8 -E -dM | grep _VECTOR
#define INT0_vect _VECTOR(1)
#define INT1_vect _VECTOR(2)
#define TIMER2_COMP_vect _VECTOR(3)
#define TIMER2_OVF_vect _VECTOR(4)
#define TIMER1_CAPT_vect _VECTOR(5)
...
```

Explanation:

**echo "#include <avr/io.h>"** This prints `#include <avr/io.h>` to the standard output, which is picked up by the following command as a C program to be preprocessed.

**avr-gcc -xc - -mmcu=atmega8 -E -dM** Set the [input language](#) to C, read the program from standard input (specified by a dash), preprocess, and print all macro definitions to the standard output.

**grep \_VECTOR** Only print lines with `_VECTOR` in them.

The output above was actually generated with an additional `| sort -t '(' -k 2n` so that the vectors are printed in order.

In order to find the respective vector numbers, use `grep _vect_num` instead.

Back to [FAQ Index](#).

## 12 Building and Installing the GNU Tool Chain

This chapter shows how to build and install, from source code, a complete development environment for the AVR processors using the GNU toolset. There are two main sections, one for Linux, FreeBSD, and other Unix-like operating systems, and another section for Windows.

- [Required AVR Tools](#)
- [Optional AVR Tools](#)
- [Building and Installing under Linux, FreeBSD, and Others](#)
  - [Preparations](#)
    - \* [Install Location](#)
    - \* [Directory Layout](#)
  - [GNU Binutils](#)
  - [GCC](#)
  - [AVR-LibC](#)
  - [AVRDUDE](#)
  - [SimulAVR](#)
  - [AVaRICE](#)
- [Building and Installing under Windows](#)
  - [Required Tools](#)
  - [Building](#)
- [Canadian Cross Builds](#)
- [Using Git](#)

### 12.1 Required AVR Tools

**GNU Binutils** Project Home: <https://sourceware.org/binutils>  
Source Downloads: <https://sourceware.org/pub/binutils/releases>  
FTP: [anonymous@ftp.gnu.org/gnu/binutils](ftp://anonymous@ftp.gnu.org/gnu/binutils)  
Git: <git://sourceware.org/git/binutils-gdb.git>  
GitHub Mirror: <https://github.com/bminor/binutils-gdb>  
[Installation](#)

**GCC** Project Home <https://gcc.gnu.org>  
Mirrors Site: <https://gcc.gnu.org/mirrors.html>  
FTP: [anonymous@ftp.gnu.org/gnu/gcc](ftp://anonymous@ftp.gnu.org/gnu/gcc)  
Git: <git://gcc.gnu.org/git/gcc.git>  
GitHub Mirror: <https://github.com/gcc-mirror/gcc>  
Installation: <https://gcc.gnu.org/install>  
[Installation](#)

**AVR-LibC** Project Home: <http://savannah.gnu.org/projects/avr-libc>  
Source Downloads: <https://download-mirror.savannah.gnu.org/releases/avr-libc>  
  
Git: <https://github.com/avrduces/avr-libc.git>  
GitHub: <https://github.com/avrduces/avr-libc>  
[Installation](#)

## 12.2 Optional AVR Tools

You can develop programs for AVR devices without the following tools. They may or may not be of use for you.

**AVRDUDE** Project Home: <http://savannah.nongnu.org/projects/avrdude>

Git: <https://github.com/avrdudes/avrdude.git>

GitHub: <https://github.com/avrdudes/avrdude>

[Installation](#)

[Usage Notes](#)

**GDB** The GNU Debugger GDB is hosted together with GNU Binutils. When you don't want or need GDB, you can configure Binutils with `--disable-gdb`.

**SimulAVR** <http://savannah.gnu.org/projects/simulavr>

[Installation](#)

**AVaRICE** GitHub: <https://github.com/avrdudes/avarice>

[Installation](#)

## 12.3 Building and Installing under Linux, FreeBSD, and Others

The default behaviour for most of these tools is to install every thing under the `/usr/local` directory. In order to keep the AVR tools separate from the base system, it is usually better to install everything into `/usr/local/avr`. If the `/usr/local/avr` directory does not exist, you should create it before trying to install anything. You will need `root` access to install there. If you don't have or want `root` access to the system, you can alternatively install in your home directory, for example, in `$HOME/local/avr`. Where you install is a completely arbitrary decision, but should be consistent for all the tools.

### Warning

If you have `CC` set to anything other than `avr-gcc` in your environment, this will cause the configure script to fail. It is best to not have `CC` set at all.

### Note

It is usually the best to use the latest released version of each of the tools.

### 12.3.1 Preparations

**12.3.1.1 Install Location** You specify the installation directory by using the `--prefix=dir` option with the `configure` script. It is important to install all the AVR tools in the same directory or some of the tools will not work correctly. To ensure consistency and simplify the discussion, we will use `$PREFIX` to refer to whatever directory you wish to install in. You can set this as an environment variable if you wish as such (using a Bourne-like shell):

```
$ PREFIX=$HOME/local/avr
$ export PREFIX
```

### Note

Be sure that you have your `PATH` environment variable set to search the directory you install everything in *before* you start installing anything. For example, if you use `--prefix=$PREFIX`, you must have `$PREFIX/bin` in your exported `PATH`. As such:

```
$ PATH=$PATH:$PREFIX/bin
$ export PATH
```



**12.3.1.2 Directory Layout** The instructions below build Binutils, GCC and AVR-LibC *outside* of the source tree, because:

- When something goes wrong, you can just remove the build directory and start all over again with a fresh build folder.
- You may want to build the tools with different configure options, e.g. build the tools for a Linux host and then build a [Canadian cross](#) to run on a Windows host.
- GCC does not support configuring anywhere in the source tree, so we'll have to use a separate build folder outside the source tree, anyway.

The instructions below assume that you have set up a directory tree like

```
+--source
+--build
```

in some place where you have write access, like in your home directory.

After successful downloads and builds, the tree will be something like:

```
+--source
|  +--gcc-<version>
|  +--binutils-<version>
|  +--avr-libc-<version>
+-- build
    +--gcc-<version>-avr
    +--binutils-<version>-avr
    +--avr-libc-<version>
```

### 12.3.2 GNU Binutils for the AVR target

The **Binutils** package provides all the low-level utilities needed in building and manipulating object files. Once installed, your environment will have an AVR assembler (`avr-as`), linker (`avr-ld`), and librarian (`avr-ar` and `avr-ranlib`). In addition, you get tools which extract data from object files (`avr-objcopy`), disassemble object file information (`avr-objdump`), and strip information from object files (`avr-strip`). Before we can build the C compiler, these tools need to be in place.

Download and unpack the source files:

```
$ # in ./source
$ tar xfv binutils-<version>.tar.bz2
```

Replace `<version>` with the version of the package you downloaded.

If you obtained a gzip compressed file (`.tar.gz` or `.tgz`), use `gunzip` instead of `bunzip2`, or `tar xfv file.tar.gz`.

The next step is to configure and build the tools. This is done by supplying arguments to the `configure` script that enable the AVR-specific options. When you also want GDB, just drop `--disable-gdb`.

```
$ # in ./build
$ mkdir binutils-<version>-avr
$ cd binutils-<version>-avr
$ ../../source/binutils-<version>/configure --prefix=$PREFIX --target=avr \
  --disable-nls --disable-sim --disable-gdb --disable-werror
```

When `configure` is run, it generates a lot of messages while it determines what is available on your operating system. When it finishes, it will have created several `Makefiles` that are custom tailored to your platform and that are run with the `make` command.

```
$ make
```

**Note**

BSD users should note that the project's `Makefile` uses GNU make syntax. This means FreeBSD users may need to build the tools by using `gmake`.

If the tools compiled cleanly, you're ready to install them. If you specified a destination that isn't owned by your account, you'll need `root` access to install them. To install:

```
$ make install
```

You should now have the programs from Binutils installed into `$PREFIX/bin`. Don't forget to [set your PATH](#) environment variable before going to build `avr-gcc`. To check that the correct assembler is found, run

```
$ avr-as --version
```

which should print the `<version>` of the used Binutils sources.

### 12.3.3 GCC for the AVR target

**Warning**

You **must** install [avr-binutils](#) and make sure your [path is set](#) properly before installing `avr-gcc`.

Before we can configure the compiler, we have to prepare the sources. GCC depends on some external host libraries, namely [GMP](#), [MPFR](#), [MPC](#) and [ISL](#). You can build and install the appropriate versions of the required prerequisites by hand and provide their location by means of `--with-gmp=` etc. Though in most situations it is easier to let GCC download and build these libraries as part of the configure and build process. All what's needed is an internet connection when running `./contrib/download_prerequisites`:

```
$ # in ./source
$ tar xvj gcc-<version>.tar.bz2
$ cd gcc-<version>
$ ./contrib/download_prerequisites

$ # in ./build
$ mkdir gcc-<version>-avr
$ cd gcc-<version>-avr
$ ../../source/gcc-<version>/configure --prefix=$PREFIX --target=avr --enable-languages=c,c++ \
  --disable-nls --disable-libssp --disable-libccl \
  --with-gnu-as --with-gnu-ld --with-dwarf2
$ make
$ make install # or make install-strip
```

The GCC binaries may consume quite some disc space. In most cases, you don't need the debug information in the compiler proper, and installing with

```
$ make install-strip
```

can save you some space.

### 12.3.4 AVR-LibC

#### Warning

You **must** install [avr-binutils](#), [avr-gcc](#) and make sure your [path is set](#) properly before installing AVR-LibC.

#### Note

If you have obtained the latest AVR-LibC from [git](#), you will have to run the `./bootstrap` script before using either of the build methods described below.

To build and install AVR-LibC:

```
$ # in ./source
$ tar xfz avr-libc-<version>.tar.gz

$ # in ./build
$ mkdir avr-libc-<version>
$ cd avr-libc-<version>
$ ../../source/avr-libc-<version>/configure --prefix=$PREFIX \
  --build=x86_64-pc-linux-gnu --host=avr
$ make
$ make install
```

Where the `--build` platform can be guessed by running

```
$ ./source/avr-libc-<version>/config.guess
```

### 12.3.5 AVRDUDE

#### Note

It has been ported to windows (via MinGW or cygwin), Linux and Solaris. Other Unix systems should be trivial to port to.

**avrdude** is part of the FreeBSD ports system. To install it, simply do the following:

```
# cd /usr/ports/devel/avrdude
# make install
```

#### Note

Installation into the default location usually requires root permissions. However, running the program only requires access permissions to the appropriate `ppi(4)` device.

Building and installing on other systems should use the `configure` system, as such:

```
$ gunzip -c avrdude-<version>.tar.gz | tar xf -
$ cd avrdude-<version>
$ mkdir obj-avr
$ cd obj-avr
$ ../configure --prefix=$PREFIX
$ make
$ make install
```

### 12.3.6 SimulAVR

SimulAVR also uses the `configure` system, so to build and install:

```
$ gunzip -c simulavr-<version>.tar.gz | tar xf -
$ cd simulavr-<version>
$ mkdir obj-avr
$ cd obj-avr
$ ../configure --prefix=$PREFIX
$ make
$ make install
```

#### Note

You might want to have already installed [avr-binutils](#), [avr-gcc](#) and [AVR-LibC](#) if you want to have the test programs built in the simulavr source.

### 12.3.7 AVaRICE

#### Note

These install notes are not applicable to avarice-1.5 or older. You probably don't want to use anything that old anyways since there have been many improvements and bug fixes since the 1.5 release.

AVaRICE also uses the `configure` system, so to build and install:

```
$ gunzip -c avarice-<version>.tar.gz | tar xf -
$ cd avarice-<version>
$ mkdir obj-avr
$ cd obj-avr
$ ../configure --prefix=$PREFIX
$ make
$ make install
```

#### Note

AVaRICE uses the BFD library for accessing various binary file formats. You may need to tell the configure script where to find the lib and headers for the link to work. This is usually done by invoking the configure script like this (Replace `<hdr_path>` with the path to the `bfd.h` file on your system. Replace `<lib_path>` with the path to `libbfd.a` on your system.):

```
$ CPPFLAGS=-I<hdr_path> LDFLAGS=-L<lib_path> ../configure --prefix=$PREFIX
```

## 12.4 Building and Installing under Windows

Building and installing the toolchain under Windows requires more effort because all of the tools required for building, and the programs themselves, are mainly designed for running under a POSIX environment such as Unix and Linux. Windows does not natively provide such an environment.

There are two projects available that provide such an environment, Cygwin and MinGW. There are advantages and disadvantages to both. Cygwin provides a very complete POSIX environment that allows one to build many Linux based tools from source with very little or no source modifications. However, POSIX functionality is provided in the form of a DLL that is linked to the application. This DLL has to be redistributed with your application and there are issues if the Cygwin DLL already exists on the installation system and different versions of the DLL. On the other hand, MinGW can compile code as native Win32 applications. However, this means that programs designed for Unix and Linux (i.e. that use POSIX functionality) will not compile as MinGW does not provide that POSIX layer for you. Therefore most programs that compile on both types of host systems, usually must provide some sort of abstraction layer to allow an application to be built cross-platform.

MinGW does provide somewhat of a POSIX environment, called MSYS, that allows you to build Unix and Linux applications as they would normally do, with a `configure` step and a `make` step. Cygwin also provides such an environment. This means that building the AVR toolchain is very similar to how it is built in Linux, described above. The main differences are in what the `PATH` environment variable gets set to, pathname differences, and the tools that are required to build the projects under Windows. We'll take a look at the tools next.

### 12.4.1 Tools Required for Building the Toolchain for Windows

These are the tools that are currently used to build an AVR tool chain. This list may change, either the version of the tools, or the tools themselves, as improvements are made.

**MinGW** Download the MinGW Automated Installer, 2013-10-04 (or later) <https://sourceforge.net/projects/mingw/files>

- Run `mingw-get-setup.exe`
- In the installation wizard, keep the default values and press the "Next" button for all installer pages except for the pages explicitly listed below.
- In the installer page "Repository Catalogues", select the "Download latest repository catalogues" radio button, and press the "Next" button
- In the installer page "License Agreement", select the "I accept the agreement" radio button, and press the "Next" button
- In the installer page "Select Components", be sure to select these items:
  - C compiler (default checked)
  - C++ compiler
  - Ada compiler
  - MinGW Developer Toolkit (which includes "MSYS Basic System").
- Install.

**Install Cygwin** Install everything, all users, UNIX line endings. This will take a *long* time. A fat internet pipe is highly recommended. It is also recommended that you download all to a directory first, and then install from that directory to your machine.

#### Note

GMP, MPFR, MPC and ISL are required to build GCC. By far the easiest way to use them is by letting GCC download the sources locally by means of running the `./contrib/download_prewerequisites` script from the GCC top source. GCC will configure and build these libs during `configure` and `make`.

Doxygen is required to build AVR-LibC documentation.

- **Install Doxygen**

- Version 1.7.2
- <https://www.doxygen.nl>
- Download and install.

NetPBM is required to build graphics in the AVR-LibC documentation.

- **Install NetPBM**

- Version 10.27.0
- From the GNUWin32 project: <http://gnuwin32.sourceforge.net/packages.html>
- Download and install.

fig2dev is required to build graphics in the AVR-LibC documentation.

- **Install fig2dev**

- Version 3.2 patchlevel 5c
- From WinFig 4.62: <http://winfig.com/downloads>

- Download the zip file version of WinFig
- Unzip the download file and install fig2dev.exe in a location of your choice, somewhere in the PATH.
- You may have to unzip and install related DLL files for fig2dev. In the version above, you have to install QtCore4.dll and QtGui4.dll.

MikTeX is required to build various documentation.

- **Install MiKTeX**

- Version 2.9
- <https://miktex.org>
- Download and install.

Ghostscript is required to build various documentation.

- **Install Ghostscript**

- Version 9.00
- <https://www.ghostscript.com>
- Download and install.
- In the \bin subdirectory of the installation, copy gswin32c.exe to gs.exe.
- Set the TEMP and TMP environment variables to `c : \temp` or to the short filename version. This helps to avoid NTVDM errors during building.

## 12.4.2 Building the Toolchain for Windows

All directories in the PATH environment variable should be specified using their short filename (8.3) version. This will also help to avoid NTVDM errors during building. These short filenames can be specific to each machine.

Build the tools below in MinGW/MSYS.

- **Binutils**

- Open source code package and patch as necessary.
- Configure and build in a directory outside of the source code tree.
- Set PATH, in order:
  - \* <MikTeX executables>
  - \* <ghostscript executables>
  - \* /usr/local/bin
  - \* /usr/bin
  - \* /bin
  - \* /mingw/bin
  - \* c:/cygwin/bin
  - \* <install directory>/bin
- Configure

```
CFLAGS=-D__USE_MINGW_ACCESS \
../$archivedir/configure \
  --prefix=$installdir \
  --target=avr \
  --disable-nls \
  --enable-doc \
  --datadir=$installdir/doc/binutils \
  2>&1 | tee binutils-configure.log
```

- Make

```
make all html install install-html 2>&1 | tee binutils-make.log
```

- Manually change documentation location.

- **GCC**

- Open source code package and patch as necessary.

- Configure and build in a directory outside of the source code tree.

- Set PATH, in order:

- \* <MikTeX executables>
- \* <ghostscript executables>
- \* /usr/local/bin
- \* /usr/bin
- \* /bin
- \* /mingw/bin
- \* c:/cygwin/bin
- \* <install directory>/bin

- Configure

```
LDLFLAGS='-L /usr/local/lib -R /usr/local/lib' \  
CFLAGS='-D__USE_MINGW_ACCESS' \  
../gcc-$version/configure \  
  --prefix=$installdir \  
  --target=$target \  
  --enable-languages=c,c++ \  
  --with-dwarf2 \  
  --enable-doc \  
  --with-docdir=$installdir/doc/$project \  
  --disable-shared \  
  --disable-libada \  
  --disable-libssp \  
  --disable-libccl \  
  --disable-nls \  
  2>&1 | tee $project-configure.log
```

- Make

```
make all html install 2>&1 | tee $package-make.log
```

- **AVR-LibC**

- Open source code package.

- Configure and build at the top of the source code tree.

- Set PATH, in order:

- \* /usr/local/bin
- \* /mingw/bin
- \* /bin
- \* <MikTeX executables>
- \* <install directory>/bin
- \* <Doxygen executables>
- \* <NetPBM executables>
- \* <fig2dev executable>
- \* <Ghostscript executables>
- \* c:/cygwin/bin

- Configure

```
./configure \
  --host=avr \
  --prefix=$installdir \
  --enable-doc \
  --disable-versioned-doc \
  --enable-html-doc \
  --enable-pdf-doc \
  --enable-man-doc \
  --mandir=$installdir/man \
  --datadir=$installdir \
  2>&1 | tee $package-configure.log
```

#### – Make

```
make all install 2>&1 | tee $package-make.log
```

- Manually change location of man page documentation.
- Move the examples to the top level of the install tree.
- Convert line endings in examples to Windows line endings.
- Convert line endings in header files to Windows line endings.

### • AVRDUDE

- Open source code package.
- Configure and build at the top of the source code tree.
- Set PATH, in order:

```
* <MikTex executables>
* /usr/local/bin
* /usr/bin
* /bin
* /mingw/bin
* c:/cygwin/bin
* <install directory>/bin
```

- Set location of LibUSB headers and libraries

```
export CPPFLAGS="-I../libusb-win32-device-bin-$libusb_version/include"
export CFLAGS="-I../libusb-win32-device-bin-$libusb_version/include"
export LDFLAGS="-L../libusb-win32-device-bin-$libusb_version/lib/gcc"
```

- Configure

```
./configure \
  --prefix=$installdir \
  --datadir=$installdir \
  --sysconfdir=$installdir/bin \
  --enable-doc \
  --disable-versioned-doc \
  2>&1 | tee $package-configure.log
```

- Make

```
make -k all install 2>&1 | tee $package-make.log
```

- Convert line endings in avrdude config file to Windows line endings.
- Delete backup copy of avrdude config file in install directory if exists.

### • Insight/GDB

- Open source code package and patch as necessary.
- Configure and build in a directory outside of the source code tree.
- Set PATH, in order:



- \* <MikTeX executables>
- \* /usr/local/bin
- \* /usr/bin
- \* /bin
- \* /mingw/bin
- \* c:/cygwin/bin
- \* <install directory>/bin

– Configure

```
CFLAGS=-D__USE_MINGW_ACCESS \
LDLFLAGS='-static' \
../$archivedir/configure \
  --prefix=$installdir \
  --target=avr \
  --with-gmp=/usr/local \
  --with-mpfr=/usr/local \
  --enable-doc \
2>&1 | tee insight-configure.log
```

– Make

```
make all install 2>&1 | tee $package-make.log
```

• **SRecord**

- Open source code package.
- Configure and build at the top of the source code tree.
- Set PATH, in order:

- \* <MikTeX executables>
- \* /usr/local/bin
- \* /usr/bin
- \* /bin
- \* /mingw/bin
- \* c:/cygwin/bin
- \* <install directory>/bin

– Configure

```
./configure \
  --prefix=$installdir \
  --infodir=$installdir/info \
  --mandir=$installdir/man \
2>&1 | tee $package-configure.log
```

– Make

```
make all install 2>&1 | tee $package-make.log
```

Build the tools below in Cygwin.

• **AVaRICE**

- Open source code package.
- Configure and build in a directory outside of the source code tree.
- Set PATH, in order:

- \* <MikTeX executables>
- \* /usr/local/bin

- \* /usr/bin
- \* /bin
- \* <install directory>/bin

– Set location of LibUSB headers and libraries

```
export CPPFLAGS=-I$startdir/libusb-win32-device-bin-$libusb_version/include
export CFLAGS=-I$startdir/libusb-win32-device-bin-$libusb_version/include
export LDFLAGS="-static -L$startdir/libusb-win32-device-bin-$libusb_version/lib/gcc "
```

– Configure

```
../$archivedir/configure \
  --prefix=$installdir \
  --datadir=$installdir/doc \
  --mandir=$installdir/man \
  --infodir=$installdir/info \
  2>&1 | tee avarice-configure.log
```

– Make

```
make all install 2>&1 | tee avarice-make.log
```

• SimulAVR

- Open source code package.
- Configure and build in a directory outside of the source code tree.
- Set PATH, in order:

- \* <MikTex executables>
- \* /usr/local/bin
- \* /usr/bin
- \* /bin
- \* <install directory>/bin

– Configure

```
export LDFLAGS="-static"
../$archivedir/configure \
  --prefix=$installdir \
  --datadir=$installdir \
  --disable-tests \
  --disable-versioned-doc \
  2>&1 | tee simulavr-configure.log
```

– Make

```
make -k all install 2>&1 | tee simulavr-make.log
make pdf install-pdf 2>&1 | tee simulavr-pdf-make.log
```

## 12.5 Canadian Cross Builds

It is also possible to build `avr-gcc` for host Windows on a Linux build system. Suppose you have installed a `i686-w64-mingw32-gcc` toolchain that can compile code to run on `host=i686-w64-mingw32`. Then the steps to build a toolchain for Windows are:

1. Build and install the AVR toolchain for the Linux build machine as explained above. Make sure that running the command

```
avr-gcc --version
```

prints the compiler version according to the used GCC sources. The native AVR cross compiler is required during configure and to build the AVR target libraries like `libgcc`. Similarly, the version of the found AVR Binutils programs must match the version of the used Binutils sources.

2. Determine the name of the `--build` platform like `x86_64-pc-linux-gnu`, for example by running the `config.guess` script as shipped with the top level GCC sources (and also with Binutils sources, and AVR-LibC sources after `./bootstrap`).
3. Use different build and install directories, e.g. `./build/binutils-<version>-avr-mingw32` to build Binutils and `--prefix=$PREFIX-mingw32` as install path.
4. Configure, build and install **Binutils** and **GCC** like for the native build, but add the following configure options:

```
--build=x86_64-pc-linux-gnu --host=i686-w64-mingw32
```

This assumes that the required host libraries like GMP are being built in one go with the compiler. This is accomplished by running the `contrib/download_prerequisites` script from the toplevel GCC sources, just like with the native build.

5. There is no need to build **AVR-LibC** again because it is a pure target library. It can be installed by means of running

```
$ # in ./build/avr-libc-<version>
$ make install prefix=$PREFIX-mingw32
```

In order to "install" the toolchain on Windows, the canadian cross installed in `$PREFIX-mingw32` can be moved to the desired location on the Windows computer. The compiler can be used by calling it by its absolute path, or by adding the `$PREFIX-mingw32/bin` directory to the `PATH` environment variable.

## 12.6 Using Git

Most of the sources of the projects above are now managed with the `git` distributed version-control tools. When you want to build from the newest development branch, you can clone the repo, like with

```
$ git clone <repo> [dirname]
```

Replace `<repo>` with the URL of the Git repository, e.g. `https://github.com/avrduces/avr-libc.git` for AVR-LibC. Notice that when building AVR-LibC from the repo source, you have to run `./bootstrap` from the top level AVR-LibC sources prior to `configure`.

Useful options for `git clone`:

**dirname** Specify an optional directory name for the cloned repository, like:

```
$ git clone https://github.com/avrduces/avr-libc.git ./source/avr-libc-main
```

Without `dirname`, the name of the git file like `avr-libc` is used.

**--depth 1** An ordinary clone will clone the complete repository with all its branches and their history. To speed up the cloning and save some disc space, you can just clone the top of the history to some depth.

**--branch branch** The default branch is the head of the latest development, which is `master` for GCC and Binutils, and `main` for AVR-LibC.

When you want a different ref, like GCC's `releases/gcc-14` for the head of the GCC v14 branch, or `releases/gcc-14.1.0` for the GCC v14.1 release tag, then you can specify that as `branch`. To see the available refs, you can use

```
$ git ls-remote <repo>
```

## 13 Using the GNU tools

This is a short summary of the AVR-specific aspects of using the GNU tools. Normally, the generic documentation of these tools is fairly large and maintained in `texinfo` files. Command-line options are explained in detail in the manual page.

### 13.1 Options for the C compiler `avr-gcc`

#### 13.1.1 Machine-specific options for the AVR

The following machine-specific options are recognized by the C compiler frontend. The preprocessor will define the macros `__AVR` and `__AVR__` (to the value 1) when compiling for an AVR target. The macro `AVR` will be defined as well, except in strict ANSI mode.

There are many options supported by `avr-gcc`, which also depend on the compiler version. For a complete overview, please see the documentation of `avr-gcc`'s command line options. Here are links to supported options of the respective release series:

- [Current development](#) (work in progress)
- [v14](#)
- [v13.2](#), [v13.3](#)
- [v12.3](#), [v12.4](#)
- [v11](#)
- [v10](#)
- [v9](#)
- [v8](#)
- [v7](#)
- [v6](#)
- [v5](#)
- [v4.9](#)
- [v4.8](#)
- [v4.7](#)

Apart from the documentation of command line options, the linked pages also contain:

- The documentation of [built-in macros](#) like `__AVR_ARCH__`, `__AVR_ATmega328P__` and `__AVR_HAVE_MUL__`, just to mention a few.
- How the compiler treats the [RAMPX, RAMPY, RAMPZ and RAMPD special function registers](#) on devices that have (one of) them.
- How the compiler treats the [EIND special function register](#) on devices with more than 128 KiB of program memory, and how indirect calls are realized on such devices.

`-mmcu=arch`

**-mmcu=*mcu*** Compile code for architecture *arch* resp. AVR device *mcu*.

Since GCC v5, the compiler no more supports individual devices, instead the compiler comes with **device specs files** that describe which options to use with each sub-processes like pre-processor, compiler proper, assembler and linker.

The purpose of these specs files is to add support for AVR devices that the compiler does not yet support.

The easiest way to add support for an unsupported device is to use device support from an **atpack archive** as provided by the hardware manufacturer. Apart from the *mcu* specific specs file, it provides device headers *io\*.h*, startup code *crtmcu.o* and device library *libmcu.a*.

### 13.1.2 Selected general compiler options

The following general gcc options might be of some interest to AVR users.

**-On** Optimization level *n*. Increasing *n* is meant to optimize more.

**-O0** reduces compilation time and makes debugging produce the expected results. This is the default. Turning off all optimizations will prevent some warnings from being issued since the generation of those warnings depends on code analysis steps that are only performed when optimizing (unreachable code, unused variables). Moreover, the delay routines in `<util/delay.h>` require optimization to be turned on.

**-O2** optimizes for speed, but without enabling very expensive optimizations like **-O3** does.

**-Os** turns on all **-O2** optimizations except those that often increase code size. In most cases, this is the preferred optimization level for AVR programs.

**-Og** optimizes debugging experience. This should be the optimization level of choice for the standard edit-compile-debug cycle, offering a reasonable level of optimization while maintaining fast compilation and a good debugging experience.

**-O3** attempts to inline all "simple" functions and might unroll some loops. For the AVR target, this will normally constitute a large pessimization due to the code increasement.

**-O** is equivalent to **-O1**. The compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

See also the [appropriate FAQ entry](#) for issues regarding debugging optimized code.

**-Wp, preprocessor-options**

**-Wa, assembler-options**

**-Wl, linker-options** Pass the listed options to the C preprocessor, the assembler or the linker, respectively. Several options can be passed at once when they are separated by a , (comma).

**-g** Generate debugging information that can be used by `avr-gdb`. GCC v12 changed the default from STABS to DWARF. Different DWARF levels can be selected by `-g2` or `-gdwarf-3`. The compiler may use GNU extensions to the DWARF format. When a debugger has problems with that, try `-gstrict-dwarf`.

**-x lang**

**-x none** Compile the following files in language *lang*. Values for *lang* are: `c`, `c++`, `assembler`, `assembler-with-cpp` and `none`.

For example, GCC does not recognize the `.asm` file extension as assembly source. With `-x assembler-with-cpp file.asm`, the compiler first runs the C preprocessor on `file.asm` (so that `#include <avr/io.h>` can be used in assembly), and then calls the assembler.

Another use case is to compile a C file that's read from standard input, which is specified as `-` (dash) instead of the name of a source file. As no source file name is specified, the compiler must be told which language to use: The command

```
$ echo '#include <avr/io.h>' | avr-gcc -xc - -mmcu=atmega8 -E -dM | grep _VECTOR
```

pre-processes the C file `#include <avr/io.h>` and writes all macro definitions to stdout. The output is then filtered by `grep` to show all possible `ISR` vector names for ATmega8.

`-x none` returns to the default for the following inputs, i.e. infer the respective source languages from the file extensions.

**`-save-temps`**

**`-save-temps=obj`**

**`-save-temps=cwd`**

**`-fverbose-asm`**

**`-dumpbase base`**

**`-dumpdir dir`** Don't remove temporary, intermediate files like C preprocessor output and assembly code generated by the compiler. The intermediate files have file extension `.i` (preprocessed C), `.ii` (preprocessed C++) and `.s` (preprocessed assembly, compiler-generated assembly).

`-dumpbase` and `-dumpdir` can be used to adjust file names and locations.

With `-fverbose-asm`, the compiler adds the high level source code to the assembly output. Compiling without debugging information (`-g0`) improves legibility of the generated assembly.

The preprocessed files can be used to check if macro expansions work as expected. With `-g3` or higher, the preprocessed files will also contain all macro definitions and indications where they are defined: Built-in, on the command line, or in some header file as indicated by `#line` notes.

**`-lname`** Locate the archive library named `libname.a`, and use it to resolve currently unresolved symbols from it. The library is searched along a path that consists of builtin pathname entries that have been specified at configure time (e. g. `/usr/local/avr/lib` on Unix systems), possibly extended by pathname entries as specified by `-L` options (that must precede the `-l` options on the command-line).

**`-Lpath`** Additional directory `path` to look for archive libraries requested by `-l` options.

**`-ffunction-sections`**

**`-fdata-sections`** Put each function resp. object in static storage in its own [input section](#). This is used with `-Wl,--gc-sections` so the linker can better garbage-collect unused sections, which are sections that are neither referenced by other sections, nor are marked as `KEEP`, nor are referenced by an entry symbol.

**`-mrelax`** Replace `JMP` and `CALL` instructions with the faster and shorter `RJMP` and `RCALL` instructions if possible. That optimization is performed by the linker, and the assembler must not resolve some expressions, which is all arranged by `-mrelax`.

**`-Tbss org`**

**`-Tdata org`**

**`-Ttext org`** Start the `.bss`, `.data`, or `.text` section at VMA address `org`, respectively.

**`-T scriptfile`** Use `scriptfile` as the linker script, replacing or augmenting the default linker script.

Default linker scripts are stored in a system-specific location (e. g. under `/usr/local/avr/lib/ldscripts` on Unix systems), and consist of the AVR architecture name (`avr2` through `avrxcmega7`) with the suffix `.x` appended. They describe how the various [memory sections](#) will be linked together and which input sections go into them. Notice that the default linker scripts are part of the linker binary, changing them on file will have no effect.

For a simple linker script augmentation, see the `avr-gcc` Wiki on [named address spaces](#).

**`-nostdlib`** Don't link against standard libraries.

**`-nodefaultlibs`** Don't link against default libraries.

**`-nodevicelib`** Don't link against AVR-LibC's `libmcu.a` that contains EEPROM support and other stuff. This can be used when no such library is available.

**-nostartfiles** Don't link against AVR-LibC's [startup code](#) `crtmcu.o`.

Notice that parts of the startup code are provided by `libgcc.a`. To get rid of that, one can `-nostdlib` or `-nodefaultlibs`; however that also removes other code like functions required for arithmetic. To just get rid of the startup bits, define the respective symbols, for example `-Wl,--defsym,__do_clear_bss=0` and similar for `__do_copy_data`, `__do_global_ctors` and `__do_global_dtors`.

**-funsigned-char** This option changes the binary interface!

Make any unqualified `char` type an unsigned char. Without this option, they default to a signed char.

**-funsigned-bitfields** This option changes the binary interface!

Make any unqualified bitfield type unsigned. By default, they are signed.

**-fshort-enums** This option changes the binary interface!

Allocate to an `enum` type only as many bytes as it needs for the declared range of possible values. Specifically, the enum type will be equivalent to the smallest integer type which has enough room.

**-fpack-struct** This option changes the binary interface!

Pack all structure members together without holes.

**-fno-jump-tables** Do not generate `tablejump` instructions. By default, jump tables can be used to optimize `switch` statements. When turned off, sequences of compare statements are used instead. Jump tables are usually faster to execute on average, but in particular for `switch` statements, where most of the jumps would go to the default label, they might waste a bit of flash memory.

Note: Since GCC v4.9.2, `tablejump` code uses the `ELPM` instruction to read from jump tables. In older version, use the `-fno-jump-tables` switch when compiling a bootloader for devices with more than 64 KiB of code memory.

**-ffreestanding** Assume a "freestanding" environment as per the C standard. This turns off automatic builtin functions (though they can still be reached by prepending `__builtin_` to the actual function name). It also makes the compiler not complain when `main()` is declared with a `void` return type. (In most cases, `main()` won't even return anyway, and hence using a return type of `int` has no downsides at all).

However, this option also turns off all optimizations normally done by the compiler which assume that functions known by a certain name behave as described by the standard. For example, applying the function `strlen()` to a literal string will normally cause the compiler to immediately replace that call by the actual length of the string, while with `-ffreestanding`, it will always call `strlen()` at run-time.

## 13.2 Options for the assembler `avr-as`

### Note

The preferred way to assemble a file is by means of using `avr-gcc`:

- `avr-gcc`, which is a driver program to call sub-programs like the compiler proper or the assembler, knows which options it has to add to the assembler's command line, like: `-mmcu=arch`, `-mno-skip-bug`, etc.
- `avr-gcc` will call the C preprocessor on the assembler input for sources with extensions `.S` and `.sx`. For other extensions, use

```
$ avr-gcc -x assembler-with-cpp file.asm ...
```

This allows to use C preprocessor directives like `#include <avr/io.h>` in assembly sources.

### 13.2.1 Machine-specific assembler options

**`-mmcu=architecture`**

**`-mmcu=mcu`** `avr-as` does not support all *mcus* supported by the compiler. As explained in the note above, the preferred way to run the assembler is by using the compiler driver `avr-gcc`.

**`-mall-opcodes`** Turns off opcode checking, and allows any possible AVR opcode to be assembled.

**`-mno-skip-bug`** Don't emit a warning when trying to skip a 2-word instruction with a `CPSE/SBIC/SBIS/↔SBRC/SBRS` instruction. Early AVR devices suffered from a hardware bug where these instructions could not be properly skipped.

**`-mno-wrap`** For `RJMP/RCALL` instructions, don't allow the target address to wrap around for devices that have more than 8 KiB of memory.

**`--gstabs`** Generate `.stabs` debugging symbols for assembler source lines. This enables `avr-gdb` to trace through assembler source files. This option *must not* be used when assembling sources that have been generated by the C compiler; these files already contain the appropriate line number information from the C source files.

**`-a [cdhlmns=file]`** Turn on the assembler listing. The sub-options are:

- `c` omit false conditionals
- `d` omit debugging directives
- `h` include high-level source
- `l` include assembly
- `m` include macro expansions
- `n` omit forms processing
- `s` include symbols
- `=file` set the name of the listing file

The various sub-options can be combined into a single `-a` option list; `=file` must be the last one in that case.

### 13.2.2 Examples for assembler options passed through the C compiler

Remember that assembler options can be passed from the C compiler frontend using `-Wa` (see [gcc\\_minusW](#) above), so in order to include the C source code into the assembler listing in file `foo.lss`, when compiling `foo.c`, the following compiler command-line can be used:

```
$ avr-gcc -mmcu=atmega8 -c foo.c -o foo.o -Wa,-ahls=foo.lss
```

In order to pass an assembler file through the C preprocessor first, and have the assembler generate line number debugging information for it, the following command can be used:

```
$ avr-gcc -c -x assembler-with-cpp -o foo.o foo.asm -Wa,--gstabs
```

Note that on Unix systems that have case-distinguishing file systems, specifying a file name with the suffix `.S` (upper-case letter `S`) will make the compiler automatically assume `-x assembler-with-cpp`, while using `.s` would pass the file directly to the assembler (no preprocessing done).



## 13.3 Controlling the linker `avr-ld`

### Note

It is highly recommended to use the compiler driver `avr-gcc` or `avr-g++` for linking.

- The driver knows which options to pass down to the linker. This includes the correct multilib path, support libraries like `libc.a`, `libm.a`, `libgcc.a` and `libmcpu.a`, as well as options for LTO (link-time optimization) and options for plugins (that call back the compiler to compile LTO byte code), startup code and many more.
- The driver program understands options like `-llib`, `-Lpath`, `Ttext`, `Tdata`, `Tbss` and `-Tscript`, so no `-Wl` is required for them.

### 13.3.1 Selected linker options

A number of the standard options might be of interest to AVR users.

`--defsym symbol=expr` Define a global symbol *symbol* using *expr* as the value.

`-M` Print a linker map to `stdout`.

`-Map mapfile` Print a linker map to *mapfile*.

`--cref` Output a cross reference table to the map file (in case `-Map` is also present), or to `stdout`.

`--gc-sections` Only keep input sections that are referenced (by other sections or the entry symbol), and that are not marked as `KEEP` in the linker script. This is used to reduce code size, usually together with compiler option `-ffunction-sections` so that input section granularity is on function level rather than on the level of compilation units.

`--section-start sectionname=org` Start section *sectionname* at absolute address *org*.

`--relax` Don't use this option directly or per `-Wl,--relax`. Instead, link with `avr-gcc ... -mrelax`.

### 13.3.2 Passing linker options from the C compiler

By default, all unknown non-option arguments on the `avr-gcc` command-line (i. e., all filename arguments that don't have a suffix that is handled by `avr-gcc`) are passed straight to the linker. Thus, all files ending in `.o` (object files) and `.a` (object libraries) are provided to the linker.

System libraries are usually not passed by their explicit filename but rather using the `-l` option which uses an abbreviated form of the archive filename (see above). AVR-LibC ships system libraries, `libc.a`, `libm.a` and `libmcpu.a`. While the standard library `libc.a` will always be searched for unresolved references when the linker is started using the C compiler frontend (i. e., there's always at least one implied `-lc` option), the mathematics library `libm.a` is only automatically added in GCC v4.7 and above. On older versions, it needs to be explicitly requested using `-lm`.

Conventionally, Makefiles use the `make` macro `LDLIBS` to keep track of `-l` (and possibly `-L`) options that should only be appended to the C compiler command-line when linking the final binary. In contrast, the macro `LD_FLAGS` is used to store other command-line options to the C compiler that should be passed as options during the linking stage. The difference is that options are placed early on the command-line, while libraries are put at the end since they are to be used to resolve global symbols that are still unresolved at this point.

Specific linker flags can be passed from the C compiler command-line using the `-Wl` compiler option, see [gcc\\_minusW](#) above. This option requires that there be no spaces in the appended linker option, while some of the linker options above (like `-Map` or `--defsym`) would require a space. In these situations, the space can be replaced by an equal sign as well. For example, the following command-line can be used to compile `foo.c` into an executable, and also produce a link map that contains a cross-reference list in the file `foo.map`:

```
$ avr-gcc -mmcu=atmega8 foo.c -o foo.elf -Wl,-Map,foo.map -Wl,--cref
```

Alternatively, a comma as a placeholder will be replaced by a space before passing the option to the linker. So for a device with external SRAM, the following command-line would cause the linker to place the data segment at address 0x2000 in the SRAM:

```
$ avr-gcc -mmcu=atmega128 foo.c -o foo.elf -Wl,-Tdata,0x802000
```

See the explanation of the [data section](#) for why 0x800000 needs to be added to the actual value. Note that the stack will still remain in internal RAM, through the symbol `__stack` that is provided by the run-time startup code. This is probably a good idea anyway (since internal RAM access is faster), and even required for some early devices that had hardware bugs preventing them from using a stack in external RAM. Note also that the heap for `malloc()` will still be placed after all the variables in the data section, so in this situation, no stack/heap collision can occur.

In order to relocate the stack from its default location at the top of internal RAM, the value of the symbol `__stack` can be changed on the linker command-line. As the linker is typically called from the compiler frontend, this can be achieved using a compiler option like

```
-Wl,--defsym=__stack=0x8003ff
```

The above will make the code use stack space from RAM address 0x3ff downwards. The amount of stack space available then depends on the bottom address of internal RAM for a particular device. It is the responsibility of the application to ensure the stack does not grow out of bounds, as well as to arrange for the stack to not collide with variable allocations made by the compiler (sections `.data` and `.bss`).

## 14 Compiler optimization

### 14.1 Problems with reordering code

Author

Jan Waclawek

Programs contain sequences of statements, and a naive compiler would execute them exactly in the order as they are written. But an optimizing compiler is free to *reorder* the statements — or even parts of them — if the resulting "net effect" is the same. The "measure" of the "net effect" is what the standard calls "side effects", and is accomplished exclusively through accesses (reads and writes) to variables qualified as `volatile`. So, as long as all volatile reads and writes are to the same addresses and in the same order (and writes write the same values), the program is correct, regardless of other operations in it. One important point to note here is, that time duration between consecutive volatile accesses is not considered at all.

Unfortunately, there are also operations which are not covered by volatile accesses. An example of this in AVR-↔GCC/AVR-LibC are the `cli()` and `sei()` macros defined in `<avr/interrupt.h>`, which convert directly to the respective assembler mnemonics through the `__asm__()` statement. They constitute a variable access by means of their memory clobber, and they are (implicitly) volatile because they don't have an output operand. So the compiler may not reorder these `inline asm` statements with respect to other memory accesses or volatile actions. However, such `asm` statements may still be reordered with other statements that are neither volatile nor access memory.

*Note that even a volatile `asm` instruction can be moved relative to other code, including across (expensive) arithmetic and jump instructions [...]*

**See also**

<http://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html>

However, not even a volatile memory barrier like

```
__asm __volatile__ (" ::: "memory");
```

keeps GCC from reordering non-volatile, non-memory accesses across such barriers. Peter Danneegger provided a nice example of this effect:

```
#define cli() __asm volatile( "cli" ::: "memory" )
#define sei() __asm volatile( "sei" ::: "memory" )

unsigned int ivar;

void test2 (unsigned int val)
{
    val = 65535U / val;

    cli();

    ivar = val;

    sei();
}
```

avr-gcc v5.4 or v14 compile with optimisations switched on (`-Os`) to

```
00000112 <test2>:
112: bc 01      movw r22, r24
114: f8 94      cli
116: 8f ef      ldi r24, 0xFF ; 255
118: 9f ef      ldi r25, 0xFF ; 255
11a: 0e 94 96 00 call 0x12c ; 0x12c <__udivmodhi4>
11e: 70 93 01 02 sts 0x0201, r23
122: 60 93 00 02 sts 0x0200, r22
126: 78 94      sei
128: 08 95      ret
```

where the potentially slow division is moved across `cli()`, resulting in interrupts to be disabled longer than intended. Note, that the volatile access occurs in order with respect to `cli()` or `sei()`; so the "net effect" required by the standard is achieved as intended, it is "only" the timing which is off. However, for most of embedded applications, timing is an important, sometimes critical factor.

**See also**

<https://www.mikrocontroller.net/topic/65923>

Unfortunately, at the moment, in `avr-gcc` (nor in the C standard), there is no mechanism to enforce complete match of written and executed code ordering — except maybe of switching the optimization completely off (`-O0`), or writing all the critical code in assembly.

**Note**

The artifact with the `__udivmodhi4` function is specific to `avr-gcc` and how the compiler represents the division internally. On other target platforms that are using a library function for division or whatever expensive operation, this effect will not occur. The reason is that `avr-gcc` does not represent the library call as a function call but rather like an ordinary instruction. Outcome is that the GCC middle-end concludes that the division is cheap (because the backend has an instruction for it) but in fact it's not.

A work around for the code from above would be to enforce that the division happens prior to the `cli()`:

```
val = 65535U / val;
__asm __volatile__ (" : "+r" (val));
cli();
```

- The `volatile` forces the asm statement prior to the `cli`.
- The asm has `val` as input operand, hence the division must be carried out prior to the asm because `val` is set by the division.

Notice that this work around does not work in general due to a variety of reasons:

- The division might be located in an inlined function.
- The variable might be read-only or may not be appropriate as an asm operand.
- There may be more such instruction prior to the division, and it is not practical to treat all of them like this.

To sum it up:

- volatile memory barriers don't ensure statements with no volatile accesses to be reordered across the barrier

## 15 Using the avrdude program

### Note

This section was contributed by Brian Dean [ [bsd@bsdhome.com](mailto:bsd@bsdhome.com) ].

The avrdude program was previously called avrprog. The name was changed to avoid confusion with the avrprog program that Atmel ships with AvrStudio.

avrdude is a program that is used to update or read the flash and EEPROM memories of Atmel AVR microcontrollers on FreeBSD Unix. It supports the Atmel serial programming protocol using the PC's parallel port and can upload either a raw binary file or an Intel Hex format file. It can also be used in an interactive mode to individually update EEPROM cells, fuse bits, and/or lock bits (if their access is supported by the Atmel serial programming protocol.) The main flash instruction memory of the AVR can also be programmed in interactive mode, however this is not very useful because one can only turn bits off. The only way to turn flash bits on is to erase the entire memory (using avrdude's `-e` option).

avrdude is part of the FreeBSD ports system. To install it, simply do the following:

```
# cd /usr/ports/devel/avrdude
# make install
```

Once installed, avrdude can program processors using the contents of the `.hex` file specified on the command line. In this example, the file `main.hex` is burned into the flash memory:

```
# avrdude -p 2313 -e -m flash -i main.hex

avrdude: AVR device initialized and ready to accept instructions

avrdude: Device signature = 0x1e9101

avrdude: erasing chip
avrdude: done.
avrdude: reading input file "main.hex"
avrdude: input file main.hex auto detected as Intel Hex

avrdude: writing flash:
1749 0x00
avrdude: 1750 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: reading on-chip flash data:
1749 0x00
avrdude: verifying ...
avrdude: 1750 bytes of flash verified

avrdude done. Thank you.
```

The `-p 2313` option lets `avrdude` know that we are operating on an AT90S2313 chip. This option specifies the device id and is matched up with the device of the same id in `avrdude`'s configuration file (`/usr/local/etc/avrdude.conf`). To list valid parts, specify the `-v` option. The `-e` option instructs `avrdude` to perform a chip-erase before programming; this is almost always necessary before programming the flash. The `-m flash` option indicates that we want to upload data into the flash memory, while `-i main.hex` specifies the name of the input file.

The EEPROM is uploaded in the same way, the only difference is that you would use `-m eeprom` instead of `-m flash`.

To use interactive mode, use the `-t` option:

```
# avrdude -p 2313 -t
avrdude: AVR device initialized and ready to accept instructions
avrdude: Device signature = 0x1e9101
avrdude>
```

The `'?'` command displays a list of valid commands:

```
avrdude> ?
>>> ?
Valid commands:

dump   : dump memory   : dump <memtype> <addr> <N-Bytes>
read   : alias for dump
write  : write memory  : write <memtype> <addr> <b1> <b2> ... <bN>
erase  : perform a chip erase
sig    : display device signature bytes
part   : display the current part information
send   : send a raw command : send <b1> <b2> <b3> <b4>
help   : help
?      : help
quit   : quit
```

Use the `'part'` command to display valid memory types for use with the `'dump'` and `'write'` commands.

```
avrdude>
```

## 16 Acknowledgments

This document tries to tie together the labors of a large group of people. Without these individuals' efforts, we wouldn't have a terrific, **free** set of tools to develop AVR projects. We all owe thanks to:

- The GCC Team, which produced a very capable set of development tools for an amazing number of platforms and processors.
- Denis Chertykov [ [denisc@overta.ru](mailto:denisc@overta.ru) ] for making the AVR-specific changes to the GNU tools.
- Denis Chertykov and Marek Michalkiewicz [ [marekm@linux.org.pl](mailto:marekm@linux.org.pl) ] for developing the standard libraries and startup code for **AVR-GCC**.
- Uros Platise for developing the AVR programmer tool, **uisp**.
- Joerg Wunsch [ [joerg@FreeBSD.ORG](mailto:joerg@FreeBSD.ORG) ] for adding all the AVR development tools to the FreeBSD [ <http://www.freebsd.org> ] ports tree and for providing the basics for the [demo project](#).
- Brian Dean [ [bsd@bsdhome.com](mailto:bsd@bsdhome.com) ] for developing **avrdude** (an alternative to **uisp**) and for contributing [documentation](#) which describes how to use it. **Avrdude** was previously called **avrprog**.

- Eric Weddington [ [eweddington@cs0.atmel.com](mailto:eweddington@cs0.atmel.com) ] for maintaining the **WinAVR** package and thus making the continued improvements to the open source AVR toolchain available to many users.
- Rich Neswold for writing the original avr-tools document (which he graciously allowed to be merged into this document) and his improvements to the [demo project](#).
- Theodore A. Roth for having been a long-time maintainer of many of the tools (**AVR-LibC**, the AVR port of **GDB**, **AVaRICE**, **uisp**, **avrdude**).
- All the people who currently maintain the tools, and/or have submitted suggestions, patches and bug reports. (See the AUTHORS files of the various tools.)
- And lastly, all the users who use the software. If nobody used the software, we would probably not be very motivated to continue to develop it. Keep those bug reports coming. ;-)

## 17 Deprecated List

Global **cbi** (port, bit)

Global **enable\_external\_int** (mask)

Global **inb** (port)

Global **inp** (port)

Global **INTERRUPT** (signame)

Global **ISR\_ALIAS** (vector, target\_vector)

For new code, the use of [ISR\(..., ISR\\_ALIASOF\(...\)\)](#) is recommended.

Global **outb** (port, val)

Global **outp** (val, port)

Global **sbi** (port, bit)

Global **SIGNAL** (vector)

Do not use [SIGNAL\(\)](#) in new code. Use [ISR\(\)](#) instead.

Global **timer\_enable\_int** (unsigned char ints)

## 18 Module Index

### 18.1 Modules

Here is a list of all modules:

<alloca.h>: Allocate space in the stack

103

---

<a href="#">&lt;assert.h&gt;: Diagnostics</a>	<a href="#">104</a>
<a href="#">&lt;ctype.h&gt;: Character Operations</a>	<a href="#">105</a>
<a href="#">&lt;errno.h&gt;: System Errors</a>	<a href="#">107</a>
<a href="#">&lt;inttypes.h&gt;: Integer Type conversions</a>	<a href="#">108</a>
<a href="#">&lt;math.h&gt;: Mathematics</a>	<a href="#">122</a>
<a href="#">&lt;setjmp.h&gt;: Non-local goto</a>	<a href="#">142</a>
<a href="#">&lt;stdint.h&gt;: Standard Integer Types</a>	<a href="#">144</a>
<a href="#">&lt;stdio.h&gt;: Standard IO facilities</a>	<a href="#">156</a>
<a href="#">&lt;stdlib.h&gt;: General utilities</a>	<a href="#">172</a>
<a href="#">&lt;string.h&gt;: Strings</a>	<a href="#">184</a>
<a href="#">&lt;time.h&gt;: Time</a>	<a href="#">197</a>
<a href="#">&lt;avr/boot.h&gt;: Bootloader Support Utilities</a>	<a href="#">207</a>
<a href="#">&lt;avr/cpufunc.h&gt;: Special AVR CPU functions</a>	<a href="#">212</a>
<a href="#">&lt;avr/eeprom.h&gt;: EEPROM handling</a>	<a href="#">214</a>
<a href="#">&lt;avr/fuse.h&gt;: Fuse Support</a>	<a href="#">219</a>
<a href="#">&lt;avr/interrupt.h&gt;: Interrupts</a>	<a href="#">222</a>
<a href="#">&lt;avr/io.h&gt;: AVR device-specific IO definitions</a>	<a href="#">228</a>
<a href="#">&lt;avr/lock.h&gt;: Lockbit Support</a>	<a href="#">230</a>
<a href="#">&lt;avr/pgmspace.h&gt;: Program Space Utilities</a>	<a href="#">232</a>
<a href="#">&lt;avr/power.h&gt;: Power Reduction Management</a>	<a href="#">259</a>
<a href="#">&lt;avr/sfr_defs.h&gt;: Special function registers</a>	<a href="#">263</a>
<a href="#">Additional notes from &lt;avr/sfr_defs.h&gt;</a>	<a href="#">263</a>
<a href="#">&lt;avr/signature.h&gt;: Signature Support</a>	<a href="#">265</a>
<a href="#">&lt;avr/sleep.h&gt;: Power Management and Sleep Modes</a>	<a href="#">266</a>
<a href="#">&lt;avr/version.h&gt;: avr-libc version macros</a>	<a href="#">268</a>
<a href="#">&lt;avr/builtins.h&gt;: avr-gcc builtins documentation</a>	<a href="#">269</a>
<a href="#">&lt;avr/wdt.h&gt;: Watchdog timer handling</a>	<a href="#">271</a>
<a href="#">&lt;util/delay.h&gt;: Convenience functions for busy-wait delay loops</a>	<a href="#">273</a>
<a href="#">&lt;util/atomic.h&gt;: Atomically and Non-Atomically Executed Code Blocks</a>	<a href="#">276</a>
<a href="#">&lt;util/crc16.h&gt;: CRC Computations</a>	<a href="#">278</a>
<a href="#">&lt;util/delay_basic.h&gt;: Basic busy-wait delay loops</a>	<a href="#">282</a>
<a href="#">&lt;util/eu_dst.h&gt;: Daylight Saving function for the European Union.</a>	<a href="#">282</a>

<a href="#">&lt;util/parity.h&gt;: Parity bit generation</a>	283
<a href="#">&lt;util/setbaud.h&gt;: Helper macros for baud rate calculations</a>	284
<a href="#">&lt;util/twi.h&gt;: TWI bit mask definitions</a>	286
<a href="#">&lt;util/usa_dst.h&gt;: Daylight Saving function for the USA.</a>	290
<a href="#">&lt;compat/deprecated.h&gt;: Deprecated items</a>	291
<a href="#">&lt;compat/ina90.h&gt;: Compatibility with IAR EWB 3.x</a>	293
<b>Demo projects</b>	294
<b>Combining C and assembly source files</b>	295
<b>A simple project</b>	297
<b>A more sophisticated project</b>	308
<b>Using the standard IO facilities</b>	312
<b>Example using the two-wire interface (TWI)</b>	317

## 19 Data Structure Index

### 19.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">div_t</a>	321
<a href="#">ldiv_t</a>	322
<a href="#">tm</a>	323
<a href="#">week_date</a>	324

## 20 File Index

### 20.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">project.h</a>	325
<a href="#">iocompat.h</a>	325
<a href="#">defines.h</a>	327
<a href="#">hd44780.h</a>	328
<a href="#">lcd.h</a>	329
<a href="#">uart.h</a>	329



<a href="#">alloca.h</a>	330
<a href="#">assert.h</a>	331
<a href="#">boot.h</a>	332
<a href="#">builtins.h</a>	341
<a href="#">cpufunc.h</a>	343
<a href="#">eeprom.h</a>	344
<a href="#">fuse.h</a>	348
<a href="#">interrupt.h</a>	352
<a href="#">io.h</a>	357
<a href="#">lock.h</a>	366
<a href="#">pgmspace.h</a>	369
<a href="#">portpins.h</a>	396
<a href="#">power.h</a>	402
<a href="#">sfr_defs.h</a>	424
<a href="#">signal.h</a>	427
<a href="#">signature.h</a>	428
<a href="#">sleep.h</a>	429
<a href="#">version.h</a>	433
<a href="#">wdt.h</a>	434
<a href="#">xmega.h</a>	442
<a href="#">deprecated.h</a>	443
<a href="#">ina90.h</a>	446
<a href="#">ctype.h</a>	447
<a href="#">errno.h</a>	450
<a href="#">inttypes.h</a>	452
<a href="#">math.h</a>	461
<a href="#">setjmp.h</a>	472
<a href="#">stdint.h</a>	474
<a href="#">stdio.h</a>	485
<a href="#">stdlib.h</a>	498
<a href="#">string.h</a>	509
<a href="#">time.h</a>	517

<a href="#">atomic.h</a>	525
<a href="#">crc16.h</a>	529
<a href="#">delay.h</a>	533
<a href="#">delay_basic.h</a>	537
<a href="#">eu_dst.h</a>	539
<a href="#">parity.h</a>	540
<a href="#">setbaud.h</a>	541
<a href="#">compat/twi.h</a>	544
<a href="#">util/twi.h</a>	544
<a href="#">usa_dst.h</a>	548
<a href="#">eedef.h</a>	549
<a href="#">fdevopen.c</a>	551
<a href="#">stdio_private.h</a>	551
<a href="#">xtoa_fast.h</a>	552
<a href="#">dtoa_conv.h</a>	552
<a href="#">stdlib_private.h</a>	553
<a href="#">ephemera_common.h</a>	554

## 21 Module Documentation

### 21.1 <alloca.h>: Allocate space in the stack

#### Functions

- void \* [alloca](#) (size\_t \_\_size)

#### 21.1.1 Detailed Description

#### 21.1.2 Function Documentation

**21.1.2.1 `alloca()`** `void * alloca (`  
`size_t __size )`

Allocate `__size` bytes of space in the stack frame of the caller.

This temporary space is automatically freed when the function that called `alloca()` returns to its caller. AVR-LibC defines the `alloca()` as a macro, which is translated into the inlined `__builtin_alloca()` function. The fact that the code is inlined, means that it is impossible to take the address of this function, or to change its behaviour by linking with a different library.

#### Returns

`alloca()` returns a pointer to the beginning of the allocated space. If the allocation causes stack overflow, program behaviour is undefined.

#### Warning

Avoid use `alloca()` inside the list of arguments of a function call.

## 21.2 `<assert.h>`: Diagnostics

### Macros

- #define `assert`(expression)

#### 21.2.1 Detailed Description

```
#include <assert.h>
```

This header file defines a debugging aid.

As there is no standard error output stream available for many applications using this library, the generation of a printable error message is not enabled by default. These messages will only be generated if the application defines the macro

```
__ASSERT_USE_STDERR
```

before including the `<assert.h>` header file. By default, only `abort()` will be called to halt the application.

#### 21.2.2 Macro Definition Documentation

**21.2.2.1 `assert`** #define `assert (`  
`expression )`

#### Parameters

<code>expression</code>	Expression to test for.
-------------------------	-------------------------

The `assert()` macro tests the given expression and if it is false, the calling process is terminated. A diagnostic

message is written to stderr and the function `abort()` is called, effectively terminating the program.

If expression is true, the `assert()` macro does nothing.

The `assert()` macro may be removed at compile time by defining `NDEBUG` as a macro (e.g., by using the compiler option `-DNDEBUG`).

## 21.3 <ctype.h>: Character Operations

### Character classification routines

These functions perform character classification. They return true or false status depending whether the character passed to the function falls into the function's classification (i.e. `isdigit()` returns true if its argument is any value '0' through '9', inclusive). If the input is not an unsigned char value, all of this function return false.

- int `isalnum` (int \_\_c)
- int `isalpha` (int \_\_c)
- int `isascii` (int \_\_c)
- int `isblank` (int \_\_c)
- int `iscntrl` (int \_\_c)
- int `isdigit` (int \_\_c)
- int `isgraph` (int \_\_c)
- int `islower` (int \_\_c)
- int `isprint` (int \_\_c)
- int `ispunct` (int \_\_c)
- int `isspace` (int \_\_c)
- int `isupper` (int \_\_c)
- int `isxdigit` (int \_\_c)

### Character conversion routines

This realization permits all possible values of integer argument. The `toascii()` function clears all highest bits. The `tolower()` and `toupper()` functions return an input argument as is, if it is not an unsigned char value.

- int `toascii` (int \_\_c)
- int `tolower` (int \_\_c)
- int `toupper` (int \_\_c)

#### 21.3.1 Detailed Description

These functions perform various operations on characters.

```
#include <ctype.h>
```

#### 21.3.2 Function Documentation

**21.3.2.1 isalnum()** `int isalnum (`  
`int __c )`

Checks for an alphanumeric character. It is equivalent to `(isalpha(c) || isdigit(c))`.

**21.3.2.2 isalpha()** `int isalpha (`  
`int __c )`

Checks for an alphabetic character. It is equivalent to `(isupper(c) || islower(c))`.

**21.3.2.3 isascii()** `int isascii (`  
`int __c )`

Checks whether `c` is a 7-bit unsigned char value that fits into the ASCII character set.

**21.3.2.4 isblank()** `int isblank (`  
`int __c )`

Checks for a blank character, that is, a space or a tab.

**21.3.2.5 iscntrl()** `int iscntrl (`  
`int __c )`

Checks for a control character.

**21.3.2.6 isdigit()** `int isdigit (`  
`int __c )`

Checks for a digit (0 through 9).

**21.3.2.7 isgraph()** `int isgraph (`  
`int __c )`

Checks for any printable character except space.

**21.3.2.8 islower()** `int islower (`  
`int __c )`

Checks for a lower-case character.

**21.3.2.9 isprint()** `int isprint (`  
`int __c )`

Checks for any printable character including space.

**21.3.2.10 ispunct()** `int ispunct (`  
`int __c )`

Checks for any printable character which is not a space or an alphanumeric character.

**21.3.2.11 isspace()** `int isspace (`  
`int __c )`

Checks for white-space characters. For the AVR-LibC library, these are: space, form-feed ('\f'), newline ('\n'), carriage return ('\r'), horizontal tab ('\t'), and vertical tab ('\v').

**21.3.2.12 isupper()** `int isupper (`  
`int __c )`

Checks for an uppercase letter.

**21.3.2.13 isxdigit()** `int isxdigit (`  
`int __c )`

Checks for a hexadecimal digits, i.e. one of 0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F.

**21.3.2.14 toascii()** `int toascii (`  
`int __c )`

Converts `c` to a 7-bit unsigned char value that fits into the ASCII character set, by clearing the high-order bits.

#### Warning

Many people will be unhappy if you use this function. This function will convert accented letters into random characters.

**21.3.2.15 tolower()** `int tolower (`  
`int __c )`

Converts the letter `c` to lower case, if possible.

**21.3.2.16 toupper()** `int toupper (`  
`int __c )`

Converts the letter `c` to upper case, if possible.

## 21.4 <errno.h>: System Errors

### Macros

- `#define` [EDOM](#) 33
- `#define` [ERANGE](#) 34

### Variables

- `int` [errno](#)

### 21.4.1 Detailed Description

```
#include <errno.h>
```

Some functions in the library set the global variable `errno` when an error occurs. The file, `<errno.h>`, provides symbolic names for various error codes.

### 21.4.2 Macro Definition Documentation

#### 21.4.2.1 EDOM `#define EDOM 33`

Domain error.

#### 21.4.2.2 ERANGE `#define ERANGE 34`

Range error.

### 21.4.3 Variable Documentation

#### 21.4.3.1 `errno` `int errno [extern]`

Error code for last error encountered by library.

The variable `errno` holds the last error code encountered by a library function. This variable must be cleared by the user prior to calling a library function.

#### Warning

The `errno` global variable is not safe to use in a threaded or multi-task system. A race condition can occur if a task is interrupted between the call which sets `error` and when the task examines `errno`. If another task changes `errno` during this time, the result will be incorrect for the interrupted task.

## 21.5 `<inttypes.h>`: Integer Type conversions

### Far pointers for memory access > 64K

- typedef `int32_t int_farptr_t`
- typedef `uint32_t uint_farptr_t`

**macros for printf and scanf format specifiers**

For C++, these are only included if `__STDC_LIMIT_MACROS` is defined before including `<inttypes.h>`.

- `#define PRId8 "d"`
- `#define PRIdLEAST8 "d"`
- `#define PRIdFAST8 "d"`
- `#define PRIi8 "i"`
- `#define PRIiLEAST8 "i"`
- `#define PRIiFAST8 "i"`
- `#define PRId16 "d"`
- `#define PRIdLEAST16 "d"`
- `#define PRIdFAST16 "d"`
- `#define PRIi16 "i"`
- `#define PRIiLEAST16 "i"`
- `#define PRIiFAST16 "i"`
- `#define PRId32 "ld"`
- `#define PRIdLEAST32 "ld"`
- `#define PRIdFAST32 "ld"`
- `#define PRIi32 "li"`
- `#define PRIiLEAST32 "li"`
- `#define PRIiFAST32 "li"`
- `#define PRIdPTR PRId16`
- `#define PRIiPTR PRIi16`
- `#define PRIo8 "o"`
- `#define PRIoLEAST8 "o"`
- `#define PRIoFAST8 "o"`
- `#define PRU8 "u"`
- `#define PRU8LEAST8 "u"`
- `#define PRU8FAST8 "u"`
- `#define PRIx8 "x"`
- `#define PRIxLEAST8 "x"`
- `#define PRIxFAST8 "x"`
- `#define PRIX8 "X"`
- `#define PRIXLEAST8 "X"`
- `#define PRIXFAST8 "X"`
- `#define PRIo16 "o"`
- `#define PRIoLEAST16 "o"`
- `#define PRIoFAST16 "o"`
- `#define PRU16 "u"`
- `#define PRU16LEAST16 "u"`
- `#define PRU16FAST16 "u"`
- `#define PRIx16 "x"`
- `#define PRIxLEAST16 "x"`
- `#define PRIxFAST16 "x"`
- `#define PRIX16 "X"`
- `#define PRIXLEAST16 "X"`
- `#define PRIXFAST16 "X"`
- `#define PRIo32 "lo"`
- `#define PRIoLEAST32 "lo"`
- `#define PRIoFAST32 "lo"`
- `#define PRU32 "lu"`
- `#define PRU32LEAST32 "lu"`
- `#define PRU32FAST32 "lu"`



- #define `PRIx32` "lx"
- #define `PRiXLEAST32` "lx"
- #define `PRiXFAST32` "lx"
- #define `PRIX32` "IX"
- #define `PRiXLEAST32` "IX"
- #define `PRiXFAST32` "IX"
- #define `PRiOPtr` `PRiO16`
- #define `PRiUPtr` `PRiU16`
- #define `PRiXPTR` `PRiX16`
- #define `PRiXPTR` `PRiX16`
- #define `SCNd8` "hhd"
- #define `SCNdLEAST8` "hhd"
- #define `SCNdFAST8` "hhd"
- #define `SCNi8` "hhi"
- #define `SCNiLEAST8` "hhi"
- #define `SCNiFAST8` "hhi"
- #define `SCNd16` "d"
- #define `SCNdLEAST16` "d"
- #define `SCNdFAST16` "d"
- #define `SCNi16` "i"
- #define `SCNiLEAST16` "i"
- #define `SCNiFAST16` "i"
- #define `SCNd32` "ld"
- #define `SCNdLEAST32` "ld"
- #define `SCNdFAST32` "ld"
- #define `SCNi32` "li"
- #define `SCNiLEAST32` "li"
- #define `SCNiFAST32` "li"
- #define `SCNdPtr` `SCNd16`
- #define `SCNiPtr` `SCNi16`
- #define `SCNo8` "hho"
- #define `SCNoLEAST8` "hho"
- #define `SCNoFAST8` "hho"
- #define `SCNu8` "hhu"
- #define `SCNuLEAST8` "hhu"
- #define `SCNuFAST8` "hhu"
- #define `SCNx8` "hxx"
- #define `SCNxLEAST8` "hxx"
- #define `SCNxFAST8` "hxx"
- #define `SCNo16` "o"
- #define `SCNoLEAST16` "o"
- #define `SCNoFAST16` "o"
- #define `SCNu16` "u"
- #define `SCNuLEAST16` "u"
- #define `SCNuFAST16` "u"
- #define `SCNx16` "x"
- #define `SCNxLEAST16` "x"
- #define `SCNxFAST16` "x"
- #define `SCNo32` "lo"
- #define `SCNoLEAST32` "lo"
- #define `SCNoFAST32` "lo"
- #define `SCNu32` "lu"
- #define `SCNuLEAST32` "lu"
- #define `SCNuFAST32` "lu"
- #define `SCNx32` "lx"

- #define `SCNxLEAST32` "lx"
- #define `SCNxFAST32` "lx"
- #define `SCNoPTR` `SCNo16`
- #define `SCNuPTR` `SCNu16`
- #define `SCNxPTR` `SCNx16`

### 21.5.1 Detailed Description

```
#include <inttypes.h>
```

This header file includes the exact-width integer definitions from `<stdint.h>`, and extends them with additional facilities provided by the implementation.

Currently, the extensions include two additional integer types that could hold a "far" pointer (i.e. a code pointer that can address more than 64 KB), as well as standard names for all printf and scanf formatting options that are supported by the `<stdio.h>`: [Standard IO facilities](#). As the library does not support the full range of conversion specifiers from ISO 9899:1999, only those conversions that are actually implemented will be listed here.

The idea behind these conversion macros is that, for each of the types defined by `<stdint.h>`, a macro will be supplied that portably allows formatting an object of that type in `printf()` or `scanf()` operations. Example:

```
#include <inttypes.h>

uint8_t smallval;
int32_t longval;
...
printf("The hexadecimal value of smallval is %" PRIx8
      ", the decimal value of longval is %" PRId32 ".\n",
      smallval, longval);
```

### 21.5.2 Macro Definition Documentation

#### 21.5.2.1 `PRId16` #define `PRId16` "d"

decimal printf format for `int16_t`

#### 21.5.2.2 `PRId32` #define `PRId32` "ld"

decimal printf format for `int32_t`

#### 21.5.2.3 `PRId8` #define `PRId8` "d"

decimal printf format for `int8_t`

#### 21.5.2.4 `PRIdFAST16` #define `PRIdFAST16` "d"

decimal printf format for `int_fast16_t`

#### 21.5.2.5 `PRIdFAST32` #define `PRIdFAST32` "ld"

decimal printf format for `int_fast32_t`

**21.5.2.6 PRIdFAST8** `#define PRIdFAST8 "d"`

decimal printf format for `int_fast8_t`

**21.5.2.7 PRIdLEAST16** `#define PRIdLEAST16 "d"`

decimal printf format for `int_least16_t`

**21.5.2.8 PRIdLEAST32** `#define PRIdLEAST32 "ld"`

decimal printf format for `int_least32_t`

**21.5.2.9 PRIdLEAST8** `#define PRIdLEAST8 "d"`

decimal printf format for `int_least8_t`

**21.5.2.10 PRIdPTR** `#define PRIdPTR PRId16`

decimal printf format for `intptr_t`

**21.5.2.11 PRIi16** `#define PRIi16 "i"`

integer printf format for `int16_t`

**21.5.2.12 PRIi32** `#define PRIi32 "li"`

integer printf format for `int32_t`

**21.5.2.13 PRIi8** `#define PRIi8 "i"`

integer printf format for `int8_t`

**21.5.2.14 PRIiFAST16** `#define PRIiFAST16 "i"`

integer printf format for `int_fast16_t`

**21.5.2.15 PRIiFAST32** `#define PRIiFAST32 "li"`

integer printf format for `int_fast32_t`

**21.5.2.16 PRIiFAST8** `#define PRIiFAST8 "i"`

integer printf format for `int_fast8_t`

**21.5.2.17 PRIiLEAST16** `#define PRIiLEAST16 "i"`

integer printf format for `int_least16_t`

**21.5.2.18 PRIiLEAST32** `#define PRIiLEAST32 "li"`

integer printf format for `int_least32_t`

**21.5.2.19 PRIiLEAST8** `#define PRIiLEAST8 "i"`

integer printf format for `int_least8_t`

**21.5.2.20 PRIiPTR** `#define PRIiPTR PRIi16`

integer printf format for `intptr_t`

**21.5.2.21 PRIo16** `#define PRIo16 "o"`

octal printf format for `uint16_t`

**21.5.2.22 PRIo32** `#define PRIo32 "lo"`

octal printf format for `uint32_t`

**21.5.2.23 PRIo8** `#define PRIo8 "o"`

octal printf format for `uint8_t`

**21.5.2.24 PRIoFAST16** `#define PRIoFAST16 "o"`

octal printf format for `uint_fast16_t`

**21.5.2.25 PRIoFAST32** `#define PRIoFAST32 "lo"`

octal printf format for `uint_fast32_t`

**21.5.2.26 PRIoFAST8** `#define PRIoFAST8 "o"`

octal printf format for `uint_fast8_t`

**21.5.2.27 PRIoLEAST16** `#define PRIoLEAST16 "o"`

octal printf format for `uint_least16_t`

**21.5.2.28 PRIoLEAST32** `#define PRIoLEAST32 "lo"`

octal printf format for `uint_least32_t`

**21.5.2.29 PRIoLEAST8** `#define PRIoLEAST8 "o"`

octal printf format for `uint_least8_t`

**21.5.2.30 PRIoPTR** `#define PRIoPTR PRIo16`

octal printf format for `uintptr_t`

**21.5.2.31 PRIu16** `#define PRIu16 "u"`

decimal printf format for `uint16_t`

**21.5.2.32 PRIu32** `#define PRIu32 "lu"`

decimal printf format for `uint32_t`

**21.5.2.33 PRIu8** `#define PRIu8 "u"`

decimal printf format for `uint8_t`

**21.5.2.34 PRIuFAST16** `#define PRIuFAST16 "u"`

decimal printf format for `uint_fast16_t`

**21.5.2.35 PRIuFAST32** `#define PRIuFAST32 "lu"`

decimal printf format for `uint_fast32_t`

**21.5.2.36 PRIuFAST8** `#define PRIuFAST8 "u"`

decimal printf format for `uint_fast8_t`

**21.5.2.37 PRIuLEAST16** `#define PRIuLEAST16 "u"`

decimal printf format for `uint_least16_t`

**21.5.2.38 PRIuLEAST32** `#define PRIuLEAST32 "lu"`

decimal printf format for `uint_least32_t`

**21.5.2.39 PRIuLEAST8** `#define PRIuLEAST8 "u"`

decimal printf format for `uint_least8_t`

**21.5.2.40 PRIuPTR** `#define PRIuPTR PRIu16`

decimal printf format for `uintptr_t`

**21.5.2.41 PRix16** `#define PRix16 "x"`

hexadecimal printf format for `uint16_t`

**21.5.2.42 PRIX16** `#define PRIX16 "X"`

uppercase hexadecimal printf format for `uint16_t`

**21.5.2.43 PRix32** `#define PRix32 "lx"`

hexadecimal printf format for `uint32_t`

**21.5.2.44 PRIX32** `#define PRIX32 "lX"`

uppercase hexadecimal printf format for `uint32_t`

**21.5.2.45 PRix8** `#define PRix8 "x"`

hexadecimal printf format for `uint8_t`

**21.5.2.46 PRIX8** `#define PRIX8 "X"`

uppercase hexadecimal printf format for `uint8_t`

**21.5.2.47 PRixFAST16** `#define PRixFAST16 "x"`

hexadecimal printf format for `uint_fast16_t`

**21.5.2.48 PRIXFAST16** `#define PRIXFAST16 "X"`

uppercase hexadecimal printf format for `uint_fast16_t`

**21.5.2.49 PRixFAST32** `#define PRixFAST32 "lx"`

hexadecimal printf format for `uint_fast32_t`

**21.5.2.50 PRIFAST32** `#define PRIFAST32 "lX"`

uppercase hexadecimal printf format for `uint_fast32_t`

**21.5.2.51 PRIFAST8** `#define PRIFAST8 "x"`

hexadecimal printf format for `uint_fast8_t`

**21.5.2.52 PRIFAST8** `#define PRIFAST8 "X"`

uppercase hexadecimal printf format for `uint_fast8_t`

**21.5.2.53 PRILEAST16** `#define PRILEAST16 "x"`

hexadecimal printf format for `uint_least16_t`

**21.5.2.54 PRILEAST16** `#define PRILEAST16 "X"`

uppercase hexadecimal printf format for `uint_least16_t`

**21.5.2.55 PRILEAST32** `#define PRILEAST32 "lx"`

hexadecimal printf format for `uint_least32_t`

**21.5.2.56 PRILEAST32** `#define PRILEAST32 "lX"`

uppercase hexadecimal printf format for `uint_least32_t`

**21.5.2.57 PRILEAST8** `#define PRILEAST8 "x"`

hexadecimal printf format for `uint_least8_t`

**21.5.2.58 PRILEAST8** `#define PRILEAST8 "X"`

uppercase hexadecimal printf format for `uint_least8_t`

**21.5.2.59 PRIPTR** `#define PRIPTR PRIx16`

hexadecimal printf format for `uintptr_t`

**21.5.2.60 PRIPTR** `#define PRIPTR PRIX16`

uppercase hexadecimal printf format for `uintptr_t`

**21.5.2.61 SCNd16** `#define SCNd16 "d"`

decimal scanf format for `int16_t`

**21.5.2.62 SCNd32** `#define SCNd32 "ld"`

decimal scanf format for `int32_t`

**21.5.2.63 SCNd8** `#define SCNd8 "hhd"`

decimal scanf format for `int8_t`

**21.5.2.64 SCNdFAST16** `#define SCNdFAST16 "d"`

decimal scanf format for `int_fast16_t`

**21.5.2.65 SCNdFAST32** `#define SCNdFAST32 "ld"`

decimal scanf format for `int_fast32_t`

**21.5.2.66 SCNdFAST8** `#define SCNdFAST8 "hhd"`

decimal scanf format for `int_fast8_t`

**21.5.2.67 SCNdLEAST16** `#define SCNdLEAST16 "d"`

decimal scanf format for `int_least16_t`

**21.5.2.68 SCNdLEAST32** `#define SCNdLEAST32 "ld"`

decimal scanf format for `int_least32_t`

**21.5.2.69 SCNdLEAST8** `#define SCNdLEAST8 "hhd"`

decimal scanf format for `int_least8_t`

**21.5.2.70 SCNdPTR** `#define SCNdPTR SCNd16`

decimal scanf format for `intptr_t`

**21.5.2.71 SCNi16** `#define SCNi16 "i"`

generic-integer scanf format for `int16_t`



**21.5.2.72 SCNi32** `#define SCNi32 "li"`

generic-integer scanf format for `int32_t`

**21.5.2.73 SCNi8** `#define SCNi8 "hhi"`

generic-integer scanf format for `int8_t`

**21.5.2.74 SCNiFAST16** `#define SCNiFAST16 "i"`

generic-integer scanf format for `int_fast16_t`

**21.5.2.75 SCNiFAST32** `#define SCNiFAST32 "li"`

generic-integer scanf format for `int_fast32_t`

**21.5.2.76 SCNiFAST8** `#define SCNiFAST8 "hhi"`

generic-integer scanf format for `int_fast8_t`

**21.5.2.77 SCNiLEAST16** `#define SCNiLEAST16 "i"`

generic-integer scanf format for `int_least16_t`

**21.5.2.78 SCNiLEAST32** `#define SCNiLEAST32 "li"`

generic-integer scanf format for `int_least32_t`

**21.5.2.79 SCNiLEAST8** `#define SCNiLEAST8 "hhi"`

generic-integer scanf format for `int_least8_t`

**21.5.2.80 SCNiPTR** `#define SCNiPTR SCNi16`

generic-integer scanf format for `intptr_t`

**21.5.2.81 SCNo16** `#define SCNo16 "o"`

octal scanf format for `uint16_t`

**21.5.2.82 SCNo32** `#define SCNo32 "lo"`

octal scanf format for `uint32_t`

**21.5.2.83 SCNo8** `#define SCNo8 "hho"`

octal scanf format for `uint8_t`

**21.5.2.84 SCNoFAST16** `#define SCNoFAST16 "o"`

octal scanf format for `uint_fast16_t`

**21.5.2.85 SCNoFAST32** `#define SCNoFAST32 "lo"`

octal scanf format for `uint_fast32_t`

**21.5.2.86 SCNoFAST8** `#define SCNoFAST8 "hho"`

octal scanf format for `uint_fast8_t`

**21.5.2.87 SCNoLEAST16** `#define SCNoLEAST16 "o"`

octal scanf format for `uint_least16_t`

**21.5.2.88 SCNoLEAST32** `#define SCNoLEAST32 "lo"`

octal scanf format for `uint_least32_t`

**21.5.2.89 SCNoLEAST8** `#define SCNoLEAST8 "hho"`

octal scanf format for `uint_least8_t`

**21.5.2.90 SCNoPTR** `#define SCNoPTR SCNo16`

octal scanf format for `uintptr_t`

**21.5.2.91 SCNu16** `#define SCNu16 "u"`

decimal scanf format for `uint16_t`

**21.5.2.92 SCNu32** `#define SCNu32 "lu"`

decimal scanf format for `uint32_t`

**21.5.2.93 SCNu8** `#define SCNu8 "hhu"`

decimal scanf format for `uint8_t`

**21.5.2.94 SCNuFAST16** `#define SCNuFAST16 "u"`

decimal scanf format for `uint_fast16_t`

**21.5.2.95 SCNuFAST32** `#define SCNuFAST32 "lu"`

decimal scanf format for `uint_fast32_t`

**21.5.2.96 SCNuFAST8** `#define SCNuFAST8 "hhu"`

decimal scanf format for `uint_fast8_t`

**21.5.2.97 SCNuLEAST16** `#define SCNuLEAST16 "u"`

decimal scanf format for `uint_least16_t`

**21.5.2.98 SCNuLEAST32** `#define SCNuLEAST32 "lu"`

decimal scanf format for `uint_least32_t`

**21.5.2.99 SCNuLEAST8** `#define SCNuLEAST8 "hhu"`

decimal scanf format for `uint_least8_t`

**21.5.2.100 SCNuPTR** `#define SCNuPTR SCNu16`

decimal scanf format for `uintptr_t`

**21.5.2.101 SCNx16** `#define SCNx16 "x"`

hexadecimal scanf format for `uint16_t`

**21.5.2.102 SCNx32** `#define SCNx32 "lx"`

hexadecimal scanf format for `uint32_t`

**21.5.2.103 SCNx8** `#define SCNx8 "hnx"`

hexadecimal scanf format for `uint8_t`

**21.5.2.104 SCNxFAST16** `#define SCNxFAST16 "x"`

hexadecimal scanf format for `uint_fast16_t`

**21.5.2.105 SCNxFAST32** `#define SCNxFAST32 "lx"`

hexadecimal scanf format for `uint_fast32_t`

**21.5.2.106 SCNxFAST8** `#define SCNxFAST8 "hhx"`

hexadecimal scanf format for `uint_fast8_t`

**21.5.2.107 SCNxLEAST16** `#define SCNxLEAST16 "x"`

hexadecimal scanf format for `uint_least16_t`

**21.5.2.108 SCNxLEAST32** `#define SCNxLEAST32 "lx"`

hexadecimal scanf format for `uint_least32_t`

**21.5.2.109 SCNxLEAST8** `#define SCNxLEAST8 "hhx"`

hexadecimal scanf format for `uint_least8_t`

**21.5.2.110 SCNxPTR** `#define SCNxPTR SCNx16`

hexadecimal scanf format for `uintptr_t`

### 21.5.3 Typedef Documentation

**21.5.3.1 int\_farptr\_t** `typedef int32_t int_farptr_t`

signed integer type that can hold a pointer > 64 KiB

**21.5.3.2 uint\_farptr\_t** `typedef uint32_t uint_farptr_t`

unsigned integer type that can hold a pointer > 64 KiB, see also [pgm\\_get\\_far\\_address\(\)](#)

## 21.6 <math.h>: Mathematics

### Macros

- #define `M_E` 2.7182818284590452354
- #define `M_LOG2E` 1.4426950408889634074
- #define `M_LOG10E` 0.43429448190325182765
- #define `M_LN2` 0.69314718055994530942
- #define `M_LN10` 2.30258509299404568402
- #define `M_PI` 3.14159265358979323846
- #define `M_PI_2` 1.57079632679489661923
- #define `M_PI_4` 0.78539816339744830962
- #define `M_1_PI` 0.31830988618379067154
- #define `M_2_PI` 0.63661977236758134308
- #define `M_2_SQRTPI` 1.12837916709551257390
- #define `M_SQRT2` 1.41421356237309504880
- #define `M_SQRT1_2` 0.70710678118654752440
- #define `NAN` `__builtin_nan("")`
- #define `nanf`(`__tag`) `__builtin_nanf(__tag)`
- #define `nan`(`__tag`) `__builtin_nan(__tag)`
- #define `nanl`(`__tag`) `__builtin_nanl(__tag)`
- #define `INFINITY` `__builtin_inf()`
- #define `HUGE_VALF` `__builtin_huge_valf()`
- #define `HUGE_VAL` `__builtin_huge_val()`
- #define `HUGE_VALL` `__builtin_huge_vall()`

### Functions

- float `cosf` (float x)
- double `cos` (double x)
- long double `cosl` (long double x)
- float `sinf` (float x)
- double `sin` (double x)
- long double `sinl` (long double x)
- float `tanf` (float x)
- double `tan` (double x)
- long double `tanl` (long double x)
- static float `fabsf` (float \_\_x)
- static double `fabs` (double \_\_x)
- static long double `fabsl` (long double \_\_x)
- float `fmodf` (float x, float y)
- double `fmod` (double x, double y)
- long double `fmodl` (long double x, long double y)
- float `modff` (float x, float \*iptr)
- double `modf` (double x, double \*iptr)
- long double `modfl` (long double x, long double \*iptr)
- float `sqrtf` (float x)
- double `sqrt` (double x)
- long double `sqrtl` (long double x)
- float `cbrtf` (float x)
- double `cbrt` (double x)
- long double `cbrtl` (long double x)
- float `hypotf` (float x, float y)

- double `hypot` (double x, double y)
- long double `hypotl` (long double x, long double y)
- float `floorf` (float x)
- double `floor` (double x)
- long double `floorl` (long double x)
- float `ceilf` (float x)
- double `ceil` (double x)
- long double `ceilf` (long double x)
- float `frexpf` (float x, int \*pexp)
- double `frexp` (double x, int \*pexp)
- long double `frexpl` (long double x, int \*pexp)
- float `ldexpf` (float x, int iexp)
- double `ldexp` (double x, int iexp)
- long double `ldexpl` (long double x, int iexp)
- float `expf` (float x)
- double `exp` (double x)
- long double `expl` (long double x)
- float `coshf` (float x)
- double `cosh` (double x)
- long double `coshl` (long double x)
- float `sinhf` (float x)
- double `sinh` (double x)
- long double `sinhl` (long double x)
- float `tanhf` (float x)
- double `tanh` (double x)
- long double `tanhf` (long double x)
- float `acosf` (float x)
- double `acos` (double x)
- long double `acosl` (long double x)
- float `asinf` (float x)
- double `asin` (double x)
- long double `asinl` (long double x)
- float `atanf` (float x)
- double `atan` (double x)
- long double `atanf` (long double x)
- float `atan2f` (float y, float x)
- double `atan2` (double y, double x)
- long double `atan2l` (long double y, long double x)
- float `logf` (float x)
- double `log` (double x)
- long double `logl` (long double x)
- float `log10f` (float x)
- double `log10` (double x)
- long double `log10l` (long double x)
- float `powf` (float x, float y)
- double `pow` (double x, double y)
- long double `powf` (long double x, long double y)
- int `isnanf` (float x)
- int `isnan` (double x)
- int `isnanl` (long double x)
- int `isinf` (float x)
- int `isinf` (double x)
- int `isinfl` (long double x)
- static int `isfinitef` (float \_\_x)
- static int `isfinite` (double \_\_x)

- static int `isfinitel` (long double `__x`)
- static float `copysignf` (float `__x`, float `__y`)
- static double `copysign` (double `__x`, double `__y`)
- static long double `copysignl` (long double `__x`, long double `__y`)
- int `signbitf` (float `x`)
- int `signbit` (double `x`)
- int `signbitl` (long double `x`)
- float `fdimf` (float `x`, float `y`)
- double `fdim` (double `x`, double `y`)
- long double `fdiml` (long double `x`, long double `y`)
- float `fmaf` (float `x`, float `y`, float `z`)
- double `fma` (double `x`, double `y`, double `z`)
- long double `fmal` (long double `x`, long double `y`, long double `z`)
- float `fmaxf` (float `x`, float `y`)
- double `fmax` (double `x`, double `y`)
- long double `fmaxl` (long double `x`, long double `y`)
- float `fminf` (float `x`, float `y`)
- double `fmin` (double `x`, double `y`)
- long double `fminl` (long double `x`, long double `y`)
- float `truncf` (float `x`)
- double `trunc` (double `x`)
- long double `truncl` (long double `x`)
- float `roundf` (float `x`)
- double `round` (double `x`)
- long double `roundl` (long double `x`)
- long `lroundf` (float `x`)
- long `lround` (double `x`)
- long `lroundl` (long double `x`)
- long `lrintf` (float `x`)
- long `lrint` (double `x`)
- long `lrintl` (long double `x`)

### Non-Standard Math Functions

- float `squaref` (float `x`)
- double `square` (double `x`)
- long double `squarel` (long double `x`)

#### 21.6.1 Detailed Description

```
#include <math.h>
```

This header file declares basic mathematics constants and functions.

#### Notes:

- Math functions do not raise exceptions and do not change the `errno` variable. Therefore the majority of them are declared with `const` attribute, for better optimization by GCC.
- 64-bit floating-point arithmetic is only available in `avr-gcc v10` and up. The size of the `double` and `long double` type can be selected at compile-time with options like `-mdouble=64` and `-mlong-double=32`. Whether such options are available, and their default values, depend on how the compiler has been configured.
- The implementation of 64-bit floating-point arithmetic has some shortcomings and limitations, see the [avr-gcc Wiki](#) for details.
- In order to access the `float` functions, in `avr-gcc v4.6` and older it is usually also required to link with `-lm`. In `avr-gcc v4.7` and up, `-lm` is added automatically to all linker invocations.

## 21.6.2 Macro Definition Documentation

**21.6.2.1 HUGE\_VAL** #define HUGE\_VAL \_\_builtin\_huge\_val()

double infinity constant.

**21.6.2.2 HUGE\_VALF** #define HUGE\_VALF \_\_builtin\_huge\_valf()

float infinity constant.

**21.6.2.3 HUGE\_VALL** #define HUGE\_VALL \_\_builtin\_huge\_vall()

long double infinity constant.

**21.6.2.4 INFINITY** #define INFINITY \_\_builtin\_inf()

double infinity constant.

**21.6.2.5 M\_1\_PI** #define M\_1\_PI 0.31830988618379067154

The constant  $1/\pi$ .

**21.6.2.6 M\_2\_PI** #define M\_2\_PI 0.63661977236758134308

The constant  $2/\pi$ .

**21.6.2.7 M\_2\_SQRTPI** #define M\_2\_SQRTPI 1.12837916709551257390

The constant  $2/\sqrt{\pi}$ .

**21.6.2.8 M\_E** #define M\_E 2.7182818284590452354

The constant Euler's number  $e$ .

**21.6.2.9 M\_LN10** #define M\_LN10 2.30258509299404568402

The constant natural logarithm of 10.

**21.6.2.10 M\_LN2** #define M\_LN2 0.69314718055994530942

The constant natural logarithm of 2.



**21.6.2.11 M\_LOG10E** `#define M_LOG10E 0.43429448190325182765`

The constant logarithm of Euler's number  $e$  to base 10.

**21.6.2.12 M\_LOG2E** `#define M_LOG2E 1.4426950408889634074`

The constant logarithm of Euler's number  $e$  to base 2.

**21.6.2.13 M\_PI** `#define M_PI 3.14159265358979323846`

The constant  $\pi$ .

**21.6.2.14 M\_PI\_2** `#define M_PI_2 1.57079632679489661923`

The constant  $\pi/2$ .

**21.6.2.15 M\_PI\_4** `#define M_PI_4 0.78539816339744830962`

The constant  $\pi/4$ .

**21.6.2.16 M\_SQRT1\_2** `#define M_SQRT1_2 0.70710678118654752440`

The constant  $1/\sqrt{2}$ .

**21.6.2.17 M\_SQRT2** `#define M_SQRT2 1.41421356237309504880`

The square root of 2.

**21.6.2.18 NAN** `#define NAN __builtin_nan("")`

The double representation of a constant quiet NaN.

**21.6.2.19 nan** `#define nan(  
 __tag ) __builtin_nan(__tag)`

The double representation of a constant quiet NaN. `__tag` is a string constant like "" or "123".

**21.6.2.20 nanf** `#define nanf(  
 __tagp ) __builtin_nanf(__tag)`

The float representation of a constant quiet NaN. `__tag` is a string constant like "" or "123".

**21.6.2.21 nanl** `#define nanl(  
 __tag ) __builtin_nanl(__tag)`

The long double representation of a constant quiet NaN. `__tag` is a string constant like "" or "123".

### 21.6.3 Function Documentation

**21.6.3.1 `acos()`** `double acos (`  
`double x )`

The `acos()` function computes the principal value of the arc cosine of  $x$ . The returned value is in the range  $[0, \pi]$  radians or NaN.

**21.6.3.2 `acosf()`** `float acosf (`  
`float x )`

The `acosf()` function computes the principal value of the arc cosine of  $x$ . The returned value is in the range  $[0, \pi]$  radians. A domain error occurs for arguments not in the range  $[-1, +1]$ .

**21.6.3.3 `acosl()`** `long double acosl (`  
`long double x )`

The `acosl()` function computes the principal value of the arc cosine of  $x$ . The returned value is in the range  $[0, \pi]$  radians or NaN.

**21.6.3.4 `asin()`** `double asin (`  
`double x )`

The `asin()` function computes the principal value of the arc sine of  $x$ . The returned value is in the range  $[-\pi/2, \pi/2]$  radians or NaN.

**21.6.3.5 `asinf()`** `float asinf (`  
`float x )`

The `asinf()` function computes the principal value of the arc sine of  $x$ . The returned value is in the range  $[-\pi/2, \pi/2]$  radians. A domain error occurs for arguments not in the range  $[-1, +1]$ .

**21.6.3.6 `asinl()`** `long double asinl (`  
`long double x )`

The `asinl()` function computes the principal value of the arc sine of  $x$ . The returned value is in the range  $[-\pi/2, \pi/2]$  radians or NaN.

**21.6.3.7 `atan()`** `double atan (`  
`double x )`

The `atan()` function computes the principal value of the arc tangent of  $x$ . The returned value is in the range  $[-\pi/2, \pi/2]$  radians.

**21.6.3.8 atan2()** `double atan2 (`  
    `double y,`  
    `double x )`

The `atan2()` function computes the principal value of the arc tangent of  $y/x$ , using the signs of both arguments to determine the quadrant of the return value. The returned value is in the range  $[-\pi, +\pi]$  radians.

**21.6.3.9 atan2f()** `float atan2f (`  
    `float y,`  
    `float x )`

The `atan2f()` function computes the principal value of the arc tangent of  $y/x$ , using the signs of both arguments to determine the quadrant of the return value. The returned value is in the range  $[-\pi, +\pi]$  radians.

**21.6.3.10 atan2l()** `long double atan2l (`  
    `long double y,`  
    `long double x )`

The `atan2l()` function computes the principal value of the arc tangent of  $y/x$ , using the signs of both arguments to determine the quadrant of the return value. The returned value is in the range  $[-\pi, +\pi]$  radians.

**21.6.3.11 atanf()** `float atanf (`  
    `float x )`

The `atanf()` function computes the principal value of the arc tangent of  $x$ . The returned value is in the range  $[-\pi/2, \pi/2]$  radians.

**21.6.3.12 atanl()** `long double atanl (`  
    `long double x )`

The `atanl()` function computes the principal value of the arc tangent of  $x$ . The returned value is in the range  $[-\pi/2, \pi/2]$  radians.

**21.6.3.13 cbrt()** `double cbrt (`  
    `double x )`

The `cbrt()` function returns the cube root of  $x$ .

**21.6.3.14 cbrtf()** `float cbrtf (`  
    `float x )`

The `cbrtf()` function returns the cube root of  $x$ .

**21.6.3.15 cbrtl()** `long double cbrtl (`  
    `long double x )`

The `cbrtl()` function returns the cube root of  $x$ .

**21.6.3.16** `ceil()` `double ceil (`  
`double x )`

The `ceil()` function returns the smallest integral value greater than or equal to  $x$ , expressed as a floating-point number.

**21.6.3.17** `ceilf()` `float ceilf (`  
`float x )`

The `ceilf()` function returns the smallest integral value greater than or equal to  $x$ , expressed as a floating-point number.

**21.6.3.18** `ceilll()` `long double ceilll (`  
`long double x )`

The `ceilll()` function returns the smallest integral value greater than or equal to  $x$ , expressed as a floating-point number.

**21.6.3.19** `copysign()` `static double copysign (`  
`double __x,`  
`double __y ) [inline], [static]`

The `copysign()` function returns `__x` but with the sign of `__y`. They work even if `__x` or `__y` are NaN or zero.

**21.6.3.20** `copysignf()` `static float copysignf (`  
`float __x,`  
`float __y ) [inline], [static]`

The `copysignf()` function returns `__x` but with the sign of `__y`. They work even if `__x` or `__y` are NaN or zero.

**21.6.3.21** `copysignl()` `static long double copysignl (`  
`long double __x,`  
`long double __y ) [inline], [static]`

The `copysignl()` function returns `__x` but with the sign of `__y`. They work even if `__x` or `__y` are NaN or zero.

**21.6.3.22** `cos()` `double cos (`  
`double x )`

The `cos()` function returns the cosine of  $x$ , measured in radians.

**21.6.3.23** `cosf()` `float cosf (`  
`float x )`

The `cosf()` function returns the cosine of  $x$ , measured in radians.

**21.6.3.24** `cosh()` `double cosh (`  
`double x )`

The `cosh()` function returns the hyperbolic cosine of  $x$ .

**21.6.3.25 coshf()** `float coshf (`  
`float x )`

The `coshf()` function returns the hyperbolic cosine of  $x$ .

**21.6.3.26 coshl()** `long double coshl (`  
`long double x )`

The `coshl()` function returns the hyperbolic cosine of  $x$ .

**21.6.3.27 cosl()** `long double cosl (`  
`long double x )`

The `cosl()` function returns the cosine of  $x$ , measured in radians.

**21.6.3.28 exp()** `double exp (`  
`double x )`

The `exp()` function returns the exponential value of  $x$ .

**21.6.3.29 expf()** `float expf (`  
`float x )`

The `expf()` function returns the exponential value of  $x$ .

**21.6.3.30 expl()** `long double expl (`  
`long double x )`

The `expl()` function returns the exponential value of  $x$ .

**21.6.3.31 fabs()** `static double fabs (`  
`double __x ) [inline], [static]`

The `fabs()` function computes the absolute value of a floating-point number  $x$ .

**21.6.3.32 fabsf()** `static float fabsf (`  
`float __x ) [inline], [static]`

The `fabsf()` function computes the absolute value of a floating-point number  $x$ .

**21.6.3.33 fabsl()** `static long double fabsl (`  
`long double __x ) [inline], [static]`

The `fabsl()` function computes the absolute value of a floating-point number  $x$ .

**21.6.3.34 fdim()** `double fdim (`  
    `double x,`  
    `double y )`

The `fdim()` function returns  $\max(x - y, 0)$ . If  $x$  or  $y$  or both are NaN, NaN is returned.

**21.6.3.35 fdimf()** `float fdimf (`  
    `float x,`  
    `float y )`

The `fdimf()` function returns  $\max(x - y, 0)$ . If  $x$  or  $y$  or both are NaN, NaN is returned.

**21.6.3.36 fdiml()** `long double fdiml (`  
    `long double x,`  
    `long double y )`

The `fdiml()` function returns  $\max(x - y, 0)$ . If  $x$  or  $y$  or both are NaN, NaN is returned.

**21.6.3.37 floor()** `double floor (`  
    `double x )`

The `floor()` function returns the largest integral value less than or equal to  $x$ , expressed as a floating-point number.

**21.6.3.38 floorf()** `float floorf (`  
    `float x )`

The `floorf()` function returns the largest integral value less than or equal to  $x$ , expressed as a floating-point number.

**21.6.3.39 floorl()** `long double floorl (`  
    `long double x )`

The `floorl()` function returns the largest integral value less than or equal to  $x$ , expressed as a floating-point number.

**21.6.3.40 fma()** `double fma (`  
    `double x,`  
    `double y,`  
    `double z )`

The `fma()` function performs floating-point multiply-add. This is the operation  $(x * y) + z$ , but the intermediate result is not rounded to the destination type. This can sometimes improve the precision of a calculation.

**21.6.3.41 fmaf()** `float fmaf (`  
    `float x,`  
    `float y,`  
    `float z )`

The `fmaf()` function performs floating-point multiply-add. This is the operation  $(x * y) + z$ , but the intermediate result is not rounded to the destination type. This can sometimes improve the precision of a calculation.

**21.6.3.42 fmal()** long double fmal (  
    long double *x*,  
    long double *y*,  
    long double *z* )

The `fmal()` function performs floating-point multiply-add. This is the operation  $(x * y) + z$ , but the intermediate result is not rounded to the destination type. This can sometimes improve the precision of a calculation.

**21.6.3.43 fmax()** double fmax (  
    double *x*,  
    double *y* )

The `fmax()` function returns the greater of the two values *x* and *y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

**21.6.3.44 fmaxf()** float fmaxf (  
    float *x*,  
    float *y* )

The `fmaxf()` function returns the greater of the two values *x* and *y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

**21.6.3.45 fmaxl()** long double fmaxl (  
    long double *x*,  
    long double *y* )

The `fmaxl()` function returns the greater of the two values *x* and *y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

**21.6.3.46 fmin()** double fmin (  
    double *x*,  
    double *y* )

The `fmin()` function returns the lesser of the two values *x* and *y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

**21.6.3.47 fminf()** float fminf (  
    float *x*,  
    float *y* )

The `fminf()` function returns the lesser of the two values *x* and *y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

**21.6.3.48 fminl()** long double fminl (  
    long double *x*,  
    long double *y* )

The `fminl()` function returns the lesser of the two values *x* and *y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

```
21.6.3.49 fmod() double fmod (
    double x,
    double y )
```

The function `fmod()` returns the floating-point remainder of  $x/y$ .

```
21.6.3.50 fmodf() float fmodf (
    float x,
    float y )
```

The function `fmodf()` returns the floating-point remainder of  $x/y$ .

```
21.6.3.51 fmodl() long double fmodl (
    long double x,
    long double y )
```

The function `fmodl()` returns the floating-point remainder of  $x/y$ .

```
21.6.3.52 frexp() double frexp (
    double x,
    int * pexp )
```

The `frexp()` function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integer in the `int` object pointed to by `pexp`.

If  $x$  is a normal float point number, the `frexp()` function returns the value  $v$ , such that  $v$  has a magnitude in the interval  $[1/2, 1)$  or zero, and  $x$  equals  $v$  times 2 raised to the power `pexp`. If  $x$  is zero, both parts of the result are zero. If  $x$  is not a finite number, the `frexp()` returns  $x$  as is and stores 0 by `pexp`.

```
21.6.3.53 frexpf() float frexpf (
    float x,
    int * pexp )
```

The `frexpf()` function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integer in the `int` object pointed to by `pexp`.

If  $x$  is a normal float point number, the `frexpf()` function returns the value  $v$ , such that  $v$  has a magnitude in the interval  $[1/2, 1)$  or zero, and  $x$  equals  $v$  times 2 raised to the power `pexp`. If  $x$  is zero, both parts of the result are zero. If  $x$  is not a finite number, the `frexpf()` returns  $x$  as is and stores 0 by `pexp`.

#### Note

This implementation permits a zero pointer as a directive to skip a storing the exponent.

```
21.6.3.54 frexpl() long double frexpl (
    long double x,
    int * pexp )
```

The `frexpl()` function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integer in the `int` object pointed to by `pexp`.

If  $x$  is a normal float point number, the `frexpl()` function returns the value  $v$ , such that  $v$  has a magnitude in the interval  $[1/2, 1)$  or zero, and  $x$  equals  $v$  times 2 raised to the power `pexp`. If  $x$  is zero, both parts of the result are zero. If  $x$  is not a finite number, the `frexpl()` returns  $x$  as is and stores 0 by `pexp`.



**21.6.3.55 hypot()** `double hypot (`  
    `double x,`  
    `double y )`

The `hypot()` function returns  $\sqrt{x*x + y*y}$ . This is the length of the hypotenuse of a right triangle with sides of length  $x$  and  $y$ , or the distance of the point  $(x, y)$  from the origin. Using this function instead of the direct formula is wise, since the error is much smaller. No underflow with small  $x$  and  $y$ . No overflow if result is in range.

**21.6.3.56 hypotf()** `float hypotf (`  
    `float x,`  
    `float y )`

The `hypotf()` function returns  $\sqrt{x*x + y*y}$ . This is the length of the hypotenuse of a right triangle with sides of length  $x$  and  $y$ , or the distance of the point  $(x, y)$  from the origin. Using this function instead of the direct formula is wise, since the error is much smaller. No underflow with small  $x$  and  $y$ . No overflow if result is in range.

**21.6.3.57 hypotl()** `long double hypotl (`  
    `long double x,`  
    `long double y )`

The `hypotl()` function returns  $\sqrt{x*x + y*y}$ . This is the length of the hypotenuse of a right triangle with sides of length  $x$  and  $y$ , or the distance of the point  $(x, y)$  from the origin. Using this function instead of the direct formula is wise, since the error is much smaller. No underflow with small  $x$  and  $y$ . No overflow if result is in range.

**21.6.3.58 isfinite()** `static int isfinite (`  
    `double __x ) [inline], [static]`

The `isfinite()` function returns a nonzero value if `__x` is finite: not plus or minus infinity, and not NaN.

**21.6.3.59 isfinitef()** `static int isfinitef (`  
    `float __x ) [inline], [static]`

The `isfinitef()` function returns a nonzero value if `__x` is finite: not plus or minus infinity, and not NaN.

**21.6.3.60 isfinitel()** `static int isfinitel (`  
    `long double __x ) [inline], [static]`

The `isfinite()` function returns a nonzero value if `__x` is finite: not plus or minus infinity, and not NaN.

**21.6.3.61 isinf()** `int isinf (`  
    `double x )`

The function `isinf()` returns 1 if the argument  $x$  is positive infinity, -1 if  $x$  is negative infinity, and 0 otherwise.

**21.6.3.62 isinff()** `int isinff (`  
    `float x )`

The function `isinff()` returns 1 if the argument  $x$  is positive infinity, -1 if  $x$  is negative infinity, and 0 otherwise.

**21.6.3.63** `isinfl()` `int isinfl (`  
`long double x )`

The function `isinfl()` returns 1 if the argument `x` is positive infinity, -1 if `x` is negative infinity, and 0 otherwise.

**21.6.3.64** `isnan()` `int isnan (`  
`double x )`

The function `isnan()` returns 1 if the argument `x` represents a "not-a-number" (NaN) object, otherwise 0.

**21.6.3.65** `isnanf()` `int isnanf (`  
`float x )`

The function `isnanf()` returns 1 if the argument `x` represents a "not-a-number" (NaN) object, otherwise 0.

**21.6.3.66** `isnanl()` `int isnanl (`  
`long double x )`

The function `isnanl()` returns 1 if the argument `x` represents a "not-a-number" (NaN) object, otherwise 0.

**21.6.3.67** `ldexp()` `double ldexp (`  
`double x,`  
`int iexp )`

The `ldexp()` function multiplies a floating-point number by an integral power of 2. It returns the value of `x` times 2 raised to the power `iexp`.

**21.6.3.68** `ldexpf()` `float ldexpf (`  
`float x,`  
`int iexp )`

The `ldexpf()` function multiplies a floating-point number by an integral power of 2. It returns the value of `x` times 2 raised to the power `iexp`.

**21.6.3.69** `ldexpl()` `long double ldexpl (`  
`long double x,`  
`int iexp )`

The `ldexpl()` function multiplies a floating-point number by an integral power of 2. It returns the value of `x` times 2 raised to the power `iexp`.

**21.6.3.70** `log()` `double log (`  
`double x )`

The `log()` function returns the natural logarithm of argument `x`.

**21.6.3.71** `log10()` `double log10 (`  
`double x )`

The `log10()` function returns the logarithm of argument `x` to base 10.

**21.6.3.72 log10f()** float log10f (  
float x )

The [log10f\(\)](#) function returns the logarithm of argument *x* to base 10.

**21.6.3.73 log10l()** long double log10l (  
long double x )

The [log10l\(\)](#) function returns the logarithm of argument *x* to base 10.

**21.6.3.74 logf()** float logf (  
float x )

The [logf\(\)](#) function returns the natural logarithm of argument *x*.

**21.6.3.75 logl()** long double logl (  
long double x )

The [logl\(\)](#) function returns the natural logarithm of argument *x*.

**21.6.3.76 lrint()** long lrint (  
double x )

The [lrint\(\)](#) function rounds *x* to the nearest integer, rounding the halfway cases to the even integer direction. (That is both 1.5 and 2.5 values are rounded to 2). This function is similar to [rint\(\)](#) function, but it differs in type of return value and in that an overflow is possible.

#### Returns

The rounded long integer value. If *x* is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

**21.6.3.77 lrintf()** long lrintf (  
float x )

The [lrintf\(\)](#) function rounds *x* to the nearest integer, rounding the halfway cases to the even integer direction. (That is both 1.5 and 2.5 values are rounded to 2). This function is similar to [rintf\(\)](#) function, but it differs in type of return value and in that an overflow is possible.

#### Returns

The rounded long integer value. If *x* is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

**21.6.3.78 lrintl()** `long lrintl (`  
`long double x )`

The `lrintl()` function rounds  $x$  to the nearest integer, rounding the halfway cases to the even integer direction. (That is both 1.5 and 2.5 values are rounded to 2). This function is similar to `rintl()` function, but it differs in type of return value and in that an overflow is possible.

#### Returns

The rounded long integer value. If  $x$  is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

**21.6.3.79 lround()** `long lround (`  
`double x )`

The `lround()` function rounds  $x$  to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). This function is similar to `round()` function, but it differs in type of return value and in that an overflow is possible.

#### Returns

The rounded long integer value. If  $x$  is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

**21.6.3.80 lroundf()** `long lroundf (`  
`float x )`

The `lroundf()` function rounds  $x$  to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). This function is similar to `round()` function, but it differs in type of return value and in that an overflow is possible.

#### Returns

The rounded long integer value. If  $x$  is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

**21.6.3.81 lroundl()** `long lroundl (`  
`long double x )`

The `lroundl()` function rounds  $x$  to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). This function is similar to `round()` function, but it differs in type of return value and in that an overflow is possible.

#### Returns

The rounded long integer value. If  $x$  is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

**21.6.3.82 modf()** `double modf (`  
    `double x,`  
    `double * iptr )`

The `modf()` function breaks the argument `x` into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a `double` in the object pointed to by `iptr`.

The `modf()` function returns the signed fractional part of `x`.

**21.6.3.83 modff()** `float modff (`  
    `float x,`  
    `float * iptr )`

The `modff()` function breaks the argument `x` into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a `float` in the object pointed to by `iptr`.

The `modff()` function returns the signed fractional part of `x`.

#### Note

This implementation skips writing by zero pointer. However, the GCC 4.3 can replace this function with inline code that does not permit to use NULL address for the avoiding of storing.

**21.6.3.84 modfl()** `long double modfl (`  
    `long double x,`  
    `long double * iptr )`

The `modfl()` function breaks the argument `x` into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a `long double` in the object pointed to by `iptr`.

The `modfl()` function returns the signed fractional part of `x`.

**21.6.3.85 pow()** `double pow (`  
    `double x,`  
    `double y )`

The function `pow()` returns the value of `x` to the exponent `y`.

Notice that for integer exponents, there is the more efficient `double __builtin_powi(double x, int y)`.

**21.6.3.86 powf()** `float powf (`  
    `float x,`  
    `float y )`

The function `powf()` returns the value of `x` to the exponent `y`.

Notice that for integer exponents, there is the more efficient `float __builtin_powif(float x, int y)`.

**21.6.3.87 powl()** long double powl (  
    long double x,  
    long double y )

The function `powl()` returns the value of  $x$  to the exponent  $y$ .

Notice that for integer exponents, there is the more efficient long double `__builtin_powil(long double x, int y)`.

**21.6.3.88 round()** double round (  
    double x )

The `round()` function rounds  $x$  to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). Overflow is impossible.

#### Returns

The rounded value. If  $x$  is an integral or infinite,  $x$  itself is returned. If  $x$  is NaN, then NaN is returned.

**21.6.3.89 roundf()** float roundf (  
    float x )

The `roundf()` function rounds  $x$  to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). Overflow is impossible.

#### Returns

The rounded value. If  $x$  is an integral or infinite,  $x$  itself is returned. If  $x$  is NaN, then NaN is returned.

**21.6.3.90 roundl()** long double roundl (  
    long double x )

The `roundl()` function rounds  $x$  to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). Overflow is impossible.

#### Returns

The rounded value. If  $x$  is an integral or infinite,  $x$  itself is returned. If  $x$  is NaN, then NaN is returned.

**21.6.3.91 signbit()** int signbit (  
    double x )

The `signbit()` function returns a nonzero value if the value of  $x$  has its sign bit set. This is not the same as `x < 0.0`, because IEEE 754 floating point allows zero to be signed. The comparison `-0.0 < 0.0` is false, but `signbit(-0.0)` will return a nonzero value.

**21.6.3.92** `signbitf()` `int signbitf (`  
`float x )`

The `signbitf()` function returns a nonzero value if the value of `x` has its sign bit set. This is not the same as ``x < 0.0'`, because IEEE 754 floating point allows zero to be signed. The comparison `'-0.0 < 0.0'` is false, but ``signbit (-0.0)'` will return a nonzero value.

**21.6.3.93** `signbitl()` `int signbitl (`  
`long double x )`

The `signbitl()` function returns a nonzero value if the value of `x` has its sign bit set. This is not the same as ``x < 0.0'`, because IEEE 754 floating point allows zero to be signed. The comparison `'-0.0 < 0.0'` is false, but ``signbit (-0.0)'` will return a nonzero value.

**21.6.3.94** `sin()` `double sin (`  
`double x )`

The `sin()` function returns the sine of `x`, measured in radians.

**21.6.3.95** `sinf()` `float sinf (`  
`float x )`

The `sinf()` function returns the sine of `x`, measured in radians.

**21.6.3.96** `sinh()` `double sinh (`  
`double x )`

The `sinh()` function returns the hyperbolic sine of `x`.

**21.6.3.97** `sinhf()` `float sinhf (`  
`float x )`

The `sinhf()` function returns the hyperbolic sine of `x`.

**21.6.3.98** `sinhl()` `long double sinhl (`  
`long double x )`

The `sinhl()` function returns the hyperbolic sine of `x`.

**21.6.3.99** `sinl()` `long double sinl (`  
`long double x )`

The `sinl()` function returns the sine of `x`, measured in radians.

**21.6.3.100** `sqrt()` `double sqrt (`  
`double x )`

The `sqrt()` function returns the non-negative square root of `x`.

**21.6.3.101** `sqrtf()` `float sqrtf (`  
`float x )`

The `sqrtf()` function returns the non-negative square root of  $x$ .

**21.6.3.102** `sqrtl()` `long double sqrtl (`  
`long double x )`

The `sqrtl()` function returns the non-negative square root of  $x$ .

**21.6.3.103** `square()` `double square (`  
`double x )`

The function `square()` returns  $x * x$ .

**Note**

This function does not belong to the C standard definition.

**21.6.3.104** `squaref()` `float squaref (`  
`float x )`

The function `squaref()` returns  $x * x$ .

**Note**

This function does not belong to the C standard definition.

**21.6.3.105** `squarel()` `long double squarel (`  
`long double x )`

The function `squarel()` returns  $x * x$ .

**Note**

This function does not belong to the C standard definition.

**21.6.3.106** `tan()` `double tan (`  
`double x )`

The `tan()` function returns the tangent of  $x$ , measured in radians.



**21.6.3.107 tanf()** `float tanf (`  
`float x )`

The `tanf()` function returns the tangent of  $x$ , measured in radians.

**21.6.3.108 tanh()** `double tanh (`  
`double x )`

The `tanh()` function returns the hyperbolic tangent of  $x$ .

**21.6.3.109 tanhf()** `float tanhf (`  
`float x )`

The `tanhf()` function returns the hyperbolic tangent of  $x$ .

**21.6.3.110 tanhl()** `long double tanhl (`  
`long double x )`

The `tanhl()` function returns the hyperbolic tangent of  $x$ .

**21.6.3.111 tanl()** `long double tanl (`  
`long double x )`

The `tanl()` function returns the tangent of  $x$ , measured in radians.

**21.6.3.112 trunc()** `double trunc (`  
`double x )`

The `trunc()` function rounds  $x$  to the nearest integer not larger in absolute value.

**21.6.3.113 truncf()** `float truncf (`  
`float x )`

The `truncf()` function rounds  $x$  to the nearest integer not larger in absolute value.

**21.6.3.114 trunc1()** `long double trunc1 (`  
`long double x )`

The `trunc1()` function rounds  $x$  to the nearest integer not larger in absolute value.

## 21.7 <setjmp.h>: Non-local goto

### Functions

- int `setjmp` (`jmp_buf __jmpb`)
- void `longjmp` (`jmp_buf __jmpb`, int `__ret`)

### 21.7.1 Detailed Description

While the C language has the dreaded `goto` statement, it can only be used to jump to a label in the same (local) function. In order to jump directly to another (non-local) function, the C library provides the `setjmp` and `longjmp` functions. `setjmp` and `longjmp` are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

#### Note

`setjmp` and `longjmp` make programs hard to understand and maintain. If possible, an alternative should be used.

`longjmp` can destroy changes made to global register variables (see [How to permanently bind a variable to a register?](#)).

For a very detailed discussion of `setjmp/longjmp`, see Chapter 7 of *Advanced Programming in the UNIX Environment*, by W. Richard Stevens.

#### Example:

```
#include <setjmp.h>

jmp_buf env;

int main (void)
{
    if (setjmp (env))
    {
        // Handle error ...
    }

    while (1)
    {
        // Main processing loop which calls foo() somewhere ...
    }
}

void foo (void)
{
    // blah, blah, blah ...

    if (err)
    {
        longjmp (env, 1);
    }
}
```

### 21.7.2 Function Documentation

**21.7.2.1 `longjmp()`** `void longjmp (`  
    `jmp_buf __jmpb,`  
    `int __ret )`

Non-local jump to a saved stack context.

```
#include <setjmp.h>
```

`longjmp()` restores the environment saved by the last call of `setjmp()` with the corresponding `__jmpb` argument. After `longjmp()` is completed, program execution continues as if the corresponding call of `setjmp()` had just returned the value `__ret`.

#### Note

`longjmp()` cannot cause 0 to be returned. If `longjmp()` is invoked with a second argument of 0, 1 will be returned instead.

**Parameters**

<code>__jmpb</code>	Information saved by a previous call to <code>setjmp()</code> .
<code>__ret</code>	Value to return to the caller of <code>setjmp()</code> .

**Returns**

This function never returns.

```
21.7.2.2 setjmp() int setjmp (  
    jmp_buf __jmpb )
```

Save stack context for non-local goto.

```
#include <setjmp.h>
```

`setjmp()` saves the stack context/environment in `__jmpb` for later use by `longjmp()`. The stack context will be invalidated if the function which called `setjmp()` returns.

**Parameters**

<code>__jmpb</code>	Variable of type <code>jmp_buf</code> which holds the stack information such that the environment can be restored.
---------------------	--

**Returns**

`setjmp()` returns 0 if returning directly, and non-zero when returning from `longjmp()` using the saved context.

## 21.8 <stdint.h>: Standard Integer Types

### Exact-width integer types

Integer types having exactly the specified width

- typedef signed char `int8_t`
- typedef unsigned char `uint8_t`
- typedef signed int `int16_t`
- typedef unsigned int `uint16_t`
- typedef signed long int `int32_t`
- typedef unsigned long int `uint32_t`
- typedef signed long long int `int64_t`
- typedef unsigned long long int `uint64_t`

### Integer types capable of holding object pointers

These allow you to declare variables of the same size as a pointer.

- typedef `int16_t` `intptr_t`
- typedef `uint16_t` `uintptr_t`

### Minimum-width integer types

Integer types having at least the specified width

- typedef `int8_t` `int_least8_t`
- typedef `uint8_t` `uint_least8_t`
- typedef `int16_t` `int_least16_t`
- typedef `uint16_t` `uint_least16_t`
- typedef `int32_t` `int_least32_t`
- typedef `uint32_t` `uint_least32_t`
- typedef `int64_t` `int_least64_t`
- typedef `uint64_t` `uint_least64_t`

### Fastest minimum-width integer types

Integer types being usually fastest having at least the specified width

- typedef `int8_t` `int_fast8_t`
- typedef `uint8_t` `uint_fast8_t`
- typedef `int16_t` `int_fast16_t`
- typedef `uint16_t` `uint_fast16_t`
- typedef `int32_t` `int_fast32_t`
- typedef `uint32_t` `uint_fast32_t`
- typedef `int64_t` `int_fast64_t`
- typedef `uint64_t` `uint_fast64_t`

### Greatest-width integer types

Types designating integer data capable of representing any value of any integer type in the corresponding signed or unsigned category

- typedef `int64_t` `intmax_t`
- typedef `uint64_t` `uintmax_t`

### Limits of specified-width integer types

C++ implementations should define these macros only when `__STDC_LIMIT_MACROS` is defined before `<stdint.h>` is included

- #define `INT8_MAX` `0x7f`
- #define `INT8_MIN` `(-INT8_MAX - 1)`
- #define `UINT8_MAX` `(INT8_MAX * 2 + 1)`
- #define `INT16_MAX` `0x7fff`
- #define `INT16_MIN` `(-INT16_MAX - 1)`
- #define `UINT16_MAX` `(__CONCAT(INT16_MAX, U) * 2U + 1U)`
- #define `INT32_MAX` `0x7fffffffL`
- #define `INT32_MIN` `(-INT32_MAX - 1L)`
- #define `UINT32_MAX` `(__CONCAT(INT32_MAX, U) * 2UL + 1UL)`
- #define `INT64_MAX` `0x7fffffffffffffffLL`
- #define `INT64_MIN` `(-INT64_MAX - 1LL)`
- #define `UINT64_MAX` `(__CONCAT(INT64_MAX, U) * 2ULL + 1ULL)`

### Limits of minimum-width integer types

- #define INT\_LEAST8\_MAX INT8\_MAX
- #define INT\_LEAST8\_MIN INT8\_MIN
- #define UINT\_LEAST8\_MAX UINT8\_MAX
- #define INT\_LEAST16\_MAX INT16\_MAX
- #define INT\_LEAST16\_MIN INT16\_MIN
- #define UINT\_LEAST16\_MAX UINT16\_MAX
- #define INT\_LEAST32\_MAX INT32\_MAX
- #define INT\_LEAST32\_MIN INT32\_MIN
- #define UINT\_LEAST32\_MAX UINT32\_MAX
- #define INT\_LEAST64\_MAX INT64\_MAX
- #define INT\_LEAST64\_MIN INT64\_MIN
- #define UINT\_LEAST64\_MAX UINT64\_MAX

### Limits of fastest minimum-width integer types

- #define INT\_FAST8\_MAX INT8\_MAX
- #define INT\_FAST8\_MIN INT8\_MIN
- #define UINT\_FAST8\_MAX UINT8\_MAX
- #define INT\_FAST16\_MAX INT16\_MAX
- #define INT\_FAST16\_MIN INT16\_MIN
- #define UINT\_FAST16\_MAX UINT16\_MAX
- #define INT\_FAST32\_MAX INT32\_MAX
- #define INT\_FAST32\_MIN INT32\_MIN
- #define UINT\_FAST32\_MAX UINT32\_MAX
- #define INT\_FAST64\_MAX INT64\_MAX
- #define INT\_FAST64\_MIN INT64\_MIN
- #define UINT\_FAST64\_MAX UINT64\_MAX

### Limits of integer types capable of holding object pointers

- #define INTPTR\_MAX INT16\_MAX
- #define INTPTR\_MIN INT16\_MIN
- #define UINTPTR\_MAX UINT16\_MAX

### Limits of greatest-width integer types

- #define INTMAX\_MAX INT64\_MAX
- #define INTMAX\_MIN INT64\_MIN
- #define UINTMAX\_MAX UINT64\_MAX

### Limits of other integer types

C++ implementations should define these macros only when `__STDC_LIMIT_MACROS` is defined before `<stdint.h>` is included

- #define PTRDIFF\_MAX INT16\_MAX
- #define PTRDIFF\_MIN INT16\_MIN
- #define SIG\_ATOMIC\_MAX INT8\_MAX
- #define SIG\_ATOMIC\_MIN INT8\_MIN
- #define SIZE\_MAX UINT16\_MAX
- #define WCHAR\_MAX \_\_WCHAR\_MAX\_\_
- #define WCHAR\_MIN \_\_WCHAR\_MIN\_\_
- #define WINT\_MAX \_\_WINT\_MAX\_\_
- #define WINT\_MIN \_\_WINT\_MIN\_\_

### Macros for integer constants

C++ implementations should define these macros only when `__STDC_CONSTANT_MACROS` is defined before `<stdint.h>` is included.

These definitions are valid for integer constants without suffix and for macros defined as integer constant without suffix

- `#define INT8_C(value) ((int8_t) value)`
- `#define UINT8_C(value) ((uint8_t) __CONCAT(value, U))`
- `#define INT16_C(value) value`
- `#define UINT16_C(value) __CONCAT(value, U)`
- `#define INT32_C(value) __CONCAT(value, L)`
- `#define UINT32_C(value) __CONCAT(value, UL)`
- `#define INT64_C(value) __CONCAT(value, LL)`
- `#define UINT64_C(value) __CONCAT(value, ULL)`
- `#define INTMAX_C(value) __CONCAT(value, LL)`
- `#define UINTMAX_C(value) __CONCAT(value, ULL)`

#### 21.8.1 Detailed Description

```
#include <stdint.h>
```

Use `[u]intN_t` if you need exactly N bits.

Since these typedefs are mandated by the C99 standard, they are preferred over rolling your own typedefs.

#### 21.8.2 Macro Definition Documentation

**21.8.2.1 INT16\_C** `#define INT16_C(  
value) value`

define a constant of type `int16_t`

**21.8.2.2 INT16\_MAX** `#define INT16_MAX 0x7fff`

largest positive value an `int16_t` can hold.

**21.8.2.3 INT16\_MIN** `#define INT16_MIN (-INT16_MAX - 1)`

smallest negative value an `int16_t` can hold.

**21.8.2.4 INT32\_C** `#define INT32_C(  
value) __CONCAT(value, L)`

define a constant of type `int32_t`

**21.8.2.5 INT32\_MAX** `#define INT32_MAX 0x7fffffffL`

largest positive value an `int32_t` can hold.

**21.8.2.6 INT32\_MIN** `#define INT32_MIN (-INT32_MAX - 1L)`

smallest negative value an `int32_t` can hold.

**21.8.2.7 INT64\_C** `#define INT64_C(  
value) __CONCAT(value, LL)`

define a constant of type `int64_t`

**21.8.2.8 INT64\_MAX** `#define INT64_MAX 0x7fffffffffffffffLL`

largest positive value an `int64_t` can hold.

**21.8.2.9 INT64\_MIN** `#define INT64_MIN (-INT64_MAX - 1LL)`

smallest negative value an `int64_t` can hold.

**21.8.2.10 INT8\_C** `#define INT8_C(  
value) ((int8_t) value)`

define a constant of type `int8_t`

**21.8.2.11 INT8\_MAX** `#define INT8_MAX 0x7f`

largest positive value an `int8_t` can hold.

**21.8.2.12 INT8\_MIN** `#define INT8_MIN (-INT8_MAX - 1)`

smallest negative value an `int8_t` can hold.

**21.8.2.13 INT\_FAST16\_MAX** `#define INT_FAST16_MAX INT16_MAX`

largest positive value an `int_fast16_t` can hold.

**21.8.2.14 INT\_FAST16\_MIN** `#define INT_FAST16_MIN INT16_MIN`

smallest negative value an `int_fast16_t` can hold.

**21.8.2.15 INT\_FAST32\_MAX** `#define INT_FAST32_MAX INT32_MAX`

largest positive value an `int_fast32_t` can hold.

**21.8.2.16 INT\_FAST32\_MIN** #define INT\_FAST32\_MIN INT32\_MIN

smallest negative value an int\_fast32\_t can hold.

**21.8.2.17 INT\_FAST64\_MAX** #define INT\_FAST64\_MAX INT64\_MAX

largest positive value an int\_fast64\_t can hold.

**21.8.2.18 INT\_FAST64\_MIN** #define INT\_FAST64\_MIN INT64\_MIN

smallest negative value an int\_fast64\_t can hold.

**21.8.2.19 INT\_FAST8\_MAX** #define INT\_FAST8\_MAX INT8\_MAX

largest positive value an int\_fast8\_t can hold.

**21.8.2.20 INT\_FAST8\_MIN** #define INT\_FAST8\_MIN INT8\_MIN

smallest negative value an int\_fast8\_t can hold.

**21.8.2.21 INT\_LEAST16\_MAX** #define INT\_LEAST16\_MAX INT16\_MAX

largest positive value an int\_least16\_t can hold.

**21.8.2.22 INT\_LEAST16\_MIN** #define INT\_LEAST16\_MIN INT16\_MIN

smallest negative value an int\_least16\_t can hold.

**21.8.2.23 INT\_LEAST32\_MAX** #define INT\_LEAST32\_MAX INT32\_MAX

largest positive value an int\_least32\_t can hold.

**21.8.2.24 INT\_LEAST32\_MIN** #define INT\_LEAST32\_MIN INT32\_MIN

smallest negative value an int\_least32\_t can hold.

**21.8.2.25 INT\_LEAST64\_MAX** #define INT\_LEAST64\_MAX INT64\_MAX

largest positive value an int\_least64\_t can hold.

**21.8.2.26 INT\_LEAST64\_MIN** #define INT\_LEAST64\_MIN INT64\_MIN

smallest negative value an int\_least64\_t can hold.



**21.8.2.27 INT\_LEAST8\_MAX** `#define INT_LEAST8_MAX INT8_MAX`

largest positive value an `int_least8_t` can hold.

**21.8.2.28 INT\_LEAST8\_MIN** `#define INT_LEAST8_MIN INT8_MIN`

smallest negative value an `int_least8_t` can hold.

**21.8.2.29 INTMAX\_C** `#define INTMAX_C(  
value) __CONCAT(value, LL)`

define a constant of type `intmax_t`

**21.8.2.30 INTMAX\_MAX** `#define INTMAX_MAX INT64_MAX`

largest positive value an `intmax_t` can hold.

**21.8.2.31 INTMAX\_MIN** `#define INTMAX_MIN INT64_MIN`

smallest negative value an `intmax_t` can hold.

**21.8.2.32 INTPTR\_MAX** `#define INTPTR_MAX INT16_MAX`

largest positive value an `intptr_t` can hold.

**21.8.2.33 INTPTR\_MIN** `#define INTPTR_MIN INT16_MIN`

smallest negative value an `intptr_t` can hold.

**21.8.2.34 PTRDIFF\_MAX** `#define PTRDIFF_MAX INT16_MAX`

largest positive value a `ptrdiff_t` can hold.

**21.8.2.35 PTRDIFF\_MIN** `#define PTRDIFF_MIN INT16_MIN`

smallest negative value a `ptrdiff_t` can hold.

**21.8.2.36 SIG\_ATOMIC\_MAX** `#define SIG_ATOMIC_MAX INT8_MAX`

largest positive value a `sig_atomic_t` can hold.

**21.8.2.37 SIG\_ATOMIC\_MIN** `#define SIG_ATOMIC_MIN INT8_MIN`

smallest negative value a `sig_atomic_t` can hold.

**21.8.2.38 SIZE\_MAX** `#define SIZE_MAX UINT16_MAX`

largest value a `size_t` can hold.

**21.8.2.39 UINT16\_C** `#define UINT16_C(  
value) __CONCAT(value, U)`

define a constant of type `uint16_t`

**21.8.2.40 UINT16\_MAX** `#define UINT16_MAX (__CONCAT(INT16_MAX, U) * 2U + 1U)`

largest value an `uint16_t` can hold.

**21.8.2.41 UINT32\_C** `#define UINT32_C(  
value) __CONCAT(value, UL)`

define a constant of type `uint32_t`

**21.8.2.42 UINT32\_MAX** `#define UINT32_MAX (__CONCAT(INT32_MAX, U) * 2UL + 1UL)`

largest value an `uint32_t` can hold.

**21.8.2.43 UINT64\_C** `#define UINT64_C(  
value) __CONCAT(value, ULL)`

define a constant of type `uint64_t`

**21.8.2.44 UINT64\_MAX** `#define UINT64_MAX (__CONCAT(INT64_MAX, U) * 2ULL + 1ULL)`

largest value an `uint64_t` can hold.

**21.8.2.45 UINT8\_C** `#define UINT8_C(  
value) ((uint8_t) __CONCAT(value, U))`

define a constant of type `uint8_t`

**21.8.2.46 UINT8\_MAX** `#define UINT8_MAX (INT8_MAX * 2 + 1)`

largest value an `uint8_t` can hold.

**21.8.2.47 UINT\_FAST16\_MAX** `#define UINT_FAST16_MAX UINT16_MAX`

largest value an `uint_fast16_t` can hold.

**21.8.2.48 UINT\_FAST32\_MAX** `#define UINT_FAST32_MAX UINT32_MAX`

largest value an `uint_fast32_t` can hold.

**21.8.2.49 UINT\_FAST64\_MAX** `#define UINT_FAST64_MAX UUINT64_MAX`

largest value an `uint_fast64_t` can hold.

**21.8.2.50 UINT\_FAST8\_MAX** `#define UINT_FAST8_MAX UUINT8_MAX`

largest value an `uint_fast8_t` can hold.

**21.8.2.51 UINT\_LEAST16\_MAX** `#define UINT_LEAST16_MAX UUINT16_MAX`

largest value an `uint_least16_t` can hold.

**21.8.2.52 UINT\_LEAST32\_MAX** `#define UINT_LEAST32_MAX UUINT32_MAX`

largest value an `uint_least32_t` can hold.

**21.8.2.53 UINT\_LEAST64\_MAX** `#define UINT_LEAST64_MAX UUINT64_MAX`

largest value an `uint_least64_t` can hold.

**21.8.2.54 UINT\_LEAST8\_MAX** `#define UINT_LEAST8_MAX UUINT8_MAX`

largest value an `uint_least8_t` can hold.

**21.8.2.55 UINTMAX\_C** `#define UINTMAX_C(  
value ) __CONCAT(value, ULL)`

define a constant of type `uintmax_t`

**21.8.2.56 UINTMAX\_MAX** `#define UINTMAX_MAX UUINT64_MAX`

largest value an `uintmax_t` can hold.

**21.8.2.57 UINTPTR\_MAX** `#define UINTPTR_MAX UUINT16_MAX`

largest value an `uintptr_t` can hold.

### 21.8.3 Typedef Documentation

**21.8.3.1 int16\_t** typedef signed int `int16_t`

16-bit signed type.

**21.8.3.2 int32\_t** typedef signed long int `int32_t`

32-bit signed type.

**21.8.3.3 int64\_t** typedef signed long long int `int64_t`

64-bit signed type.

**Note**

This type is not available when the compiler option `-mint8` is in effect.

**21.8.3.4 int8\_t** typedef signed char `int8_t`

8-bit signed type.

**21.8.3.5 int\_fast16\_t** typedef `int16_t int_fast16_t`

fastest signed int with at least 16 bits.

**21.8.3.6 int\_fast32\_t** typedef `int32_t int_fast32_t`

fastest signed int with at least 32 bits.

**21.8.3.7 int\_fast64\_t** typedef `int64_t int_fast64_t`

fastest signed int with at least 64 bits.

**Note**

This type is not available when the compiler option `-mint8` is in effect.

**21.8.3.8 int\_fast8\_t** typedef `int8_t int_fast8_t`

fastest signed int with at least 8 bits.

**21.8.3.9 int\_least16\_t** typedef `int16_t int_least16_t`

signed int with at least 16 bits.

**21.8.3.10 int\_least32\_t** typedef `int32_t int_least32_t`

signed int with at least 32 bits.

**21.8.3.11 `int_least64_t`** typedef `int64_t` `int_least64_t`

signed int with at least 64 bits.

**Note**

This type is not available when the compiler option `-mint8` is in effect.

**21.8.3.12 `int_least8_t`** typedef `int8_t` `int_least8_t`

signed int with at least 8 bits.

**21.8.3.13 `intmax_t`** typedef `int64_t` `intmax_t`

largest signed int available.

**21.8.3.14 `intptr_t`** typedef `int16_t` `intptr_t`

Signed pointer compatible type.

**21.8.3.15 `uint16_t`** typedef unsigned int `uint16_t`

16-bit unsigned type.

**21.8.3.16 `uint32_t`** typedef unsigned long int `uint32_t`

32-bit unsigned type.

**21.8.3.17 `uint64_t`** typedef unsigned long long int `uint64_t`

64-bit unsigned type.

**Note**

This type is not available when the compiler option `-mint8` is in effect.

**21.8.3.18 `uint8_t`** typedef unsigned char `uint8_t`

8-bit unsigned type.

**21.8.3.19 `uint_fast16_t`** typedef `uint16_t` `uint_fast16_t`

fastest unsigned int with at least 16 bits.

**21.8.3.20 uint\_fast32\_t** typedef uint32\_t uint\_fast32\_t

fastest unsigned int with at least 32 bits.

**21.8.3.21 uint\_fast64\_t** typedef uint64\_t uint\_fast64\_t

fastest unsigned int with at least 64 bits.

**Note**

This type is not available when the compiler option `-mint8` is in effect.

**21.8.3.22 uint\_fast8\_t** typedef uint8\_t uint\_fast8\_t

fastest unsigned int with at least 8 bits.

**21.8.3.23 uint\_least16\_t** typedef uint16\_t uint\_least16\_t

unsigned int with at least 16 bits.

**21.8.3.24 uint\_least32\_t** typedef uint32\_t uint\_least32\_t

unsigned int with at least 32 bits.

**21.8.3.25 uint\_least64\_t** typedef uint64\_t uint\_least64\_t

unsigned int with at least 64 bits.

**Note**

This type is not available when the compiler option `-mint8` is in effect.

**21.8.3.26 uint\_least8\_t** typedef uint8\_t uint\_least8\_t

unsigned int with at least 8 bits.

**21.8.3.27 uintmax\_t** typedef uint64\_t uintmax\_t

largest unsigned int available.

**21.8.3.28 uintptr\_t** typedef uint16\_t uintptr\_t

Unsigned pointer compatible type.

## 21.9 <stdio.h>: Standard IO facilities

### Macros

- #define `stdin` (`__iob[0]`)
- #define `stdout` (`__iob[1]`)
- #define `stderr` (`__iob[2]`)
- #define `EOF` (-1)
- #define `fdev_set_udata`(stream, u) do { (stream)->udata = u; } while(0)
- #define `fdev_get_udata`(stream) ((stream)->udata)
- #define `fdev_setup_stream`(stream, put, get, rflag)
- #define `_FDEV_SETUP_READ` `__SRD`
- #define `_FDEV_SETUP_WRITE` `__SWR`
- #define `_FDEV_SETUP_RW` (`__SRD|__SWR`)
- #define `_FDEV_ERR` (-1)
- #define `_FDEV_EOF` (-2)
- #define `FDEV_SETUP_STREAM`(put, get, rflag)
- #define `fdev_close`()
- #define `putc`(`__c`, `__stream`) `fputc`(`__c`, `__stream`)
- #define `putchar`(`__c`) `fputc`(`__c`, `stdout`)
- #define `getc`(`__stream`) `fgetc`(`__stream`)
- #define `getchar`() `fgetc`(`stdin`)

### Typedefs

- typedef struct `__file` `FILE`

### Functions

- int `fclose` (`FILE *__stream`)
- int `vfprintf` (`FILE *__stream`, const char \*`__fmt`, va\_list `__ap`)
- int `vfprintf_P` (`FILE *__stream`, const char \*`__fmt`, va\_list `__ap`)
- int `fputc` (int `__c`, `FILE *__stream`)
- int `printf` (const char \*`__fmt`,...)
- int `printf_P` (const char \*`__fmt`,...)
- int `vprintf` (const char \*`__fmt`, va\_list `__ap`)
- int `sprintf` (char \*`__s`, const char \*`__fmt`,...)
- int `sprintf_P` (char \*`__s`, const char \*`__fmt`,...)
- int `snprintf` (char \*`__s`, size\_t `__n`, const char \*`__fmt`,...)
- int `snprintf_P` (char \*`__s`, size\_t `__n`, const char \*`__fmt`,...)
- int `vsprintf` (char \*`__s`, const char \*`__fmt`, va\_list `ap`)
- int `vsprintf_P` (char \*`__s`, const char \*`__fmt`, va\_list `ap`)
- int `vsprintf` (char \*`__s`, size\_t `__n`, const char \*`__fmt`, va\_list `ap`)
- int `vsprintf_P` (char \*`__s`, size\_t `__n`, const char \*`__fmt`, va\_list `ap`)
- int `fprintf` (`FILE *__stream`, const char \*`__fmt`,...)
- int `fprintf_P` (`FILE *__stream`, const char \*`__fmt`,...)
- int `fputs` (const char \*`__str`, `FILE *__stream`)
- int `fputs_P` (const char \*`__str`, `FILE *__stream`)
- int `puts` (const char \*`__str`)
- int `puts_P` (const char \*`__str`)
- size\_t `fwrite` (const void \*`__ptr`, size\_t `__size`, size\_t `__nmemb`, `FILE *__stream`)
- int `fgetc` (`FILE *__stream`)
- int `ungetc` (int `__c`, `FILE *__stream`)

- char \* [fgets](#) (char \*\_\_str, int \_\_size, FILE \*\_\_stream)
- char \* [gets](#) (char \*\_\_str)
- size\_t [fread](#) (void \*\_\_ptr, size\_t \_\_size, size\_t \_\_nmemb, FILE \*\_\_stream)
- void [clearerr](#) (FILE \*\_\_stream)
- int [feof](#) (FILE \*\_\_stream)
- int [ferror](#) (FILE \*\_\_stream)
- int [vfscanf](#) (FILE \*\_\_stream, const char \*\_\_fmt, va\_list \_\_ap)
- int [vfscanf\\_P](#) (FILE \*\_\_stream, const char \*\_\_fmt, va\_list \_\_ap)
- int [fscanf](#) (FILE \*\_\_stream, const char \*\_\_fmt,...)
- int [fscanf\\_P](#) (FILE \*\_\_stream, const char \*\_\_fmt,...)
- int [scanf](#) (const char \*\_\_fmt,...)
- int [scanf\\_P](#) (const char \*\_\_fmt,...)
- int [vscanf](#) (const char \*\_\_fmt, va\_list \_\_ap)
- int [sscanf](#) (const char \*\_\_buf, const char \*\_\_fmt,...)
- int [sscanf\\_P](#) (const char \*\_\_buf, const char \*\_\_fmt,...)
- int [fflush](#) (FILE \*stream)
- FILE \* [fdevopen](#) (int(\*put)(char, FILE \*), int(\*get)(FILE \*))

### 21.9.1 Detailed Description

```
#include <stdio.h>
```

**Introduction to the Standard IO facilities** This file declares the standard IO facilities that are implemented in AVR-LibC. Due to the nature of the underlying hardware, only a limited subset of standard IO is implemented. There is no actual file implementation available, so only device IO can be performed. Since there's no operating system, the application needs to provide enough details about their devices in order to make them usable by the standard IO facilities.

Due to space constraints, some functionality has not been implemented at all (like some of the `printf` conversions that have been left out). Nevertheless, potential users of this implementation should be warned: the `printf` and `scanf` families of functions, although usually associated with presumably simple things like the famous "Hello, world!" program, are actually fairly complex which causes their inclusion to eat up a fair amount of code space. Also, they are not fast due to the nature of interpreting the format string at run-time. Whenever possible, resorting to the (sometimes non-standard) predetermined conversion facilities that are offered by AVR-LibC will usually cost much less in terms of speed and code size.

**Tunable options for code size vs. feature set** In order to allow programmers a code size vs. functionality tradeoff, the function `fprintf()` which is the heart of the `printf` family can be selected in different flavours using linker options. See the documentation of `fprintf()` for a detailed description. The same applies to `vfscanf()` and the `scanf` family of functions.

**Outline of the chosen API** The standard streams `stdin`, `stdout`, and `stderr` are provided, but contrary to the C standard, since AVR-LibC has no knowledge about applicable devices, these streams are not already pre-initialized at application startup. Also, since there is no notion of "file" whatsoever to AVR-LibC, there is no function `fopen()` that could be used to associate a stream to some device. (See [note 1](#).) Instead, the function `fdevopen()` is provided to associate a stream to a device, where the device needs to provide a function to send a character, to receive a character, or both. There is no differentiation between "text" and "binary" streams inside AVR-LibC. Character `\n` is sent literally down to the device's `put()` function. If the device requires a carriage return (`\r`) character to be sent before the linefeed, its `put()` routine must implement this (see [note 2](#)).

As an alternative method to `fdevopen()`, the macro `fdev_setup_stream()` might be used to setup a user-supplied FILE structure.



It should be noted that the automatic conversion of a newline character into a carriage return - newline sequence breaks binary transfers. If binary transfers are desired, no automatic conversion should be performed, but instead any string that aims to issue a CR-LF sequence must use `"\r\n"` explicitly.

For convenience, the first call to `fdevopen()` that opens a stream for reading will cause the resulting stream to be aliased to `stdin`. Likewise, the first call to `fdevopen()` that opens a stream for writing will cause the resulting stream to be aliased to both, `stdout`, and `stderr`. Thus, if the open was done with both, read and write intent, all three standard streams will be identical. Note that these aliases are indistinguishable from each other, thus calling `fclose()` on such a stream will also effectively close all of its aliases ([note 3](#)).

It is possible to tie additional user data to a stream, using `fdev_set_udata()`. The backend put and get functions can then extract this user data using `fdev_get_udata()`, and act appropriately. For example, a single put function could be used to talk to two different UARTs that way, or the put and get functions could keep internal state between calls there.

**Format strings in flash ROM** All the `printf` and `scanf` family functions come in two flavours: the standard name, where the format string is expected to be in SRAM, as well as a version with the suffix `"_P"` where the format string is expected to reside in the flash ROM. The macro `PSTR` (explained in [<avr/pgmspace.h>: Program Space Utilities](#)) becomes very handy for declaring these format strings.

**Running stdio without malloc()** By default, `fdevopen()` requires `malloc()`. As this is often not desired in the limited environment of a microcontroller, an alternative option is provided to run completely without `malloc()`.

The macro `fdev_setup_stream()` is provided to prepare a user-supplied FILE buffer for operation with stdio.

**Example** `#include <stdio.h>`

```
static int uart_putchar(char c, FILE *stream);

static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,
                                        _FDEV_SETUP_WRITE);

static int
uart_putchar(char c, FILE *stream)
{
    if (c == '\n')
        uart_putchar('\r', stream);
    loop_until_bit_is_set(UCSRA, UDRE);
    UDR = c;
    return 0;
}

int
main(void)
{
    init_uart();
    stdout = &mystdout;
    printf("Hello, world!\n");

    return 0;
}
```

This example uses the initializer form `FDEV_SETUP_STREAM()` rather than the function-like `fdev_setup_stream()`, so all data initialization happens during C start-up.

If streams initialized that way are no longer needed, they can be destroyed by first calling the macro `fdev_close()`, and then destroying the object itself. No call to `fclose()` should be issued for these streams. While calling `fclose()` itself is harmless, it will cause an undefined reference to `free()` and thus cause the linker to link the `malloc` module into the application.

## Notes

**Note 1:** It might have been possible to implement a device abstraction that is compatible with `fopen()` but since this would have required to parse a string, and to take all the information needed either out of this string, or out of an additional table that would need to be provided by the application, this approach was not taken.

**Note 2:** This basically follows the Unix approach: if a device such as a terminal needs special handling, it is in the domain of the terminal device driver to provide this functionality. Thus, a simple function suitable as `put()` for `fdevopen()` that talks to a UART interface might look like this:

```
int
uart_putchar(char c, FILE *stream)
{
    if (c == '\n')
        uart_putchar('\r', stream);
    loop_until_bit_is_set(UCSRA, UDRE);
    UDR = c;
    return 0;
}
```

**Note 3:** This implementation has been chosen because the cost of maintaining an alias is considerably smaller than the cost of maintaining full copies of each stream. Yet, providing an implementation that offers the complete set of standard streams was deemed to be useful. Not only that writing `printf()` instead of `fprintf(mystream, ...)` saves typing work, but since `avr-gcc` needs to resort to pass all arguments of variadic functions on the stack (as opposed to passing them in registers for functions that take a fixed number of parameters), the ability to pass one parameter less by implying `stdin` or `stdout` will also save some execution time.

## 21.9.2 Macro Definition Documentation

**21.9.2.1 `_FDEV_EOF`** `#define _FDEV_EOF (-2)`

Return code for an end-of-file condition during device read.

To be used in the `get` function of `fdevopen()`.

**21.9.2.2 `_FDEV_ERR`** `#define _FDEV_ERR (-1)`

Return code for an error condition during device read.

To be used in the `get` function of `fdevopen()`.

**21.9.2.3 `_FDEV_SETUP_READ`** `#define _FDEV_SETUP_READ __SRD`

`fdev_setup_stream()` with read intent

**21.9.2.4 `_FDEV_SETUP_RW`** `#define _FDEV_SETUP_RW (__SRD|__SWR)`

`fdev_setup_stream()` with read/write intent

**21.9.2.5 `_FDEV_SETUP_WRITE`** `#define _FDEV_SETUP_WRITE __SWR`

`fdev_setup_stream()` with write intent

**21.9.2.6 EOF** `#define EOF (-1)`

EOF declares the value that is returned by various standard IO functions in case of an error. Since the AVR platform (currently) doesn't contain an abstraction for actual files, its origin as "end of file" is somewhat meaningless here.

**21.9.2.7 fdev\_close** `#define fdev_close( )`

This macro frees up any library resources that might be associated with `stream`. It should be called if `stream` is no longer needed, right before the application is going to destroy the `stream` object itself.

(Currently, this macro evaluates to nothing, but this might change in future versions of the library.)

**21.9.2.8 fdev\_get\_udata** `#define fdev_get_udata( stream ) ((stream)->udata)`

This macro retrieves a pointer to user defined data from a FILE stream object.

**21.9.2.9 fdev\_set\_udata** `#define fdev_set_udata( stream, u ) do { (stream)->udata = u; } while(0)`

This macro inserts a pointer to user defined data into a FILE stream object.

The user data can be useful for tracking state in the put and get functions supplied to the [fdevopen\(\)](#) function.

**21.9.2.10 FDEV\_SETUP\_STREAM** `#define FDEV_SETUP_STREAM( put, get, rwflag )`

Initializer for a user-supplied stdio stream.

This macro acts similar to [fdev\\_setup\\_stream\(\)](#), but it is to be used as the initializer of a variable of type FILE.

The remaining arguments are to be used as explained in [fdev\\_setup\\_stream\(\)](#).

**21.9.2.11 fdev\_setup\_stream** `#define fdev_setup_stream( stream, put, get, rwflag )`

Setup a user-supplied buffer as an stdio stream.

This macro takes a user-supplied buffer `stream`, and sets it up as a stream that is valid for stdio operations, similar to one that has been obtained dynamically from [fdevopen\(\)](#). The buffer to setup must be of type FILE.

The arguments `put` and `get` are identical to those that need to be passed to [fdevopen\(\)](#).

The `rwflag` argument can take one of the values `_FDEV_SETUP_READ`, `_FDEV_SETUP_WRITE`, or `_FDEV_SETUP_RW`, for read, write, or read/write intent, respectively.

**Note**

No assignments to the standard streams will be performed by [fdev\\_setup\\_stream\(\)](#). If standard streams are to be used, these need to be assigned by the user. See also under [Running stdio without malloc\(\)](#).

```
21.9.2.12 getc #define getc(  
    __stream ) fgetc(__stream)
```

The macro `getc` used to be a "fast" macro implementation with a functionality identical to `fgetc()`. For space constraints, in AVR-LibC, it is just an alias for `fgetc`.

```
21.9.2.13 getchar #define getchar(  
    void ) fgetc(stdin)
```

The macro `getchar` reads a character from `stdin`. Return values and error handling is identical to `fgetc()`.

```
21.9.2.14 putc #define putc(  
    __c,  
    __stream ) fputc(__c, __stream)
```

The macro `putc` used to be a "fast" macro implementation with a functionality identical to `fputc()`. For space constraints, in AVR-LibC, it is just an alias for `fputc`.

```
21.9.2.15 putchar #define putchar(  
    __c ) fputc(__c, stdout)
```

The macro `putchar` sends character `c` to `stdout`.

```
21.9.2.16 stderr #define stderr (__iob[2])
```

Stream destined for error output. Unless specifically assigned, identical to `stdout`.

If `stderr` should point to another stream, the result of another `fdevopen()` must be explicitly assigned to it without closing the previous `stderr` (since this would also close `stdout`).

```
21.9.2.17 stdin #define stdin (__iob[0])
```

Stream that will be used as an input stream by the simplified functions that don't take a `stream` argument.

The first stream opened with read intent using `fdevopen()` will be assigned to `stdin`.

```
21.9.2.18 stdout #define stdout (__iob[1])
```

Stream that will be used as an output stream by the simplified functions that don't take a `stream` argument.

The first stream opened with write intent using `fdevopen()` will be assigned to both, `stdin`, and `stderr`.

## 21.9.3 Typedef Documentation

```
21.9.3.1 FILE typedef struct __file FILE
```

`FILE` is the opaque structure that is passed around between the various standard IO functions.

## 21.9.4 Function Documentation

**21.9.4.1 clearerr()** `void clearerr (`  
`FILE * __stream )`

Clear the error and end-of-file flags of `stream`.

**21.9.4.2 fclose()** `int fclose (`  
`FILE * __stream )`

This function closes `stream`, and disallows any further IO to and from it.

When using `fdevopen()` to setup the stream, a call to `fclose()` is needed in order to free the internal resources allocated.

If the stream has been set up using `fdev_setup_stream()` or `FDEV_SETUP_STREAM()`, use `fdev_close()` instead.

It currently always returns 0 (for success).

**21.9.4.3 fdevopen()** `FILE * fdevopen (`  
`int (*) (char, FILE *) put,`  
`int (*) (FILE *) get )`

This function is a replacement for `fopen()`.

It opens a stream for a device where the actual device implementation needs to be provided by the application. If successful, a pointer to the structure for the opened stream is returned. Reasons for a possible failure currently include that neither the `put` nor the `get` argument have been provided, thus attempting to open a stream with no IO intent at all, or that insufficient dynamic memory is available to establish a new stream.

If the `put` function pointer is provided, the stream is opened with write intent. The function passed as `put` shall take two arguments, the first a character to write to the device, and the second a pointer to `FILE`, and shall return 0 if the output was successful, and a nonzero value if the character could not be sent to the device.

If the `get` function pointer is provided, the stream is opened with read intent. The function passed as `get` shall take a pointer to `FILE` as its single argument, and return one character from the device, passed as an `int` type. If an error occurs when trying to read from the device, it shall return `_FDEV_ERR`. If an end-of-file condition was reached while reading from the device, `_FDEV_EOF` shall be returned.

If both functions are provided, the stream is opened with read and write intent.

The first stream opened with read intent is assigned to `stdin`, and the first one opened with write intent is assigned to both, `stdout` and `stderr`.

`fdevopen()` uses `calloc()` (and thus `malloc()`) in order to allocate the storage for the new stream.

### Note

If the macro `__STDIO_FDEVOPEN_COMPAT_12` is declared before including `<stdio.h>`, a function prototype for `fdevopen()` will be chosen that is backwards compatible with AVR-LibC version 1.2 and before. This is solely intended for providing a simple migration path without the need to immediately change all source code. Do not use for new code.

**21.9.4.4 feof()** `int feof (`  
`FILE * __stream )`

Test the end-of-file flag of `stream`. This flag can only be cleared by a call to `clearerr()`.

**21.9.4.5 ferror()** `int ferror (`  
`FILE * __stream )`

Test the error flag of `stream`. This flag can only be cleared by a call to `clearerr()`.

**21.9.4.6 fflush()** `int fflush (`  
`FILE * stream )`

Flush `stream`.

This is a null operation provided for source-code compatibility only, as the standard IO implementation currently does not perform any buffering.

**21.9.4.7 fgetc()** `int fgetc (`  
`FILE * __stream )`

The function `fgetc` reads a character from `stream`. It returns the character, or EOF in case end-of-file was encountered or an error occurred. The routines `feof()` or `ferror()` must be used to distinguish between both situations.

**21.9.4.8 fgets()** `char * fgets (`  
`char * __str,`  
`int __size,`  
`FILE * __stream )`

Read at most `size - 1` bytes from `stream`, until a newline character was encountered, and store the characters in the buffer pointed to by `str`. Unless an error was encountered while reading, the string will then be terminated with a NUL character.

If an error was encountered, the function returns NULL and sets the error flag of `stream`, which can be tested using `ferror()`. Otherwise, a pointer to the string will be returned.

**21.9.4.9 fprintf()** `int fprintf (`  
`FILE * __stream,`  
`const char * __fmt,`  
`... )`

The function `fprintf` performs formatted output to `stream`. See `vfprintf()` for details.

**21.9.4.10 fprintf\_P()** `int fprintf_P (`  
`FILE * __stream,`  
`const char * __fmt,`  
`... )`

Variant of `fprintf()` that uses a `fmt` string that resides in program memory.

**21.9.4.11 fputc()** `int fputc (`  
    `int __c,`  
    `FILE * __stream )`

The function `fputc` sends the character `c` (though given as type `int`) to `stream`. It returns the character, or EOF in case an error occurred.

**21.9.4.12 fputs()** `int fputs (`  
    `const char * __str,`  
    `FILE * __stream )`

Write the string pointed to by `str` to stream `stream`.

Returns 0 on success and EOF on error.

**21.9.4.13 fputs\_P()** `int fputs_P (`  
    `const char * __str,`  
    `FILE * __stream )`

Variant of `fputs()` where `str` resides in program memory.

**21.9.4.14 fread()** `size_t fread (`  
    `void * __ptr,`  
    `size_t __size,`  
    `size_t __nmemb,`  
    `FILE * __stream )`

Read `nmemb` objects, `size` bytes each, from `stream`, to the buffer pointed to by `ptr`.

Returns the number of objects successfully read, i. e. `nmemb` unless an input error occurred or end-of-file was encountered. `feof()` and `ferror()` must be used to distinguish between these two conditions.

**21.9.4.15 fscanf()** `int fscanf (`  
    `FILE * __stream,`  
    `const char * __fmt,`  
    `... )`

The function `fscanf` performs formatted input, reading the input data from `stream`.

See `vfscanf()` for details.

**21.9.4.16 fscanf\_P()** `int fscanf_P (`  
    `FILE * __stream,`  
    `const char * __fmt,`  
    `... )`

Variant of `fscanf()` using a `fmt` string in program memory.

```
21.9.4.17 fwrite() size_t fwrite (  
    const void * __ptr,  
    size_t __size,  
    size_t __nmemb,  
    FILE * __stream )
```

Write `nmemb` objects, `size` bytes each, to `stream`. The first byte of the first object is referenced by `ptr`.

Returns the number of objects successfully written, i. e. `nmemb` unless an output error occurred.

```
21.9.4.18 gets() char * gets (  
    char * __str )
```

Similar to `fgets()` except that it will operate on stream `stdin`, and the trailing newline (if any) will not be stored in the string. It is the caller's responsibility to provide enough storage to hold the characters read.

```
21.9.4.19 printf() int printf (  
    const char * __fmt,  
    ... )
```

The function `printf` performs formatted output to stream `stdout`. See `vfprintf()` for details.

```
21.9.4.20 printf_P() int printf_P (  
    const char * __fmt,  
    ... )
```

Variant of `printf()` that uses a `fmt` string that resides in program memory.

```
21.9.4.21 puts() int puts (  
    const char * __str )
```

Write the string pointed to by `str`, and a trailing newline character, to `stdout`.

```
21.9.4.22 puts_P() int puts_P (  
    const char * __str )
```

Variant of `puts()` where `str` resides in program memory.

```
21.9.4.23 scanf() int scanf (  
    const char * __fmt,  
    ... )
```

The function `scanf` performs formatted input from stream `stdin`.

See `vfscanf()` for details.

```
21.9.4.24 scanf_P() int scanf_P (  
    const char * __fmt,  
    ... )
```

Variant of `scanf()` where `fmt` resides in program memory.



**21.9.4.25 `snprintf()`** `int snprintf (`  
    `char * __s,`  
    `size_t __n,`  
    `const char * __fmt,`  
    `... )`

Like `sprintf()`, but instead of assuming `s` to be of infinite size, no more than `n` characters (including the trailing NUL character) will be converted to `s`.

Returns the number of characters that would have been written to `s` if there were enough space.

**21.9.4.26 `snprintf_P()`** `int snprintf_P (`  
    `char * __s,`  
    `size_t __n,`  
    `const char * __fmt,`  
    `... )`

Variant of `snprintf()` that uses a `fmt` string that resides in program memory.

**21.9.4.27 `sprintf()`** `int sprintf (`  
    `char * __s,`  
    `const char * __fmt,`  
    `... )`

Variant of `printf()` that sends the formatted characters to string `s`.

**21.9.4.28 `sprintf_P()`** `int sprintf_P (`  
    `char * __s,`  
    `const char * __fmt,`  
    `... )`

Variant of `sprintf()` that uses a `fmt` string that resides in program memory.

**21.9.4.29 `sscanf()`** `int sscanf (`  
    `const char * __buf,`  
    `const char * __fmt,`  
    `... )`

The function `sscanf` performs formatted input, reading the input data from the buffer pointed to by `buf`.

See `vfscanf()` for details.

**21.9.4.30 `sscanf_P()`** `int sscanf_P (`  
    `const char * __buf,`  
    `const char * __fmt,`  
    `... )`

Variant of `sscanf()` using a `fmt` string in program memory.

```

21.9.4.31 ungetc() int ungetc (
    int __c,
    FILE * __stream )

```

The `ungetc()` function pushes the character `c` (converted to an unsigned char) back onto the input stream pointed to by `stream`. The pushed-back character will be returned by a subsequent read on the stream.

Currently, only a single character can be pushed back onto the stream.

The `ungetc()` function returns the character pushed back after the conversion, or `EOF` if the operation fails. If the value of the argument `c` character equals `EOF`, the operation will fail and the stream will remain unchanged.

```

21.9.4.32 vfprintf() int vfprintf (
    FILE * __stream,
    const char * __fmt,
    va_list __ap )

```

`vfprintf` is the central facility of the `printf` family of functions. It outputs values to `stream` under control of a format string passed in `fmt`. The actual values to print are passed as a variable argument list `ap`.

`vfprintf` returns the number of characters written to `stream`, or `EOF` in case of an error. Currently, this will only happen if `stream` has not been opened with write intent.

The format string is composed of zero or more directives: ordinary characters (not `%`), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the `%` character. The arguments must properly correspond (after type promotion) with the conversion specifier. After the `%`, the following appear in sequence:

- Zero or more of the following flags:
  - # The value should be converted to an "alternate form". For `c`, `d`, `i`, `s`, and `u` conversions, this option has no effect. For `o` conversions, the precision of the number is increased to force the first character of the output string to a zero (except if a zero value is printed with an explicit precision of zero). For `x` and `X` conversions, a non-zero result has the string ``0x'` (or ``0X'` for `X` conversions) prepended to it.
  - 0 (zero) Zero padding. For all conversions, the converted value is padded on the left with zeros rather than blanks. If a precision is given with a numeric conversion (`d`, `i`, `o`, `u`, `i`, `x`, and `X`), the `0` flag is ignored.
  - - A negative field width flag; the converted value is to be left adjusted on the field boundary. The converted value is padded on the right with blanks, rather than on the left with blanks or zeros. `A -` overrides a `0` if both are given.
  - ' ' (space) A blank should be left before a positive number produced by a signed conversion (`d`, or `i`).
  - + A sign must always be placed before a number produced by a signed conversion. `A +` overrides a space if both are used.
- An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given) to fill out the field width.
- An optional precision, in the form of a period `.` followed by an optional digit string. If the digit string is omitted, the precision is taken as zero. This gives the minimum number of digits to appear for `d`, `i`, `o`, `u`, `x`, and `X` conversions, or the maximum number of characters to be printed from a string for `s` conversions.
- An optional `l` or `h` length modifier, that specifies that the argument for the `d`, `i`, `o`, `u`, `x`, or `X` conversion is a "long int" rather than `int`. The `h` is ignored, as "short int" is equivalent to `int`.
- A character that specifies the type of conversion to be applied.

The conversion specifiers and their meanings are:

- `diouxX` The `int` (or appropriate variant) argument is converted to signed decimal (`d` and `i`), unsigned octal (`o`), unsigned decimal (`u`), or unsigned hexadecimal (`x` and `X`) notation. The letters "abcdef" are used for `x` conversions; the letters "ABCDEF" are used for `X` conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.
- `p` The `void *` argument is taken as an unsigned integer, and converted similarly as a `%#x` command would do.
- `c` The `int` argument is converted to an "unsigned char", and the resulting character is written.
- `s` The "`char *`" argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating NUL character; if a precision is specified, no more than the number specified are written. If a precision is given, no null character need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating NUL character.
- `% A %` is written. No argument is converted. The complete conversion specification is "%%".
- `eE` The double argument is rounded and converted in the format "`[-]d.ddde±dd`" where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal-point character appears. An `E` conversion uses the letter 'E' (rather than 'e') to introduce the exponent. The exponent always contains two digits; if the value is zero, the exponent is 00.
- `fF` The double argument is rounded and converted to decimal notation in the format "`[-]ddd.ddd`", where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no decimal-point character appears. If a decimal point appears, at least one digit appears before it.
- `gG` The double argument is converted in style `f` or `e` (or `F` or `E` for `G` conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style `e` is used if the exponent from its conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.
- `S` Similar to the `s` format, except the pointer is expected to point to a program-memory (ROM) string instead of a RAM string.

In no case does a non-existent or small field width cause truncation of a numeric field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

Since the full implementation of all the mentioned features becomes fairly large, three different flavours of `vfprintf()` can be selected using linker options. The default `vfprintf()` implements all the mentioned functionality except floating point conversions. A minimized version of `vfprintf()` is available that only implements the very basic integer and string conversion facilities, but only the `#` additional option can be specified using conversion flags (these flags are parsed correctly from the format specification, but then simply ignored). This version can be requested using the following [compiler options](#):

```
-Wl,-u,vfprintf -lprintf_min
```

If the full functionality including the floating point conversions is required, the following options should be used:

```
-Wl,-u,vfprintf -lprintf_float -lm
```

#### Limitations:

- The specified width and precision can be at most 255.

## Notes:

- For floating-point conversions, if you link default or minimized version of `vfprintf()`, the symbol `?` will be output and double argument will be skipped. So you output below will not be crashed. For default version the width field and the "pad to left" ( symbol minus ) option will work in this case.
- The `hh` length modifier is ignored (`char` argument is promoted to `int`). More exactly, this realization does not check the number of `h` symbols.
- But the `ll` length modifier will to abort the output, as this realization does not operate `long long` arguments.
- The variable width or precision field (an asterisk `*` symbol) is not realized and will to abort the output.

**21.9.4.33 `vfprintf_P()`** `int vfprintf_P (`  
`FILE * __stream,`  
`const char * __fmt,`  
`va_list __ap )`

Variant of `vfprintf()` that uses a `fmt` string that resides in program memory.

**21.9.4.34 `vfscanf()`** `int vfscanf (`  
`FILE * stream,`  
`const char * fmt,`  
`va_list ap )`

Formatted input. This function is the heart of the **`scanf`** family of functions.

Characters are read from `stream` and processed in a way described by `fmt`. Conversion results will be assigned to the parameters passed via `ap`.

The format string `fmt` is scanned for conversion specifications. Anything that doesn't comprise a conversion specification is taken as text that is matched literally against the input. White space in the format string will match any white space in the data (including none), all other characters match only itself. Processing is aborted as soon as the data and format string no longer match, or there is an error or end-of-file condition on `stream`.

Most conversions skip leading white space before starting the actual conversion.

Conversions are introduced with the character `%`. Possible options can follow the `%`:

- a `*` indicating that the conversion should be performed but the conversion result is to be discarded; no parameters will be processed from `ap`,
- the character `h` indicating that the argument is a pointer to `short int` (rather than `int`),
- the 2 characters `hh` indicating that the argument is a pointer to `char` (rather than `int`).
- the character `l` indicating that the argument is a pointer to `long int` (rather than `int`, for integer type conversions), or a pointer to `float` (for floating point conversions),

In addition, a maximal field width may be specified as a nonzero positive decimal integer, which will restrict the conversion to at most this many characters from the input stream. This field width is limited to at most 255 characters which is also the default value (except for the `c` conversion that defaults to 1).

The following conversion flags are supported:

- `%` Matches a literal `%` character. This is not a conversion.
- `d` Matches an optionally signed decimal integer; the next pointer must be a pointer to `int`.
- `i` Matches an optionally signed integer; the next pointer must be a pointer to `int`. The integer is read in base 16 if it begins with `0x` or `0X`, in base 8 if it begins with `0`, and in base 10 otherwise. Only characters that correspond to the base are used.
- `o` Matches an octal integer; the next pointer must be a pointer to `unsigned int`.
- `u` Matches an optionally signed decimal integer; the next pointer must be a pointer to `unsigned int`.
- `x` Matches an optionally signed hexadecimal integer; the next pointer must be a pointer to `unsigned int`.
- `f` Matches an optionally signed floating-point number; the next pointer must be a pointer to `float`.
- `e`, `g`, `F`, `E`, `G` Equivalent to `f`.
- `s` Matches a sequence of non-white-space characters; the next pointer must be a pointer to `char`, and the array must be large enough to accept all the sequence and the terminating `NUL` character. The input string stops at white space or at the maximum field width, whichever occurs first.
- `c` Matches a sequence of width count characters (default 1); the next pointer must be a pointer to `char`, and there must be enough room for all the characters (no terminating `NUL` is added). The usual skip of leading white space is suppressed. To skip white space first, use an explicit space in the format.
- `[` Matches a nonempty sequence of characters from the specified set of accepted characters; the next pointer must be a pointer to `char`, and there must be enough room for all the characters in the string, plus a terminating `NUL` character. The usual skip of leading white space is suppressed. The string is to be made up of characters in (or not in) a particular set; the set is defined by the characters between the open bracket `[` character and a close bracket `]` character. The set excludes those characters if the first character after the open bracket is a circumflex `^`. To include a close bracket in the set, make it the first character after the open bracket or the circumflex; any other position will end the set. The hyphen character `-` is also special; when placed between two other characters, it adds all intervening characters to the set. To include a hyphen, make it the last character before the final close bracket. For instance, `[^]0-9-]` means the set of *everything except close bracket, zero through nine, and hyphen*. The string ends with the appearance of a character not in the (or, with a circumflex, in) set or when the field width runs out. Note that usage of this conversion enlarges the stack expense.
- `p` Matches a pointer value (as printed by `p` in `printf()`); the next pointer must be a pointer to `void`.
- `n` Nothing is expected; instead, the number of characters consumed thus far from the input is stored through the next pointer, which must be a pointer to `int`. This is not a conversion, although it can be suppressed with the `*` flag.

These functions return the number of input items assigned, which can be fewer than provided for, or even zero, in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned; typically this is due to an invalid input character, such as an alphabetic character for a `d` conversion. The value `EOF` is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions which were successfully completed is returned.

By default, all the conversions described above are available except the floating-point conversions and the width is limited to 255 characters. The float-point conversion will be available in the extended version provided by the library `libscanf_flt.a`. Also in this case the width is not limited (exactly, it is limited to 65535 characters). To link a program against the extended version, use the following compiler flags in the link stage:

```
-Wl,-u,vfscanf -lscanf_flt -lm
```

A third version is available for environments that are tight on space. In addition to the restrictions of the standard one, this version implements no `[%]` specification. This version is provided in the library `libscanf_min.a`, and can be requested using the following options in the link stage:

```
-Wl,-u,vfscanf -lscanf_min -lm
```

```
21.9.4.35 vfscanf_P() int vfscanf_P (
    FILE * __stream,
    const char * __fmt,
    va_list __ap )
```

Variant of [vfscanf\(\)](#) using a `fmt` string in program memory.

```
21.9.4.36 vprintf() int vprintf (
    const char * __fmt,
    va_list __ap )
```

The function `vprintf` performs formatted output to stream `stdout`, taking a variable argument list as in [vprintf\(\)](#).

See [vprintf\(\)](#) for details.

```
21.9.4.37 vscanf() int vscanf (
    const char * __fmt,
    va_list __ap )
```

The function `vscanf` performs formatted input from stream `stdin`, taking a variable argument list as in [vfscanf\(\)](#).

See [vfscanf\(\)](#) for details.

```
21.9.4.38 vsnprintf() int vsnprintf (
    char * __s,
    size_t __n,
    const char * __fmt,
    va_list ap )
```

Like [vsprintf\(\)](#), but instead of assuming `s` to be of infinite size, no more than `n` characters (including the trailing NUL character) will be converted to `s`.

Returns the number of characters that would have been written to `s` if there were enough space.

```
21.9.4.39 vsprintf_P() int vsprintf_P (
    char * __s,
    size_t __n,
    const char * __fmt,
    va_list ap )
```

Variant of [vsnprintf\(\)](#) that uses a `fmt` string that resides in program memory.

```
21.9.4.40 vsprintf() int vsprintf (
    char * __s,
    const char * __fmt,
    va_list ap )
```

Like [sprintf\(\)](#) but takes a variable argument list for the arguments.

```
21.9.4.41 vsprintf_P() int vsprintf_P (  
    char * __s,  
    const char * __fmt,  
    va_list ap )
```

Variant of `vsprintf()` that uses a `fmt` string that resides in program memory.

## 21.10 <stdlib.h>: General utilities

### Data Structures

- struct [div\\_t](#)
- struct [ldiv\\_t](#)

### Macros

- `#define RAND\_MAX 0x7FFF`

### Typedefs

- typedef `int(* \_\_compar\_fn\_t) (const void *, const void *)`

### Functions

- void [abort](#) (void)
- int [abs](#) (int \_\_i)
- long [labs](#) (long \_\_i)
- void \* [bsearch](#) (const void \* \_\_key, const void \* \_\_base, size\_t \_\_nmemb, size\_t \_\_size, int(\* \_\_compar)(const void \*, const void \*))
- [div\\_t](#) [div](#) (int \_\_num, int \_\_denom) `__asm__ ("__divmodhi4")`
- [ldiv\\_t](#) [ldiv](#) (long \_\_num, long \_\_denom) `__asm__ ("__divmodsi4")`
- void [qsort](#) (void \* \_\_base, size\_t \_\_nmemb, size\_t \_\_size, [\\_\\_compar\\_fn\\_t](#) \_\_compar)
- long [strtol](#) (const char \* \_\_nptr, char \*\* \_\_endptr, int \_\_base)
- unsigned long [strtoul](#) (const char \* \_\_nptr, char \*\* \_\_endptr, int \_\_base)
- long [atol](#) (const char \* \_\_s)
- int [atoi](#) (const char \* \_\_s)
- void [exit](#) (int \_\_status)
- void \* [malloc](#) (size\_t \_\_size)
- void [free](#) (void \* \_\_ptr)
- void \* [calloc](#) (size\_t \_\_nele, size\_t \_\_size)
- void \* [realloc](#) (void \* \_\_ptr, size\_t \_\_size)
- float [strtof](#) (const char \* \_\_nptr, char \*\* \_\_endptr)
- double [strtod](#) (const char \* \_\_nptr, char \*\* \_\_endptr)
- long double [strtold](#) (const char \* \_\_nptr, char \*\* \_\_endptr)
- int [atexit](#) (void(\*func)(void))
- float [atoff](#) (const char \* \_\_nptr)
- double [atof](#) (const char \* \_\_nptr)
- long double [atofl](#) (const char \* \_\_nptr)
- int [rand](#) (void)
- void [srand](#) (unsigned int \_\_seed)
- int [rand\\_r](#) (unsigned long \* \_\_ctx)

**Variables**

- `size_t __malloc_margin`
- `char * __malloc_heap_start`
- `char * __malloc_heap_end`

**Non-standard (i.e. non-ISO C) functions.**

- `char * ltoa` (long val, char \*s, int radix)
- `char * utoa` (unsigned int val, char \*s, int radix)
- `char * ultoa` (unsigned long val, char \*s, int radix)
- `long random` (void)
- `void srandom` (unsigned long \_\_seed)
- `long random_r` (unsigned long \* \_\_ctx)
- `char * itoa` (int val, char \*s, int radix)
- `#define RANDOM_MAX` 0x7FFFFFFF

**Conversion functions for double arguments.**

- `char * fostre` (float \_\_val, char \*\_\_s, unsigned char \_\_prec, unsigned char \_\_flags)
- `char * dtostre` (double \_\_val, char \*\_\_s, unsigned char \_\_prec, unsigned char \_\_flags)
- `char * ldostre` (long double \_\_val, char \*\_\_s, unsigned char \_\_prec, unsigned char \_\_flags)
- `char * ftostrf` (float \_\_val, signed char \_\_width, unsigned char \_\_prec, char \*\_\_s)
- `char * dtostrf` (double \_\_val, signed char \_\_width, unsigned char \_\_prec, char \*\_\_s)
- `char * ldostrf` (long double \_\_val, signed char \_\_width, unsigned char \_\_prec, char \*\_\_s)
- `#define DTOSTR_ALWAYS_SIGN` 0x01 /\* put '+' or '-' for positives \*/
- `#define DTOSTR_PLUS_SIGN` 0x02 /\* put '+' rather than '-' \*/
- `#define DTOSTR_UPPERCASE` 0x04 /\* put 'E' rather 'e' \*/
- `#define EXIT_SUCCESS` 0
- `#define EXIT_FAILURE` 1

**21.10.1 Detailed Description**

```
#include <stdlib.h>
```

This file declares some basic C macros and functions as defined by the ISO standard, plus some AVR-specific extensions.

**21.10.2 Macro Definition Documentation**

**21.10.2.1 DTOSTR\_ALWAYS\_SIGN** `#define DTOSTR_ALWAYS_SIGN 0x01 /* put '+' or '-' for positives */`

Bit value that can be passed in `flags` to `fostre()`, `dtostre()` and `ldostre()`.

**21.10.2.2 DTOSTR\_PLUS\_SIGN** `#define DTOSTR_PLUS_SIGN 0x02 /* put '+' rather than '-' */`

Bit value that can be passed in `flags` to `fostre()`, `dtostre()` and `ldostre()`.



**21.10.2.3 DOSTR\_UPPERCASE** `#define DOSTR_UPPERCASE 0x04 /* put 'E' rather 'e' */`

Bit value that can be passed in `flags` to `fostre()`, `dstostre()` and `ldtostre()`.

**21.10.2.4 EXIT\_FAILURE** `#define EXIT_FAILURE 1`

Unsuccessful termination for `exit()`; evaluates to a non-zero value.

**21.10.2.5 EXIT\_SUCCESS** `#define EXIT_SUCCESS 0`

Successful termination for `exit()`; evaluates to 0.

**21.10.2.6 RAND\_MAX** `#define RAND_MAX 0x7FFF`

Highest number that can be generated by `rand()`.

**21.10.2.7 RANDOM\_MAX** `#define RANDOM_MAX 0x7FFFFFFF`

Highest number that can be generated by `random()`.

### 21.10.3 Typedef Documentation

**21.10.3.1 \_\_compar\_fn\_t** `typedef int (* __compar_fn_t) (const void *, const void *)`

Comparison function type for `qsort()`, just for convenience.

### 21.10.4 Function Documentation

**21.10.4.1 abort()** `void abort (`  
    `void )`

The `abort()` function causes abnormal program termination to occur. This realization disables interrupts and jumps to `_exit()` function with argument equal to 1. In the limited AVR environment, execution is effectively halted by entering an infinite loop.

**21.10.4.2 abs()** `int abs (`  
    `int __i )`

The `abs()` function computes the absolute value of the integer `i`.

#### Note

The `abs()` and `labs()` functions are builtins of gcc.

**21.10.4.3 atexit()** `int atexit (`  
`void(*) (void) func )`

The `atexit()` function registers function `func` to be run as part of the `exit()` function during `.fini8`. `atexit()` calls `malloc()`.

**21.10.4.4 atof()** `double atof (`  
`const char * nptr )`

The `atof()` function converts the initial portion of the string pointed to by `nptr` to `double` representation.

It is equivalent to calling  
`strtod(nptr, (char**) 0);`

**21.10.4.5 atoff()** `float atoff (`  
`const char * nptr )`

The `atoff()` function converts the initial portion of the string pointed to by `nptr` to `float` representation.

It is equivalent to calling  
`strtof(nptr, (char**) 0);`

**21.10.4.6 atofl()** `long double atofl (`  
`const char * nptr )`

The `atofl()` function converts the initial portion of the string pointed to by `nptr` to `long double` representation.

It is equivalent to calling  
`strtold(nptr, (char**) 0);`

**21.10.4.7 atoi()** `int atoi (`  
`const char * __s )`

The `atoi()` function converts the initial portion of the string pointed to by `s` to integer representation. In contrast to `(int) strtol(s, (char **) NULL, 10);`

this function does not detect overflow (`errno` is not changed and the result value is not predictable), uses smaller memory (flash and stack) and works more quickly.

**21.10.4.8 atol()** `long atol (`  
`const char * __s )`

The `atol()` function converts the initial portion of the string pointed to by `s` to long integer representation. In contrast to

`strtol(s, (char **) NULL, 10);`

this function does not detect overflow (`errno` is not changed and the result value is not predictable), uses smaller memory (flash and stack) and works more quickly.

**21.10.4.9 bsearch()** `void * bsearch (`  
    `const void * __key,`  
    `const void * __base,`  
    `size_t __nmemb,`  
    `size_t __size,`  
    `int (*)(const void *, const void *) __compar )`

The `bsearch()` function searches an array of `nmemb` objects, the initial member of which is pointed to by `base`, for a member that matches the object pointed to by `key`. The size of each member of the array is specified by `size`.

The contents of the array should be in ascending sorted order according to the comparison function referenced by `compar`. The `compar` routine is expected to have two arguments which point to the key object and to an array member, in that order, and should return an integer less than, equal to, or greater than zero if the key object is found, respectively, to be less than, to match, or be greater than the array member.

The `bsearch()` function returns a pointer to a matching member of the array, or a null pointer if no match is found. If two members compare as equal, which member is matched is unspecified.

**21.10.4.10 calloc()** `void * calloc (`  
    `size_t __nele,`  
    `size_t __size )`

Allocate `nele` elements of `size` each. Identical to calling `malloc()` using `nele * size` as argument, except the allocated memory will be cleared to zero.

**21.10.4.11 div()** `div_t div (`  
    `int __num,`  
    `int __denom )`

The `div()` function computes the value `num/denom` and returns the quotient and remainder in a structure named `div_t` that contains two `int` members named `quot` and `rem`.

**21.10.4.12 dtostre()** `char * dtostre (`  
    `double __val,`  
    `char * __s,`  
    `unsigned char __prec,`  
    `unsigned char __flags )`

The `dtostre()` function is similar to the `fstre()` function, except that it converts a `double` value instead of a `float` value.

`dtostre()` is currently only supported when `double` is a 32-bit type.

**21.10.4.13 dtostrf()** `char * dtostrf (`  
    `double __val,`  
    `signed char __width,`  
    `unsigned char __prec,`  
    `char * __s )`

The `dtostrf()` function is similar to the `fstrof()` function, except that converts a `double` value instead of a `float` value.

`ldtostre()` is currently only supported when `double` is a 32-bit type.

**21.10.4.14** `exit()` `void exit (`  
`int __status )`

The `exit()` function terminates the application. Since there is no environment to return to, `status` is ignored, and code execution will eventually reach an infinite loop, thereby effectively halting all code processing. Before entering the infinite loop, interrupts are globally disabled.

Global destructors will be called before halting execution, see the `.fini` sections.

**21.10.4.15** `free()` `void free (`  
`void * __ptr )`

The `free()` function causes the allocated memory referenced by `ptr` to be made available for future allocations. If `ptr` is `NULL`, no action occurs.

**21.10.4.16** `fstre()` `char * fstre (`  
`float __val,`  
`char * __s,`  
`unsigned char __prec,`  
`unsigned char __flags )`

The `fstre()` function converts the `float` value passed in `val` into an ASCII representation that will be stored under `s`. The caller is responsible for providing sufficient storage in `s`.

Conversion is done in the format "`[-]d.ddde±dd`" where there is one digit before the decimal-point character and the number of digits after it is equal to the precision `prec`; if the precision is zero, no decimal-point character appears. If `flags` has the `DTOSTR_UPPERCASE` bit set, the letter 'E' (rather than 'e' ) will be used to introduce the exponent. The exponent always contains two digits; if the value is zero, the exponent is "00".

If `flags` has the `DTOSTR_ALWAYS_SIGN` bit set, a space character will be placed into the leading position for positive numbers.

If `flags` has the `DTOSTR_PLUS_SIGN` bit set, a plus sign will be used instead of a space character in this case.

The `fstre()` function returns the pointer to the converted string `s`.

**21.10.4.17** `fstorf()` `char * fstorf (`  
`float __val,`  
`signed char __width,`  
`unsigned char __prec,`  
`char * __s )`

The `fstorf()` function converts the `float` value passed in `val` into an ASCII representation that will be stored in `s`. The caller is responsible for providing sufficient storage in `s`.

Conversion is done in the format "`[-]d.ddd`". The minimum field width of the output string (including the possible '.' and the possible sign for negative values) is given in `width`, and `prec` determines the number of digits after the decimal sign. `width` is signed value, negative for left adjustment.

The `fstorf()` function returns the pointer to the converted string `s`.

**21.10.4.18 itoa()** `char * itoa (`  
    `int val,`  
    `char * s,`  
    `int radix )`

Convert an integer to a string.

The function `itoa()` converts the integer value from `val` into an ASCII representation that will be stored under `s`. The caller is responsible for providing sufficient storage in `s`.

#### Note

The minimal size of the buffer `s` depends on the choice of `radix`. For example, if the `radix` is 2 (binary), you need to supply a buffer with a minimal length of `8 * sizeof (int) + 1` characters, i.e. one character for each bit plus one for the string terminator. Using a larger `radix` will require a smaller minimal buffer size.

#### Warning

If the buffer is too small, you risk a buffer overflow.

Conversion is done using the `radix` as base, which may be a number between 2 (binary conversion) and up to 36. If `radix` is greater than 10, the next digit after '9' will be the letter 'a'.

If `radix` is 10 and `val` is negative, a minus sign will be prepended.

The `itoa()` function returns the pointer passed as `s`.

**21.10.4.19 labs()** `long labs (`  
    `long __i )`

The `labs()` function computes the absolute value of the long integer `i`.

#### Note

The `abs()` and `labs()` functions are builtins of gcc.

**21.10.4.20 ldiv()** `ldiv_t ldiv (`  
    `long __num,`  
    `long __denom )`

The `ldiv()` function computes the value `num/denom` and returns the quotient and remainder in a structure named `ldiv_t` that contains two long integer members named `quot` and `rem`.

**21.10.4.21 ldostre()** `char * ldostre (`  
    `long double __val,`  
    `char * __s,`  
    `unsigned char __prec,`  
    `unsigned char __flags )`

The `ldostre()` function is similar to the `fstre()` function, except that it converts a long double value instead of a float value.

`ldostre()` is currently only supported when long double is a 32-bit type.

```
21.10.4.22 ldtostrf() char * ldtostrf (
    long double __val,
    signed char __width,
    unsigned char __prec,
    char * __s )
```

The `ldtostrf()` function is similar to the `ftostrf()` function, except that converts a long double value instead of a float value.

`ldtostre()` is currently only supported when long double is a 32-bit type.

```
21.10.4.23 ltoa() char * ltoa (
    long val,
    char * s,
    int radix )
```

Convert a long integer to a string.

The function `ltoa()` converts the long integer value from `val` into an ASCII representation that will be stored under `s`. The caller is responsible for providing sufficient storage in `s`.

#### Note

The minimal size of the buffer `s` depends on the choice of `radix`. For example, if the `radix` is 2 (binary), you need to supply a buffer with a minimal length of  $8 * \text{sizeof}(\text{long int}) + 1$  characters, i.e. one character for each bit plus one for the string terminator. Using a larger `radix` will require a smaller minimal buffer size.

#### Warning

If the buffer is too small, you risk a buffer overflow.

Conversion is done using the `radix` as base, which may be a number between 2 (binary conversion) and up to 36. If `radix` is greater than 10, the next digit after '9' will be the letter 'a'.

If `radix` is 10 and `val` is negative, a minus sign will be prepended.

The `ltoa()` function returns the pointer passed as `s`.

```
21.10.4.24 malloc() void * malloc (
    size_t __size )
```

The `malloc()` function allocates `size` bytes of memory. If `malloc()` fails, a NULL pointer is returned.

Note that `malloc()` does *not* initialize the returned memory to zero bytes.

See the chapter about [malloc\(\) usage](#) for implementation details.

**21.10.4.25 `qsort()`** `void qsort (`  
    `void * __base,`  
    `size_t __nmemb,`  
    `size_t __size,`  
    `__compar_fn_t __compar )`

The `qsort()` function is a modified partition-exchange sort, or quicksort.

The `qsort()` function sorts an array of `nmemb` objects, the initial member of which is pointed to by `base`. The size of each object is specified by `size`. The contents of the array `base` are sorted in ascending order according to a comparison function pointed to by `compar`, which requires two arguments pointing to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

**21.10.4.26 `rand()`** `int rand (`  
    `void )`

The `rand()` function computes a sequence of pseudo-random integers in the range of 0 to `RAND_MAX` (as defined by the header file `<stdlib.h>`).

The `srand()` function sets its argument `seed` as the seed for a new sequence of pseudo-random numbers to be returned by `rand()`. These sequences are repeatable by calling `srand()` with the same seed value.

If no seed value is provided, the functions are automatically seeded with a value of 1.

In compliance with the C standard, these functions operate on `int` arguments. Since the underlying algorithm already uses 32-bit calculations, this causes a loss of precision. See `random()` for an alternate set of functions that retains full 32-bit precision.

**21.10.4.27 `rand_r()`** `int rand_r (`  
    `unsigned long * __ctx )`

Variant of `rand()` that stores the context in the user-supplied variable located at `ctx` instead of a static library variable so the function becomes re-entrant.

**21.10.4.28 `random()`** `long random (`  
    `void )`

The `random()` function computes a sequence of pseudo-random integers in the range of 0 to `RANDOM_MAX` (as defined by the header file `<stdlib.h>`).

The `srandom()` function sets its argument `seed` as the seed for a new sequence of pseudo-random numbers to be returned by `rand()`. These sequences are repeatable by calling `srandom()` with the same seed value.

If no seed value is provided, the functions are automatically seeded with a value of 1.

**21.10.4.29 `random_r()`** `long random_r (`  
    `unsigned long * __ctx )`

Variant of `random()` that stores the context in the user-supplied variable located at `ctx` instead of a static library variable so the function becomes re-entrant.

```
21.10.4.30 realloc() void * realloc (
    void * __ptr,
    size_t __size )
```

The `realloc()` function tries to change the size of the region allocated at `ptr` to the new `size` value. It returns a pointer to the new region. The returned pointer might be the same as the old pointer, or a pointer to a completely different region.

The contents of the returned region up to either the old or the new size value (whatever is less) will be identical to the contents of the old region, even in case a new region had to be allocated.

It is acceptable to pass `ptr` as `NULL`, in which case `realloc()` will behave identical to `malloc()`.

If the new memory cannot be allocated, `realloc()` returns `NULL`, and the region at `ptr` will not be changed.

```
21.10.4.31 srand() void srand (
    unsigned int __seed )
```

Pseudo-random number generator seeding; see [rand\(\)](#).

```
21.10.4.32 srandom() void srandom (
    unsigned long __seed )
```

Pseudo-random number generator seeding; see [random\(\)](#).

```
21.10.4.33 strtod() double strtod (
    const char * __nptr,
    char ** __endptr )
```

The `strtod()` function is similar to `strtof()`, except that the conversion result is of type `double` instead of `float`.

`strtod()` is currently only supported when `double` is a 32-bit type.

```
21.10.4.34 strtof() float strtof (
    const char * nptr,
    char ** endptr )
```

The `strtof()` function converts the initial portion of the string pointed to by `nptr` to `float` representation.

The expected form of the string is an optional plus ( '+' ) or minus sign ( '-' ) followed by a sequence of digits optionally containing a decimal-point character, optionally followed by an exponent. An exponent consists of an 'E' or 'e', followed by an optional plus or minus sign, followed by a sequence of digits.

Leading white-space characters in the string are skipped.

The `strtof()` function returns the converted value, if any.

If `endptr` is not `NULL`, a pointer to the character after the last character used in the conversion is stored in the location referenced by `endptr`.

If no conversion is performed, zero is returned and the value of `nptr` is stored in the location referenced by `endptr`.

If the correct value would cause overflow, plus or minus `INFINITY` is returned (according to the sign of the value), and `ERANGE` is stored in `errno`. If the correct value would cause underflow, zero is returned and `ERANGE` is stored in `errno`.



```
21.10.4.35 strtol() long strtol (
    const char * __nptr,
    char ** __endptr,
    int __base )
```

The `strtol()` function converts the string in `nptr` to a long value. The conversion is done according to the given base, which must be between 2 and 36 inclusive, or be the special value 0.

The string may begin with an arbitrary amount of white space (as determined by `isspace()`) followed by a single optional '+' or '-' sign. If `base` is zero or 16, the string may then include a "0x" prefix, and the number will be read in base 16; otherwise, a zero base is taken as 10 (decimal) unless the next character is '0', in which case it is taken as 8 (octal).

The remainder of the string is converted to a long value in the obvious manner, stopping at the first character which is not a valid digit in the given base. (In bases above 10, the letter 'A' in either upper or lower case represents 10, 'B' represents 11, and so forth, with 'Z' representing 35.)

If `endptr` is not NULL, `strtol()` stores the address of the first invalid character in `*endptr`. If there were no digits at all, however, `strtol()` stores the original value of `nptr` in `endptr`. (Thus, if `*nptr` is not '\0' but `**endptr` is '\0' on return, the entire string was valid.)

The `strtol()` function returns the result of the conversion, unless the value would underflow or overflow. If no conversion could be performed, 0 is returned. If an overflow or underflow occurs, `errno` is set to `ERANGE` and the function return value is clamped to `LONG_MIN` or `LONG_MAX`, respectively.

```
21.10.4.36 strtold() long double strtold (
    const char * __nptr,
    char ** __endptr )
```

The `strtold()` function is similar to `strtod()`, except that the conversion result is of type `long double` instead of `float`.

`strtold()` is currently only supported when `long double` is a 32-bit type.

```
21.10.4.37 strtoul() unsigned long strtoul (
    const char * __nptr,
    char ** __endptr,
    int __base )
```

The `strtoul()` function converts the string in `nptr` to an unsigned long value. The conversion is done according to the given base, which must be between 2 and 36 inclusive, or be the special value 0.

The string may begin with an arbitrary amount of white space (as determined by `isspace()`) followed by a single optional '+' or '-' sign. If `base` is zero or 16, the string may then include a "0x" prefix, and the number will be read in base 16; otherwise, a zero base is taken as 10 (decimal) unless the next character is '0', in which case it is taken as 8 (octal).

The remainder of the string is converted to an unsigned long value in the obvious manner, stopping at the first character which is not a valid digit in the given base. (In bases above 10, the letter 'A' in either upper or lower case represents 10, 'B' represents 11, and so forth, with 'Z' representing 35.)

If `endptr` is not NULL, `strtoul()` stores the address of the first invalid character in `*endptr`. If there were no digits at all, however, `strtoul()` stores the original value of `nptr` in `endptr`. (Thus, if `*nptr` is not '\0' but `**endptr` is '\0' on return, the entire string was valid.)

The `strtoul()` function return either the result of the conversion or, if there was a leading minus sign, the negation of the result of the conversion, unless the original (non-negated) value would overflow; in the latter case, `strtoul()` returns `ULONG_MAX`, and `errno` is set to `ERANGE`. If no conversion could be performed, 0 is returned.

```
21.10.4.38 ultoa() char * ultoa (  
    unsigned long val,  
    char * s,  
    int radix )
```

Convert an unsigned long integer to a string.

The function `ultoa()` converts the unsigned long integer value from `val` into an ASCII representation that will be stored under `s`. The caller is responsible for providing sufficient storage in `s`.

#### Note

The minimal size of the buffer `s` depends on the choice of `radix`. For example, if the `radix` is 2 (binary), you need to supply a buffer with a minimal length of  $8 * \text{sizeof}(\text{unsigned long int}) + 1$  characters, i.e. one character for each bit plus one for the string terminator. Using a larger `radix` will require a smaller minimal buffer size.

#### Warning

If the buffer is too small, you risk a buffer overflow.

Conversion is done using the `radix` as base, which may be a number between 2 (binary conversion) and up to 36. If `radix` is greater than 10, the next digit after '9' will be the letter 'a'.

The `ultoa()` function returns the pointer passed as `s`.

```
21.10.4.39 utoa() char * utoa (  
    unsigned int val,  
    char * s,  
    int radix )
```

Convert an unsigned integer to a string.

The function `utoa()` converts the unsigned integer value from `val` into an ASCII representation that will be stored under `s`. The caller is responsible for providing sufficient storage in `s`.

#### Note

The minimal size of the buffer `s` depends on the choice of `radix`. For example, if the `radix` is 2 (binary), you need to supply a buffer with a minimal length of  $8 * \text{sizeof}(\text{unsigned int}) + 1$  characters, i.e. one character for each bit plus one for the string terminator. Using a larger `radix` will require a smaller minimal buffer size.

#### Warning

If the buffer is too small, you risk a buffer overflow.

Conversion is done using the `radix` as base, which may be a number between 2 (binary conversion) and up to 36. If `radix` is greater than 10, the next digit after '9' will be the letter 'a'.

The `utoa()` function returns the pointer passed as `s`.

## 21.10.5 Variable Documentation

**21.10.5.1** `__malloc_heap_end` `char* __malloc_heap_end` [extern]

`malloc()` [tunable](#).

**21.10.5.2** `__malloc_heap_start` `char* __malloc_heap_start` [extern]

`malloc()` [tunable](#).

**21.10.5.3** `__malloc_margin` `size_t __malloc_margin` [extern]

`malloc()` [tunable](#).

## 21.11 <string.h>: Strings

### Macros

- `#define _FFS(x)`

### Functions

- `int ffs` (`int __val`)
- `int ffsi` (`long __val`)
- `int ffsll` (`long long __val`)
- `void * memccpy` (`void *`, `const void *`, `int`, `size_t`)
- `void * memchr` (`const void *`, `int`, `size_t`)
- `int memcmp` (`const void *`, `const void *`, `size_t`)
- `void * memcpy` (`void *`, `const void *`, `size_t`)
- `void * memmem` (`const void *`, `size_t`, `const void *`, `size_t`)
- `void * memmove` (`void *`, `const void *`, `size_t`)
- `void * memrchr` (`const void *`, `int`, `size_t`)
- `void * memset` (`void *`, `int`, `size_t`)
- `char * strcat` (`char *`, `const char *`)
- `char * strchr` (`const char *`, `int`)
- `char * strchrnul` (`const char *`, `int`)
- `int strcmp` (`const char *`, `const char *`)
- `char * strcpy` (`char *`, `const char *`)
- `int strcasecmp` (`const char *`, `const char *`)
- `char * strcasestr` (`const char *`, `const char *`)
- `size_t strcspn` (`const char *__s`, `const char *__reject`)
- `char * strdup` (`const char *s1`)
- `char * strndup` (`const char *s`, `size_t n`)
- `size_t strlcat` (`char *`, `const char *`, `size_t`)
- `size_t strlcpy` (`char *`, `const char *`, `size_t`)
- `size_t strlen` (`const char *`)
- `char * strlwr` (`char *`)
- `char * strncat` (`char *`, `const char *`, `size_t`)
- `int strncmp` (`const char *`, `const char *`, `size_t`)
- `char * strncpy` (`char *`, `const char *`, `size_t`)
- `int strncasecmp` (`const char *`, `const char *`, `size_t`)
- `size_t strnlen` (`const char *`, `size_t`)
- `char * strpbrk` (`const char *__s`, `const char *__accept`)
- `char * strrchr` (`const char *`, `int`)
- `char * strev` (`char *`)
- `char * strsep` (`char **`, `const char *`)
- `size_t strspn` (`const char *__s`, `const char *__accept`)
- `char * strstr` (`const char *`, `const char *`)
- `char * strtok` (`char *`, `const char *`)
- `char * strtok_r` (`char *`, `const char *`, `char **`)
- `char *strupr` (`char *`)

### 21.11.1 Detailed Description

```
#include <string.h>
```

The string functions perform string operations on `NULL` terminated strings.

#### Note

If the strings you are working on resident in program space (flash), you will need to use the string functions described in [<avr/pgmspace.h>: Program Space Utilities](#).

### 21.11.2 Macro Definition Documentation

**21.11.2.1** `_FFS`

```
#define _FFS(  
    x )
```

This macro finds the first (least significant) bit set in the input value.

This macro is very similar to the function `ffs()` except that it evaluates its argument at compile-time, so it should only be applied to compile-time constant expressions where it will reduce to a constant itself. Application of this macro to expressions that are not constant at compile-time is not recommended, and might result in a huge amount of code generated.

#### Returns

The `_FFS()` macro returns the position of the first (least significant) bit set in the word `val`, or 0 if no bits are set. The least significant bit is position 1. Only 16 bits of argument are evaluated.

### 21.11.3 Function Documentation

**21.11.3.1** `ffs()`

```
int ffs (  
    int val )
```

This function finds the first (least significant) bit set in the input value.

#### Returns

The `ffs()` function returns the position of the first (least significant) bit set in the word `val`, or 0 if no bits are set. The least significant bit is position 1.

#### Note

For expressions that are constant at compile time, consider using the `_FFS` macro instead.

**21.11.3.2 ffs1()** `int ffs1 (`  
`long __val )`

Same as [ffs\(\)](#), for an argument of type `long`.

**21.11.3.3 ffsll()** `int ffsll (`  
`long long __val )`

Same as [ffs\(\)](#), for an argument of type `long long`.

**21.11.3.4 memccpy()** `void * memccpy (`  
`void * dest,`  
`const void * src,`  
`int val,`  
`size_t len )`

Copy memory area.

The [memccpy\(\)](#) function copies no more than `len` bytes from memory area `src` to memory area `dest`, stopping when the character `val` is found.

#### Returns

The [memccpy\(\)](#) function returns a pointer to the next character in `dest` after `val`, or `NULL` if `val` was not found in the first `len` characters of `src`.

**21.11.3.5 memchr()** `void * memchr (`  
`const void * src,`  
`int val,`  
`size_t len )`

Scan memory for a character.

The [memchr\(\)](#) function scans the first `len` bytes of the memory area pointed to by `src` for the character `val`. The first byte to match `val` (interpreted as an unsigned character) stops the operation.

#### Returns

The [memchr\(\)](#) function returns a pointer to the matching byte or `NULL` if the character does not occur in the given memory area.

```
21.11.3.6 memcmp() int memcmp (  
    const void * s1,  
    const void * s2,  
    size_t len )
```

Compare memory areas.

The `memcmp()` function compares the first `len` bytes of the memory areas `s1` and `s2`. The comparison is performed using unsigned char operations.

#### Returns

The `memcmp()` function returns an integer less than, equal to, or greater than zero if the first `len` bytes of `s1` is found, respectively, to be less than, to match, or be greater than the first `len` bytes of `s2`.

#### Note

Be sure to store the result in a 16 bit variable since you may get incorrect results if you use an unsigned char or char due to truncation.

#### Warning

This function is not -mint8 compatible, although if you only care about testing for equality, this function should be safe to use.

```
21.11.3.7 memcpy() void * memcpy (  
    void * dest,  
    const void * src,  
    size_t len )
```

Copy a memory area.

The `memcpy()` function copies `len` bytes from memory area `src` to memory area `dest`. The memory areas may not overlap. Use `memmove()` if the memory areas do overlap.

#### Returns

The `memcpy()` function returns a pointer to `dest`.

```
21.11.3.8 memmem() void * memmem (  
    const void * s1,  
    size_t len1,  
    const void * s2,  
    size_t len2 )
```

The `memmem()` function finds the start of the first occurrence of the substring `s2` of length `len2` in the memory area `s1` of length `len1`.

#### Returns

The `memmem()` function returns a pointer to the beginning of the substring, or `NULL` if the substring is not found. If `len2` is zero, the function returns `s1`.

**21.11.3.9 memmove()** `void * memmove (`  
    `void * dest,`  
    `const void * src,`  
    `size_t len )`

Copy memory area.

The `memmove()` function copies `len` bytes from memory area `src` to memory area `dest`. The memory areas may overlap.

#### Returns

The `memmove()` function returns a pointer to `dest`.

**21.11.3.10 memrchr()** `void * memrchr (`  
    `const void * src,`  
    `int val,`  
    `size_t len )`

The `memrchr()` function is like the `memchr()` function, except that it searches backwards from the end of the `len` bytes pointed to by `src` instead of forwards from the front. (Glibc, GNU extension.)

#### Returns

The `memrchr()` function returns a pointer to the matching byte or `NULL` if the character does not occur in the given memory area.

**21.11.3.11 memset()** `void * memset (`  
    `void * dest,`  
    `int val,`  
    `size_t len )`

Fill memory with a constant byte.

The `memset()` function fills the first `len` bytes of the memory area pointed to by `dest` with the constant byte `val`.

#### Returns

The `memset()` function returns a pointer to the memory area `dest`.

```
21.11.3.12 strcasecmp() int strcasecmp (  
    const char * s1,  
    const char * s2 )
```

Compare two strings ignoring case.

The [strcasecmp\(\)](#) function compares the two strings `s1` and `s2`, ignoring the case of the characters.

#### Returns

The [strcasecmp\(\)](#) function returns an integer less than, equal to, or greater than zero if `s1` is found, respectively, to be less than, to match, or be greater than `s2`. A consequence of the ordering used by [strcasecmp\(\)](#) is that if `s1` is an initial substring of `s2`, then `s1` is considered to be "less than" `s2`.

```
21.11.3.13 strcasestr() char * strcasestr (  
    const char * s1,  
    const char * s2 )
```

The [strcasestr\(\)](#) function finds the first occurrence of the substring `s2` in the string `s1`. This is like [strstr\(\)](#), except that it ignores case of alphabetic symbols in searching for the substring. (Glibc, GNU extension.)

#### Returns

The [strcasestr\(\)](#) function returns a pointer to the beginning of the substring, or `NULL` if the substring is not found. If `s2` points to a string of zero length, the function returns `s1`.

```
21.11.3.14 strcat() char * strcat (  
    char * dest,  
    const char * src )
```

Concatenate two strings.

The [strcat\(\)](#) function appends the `src` string to the `dest` string overwriting the `'\0'` character at the end of `dest`, and then adds a terminating `'\0'` character. The strings may not overlap, and the `dest` string must have enough space for the result.

#### Returns

The [strcat\(\)](#) function returns a pointer to the resulting string `dest`.

```
21.11.3.15 strchr() char * strchr (  
    const char * src,  
    int val )
```

Locate character in string.

#### Returns

The [strchr\(\)](#) function returns a pointer to the first occurrence of the character `val` in the string `src`, or `NULL` if the character is not found.

Here "character" means "byte" – these functions do not work with wide or multi-byte characters.



**21.11.3.16 strchrnul()** `char * strchrnul (`  
    `const char * s,`  
    `int c )`

The `strchrnul()` function is like `strchr()` except that if `c` is not found in `s`, then it returns a pointer to the null byte at the end of `s`, rather than `NULL`. (Glibc, GNU extension.)

#### Returns

The `strchrnul()` function returns a pointer to the matched character, or a pointer to the null byte at the end of `s` (i.e., `s+strlen(s)`) if the character is not found.

**21.11.3.17 strcmp()** `int strcmp (`  
    `const char * s1,`  
    `const char * s2 )`

Compare two strings.

The `strcmp()` function compares the two strings `s1` and `s2`.

#### Returns

The `strcmp()` function returns an integer less than, equal to, or greater than zero if `s1` is found, respectively, to be less than, to match, or be greater than `s2`. A consequence of the ordering used by `strcmp()` is that if `s1` is an initial substring of `s2`, then `s1` is considered to be "less than" `s2`.

**21.11.3.18 strcpy()** `char * strcpy (`  
    `char * dest,`  
    `const char * src )`

Copy a string.

The `strcpy()` function copies the string pointed to by `src` (including the terminating `\0` character) to the array pointed to by `dest`. The strings may not overlap, and the destination string `dest` must be large enough to receive the copy.

#### Returns

The `strcpy()` function returns a pointer to the destination string `dest`.

#### Note

If the destination string of a `strcpy()` is not large enough (that is, if the programmer was stupid/lazy, and failed to check the size before copying) then anything might happen. Overflowing fixed length strings is a favourite cracker technique.

**21.11.3.19 strcspn()** `size_t strcspn (`  
    `const char * s,`  
    `const char * reject )`

The `strcspn()` function calculates the length of the initial segment of `s` which consists entirely of characters not in `reject`.

#### Returns

The `strcspn()` function returns the number of characters in the initial segment of `s` which are not in the string `reject`. The terminating zero is not considered as a part of string.

**21.11.3.20 strdup()** `char * strdup (`  
    `const char * s1 )`

Duplicate a string.

The `strdup()` function allocates memory and copies into it the string addressed by `s1`, including the terminating null character.

#### Warning

The `strdup()` function calls `malloc()` to allocate the memory for the duplicated string! The user is responsible for freeing the memory by calling `free()`.

#### Returns

The `strdup()` function returns a pointer to the resulting string `dest`. If `malloc()` cannot allocate enough storage for the string, `strdup()` will return `NULL`.

#### Warning

Be sure to check the return value of the `strdup()` function to make sure that the function has succeeded in allocating the memory!

**21.11.3.21 strlcat()** `size_t strlcat (`  
    `char * dst,`  
    `const char * src,`  
    `size_t siz )`

Concatenate two strings.

Appends `src` to string `dst` of size `siz` (unlike `strncat()`, `siz` is the full size of `dst`, not space left). At most `siz-1` characters will be copied. Always `'\0'` terminated (unless `siz <= strlen(dst)`).

#### Returns

The `strlcat()` function returns `strlen(src) + MIN(siz, strlen(initial dst))`. If `retval >= siz`, truncation occurred.

Appends `src` to string `dst` of size `siz` (unlike `strncat()`, `siz` is the full size of `dst`, not space left). At most `siz-1` characters will be copied. Always `NULL` terminates (unless `siz <= strlen(dst)`).

#### Returns

The `strlcat()` function returns `strlen(src) + MIN(siz, strlen(initial dst))`. If `retval >= siz`, truncation occurred.

**21.11.3.22 strcpy()** `size_t strcpy (`  
    `char * dst,`  
    `const char * src,`  
    `size_t siz )`

Copy a string.

Copy `src` to string `dst` of size `siz`. At most `siz-1` characters will be copied. Always `'\0'` terminated (unless `siz == 0`).

#### Returns

The `strcpy()` function returns `strlen(src)`. If `retval >= siz`, truncation occurred.

Copy `src` to string `dst` of size `siz`. At most `siz-1` characters will be copied. Always `NULL` terminates (unless `siz == 0`).

#### Returns

The `strcpy()` function returns `strlen(src)`. If `retval >= siz`, truncation occurred.

**21.11.3.23 strlen()** `size_t strlen (`  
    `const char * src )`

Calculate the length of a string.

The `strlen()` function calculates the length of the string `src`, not including the terminating `'\0'` character.

#### Returns

The `strlen()` function returns the number of characters in `src`.

**21.11.3.24 strlwr()** `char * strlwr (`  
    `char * s )`

Convert a string to lower case.

The `strlwr()` function will convert a string to lower case. Only the upper case alphabetic characters [A .. Z] are converted. Non-alphabetic characters will not be changed.

#### Returns

The `strlwr()` function returns a pointer to the converted string. Conversion is performed in-place.

```
21.11.3.25 strncasecmp() int strncasecmp (  
    const char * s1,  
    const char * s2,  
    size_t len )
```

Compare two strings ignoring case.

The `strncasecmp()` function is similar to `strcasecmp()`, except it only compares the first `len` characters of `s1`.

#### Returns

The `strncasecmp()` function returns an integer less than, equal to, or greater than zero if `s1` (or the first `len` bytes thereof) is found, respectively, to be less than, to match, or be greater than `s2`. A consequence of the ordering used by `strncasecmp()` is that if `s1` is an initial substring of `s2`, then `s1` is considered to be "less than" `s2`.

```
21.11.3.26 strncat() char * strncat (  
    char * dest,  
    const char * src,  
    size_t len )
```

Concatenate two strings.

The `strncat()` function is similar to `strcat()`, except that only the first `len` characters of `src` are appended to `dest`.

#### Returns

The `strncat()` function returns a pointer to the resulting string `dest`.

```
21.11.3.27 strncmp() int strncmp (  
    const char * s1,  
    const char * s2,  
    size_t len )
```

Compare two strings.

The `strncmp()` function is similar to `strcmp()`, except it only compares the first (at most) `len` characters of `s1` and `s2`.

#### Returns

The `strncmp()` function returns an integer less than, equal to, or greater than zero if `s1` (or the first `len` bytes thereof) is found, respectively, to be less than, to match, or be greater than `s2`.

**21.11.3.28 strncpy()** `char * strncpy (`  
    `char * dest,`  
    `const char * src,`  
    `size_t len )`

Copy a string.

The `strncpy()` function is similar to `strcpy()`, except that not more than `len` bytes of `src` are copied. Thus, if there is no null byte among the first `len` bytes of `src`, the result will not be null-terminated.

In the case where the length of `src` is less than that of `len`, the remainder of `dest` will be padded with nulls (`'\0'`).

#### Returns

The `strncpy()` function returns a pointer to the destination string `dest`.

**21.11.3.29 strdup()** `char * strdup (`  
    `const char * s,`  
    `size_t len )`

Duplicate a string.

The `strdup()` function is similar to `strdup()`, but copies at most `len` bytes. If `s` is longer than `len`, only `len` bytes are copied, and a terminating null byte (`'\0'`) is added.

Memory for the new string is obtained with `malloc()`, and can be freed with `free()`.

#### Returns

The `strdup()` function returns the location of the newly malloc'ed memory, or `NULL` if the allocation failed.

**21.11.3.30 strlen()** `size_t strlen (`  
    `const char * src,`  
    `size_t len )`

Determine the length of a fixed-size string.

The `strlen()` function returns the number of characters in the string pointed to by `src`, not including the terminating `'\0'` character, but at most `len`. In doing this, `strlen()` looks only at the first `len` characters at `src` and never beyond `src + len`.

#### Returns

The `strlen` function returns `strlen(src)`, if that is less than `len`, or `len` if there is no `'\0'` character among the first `len` characters pointed to by `src`.

**21.11.331 strpbrk()** `char * strpbrk (`  
    `const char * s,`  
    `const char * accept )`

The `strpbrk()` function locates the first occurrence in the string `s` of any of the characters in the string `accept`.

#### Returns

The `strpbrk()` function returns a pointer to the character in `s` that matches one of the characters in `accept`, or `NULL` if no such character is found. The terminating zero is not considered as a part of string: if one or both args are empty, the result will be `NULL`.

**21.11.332 strrchr()** `char * strrchr (`  
    `const char * src,`  
    `int val )`

Locate character in string.

The `strrchr()` function returns a pointer to the last occurrence of the character `val` in the string `src`.

Here "character" means "byte" – these functions do not work with wide or multi-byte characters.

#### Returns

The `strrchr()` function returns a pointer to the matched character or `NULL` if the character is not found.

**21.11.333 strrev()** `char * strrev (`  
    `char * s )`

Reverse a string.

The `strrev()` function reverses the order of the string.

#### Returns

The `strrev()` function returns a pointer to the beginning of the reversed string.

**21.11.334 strsep()** `char * strsep (`  
    `char ** sp,`  
    `const char * delim )`

Parse a string into tokens.

The `strsep()` function locates, in the string referenced by `*sp`, the first occurrence of any character in the string `delim` (or the terminating `\0` character) and replaces it with a `\0`. The location of the next character after the delimiter character (or `NULL`, if the end of the string was reached) is stored in `*sp`. An "empty" field, i.e. one caused by two adjacent delimiter characters, can be detected by comparing the location referenced by the pointer returned in `*sp` to `\0`.

#### Returns

The `strsep()` function returns a pointer to the original value of `*sp`. If `*sp` is initially `NULL`, `strsep()` returns `NULL`.

**21.11.3.35 `strspn()`** `size_t strspn (`  
    `const char * s,`  
    `const char * accept )`

The `strspn()` function calculates the length of the initial segment of `s` which consists entirely of characters in `accept`.

#### Returns

The `strspn()` function returns the number of characters in the initial segment of `s` which consist only of characters from `accept`. The terminating zero is not considered as a part of string.

**21.11.3.36 `strstr()`** `char * strstr (`  
    `const char * s1,`  
    `const char * s2 )`

Locate a substring.

The `strstr()` function finds the first occurrence of the substring `s2` in the string `s1`. The terminating '\0' characters are not compared.

#### Returns

The `strstr()` function returns a pointer to the beginning of the substring, or `NULL` if the substring is not found. If `s2` points to a string of zero length, the function returns `s1`.

**21.11.3.37 `strtok()`** `char * strtok (`  
    `char * s,`  
    `const char * delim )`

Parses the string `s` into tokens.

`strtok` parses the string `s` into tokens. The first call to `strtok` should have `s` as its first argument. Subsequent calls should have the first argument set to `NULL`. If a token ends with a delimiter, this delimiting character is overwritten with a '\0' and a pointer to the next character is saved for the next call to `strtok`. The delimiter string `delim` may be different for each call.

#### Returns

The `strtok()` function returns a pointer to the next token or `NULL` when no more tokens are found.

#### Note

`strtok()` is NOT reentrant. For a reentrant version of this function see `strtok_r()`.

```
21.11.3.38 strtok_r() char * strtok_r (  
    char * string,  
    const char * delim,  
    char ** last )
```

Parses string into tokens.

`strtok_r` parses string into tokens. The first call to `strtok_r` should have string as its first argument. Subsequent calls should have the first argument set to `NULL`. If a token ends with a delimiter, this delimiting character is overwritten with a `'\0'` and a pointer to the next character is saved for the next call to `strtok_r`. The delimiter string `delim` may be different for each call. `last` is a user allocated `char*` pointer. It must be the same while parsing the same string. `strtok_r` is a reentrant version of `strtok()`.

#### Returns

The `strtok_r()` function returns a pointer to the next token or `NULL` when no more tokens are found.

```
21.11.3.39 strupr() char * strupr (  
    char * s )
```

Convert a string to upper case.

The `strupr()` function will convert a string to upper case. Only the lower case alphabetic characters [a .. z] are converted. Non-alphabetic characters will not be changed.

#### Returns

The `strupr()` function returns a pointer to the converted string. The pointer is the same as that passed in since the operation is performed in place.

## 21.12 <time.h>: Time

### Data Structures

- struct [tm](#)
- struct [week\\_date](#)

### Macros

- #define [ONE\\_HOUR](#) 3600
- #define [ONE\\_DEGREE](#) 3600
- #define [ONE\\_DAY](#) 86400
- #define [UNIX\\_OFFSET](#) 946684800
- #define [NTP\\_OFFSET](#) 3155673600

### Typedefs

- typedef [uint32\\_t](#) [time\\_t](#)



## Enumerations

- enum `_WEEK_DAYS_` {  
**SUNDAY** , **MONDAY** , **TUESDAY** , **WEDNESDAY** ,  
**THURSDAY** , **FRIDAY** , **SATURDAY** }
- enum `_MONTHS_` {  
**JANUARY** , **FEBRUARY** , **MARCH** , **APRIL** ,  
**MAY** , **JUNE** , **JULY** , **AUGUST** ,  
**SEPTEMBER** , **OCTOBER** , **NOVEMBER** , **DECEMBER** }

## Functions

- `time_t` `time` (`time_t` \*timer)
- `int32_t` `difftime` (`time_t` time1, `time_t` time0)
- `time_t` `mktime` (`struct tm` \*timeptr)
- `time_t` `mk_gmtime` (`const struct tm` \*timeptr)
- `struct tm` \* `gmtime` (`const time_t` \*timer)
- void `gmtime_r` (`const time_t` \*timer, `struct tm` \*timeptr)
- `struct tm` \* `localtime` (`const time_t` \*timer)
- void `localtime_r` (`const time_t` \*timer, `struct tm` \*timeptr)
- char \* `asctime` (`const struct tm` \*timeptr)
- void `asctime_r` (`const struct tm` \*timeptr, char \*buf)
- char \* `ctime` (`const time_t` \*timer)
- void `ctime_r` (`const time_t` \*timer, char \*buf)
- char \* `isotime` (`const struct tm` \*tmptr)
- void `isotime_r` (`const struct tm` \*, char \*)
- `size_t` `strftime` (char \*s, `size_t` maxsize, `const char` \*format, `const struct tm` \*timeptr)
- void `set_dst` (`int`\*(\*)(`const time_t` \*, `int32_t` \*))
- void `set_zone` (`int32_t`)
- void `set_system_time` (`time_t` timestamp)
- void `system_tick` (void)
- `uint8_t` `is_leap_year` (`int16_t` year)
- `uint8_t` `month_length` (`int16_t` year, `uint8_t` month)
- `uint8_t` `week_of_year` (`const struct tm` \*timeptr, `uint8_t` start)
- `uint8_t` `week_of_month` (`const struct tm` \*timeptr, `uint8_t` start)
- `struct week_date` \* `iso_week_date` (`int` year, `int` yday)
- void `iso_week_date_r` (`int` year, `int` yday, `struct week_date` \*)
- `uint32_t` `fatfs_time` (`const struct tm` \*timeptr)
- void `set_position` (`int32_t` latitude, `int32_t` longitude)
- `int16_t` `equation_of_time` (`const time_t` \*timer)
- `int32_t` `daylight_seconds` (`const time_t` \*timer)
- `time_t` `solar_noon` (`const time_t` \*timer)
- `time_t` `sun_rise` (`const time_t` \*timer)
- `time_t` `sun_set` (`const time_t` \*timer)
- float `solar_declinationf` (`const time_t` \*timer)
- double `solar_declination` (`const time_t` \*timer)
- long double `solar_declinationl` (`const time_t` \*timer)
- `int8_t` `moon_phase` (`const time_t` \*timer)
- unsigned long `gm_sidereal` (`const time_t` \*timer)
- unsigned long `lm_sidereal` (`const time_t` \*timer)

### 21.12.1 Detailed Description

```
#include <time.h>
```

**Introduction to the Time functions** This file declares the time functions implemented in AVR-LibC.

The implementation aspires to conform with ISO/IEC 9899 (C90). However, due to limitations of the target processor and the nature of its development environment, a practical implementation must of necessity deviate from the standard.

Section 7.23.2.1 `clock()` The type `clock_t`, the macro `CLOCKS_PER_SEC`, and the function `clock()` are not implemented. We consider these items belong to operating system code, or to application code when no operating system is present.

Section 7.23.2.3 `mktime()` The standard specifies that `mktime()` should return `(time_t) -1`, if the time cannot be represented. This implementation always returns a 'best effort' representation.

Section 7.23.2.4 `time()` The standard specifies that `time()` should return `(time_t) -1`, if the time is not available. Since the application must initialize the time system, this functionality is not implemented.

Section 7.23.2.2, `difftime()` Due to the lack of a 64 bit double, the function `difftime()` returns a long integer. In most cases this change will be invisible to the user, handled automatically by the compiler.

Section 7.23.1.4 `struct tm` Per the standard, `tm->tm_isdst` is greater than zero when Daylight Saving time is in effect. This implementation further specifies that, when positive, the value of `tm_isdst` represents the amount time is advanced during Daylight Saving time.

Section 7.23.3.5 `strftime()` Only the 'C' locale is supported, therefore the modifiers 'E' and 'O' are ignored. The 'Z' conversion is also ignored, due to the lack of time zone name.

In addition to the above departures from the standard, there are some behaviors which are different from what is often expected, though allowed under the standard.

There is no 'platform standard' method to obtain the current time, time zone, or daylight savings 'rules' in the AVR environment. Therefore the application must initialize the time system with this information. The functions `set_zone()`, `set_dst()`, and `set_system_time()` are provided for initialization. Once initialized, system time is maintained by calling the function `system_tick()` at one second intervals.

Though not specified in the standard, it is often expected that `time_t` is a signed integer representing an offset in seconds from Midnight Jan 1 1970... i.e. 'Unix time'. This implementation uses an unsigned 32 bit integer offset from Midnight Jan 1 2000. The use of this 'epoch' helps to simplify the conversion functions, while the 32 bit value allows time to be properly represented until Tue Feb 7 06:28:15 2136 UTC. The macros `UNIX_OFFSET` and `NTP_OFFSET` are defined to assist in converting to and from Unix and NTP time stamps.

Unlike desktop counterparts, it is impractical to implement or maintain the 'zoneinfo' database. Therefore no attempt is made to account for time zone, daylight saving, or leap seconds in past dates. All calculations are made according to the currently configured time zone and daylight saving 'rule'.

In addition to C standard functions, re-entrant versions of `ctime()`, `asctime()`, `gmtime()` and `localtime()` are provided which, in addition to being re-entrant, have the property of claiming less permanent storage in RAM. An additional time conversion, `isotime()` and its re-entrant version, uses far less storage than either `ctime()` or `asctime()`.

Along with the usual smattering of utility functions, such as `is_leap_year()`, this library includes a set of functions related the sun and moon, as well as sidereal time functions.

## 21.12.2 Macro Definition Documentation

**21.12.2.1 NTP\_OFFSET** `#define NTP_OFFSET 3155673600`

Difference between the Y2K and the NTP epochs, in seconds. To convert a Y2K timestamp to NTP...

```
unsigned long ntp;  
time_t y2k;  
  
y2k = time(NULL);  
ntp = y2k + NTP_OFFSET;
```

**21.12.2.2 ONE\_DAY** `#define ONE_DAY 86400`

One day, expressed in seconds

**21.12.2.3 ONE\_DEGREE** `#define ONE_DEGREE 3600`

Angular degree, expressed in arc seconds

**21.12.2.4 ONE\_HOUR** `#define ONE_HOUR 3600`

One hour, expressed in seconds

**21.12.2.5 UNIX\_OFFSET** `#define UNIX_OFFSET 946684800`

Difference between the Y2K and the UNIX epochs, in seconds. To convert a Y2K timestamp to UNIX...

```
long unix;  
time_t y2k;  
  
y2k = time(NULL);  
unix = y2k + UNIX_OFFSET;
```

**21.12.3 Typedef Documentation****21.12.3.1 time\_t** `typedef uint32_t time_t`

`time_t` represents seconds elapsed from Midnight, Jan 1 2000 UTC (the Y2K 'epoch'). Its range allows this implementation to represent time up to Tue Feb 7 06:28:15 2136 UTC.

**21.12.4 Enumeration Type Documentation****21.12.4.1 \_MONTHS\_** `enum _MONTHS_`

Enumerated labels for the months.

#### 21.12.4.2 `_WEEK_DAYS_` enum `_WEEK_DAYS_`

Enumerated labels for the days of the week.

### 21.12.5 Function Documentation

**21.12.5.1 `asctime()`** `char * asctime (`  
`const struct tm * timeptr )`

The `asctime` function converts the broken-down time of `timeptr`, into an ascii string in the form

```
Sun Mar 23 01:03:52 2013
```

**21.12.5.2 `asctime_r()`** `void asctime_r (`  
`const struct tm * timeptr,`  
`char * buf )`

Re entrant version of [`asctime\(\)`](#).

**21.12.5.3 `ctime()`** `char * ctime (`  
`const time_t * timer )`

The `ctime` function is equivalent to `asctime(localtime(timer))`

**21.12.5.4 `ctime_r()`** `void ctime_r (`  
`const time_t * timer,`  
`char * buf )`

Re entrant version of [`ctime\(\)`](#).

**21.12.5.5 `daylight_seconds()`** `int32_t daylight_seconds (`  
`const time_t * timer )`

Computes the amount of time the sun is above the horizon, at the location of the observer.

NOTE: At observer locations inside a polar circle, this value can be zero during the winter, and can exceed `ONE ←`  
`_DAY` during the summer.

The returned value is in seconds.

**21.12.5.6 `difftime()`** `int32_t difftime (`  
`time_t time1,`  
`time_t time0 )`

The `difftime` function returns the difference between two binary time stamps, `time1 - time0`.

**21.12.5.7 equation\_of\_time()** `int16_t equation_of_time (`  
`const time_t * timer )`

Computes the difference between apparent solar time and mean solar time. The returned value is in seconds.

**21.12.5.8 fatfs\_time()** `uint32_t fatfs_time (`  
`const struct tm * timeptr )`

Convert a Y2K time stamp into a FAT file system time stamp.

**21.12.5.9 gm\_sidereal()** `unsigned long gm_sidereal (`  
`const time_t * timer )`

Returns Greenwich Mean Sidereal Time, as seconds into the sidereal day. The returned value will range from 0 through 86399 seconds.

**21.12.5.10 gmtime()** `struct tm * gmtime (`  
`const time_t * timer )`

The gmtime function converts the time stamp pointed to by timer into broken-down time, expressed as UTC.

**21.12.5.11 gmtime\_r()** `void gmtime_r (`  
`const time_t * timer,`  
`struct tm * timeptr )`

Re-entrant version of [gmtime\(\)](#).

**21.12.5.12 is\_leap\_year()** `uint8_t is_leap_year (`  
`int16_t year )`

Return 1 if year is a leap year, zero if it is not.

**21.12.5.13 iso\_week\_date()** `struct week_date * iso_week_date (`  
`int year,`  
`int yday )`

Return a [week\\_date](#) structure with the ISO\_8601 week based date corresponding to the given year and day of year. See [http://en.wikipedia.org/wiki/ISO\\_week\\_date](http://en.wikipedia.org/wiki/ISO_week_date) for more information.

**21.12.5.14 iso\_week\_date\_r()** `void iso_week_date_r (`  
`int year,`  
`int yday,`  
`struct week_date * iso )`

Re-entrant version of [iso-week\\_date](#).

**21.12.5.15** `isotime()` `char * isotime (`  
`const struct tm * tm_ptr )`

The `isotime` function constructs an ascii string in the form  
2013-03-23 01:03:52

**21.12.5.16** `isotime_r()` `void isotime_r (`  
`const struct tm * tm_ptr,`  
`char * buffer )`

Re entrant version of `isotime()`

**21.12.5.17** `lm_sidereal()` `unsigned long lm_sidereal (`  
`const time_t * timer )`

Returns Local Mean Sidereal Time, as seconds into the sidereal day. The returned value will range from 0 through 86399 seconds.

**21.12.5.18** `localtime()` `struct tm * localtime (`  
`const time_t * timer )`

The `localtime` function converts the time stamp pointed to by `timer` into broken-down time, expressed as Local time.

**21.12.5.19** `localtime_r()` `void localtime_r (`  
`const time_t * timer,`  
`struct tm * timeptr )`

Re entrant version of `localtime()`.

**21.12.5.20** `mk_gmtime()` `time_t mk_gmtime (`  
`const struct tm * timeptr )`

This function 'compiles' the elements of a broken-down time structure, returning a binary time stamp. The elements of `timeptr` are interpreted as representing UTC.

The original values of the `tm_wday` and `tm_yday` elements of the structure are ignored, and the original values of the other elements are not restricted to the ranges stated for `struct tm`.

Unlike `mktime()`, this function DOES NOT modify the elements of `timeptr`.

**21.12.5.21** `mktime()` `time_t mktime (`  
`struct tm * timeptr )`

This function 'compiles' the elements of a broken-down time structure, returning a binary time stamp. The elements of `timeptr` are interpreted as representing Local Time.

The original values of the `tm_wday` and `tm_yday` elements of the structure are ignored, and the original values of the other elements are not restricted to the ranges stated for `struct tm`.

The element `tm_isdst` is used for input and output. If set to 0 or a positive value on input, this requests calculation for Daylight Savings Time being off or on, respectively. If set to a negative value on input, it requests calculation to return whether Daylight Savings Time is in effect or not according to the other values.

On successful completion, the values of all elements of `timeptr` are set to the appropriate range.

**21.12.5.22 month\_length()** `uint8_t month_length (`  
    `int16_t year,`  
    `uint8_t month )`

Return the length of month, given the year and month, where month is in the range 1 to 12.

**21.12.5.23 moon\_phase()** `int8_t moon_phase (`  
    `const time_t * timer )`

Returns an approximation to the phase of the moon. The sign of the returned value indicates a waning or waxing phase. The magnitude of the returned value indicates the percentage illumination.

**21.12.5.24 set\_dst()** `void set_dst (`  
    `int(*) (const time_t *, int32_t *) d )`

Specify the Daylight Saving function.

The Daylight Saving function should examine its parameters to determine whether Daylight Saving is in effect, and return a value appropriate for `tm_isdst`.

Working examples for the USA and the EU are available..

```
#include <util/eu_dst.h>
```

for the European Union, and

```
#include <util/usa_dst.h>
```

for the United States

If a Daylight Saving function is not specified, the system will ignore Daylight Saving.

**21.12.5.25 set\_position()** `void set_position (`  
    `int32_t latitude,`  
    `int32_t longitude )`

Set the geographic coordinates of the 'observer', for use with several of the following functions. Parameters are passed as seconds of North Latitude, and seconds of East Longitude.

For New York City..

```
set_position( 40.7142 * ONE_DEGREE, -74.0064 * ONE_DEGREE);
```

**21.12.5.26 set\_system\_time()** `void set_system_time (`  
    `time_t timestamp )`

Initialize the system time. Examples are...

From a Clock / Calendar type RTC:

```
struct tm rtc_time;  
  
read_rtc(&rtc_time);  
rtc_time.tm_isdst = 0;  
set_system_time( mktime(&rtc_time) );
```

From a Network Time Protocol time stamp:

```
set_system_time(ntp_timestamp - NTP_OFFSET);
```

From a UNIX time stamp:

```
set_system_time(unix_timestamp - UNIX_OFFSET);
```

**21.12.5.27 set\_zone()** `void set_zone (`  
`int32_t )`

Set the 'time zone'. The parameter is given in seconds East of the Prime Meridian. Example for New York City:  
`set_zone(-5 * ONE_HOUR);`

If the time zone is not set, the time system will operate in UTC only.

**21.12.5.28 solar\_declination()** `double solar_declination (`  
`const time_t * timer )`

Returns the declination of the sun in radians.

This implementation is only available when `double` is a 32-bit type.

**21.12.5.29 solar\_declinationf()** `float solar_declinationf (`  
`const time_t * timer )`

Returns the declination of the sun in radians.

**21.12.5.30 solar\_declinationl()** `long double solar_declinationl (`  
`const time_t * timer )`

Returns the declination of the sun in radians.

This implementation is only available when `long double` is a 32-bit type.

**21.12.5.31 solar\_noon()** `time_t solar_noon (`  
`const time_t * timer )`

Computes the time of solar noon, at the location of the observer.

**21.12.5.32 strftime()** `size_t strftime (`  
`char * s,`  
`size_t maxsize,`  
`const char * format,`  
`const struct tm * timeptr )`

A complete description of `strftime()` is beyond the pale of this document. Refer to ISO/IEC document 9899 for details.

All conversions are made using the 'C Locale', ignoring the E or O modifiers. Due to the lack of a time zone 'name', the 'Z' conversion is also ignored.

**21.12.5.33 sun\_rise()** `time_t sun_rise (`  
`const time_t * timer )`

Return the time of sunrise, at the location of the observer. See the note about `daylight_seconds()`.



**21.12.5.34 sun\_set()** `time_t sun_set (`  
`const time_t * timer )`

Return the time of sunset, at the location of the observer. See the note about [daylight\\_seconds\(\)](#).

**21.12.5.35 system\_tick()** `void system_tick (`  
`void )`

Maintain the system time by calling this function at a rate of 1 Hertz.

It is anticipated that this function will typically be called from within an Interrupt Service Routine, (though that is not required). It therefore includes code which makes it simple to use from within a 'Naked' ISR, avoiding the cost of saving and restoring all the cpu registers.

Such an ISR may resemble the following example...

```
ISR(RTC_OVF_vect, ISR_NAKED)
{
    system_tick();
    reti();
}
```

**21.12.5.36 time()** `time_t time (`  
`time_t * timer )`

The time function returns the systems current time stamp. If timer is not a null pointer, the return value is also assigned to the object it points to.

**21.12.5.37 week\_of\_month()** `uint8_t week_of_month (`  
`const struct tm * timeptr,`  
`uint8_t start )`

Return the calendar week of month, where the first week is considered to begin on the day of week specified by 'start'. The returned value may range from zero to 5.

**21.12.5.38 week\_of\_year()** `uint8_t week_of_year (`  
`const struct tm * timeptr,`  
`uint8_t start )`

Return the calendar week of year, where week 1 is considered to begin on the day of week specified by 'start'. The returned value may range from zero to 52.

## 21.13 <avr/boot.h>: Bootloader Support Utilities

### Macros

- #define `BOOTLOADER_SECTION` `__attribute__((__section__(".bootloader")))`
- #define `boot_spm_interrupt_enable()` `(__SPM_REG |= (uint8_t)_BV(SPMIE))`
- #define `boot_spm_interrupt_disable()` `(__SPM_REG &= (uint8_t)~_BV(SPMIE))`
- #define `boot_is_spm_interrupt()` `(__SPM_REG & (uint8_t)_BV(SPMIE))`
- #define `boot_rww_busy()` `(__SPM_REG & (uint8_t)_BV(__COMMON_ASB))`
- #define `boot_spm_busy()` `(__SPM_REG & (uint8_t)_BV(__SPM_ENABLE))`
- #define `boot_spm_busy_wait()` `do{}while(boot_spm_busy())`
- #define `GET_LOW_FUSE_BITS` `(0x0000)`
- #define `GET_LOCK_BITS` `(0x0001)`
- #define `GET_EXTENDED_FUSE_BITS` `(0x0002)`
- #define `GET_HIGH_FUSE_BITS` `(0x0003)`
- #define `boot_lock_fuse_bits_get(address)`
- #define `boot_signature_byte_get(addr)`
- #define `boot_page_fill(address, data)` `__boot_page_fill_normal(address, data)`
- #define `boot_page_erase(address)` `__boot_page_erase_normal(address)`
- #define `boot_page_write(address)` `__boot_page_write_normal(address)`
- #define `boot_rww_enable()` `__boot_rww_enable()`
- #define `boot_lock_bits_set(lock_bits)` `__boot_lock_bits_set(lock_bits)`
- #define `boot_page_fill_safe(address, data)`
- #define `boot_page_erase_safe(address)`
- #define `boot_page_write_safe(address)`
- #define `boot_rww_enable_safe()`
- #define `boot_lock_bits_set_safe(lock_bits)`

### 21.13.1 Detailed Description

```
#include <avr/io.h>
#include <avr/boot.h>
```

The macros in this module provide a C language interface to the bootloader support functionality of certain AVR processors. These macros are designed to work with all sizes of flash memory.

Global interrupts are not automatically disabled for these macros. It is left up to the programmer to do this. See the code example below. Also see the processor datasheet for caveats on having global interrupts enabled during writing of the Flash.

#### Note

Not all AVR processors provide bootloader support. See your processor datasheet to see if it provides bootloader support.

## API Usage Example

The following code shows typical usage of the boot API.

```
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

void boot_program_page (uint32_t page, uint8_t *buf)
{
    // Disable interrupts.
    uint8_t sreg = SREG;
    cli();

    eeprom_busy_wait ();

    boot_page_erase (page);
    boot_spm_busy_wait ();      // Wait until the memory is erased.

    for (uint16_t i = 0; i < SPM_PAGESIZE; i += 2)
    {
        // Set up little-endian word.
        uint16_t w = *buf++;
        w += (*buf++) << 8;

        boot_page_fill (page + i, w);
    }

    boot_page_write (page);     // Store buffer in flash page.
    boot_spm_busy_wait ();     // Wait until the memory is written.

    // Reenable RWW-section again. We need this if we want to jump back
    // to the application after bootloading.
    boot_rww_enable ();

    // Re-enable interrupts (if they were ever enabled).
    SREG = sreg;
}
```

## 21.13.2 Macro Definition Documentation

**21.13.2.1 boot\_is\_spm\_interrupt** #define boot\_is\_spm\_interrupt( ) (\_\_SPM\_REG & (uint8\_t)\_BV(SPMIE))

Check if the SPM interrupt is enabled.

**21.13.2.2 boot\_lock\_bits\_set** #define boot\_lock\_bits\_set(  
lock\_bits ) \_\_boot\_lock\_bits\_set(lock\_bits)

Set the bootloader lock bits.

### Parameters

<i>lock_bits</i>	A mask of which Boot Loader Lock Bits to set.
------------------	---

### Note

In this context, a 'set bit' will be written to a zero value. Note also that only BLBxx bits can be programmed by this command.

For example, to disallow the SPM instruction from writing to the Boot Loader memory section of flash, you would use this macro as such:

```
boot_lock_bits_set (_BV (BLB11));
```

**Note**

Like any lock bits, the Boot Loader Lock Bits, once set, cannot be cleared again except by a chip erase which will in turn also erase the boot loader itself.

**21.13.2.3 boot\_lock\_bits\_set\_safe** #define boot\_lock\_bits\_set\_safe(  
lock\_bits )

**Value:**

```
do { \
    boot_spm_busy_wait();           \
    eeprom_busy_wait();           \
    boot_lock_bits_set (lock_bits); \
} while (0)
```

Same as `boot_lock_bits_set()` except waits for eeprom and spm operations to complete before setting the lock bits.

**21.13.2.4 boot\_lock\_fuse\_bits\_get** #define boot\_lock\_fuse\_bits\_get(  
address )

**Value:**

```
(( {
    uint8_t __result;
    __asm__ __volatile__
    (
        "sts %1, %2\n\t"
        "lpm %0, Z\n\t"
        : "=r" (__result)
        : "i" (_SFR_MEM_ADDR(__SPM_REG)),
          "r" ((uint8_t)(__BOOT_LOCK_BITS_SET)),
          "z" ((uint16_t)(address))
        );
    __result;
    })
```

Read the lock or fuse bits at address.

Parameter `address` can be any of `GET_LOW_FUSE_BITS`, `GET_LOCK_BITS`, `GET_EXTENDED_FUSE_BITS`, or `GET_HIGH_FUSE_BITS`.

**Note**

The lock and fuse bits returned are the physical values, i.e. a bit returned as 0 means the corresponding fuse or lock bit is programmed.

**21.13.2.5 boot\_page\_erase** #define boot\_page\_erase(  
address ) \_\_boot\_page\_erase\_normal(address)

Erase the flash page that contains address.

**Note**

`address` is a byte address in flash, not a word address.

**21.13.2.6 boot\_page\_erase\_safe** #define boot\_page\_erase\_safe(  
*address* )

**Value:**

```
do { \
    boot_spm_busy_wait();           \
    eeprom_busy_wait();           \
    boot_page_erase (address);     \
} while (0)
```

Same as [boot\\_page\\_erase\(\)](#) except it waits for eeprom and spm operations to complete before erasing the page.

**21.13.2.7 boot\_page\_fill** #define boot\_page\_fill(  
*address*,  
*data* ) \_\_boot\_page\_fill\_normal(*address*, *data*)

Fill the bootloader temporary page buffer for flash address with data word.

**Note**

The address is a byte address. The data is a word. The AVR writes data to the buffer a word at a time, but addresses the buffer per byte! So, increment your address by 2 between calls, and send 2 data bytes in a word format! The LSB of the data is written to the lower address; the MSB of the data is written to the higher address.

**21.13.2.8 boot\_page\_fill\_safe** #define boot\_page\_fill\_safe(  
*address*,  
*data* )

**Value:**

```
do { \
    boot_spm_busy_wait();           \
    eeprom_busy_wait();           \
    boot_page_fill(address, data);  \
} while (0)
```

Same as [boot\\_page\\_fill\(\)](#) except it waits for eeprom and spm operations to complete before filling the page.

**21.13.2.9 boot\_page\_write** #define boot\_page\_write(  
*address* ) \_\_boot\_page\_write\_normal(*address*)

Write the bootloader temporary page buffer to flash page that contains address.

**Note**

address is a byte address in flash, not a word address.

**21.13.2.10 boot\_page\_write\_safe** #define boot\_page\_write\_safe(  
     *address* )

**Value:**

```
do { \
    boot_spm_busy_wait();           \
    eeprom_busy_wait();           \
    boot_page_write (address);     \
} while (0)
```

Same as [boot\\_page\\_write\(\)](#) except it waits for eeprom and spm operations to complete before writing the page.

**21.13.2.11 boot\_rww\_busy** #define boot\_rww\_busy( ) ( \_\_SPM\_REG & (uint8\_t)\_BV(\_\_COMMON\_ASB) )

Check if the RWW section is busy.

**21.13.2.12 boot\_rww\_enable** #define boot\_rww\_enable( ) \_\_boot\_rww\_enable()

Enable the Read-While-Write memory section.

**21.13.2.13 boot\_rww\_enable\_safe** #define boot\_rww\_enable\_safe( )

**Value:**

```
do { \
    boot_spm_busy_wait();           \
    eeprom_busy_wait();           \
    boot_rww_enable();             \
} while (0)
```

Same as [boot\\_rww\\_enable\(\)](#) except waits for eeprom and spm operations to complete before enabling the RWW mameory.

**21.13.2.14 boot\_signature\_byte\_get** #define boot\_signature\_byte\_get(  
     *addr* )

**Value:**

```
((
    uint8_t __result;               \
    __asm__ __volatile__           \
    (                               \
        "sts %1, %2"   "\n\t"      \
        "lpm %0, Z"      \
        : "=r" (__result)         \
        : "i" (_SFR_MEM_ADDR(__SPM_REG)), \
          "x" ((uint8_t)(__BOOT_SIGROW_READ)), \
          "z" ((uint16_t)(addr))   \
        );                       \
    __result;                     \
}))
```

Read the Signature Row byte at *address*. For some MCU types, this function can also retrieve the factory-stored oscillator calibration bytes.

Parameter *address* can be 0-0x1f as documented by the datasheet.

**Note**

The values are MCU type dependent.

**21.13.2.15 boot\_spm\_busy** `#define boot_spm_busy( ) ( __SPM_REG & (uint8_t)_BV(__SPM_ENABLE) )`

Check if the SPM instruction is busy.

**21.13.2.16 boot\_spm\_busy\_wait** `#define boot_spm_busy_wait( ) do{}while(boot_spm_busy())`

Wait while the SPM instruction is busy.

**21.13.2.17 boot\_spm\_interrupt\_disable** `#define boot_spm_interrupt_disable( ) ( __SPM_REG &= (uint8_t)~_BV(SPMIE) )`

Disable the SPM interrupt.

**21.13.2.18 boot\_spm\_interrupt\_enable** `#define boot_spm_interrupt_enable( ) ( __SPM_REG |= (uint8_t)_BV(SPMIE) )`

Enable the SPM interrupt.

**21.13.2.19 BOOTLOADER\_SECTION** `#define BOOTLOADER_SECTION __attribute__((section(↵  
(".bootloader"))))`

Used to declare a function or variable to be placed into a new section called `.bootloader`. This section and its contents can then be relocated to any address (such as the bootloader NRWW area) at link-time.

**21.13.2.20 GET\_EXTENDED\_FUSE\_BITS** `#define GET_EXTENDED_FUSE_BITS (0x0002)`

address to read the extended fuse bits, using `boot_lock_fuse_bits_get`

**21.13.2.21 GET\_HIGH\_FUSE\_BITS** `#define GET_HIGH_FUSE_BITS (0x0003)`

address to read the high fuse bits, using `boot_lock_fuse_bits_get`

**21.13.2.22 GET\_LOCK\_BITS** `#define GET_LOCK_BITS (0x0001)`

address to read the lock bits, using `boot_lock_fuse_bits_get`

**21.13.2.23 GET\_LOW\_FUSE\_BITS** `#define GET_LOW_FUSE_BITS (0x0000)`

address to read the low fuse bits, using `boot_lock_fuse_bits_get`

## 21.14 <avr/cpufunc.h>: Special AVR CPU functions

### Macros

- `#define _NOP()`
- `#define _MemoryBarrier()`

## Functions

- void `ccp_write_io` (volatile `uint8_t` \* \_\_ioaddr, `uint8_t` \_\_value)
- void `ccp_write_spm` (volatile `uint8_t` \* \_\_ioaddr, `uint8_t` \_\_value)

### 21.14.1 Detailed Description

```
#include <avr/cpufunc.h>
```

This header file contains macros that access special functions of the AVR CPU which do not fit into any of the other header files.

### 21.14.2 Macro Definition Documentation

#### 21.14.2.1 `_MemoryBarrier` `#define _MemoryBarrier( )`

Implement a read/write *memory barrier*. A memory barrier instructs the compiler to not cache any memory data in registers beyond the barrier. This can sometimes be more effective than blocking certain optimizations by declaring some object with a `volatile` qualifier.

See [Problems with reordering code](#) for things to be taken into account with respect to compiler optimizations.

#### 21.14.2.2 `_NOP` `#define _NOP( )`

Execute a *no operation* (NOP) CPU instruction. This should not be used to implement delays, better use the functions from `<util/delay_basic.h>` or `<util/delay.h>` for this. For debugging purposes, a NOP can be useful to have an instruction that is guaranteed to be not optimized away by the compiler, so it can always become a breakpoint in the debugger.

### 21.14.3 Function Documentation

#### 21.14.3.1 `ccp_write_io()` `void ccp_write_io (` `volatile uint8_t * __ioaddr,` `uint8_t __value )`

Write `__value` to IO Register Protected (CCP) IO register at `__ioaddr`. . See also [\\_PROTECTED\\_WRITE\( \)](#) .

#### 21.14.3.2 `ccp_write_spm()` `void ccp_write_spm (` `volatile uint8_t * __ioaddr,` `uint8_t __value )`

Write `__value` to SPM Instruction Protected (CCP) IO register at `__ioaddr`. See also [\\_PROTECTED\\_WRITE\\_SPM\( \)](#) .



## 21.15 <avr/eeprom.h>: EEPROM handling

### Macros

- #define `EEMEM __attribute__((__section__(".eeprom")))`
- #define `eeprom_is_ready()`
- #define `eeprom_busy_wait()` do {} while (!`eeprom_is_ready()`)

### Functions

- `uint8_t eeprom_read_byte (const uint8_t * __p)`
- `uint16_t eeprom_read_word (const uint16_t * __p)`
- `uint32_t eeprom_read_dword (const uint32_t * __p)`
- `uint64_t eeprom_read_qword (const uint64_t * __p)`
- `float eeprom_read_float (const float * __p)`
- `double eeprom_read_double (const double * __p)`
- `long double eeprom_read_long_double (const long double * __p)`
- `void eeprom_read_block (void * __dst, const void * __src, size_t __n)`
- `void eeprom_write_byte (uint8_t * __p, uint8_t __value)`
- `void eeprom_write_word (uint16_t * __p, uint16_t __value)`
- `void eeprom_write_dword (uint32_t * __p, uint32_t __value)`
- `void eeprom_write_qword (uint64_t * __p, uint64_t __value)`
- `void eeprom_write_float (float * __p, float __value)`
- `void eeprom_write_double (double * __p, double __value)`
- `void eeprom_write_long_double (long double * __p, long double __value)`
- `void eeprom_write_block (const void * __src, void * __dst, size_t __n)`
- `void eeprom_update_byte (uint8_t * __p, uint8_t __value)`
- `void eeprom_update_word (uint16_t * __p, uint16_t __value)`
- `void eeprom_update_dword (uint32_t * __p, uint32_t __value)`
- `void eeprom_update_qword (uint64_t * __p, uint64_t __value)`
- `void eeprom_update_float (float * __p, float __value)`
- `void eeprom_update_double (double * __p, double __value)`
- `void eeprom_update_long_double (long double * __p, long double __value)`
- `void eeprom_update_block (const void * __src, void * __dst, size_t __n)`

### IAR C compatibility defines

- #define `_EEPWRITE(addr, val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))`
- #define `__EEPWRITE(addr, val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))`
- #define `_EEGET(var, addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))`
- #define `__EEGET(var, addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))`

### 21.15.1 Detailed Description

```
#include <avr/eeprom.h>
```

This header file declares the interface to some simple library routines suitable for handling the data EEPROM contained in the AVR microcontrollers. The implementation uses a simple polled mode interface. Applications that require interrupt-controlled EEPROM access to ensure that no time will be wasted in spinloops will have to deploy their own implementation.

#### Notes:

- In addition to the write functions there is a set of update ones. This functions read each byte first and skip the burning if the old value is the same with new. The scanning direction is from high address to low, to obtain quick return in common cases.
- All of the read/write functions first make sure the EEPROM is ready to be accessed. Since this may cause long delays if a write operation is still pending, time-critical applications should first poll the EEPROM e. g. using `eeprom_is_ready()` before attempting any actual I/O. But this functions does not wait until `SELFPRGEN` in `SPMCSR` becomes zero. Do this manually, if your softwate contains the Flash burning.
- As these functions modify IO registers, they are known to be non-reentrant. If any of these functions are used from both, standard and interrupt context, the applications must ensure proper protection (e.g. by disabling interrupts before accessing them).
- All write functions force `erase_and_write` programming mode.
- For Xmega the EEPROM start address is 0, like other architectures. The reading functions add the 0x2000 value to use EEPROM mapping into data space.

### 21.15.2 Macro Definition Documentation

**21.15.2.1 `__EEGET`** `#define __EEGET(  
var,  
addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))`

Read a byte from EEPROM. Compatibility define for IAR C.

**21.15.2.2 `__EEPUT`** `#define __EEPUT(  
addr,  
val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))`

Write a byte to EEPROM. Compatibility define for IAR C.

**21.15.2.3 `_EEGET`** `#define _EEGET(  
var,  
addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))`

Read a byte from EEPROM. Compatibility define for IAR C.

**21.15.2.4** `_EEPWRITE` `#define _EEPWRITE(`  
    `addr,`  
    `val ) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))`

Write a byte to EEPROM. Compatibility define for IAR C.

**21.15.2.5** `EEMEM` `#define EEMEM __attribute__((__section__(".eeprom")))`

Attribute expression causing a variable to be allocated within the `.eeprom` section.

**21.15.2.6** `eeprom_busy_wait` `#define eeprom_busy_wait( ) do {} while (!eeprom_is_ready())`

Loops until the eeprom is no longer busy.

Returns

Nothing.

**21.15.2.7** `eeprom_is_ready` `#define eeprom_is_ready( )`

Returns

1 if EEPROM is ready for a new read/write operation, 0 if not.

### 21.15.3 Function Documentation

**21.15.3.1** `eeprom_read_block()` `void eeprom_read_block (`  
    `void * __dst,`  
    `const void * __src,`  
    `size_t __n )`

Read a block of `__n` bytes from EEPROM address `__src` to SRAM `__dst`.

**21.15.3.2** `eeprom_read_byte()` `uint8_t eeprom_read_byte (`  
    `const uint8_t * __p )`

Read one byte from EEPROM address `__p`.

**21.15.3.3** `eeprom_read_double()` `double eeprom_read_double (`  
    `const double * __p )`

Read one double value (little endian) from EEPROM address `__p`.

**21.15.3.4 eeprom\_read\_dword()** `uint32_t eeprom_read_dword (`  
`const uint32_t * __p )`

Read one 32-bit double word (little endian) from EEPROM address `__p`.

**21.15.3.5 eeprom\_read\_float()** `float eeprom_read_float (`  
`const float * __p )`

Read one float value (little endian) from EEPROM address `__p`.

**21.15.3.6 eeprom\_read\_long\_double()** `long double eeprom_read_long_double (`  
`const long double * __p )`

Read one long double value (little endian) from EEPROM address `__p`.

**21.15.3.7 eeprom\_read\_qword()** `uint64_t eeprom_read_qword (`  
`const uint64_t * __p )`

Read one 64-bit quad word (little endian) from EEPROM address `__p`.

**21.15.3.8 eeprom\_read\_word()** `uint16_t eeprom_read_word (`  
`const uint16_t * __p )`

Read one 16-bit word (little endian) from EEPROM address `__p`.

**21.15.3.9 eeprom\_update\_block()** `void eeprom_update_block (`  
`const void * __src,`  
`void * __dst,`  
`size_t __n )`

Update a block of `__n` bytes at EEPROM address `__dst` from `__src`.

#### Note

The argument order is mismatch with common functions like `strcpy()`.

**21.15.3.10 eeprom\_update\_byte()** `void eeprom_update_byte (`  
`uint8_t * __p,`  
`uint8_t __value )`

Update a byte `__value` at EEPROM address `__p`.

**21.15.3.11 eeprom\_update\_double()** `void eeprom_update_double (`  
`double * __p,`  
`double __value )`

Update a double `__value` at EEPROM address `__p`.

**21.15.3.12 eeprom\_update\_dword()** void eeprom\_update\_dword (   
    uint32\_t \* \_\_p,   
    uint32\_t \_\_value )

Update a 32-bit double word `__value` at EEPROM address `__p`.

**21.15.3.13 eeprom\_update\_float()** void eeprom\_update\_float (   
    float \* \_\_p,   
    float \_\_value )

Update a float `__value` at EEPROM address `__p`.

**21.15.3.14 eeprom\_update\_long\_double()** void eeprom\_update\_long\_double (   
    long double \* \_\_p,   
    long double \_\_value )

Update a long double `__value` at EEPROM address `__p`.

**21.15.3.15 eeprom\_update\_qword()** void eeprom\_update\_qword (   
    uint64\_t \* \_\_p,   
    uint64\_t \_\_value )

Update a 64-bit quad word `__value` at EEPROM address `__p`.

**21.15.3.16 eeprom\_update\_word()** void eeprom\_update\_word (   
    uint16\_t \* \_\_p,   
    uint16\_t \_\_value )

Update a word `__value` at EEPROM address `__p`.

**21.15.3.17 eeprom\_write\_block()** void eeprom\_write\_block (   
    const void \* \_\_src,   
    void \* \_\_dst,   
    size\_t \_\_n )

Write a block of `__n` bytes to EEPROM address `__dst` from `__src`.

#### Note

The argument order is mismatch with common functions like `strcpy()`.

**21.15.3.18 eeprom\_write\_byte()** void eeprom\_write\_byte (   
    uint8\_t \* \_\_p,   
    uint8\_t \_\_value )

Write a byte `__value` to EEPROM address `__p`.

**21.15.3.19 eeprom\_write\_double()** void eeprom\_write\_double (   
double \* \_\_p,   
double \_\_value )

Write a double \_\_value to EEPROM address \_\_p.

**21.15.3.20 eeprom\_write\_dword()** void eeprom\_write\_dword (   
uint32\_t \* \_\_p,   
uint32\_t \_\_value )

Write a 32-bit double word \_\_value to EEPROM address \_\_p.

**21.15.3.21 eeprom\_write\_float()** void eeprom\_write\_float (   
float \* \_\_p,   
float \_\_value )

Write a float \_\_value to EEPROM address \_\_p.

**21.15.3.22 eeprom\_write\_long\_double()** void eeprom\_write\_long\_double (   
long double \* \_\_p,   
long double \_\_value )

Write a long double \_\_value to EEPROM address \_\_p.

**21.15.3.23 eeprom\_write\_qword()** void eeprom\_write\_qword (   
uint64\_t \* \_\_p,   
uint64\_t \_\_value )

Write a 64-bit quad word \_\_value to EEPROM address \_\_p.

**21.15.3.24 eeprom\_write\_word()** void eeprom\_write\_word (   
uint16\_t \* \_\_p,   
uint16\_t \_\_value )

Write a word \_\_value to EEPROM address \_\_p.

## 21.16 <avr/fuse.h>: Fuse Support

### Introduction

The Fuse API allows a user to specify the fuse settings for the specific AVR device they are compiling for. These fuse settings will be placed in a special section in the ELF output file, after linking.

Programming tools can take advantage of the fuse information embedded in the ELF file, by extracting this information and determining if the fuses need to be programmed before programming the Flash and EEPROM memories. This also allows a single ELF file to contain all the information needed to program an AVR.

To use the Fuse API, include the <avr/io.h> header file, which in turn automatically includes the individual I/O header file and the <avr/fuse.h> file. These other two files provides everything necessary to set the AVR fuses.

## Fuse API

Each I/O header file must define the `FUSE_MEMORY_SIZE` macro which is defined to the number of fuse bytes that exist in the AVR device.

A new type, `__fuse_t`, is defined as a structure. The number of fields in this structure are determined by the number of fuse bytes in the `FUSE_MEMORY_SIZE` macro.

If `FUSE_MEMORY_SIZE == 1`, there is only a single field: byte, of type unsigned char.

If `FUSE_MEMORY_SIZE == 2`, there are two fields: low, and high, of type unsigned char.

If `FUSE_MEMORY_SIZE == 3`, there are three fields: low, high, and extended, of type unsigned char.

If `FUSE_MEMORY_SIZE > 3`, there is a single field: byte, which is an array of unsigned char with the size of the array being `FUSE_MEMORY_SIZE`.

A convenience macro, `FUSEMEM`, is defined as a GCC attribute for a custom-named section of ".fuse".

A convenience macro, `FUSES`, is defined that declares a variable, `__fuse`, of type `__fuse_t` with the attribute defined by `FUSEMEM`. This variable allows the end user to easily set the fuse data.

### Note

If a device-specific I/O header file has previously defined `FUSEMEM`, then `FUSEMEM` is not redefined. If a device-specific I/O header file has previously defined `FUSES`, then `FUSES` is not redefined.

Each AVR device I/O header file has a set of defined macros which specify the actual fuse bits available on that device. The AVR fuses have inverted values, logical 1 for an unprogrammed (disabled) bit and logical 0 for a programmed (enabled) bit. The defined macros for each individual fuse bit represent this in their definition by a bit-wise inversion of a mask. For example, the `FUSE_EESAVE` fuse in the ATmega128 is defined as:

```
#define FUSE_EESAVE    ~_BV(3)
```

### Note

The `_BV` macro creates a bit mask from a bit number. It is then inverted to represent logical values for a fuse memory byte.

To combine the fuse bits macros together to represent a whole fuse byte, use the bitwise AND operator, like so:

```
(FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN)
```

Each device I/O header file also defines macros that provide default values for each fuse byte that is available. `LFUSE_DEFAULT` is defined for a Low Fuse byte. `HFUSE_DEFAULT` is defined for a High Fuse byte. `EFUSE_DEFAULT` is defined for an Extended Fuse byte.

If `FUSE_MEMORY_SIZE > 3`, then the I/O header file defines macros that provide default values for each fuse byte like so: `FUSE0_DEFAULT FUSE1_DEFAULT FUSE2_DEFAULT FUSE3_DEFAULT FUSE4_DEFAULT ....`

## API Usage Example

Putting all of this together is easy. Using C99's designated initializers:

```
#include <avr/io.h>

FUSES =
{
    .low = LFUSE_DEFAULT,
    .high = (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN),
    .extended = EFUSE_DEFAULT,
};

int main(void)
{
    return 0;
}
```

Or, using the variable directly instead of the FUSES macro,

```
#include <avr/io.h>

__fuse_t __fuse __attribute__((section (".fuse"))) =
{
    .low = LFUSE_DEFAULT,
    .high = (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN),
    .extended = EFUSE_DEFAULT,
};

int main(void)
{
    return 0;
}
```

If you are compiling in C++, you cannot use the designated initializers so you must do:

```
#include <avr/io.h>

FUSES =
{
    LFUSE_DEFAULT, // .low
    (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN), // .high
    EFUSE_DEFAULT, // .extended
};

int main(void)
{
    return 0;
}
```

However there are a number of caveats that you need to be aware of to use this API properly.

Be sure to include <avr/io.h> to get all of the definitions for the API. The FUSES macro defines a global variable to store the fuse data. This variable is assigned to its own linker section. Assign the desired fuse values immediately in the variable initialization.

The .fuse section in the ELF file will get its values from the initial variable assignment ONLY. This means that you can NOT assign values to this variable in functions and the new values will not be put into the ELF .fuse section.

The global variable is declared in the FUSES macro has two leading underscores, which means that it is reserved for the "implementation", meaning the library, so it will not conflict with a user-named variable.

You must initialize ALL fields in the \_\_fuse\_t structure. This is because the fuse bits in all bytes default to a logical 1, meaning unprogrammed. Normal uninitialized data defaults to all logical zeros. So it is vital that all fuse bytes are initialized, even with default data. If they are not, then the fuse bits may not be programmed to the desired settings.

Be sure to have the -mmcu=*device* flag in your compile command line and your linker command line to have the correct device selected and to have the correct I/O header file included when you include <avr/io.h>.

You can print out the contents of the .fuse section in the ELF file by using this command line:

```
avr-objdump -s -j .fuse <ELF file>
```

The section contents shows the address on the left, then the data going from lower address to a higher address, left to right.



## 21.17 <avr/interrupt.h>: Interrupts

### Global manipulation of the interrupt flag

The global interrupt flag is maintained in the I bit of the status register (SREG).

Handling interrupts frequently requires attention regarding atomic access to objects that could be altered by code running within an interrupt context, see <util/atomic.h>.

Frequently, interrupts are being disabled for periods of time in order to perform certain operations without being disturbed; see [Problems with reordering code](#) for things to be taken into account with respect to compiler optimizations.

- #define [sei\(\)](#) `__asm__ __volatile__ ("sei" ::: "memory")`
- #define [cli\(\)](#) `__asm__ __volatile__ ("cli" ::: "memory")`

### Macros for writing interrupt handler functions

- #define [ISR](#)(vector, attributes)
- #define [SIGNAL](#)(vector)
- #define [EMPTY\\_INTERRUPT](#)(vector)
- #define [ISR\\_ALIAS](#)(vector, target\_vector)
- #define [reti\(\)](#) `__asm__ __volatile__ ("reti" ::: "memory")`
- #define [BADISR\\_vect](#)

### ISR attributes

- #define [ISR\\_BLOCK](#)
- #define [ISR\\_NOBLOCK](#)
- #define [ISR\\_NAKED](#)
- #define [ISR\\_FLATTEN](#)
- #define [ISR\\_NOICF](#)
- #define [ISR\\_NOGCCISR](#)
- #define [ISR\\_ALIASOF](#)(target\_vector)

#### 21.17.1 Detailed Description

##### Note

This discussion of interrupts was originally taken from Rich Neswold's document. See [Acknowledgments](#).

**Introduction to AVR-LibC's interrupt handling** It's nearly impossible to find compilers that agree on how to handle interrupt code. Since the C language tries to stay away from machine dependent details, each compiler writer is forced to design their method of support.

In the AVR-GCC environment, the vector table is predefined to point to interrupt routines with predetermined names. By using the appropriate name, your routine will be called when the corresponding interrupt occurs. The device library provides a set of default interrupt routines, which will get used if you don't define your own.

Patching into the vector table is only one part of the problem. The compiler uses, by convention, a set of registers when it's normally executing compiler-generated code. It's important that these registers, as well as the status register, get saved and restored.

These details seem to make interrupt routines a little messy, but all these details are handled by the Interrupt API. An interrupt routine is defined with `ISR()`. This macro registers and marks the routine as an interrupt handler for the specified peripheral. The following is an example definition of a handler for the ADC interrupt.

```
#include <avr/interrupt.h>

ISR(ADC_vect)
{
    // user code here
}
```

Refer to the chapter explaining [assembler programming](#) for an explanation about interrupt routines written solely in assembly.

**Catch-all interrupt vector** If an unexpected interrupt occurs (interrupt is enabled but no handler is installed, which usually indicates a bug), then the default action is to reset the device by jumping to the reset vector. You can override this by supplying a function named `BADISR_vect` which should be defined with `ISR()` as such. The name `BADISR_vect` is actually an alias for `__vector_default`. The latter must be used inside assembly code in case `<avr/interrupt.h>` is not included.

```
#include <avr/interrupt.h>

ISR(BADISR_vect)
{
    // user code here
}
```

**Nested interrupts** The AVR hardware clears the global interrupt flag in SREG when an interrupt request is serviced. Thus, normally interrupts will remain disabled inside the handler until the handler exits, where the `RETI` instruction (that is emitted by the compiler as part of the normal function epilogue for an ISR) will eventually re-enable further interrupts. For that reason, interrupt handlers normally do not nest. For most interrupt handlers, this is the desired behaviour, for some it is even required in order to prevent infinitely recursive interrupts (like UART interrupts, or level-triggered external interrupts). In rare circumstances though it might be desired to re-enable the global interrupt flag as early as possible in the interrupt handler, in order to not defer any other interrupt more than absolutely needed. This could be done using an `sei()` instruction right at the beginning of the interrupt handler, but this still leaves few instructions inside the compiler-generated function prologue to run with global interrupts disabled. The compiler can be instructed to insert a `SEI` instruction right at the beginning of an interrupt handler by declaring the handler the following way:

```
ISR(XXX_vect, ISR_NOBLOCK)
{
    ...
}
```

where `XXX_vect` is the name of a valid interrupt vector for the MCU type in question, as explained below.

**Two vectors sharing the same code** In some circumstances, the actions to be taken upon two different interrupts might be completely identical so a single implementation for the ISR would suffice. For example, pin-change interrupts arriving from two different ports could logically signal an event that is independent from the actual port (and thus interrupt vector) where it happened. Sharing interrupt vector code can be accomplished using the `ISR_ALIASOF()` attribute to the `ISR` macro:

```
ISR(PCINT0_vect)
{
    ...
    // Code to handle the event.
}

ISR(PCINT1_vect, ISR_ALIASOF(PCINT0_vect));
```

#### Note

There is no body to the aliased ISR.

Note that the `ISR_ALIASOF()` feature requires GCC 4.2 or above (or a patched version of GCC 4.1.x). See the documentation of the `ISR_ALIAS()` macro for an implementation which is less elegant but could be applied to all compiler versions.

**Empty interrupt service routines** In rare circumstances, an interrupt vector does not need any code to be implemented at all. The vector must be declared anyway, so when the interrupt triggers it won't execute the `BADISR_vect` code (which by default restarts the application).

This could for example be the case for interrupts that are solely enabled for the purpose of getting the controller out of `sleep_mode()`.

A handler for such an interrupt vector can be declared using the `EMPTY_INTERRUPT()` macro:

```
EMPTY_INTERRUPT(ADC_vect);
```

#### Note

There is no body to this macro.

**Manually defined ISRs** In some circumstances, the compiler-generated prologue and epilogue of the ISR might not be optimal for the job, and a manually defined ISR could be considered particularly to speedup the interrupt handling.

One solution to this could be to implement the entire ISR as manual assembly code in a separate (assembly) file. See [Combining C and assembly source files](#) for an example of how to implement it that way.

Another solution is to still implement the ISR in C language but take over the compiler's job of generating the prologue and epilogue. This can be done using the `ISR_NAKED` attribute to the `ISR()` macro. Note that the compiler does not generate *anything* as prologue or epilogue, so the final `reti()` must be provided by the actual implementation. `SREG` must be manually saved if the ISR code modifies it, and the compiler-implied assumption of `__zero_reg__` always being 0 could be wrong (e. g. when interrupting right after of a `MUL` instruction).

#### Warning

According to the GCC documentation, only [inline assembly](#) is supported in `naked` functions, like with `ISR_NAKED`.

```
ISR(TIMER1_OVF_vect, ISR_NAKED)
{
    PORTB |= _BV(0); // results in SBI which does not affect SREG
    reti();
}
```

**Choosing the vector: Interrupt vector names** The interrupt is chosen by supplying one of the vector names in the following table.

There are currently two different styles present for naming the vectors.

- Starting with AVR-LibC v1.4, the style of interrupt vector names is a short phrase for the vector description followed by `_vect`. The short phrase matches the vector name as described in the datasheet of the respective device (and in the hardware manufacturer's XML/ATDF files), with spaces replaced by an underscore and other non-alphanumeric characters dropped. Using the suffix `_vect` is intended to improve portability to other C compilers available for the AVR that use a similar naming convention.
- A **deprecated** form that uses names starting with `SIG_`, followed by a relatively verbose but arbitrarily chosen name describing the interrupt vector. This has been the only available style in AVR-LibC up to version 1.2.x. This historical naming style is not recommended for new projects, and some headers require that the macro `__AVR_LIBC_DEPRECATED_ENABLE__` is defined so that the `SIG_` names ISR names are available.

#### Note

The `ISR()` macro cannot really spell-check the argument passed to them. Thus, by misspelling one of the names below used in `ISR()`, a function will be created that, while possibly being usable as an interrupt function, is not actually wired into the interrupt vector table. The compiler will generate a warning if it detects a suspiciously looking name of an `ISR()` function (i.e. one that after macro replacement does not start with `"__vector_"`).

Apart from the `NAME_vect` macros listed below, for each such ISR name there is also a macro `NAME_vect_num` defined which resolves to the IRQ number. This is the index into the vector table, where indices start at index 0 (the reset vector).

See also [What ISR names are available for my device?](#) in the FAQ for how find all the IRQ names for a specific device.

**Table 24** Due to its sheer size, the following table is only available in the HTML version of the documentation.

Vector Name	Description	Applicable for Device
-------------	-------------	-----------------------

#### Note

For the following devices, only the deprecated `SIG_` names are available: AT43USB320, AT43USB355, AT76C711, AT90C8534, AT94K, M3000.

## 21.17.2 Macro Definition Documentation

### 21.17.2.1 `BADISR_vect` `#define BADISR_vect` `#include <avr/interrupt.h>`

This is a vector which is aliased to `__vector_default`, the vector executed when an IRQ fires with no accompanying ISR handler. This may be used along with the `ISR()` macro to create a catch-all for undefined but used ISRs for debugging purposes.

**21.17.2.2 cli** `#define cli( ) __asm__ __volatile__ ("cli" ::: "memory")`

Disables all interrupts by clearing the global interrupt mask. This function actually compiles into a single line of assembly, so there is no function call overhead. However, the macro also implies a *memory barrier* which can cause additional loss of optimization.

In order to implement atomic access to multi-byte objects, consider using the macros from `<util/atomic.h>`, rather than implementing them manually with `cli()` and `sei()`.

**21.17.2.3 EMPTY\_INTERRUPT** `#define EMPTY_INTERRUPT(  
vector )`

Defines an empty interrupt handler function. This will not generate any prolog or epilog code and will only return from the `ISR`. Do not define a function body as this will define it for you. Example:

```
EMPTY_INTERRUPT(ADC_vect);
```

**21.17.2.4 ISR** `#define ISR(  
vector,  
attributes )`

Introduces an interrupt handler function (interrupt service routine) that runs with global interrupts initially disabled by default with no attributes specified.

The attributes are optional and alter the behaviour and resultant generated code of the interrupt routine. Multiple attributes may be used for a single function, with a space separating each attribute.

Valid attributes are `ISR_BLOCK`, `ISR_NOBLOCK`, `ISR_NAKED`, `ISR_FLATTEN`, `ISR_NOICF`, `ISR_NOGCCISR` and `ISR_ALIASOF(vect)`.

`vector` must be one of the interrupt vector names that are valid for the particular MCU type.

**21.17.2.5 ISR\_ALIAS** `#define ISR_ALIAS(  
vector,  
target_vector )`

Aliases a given vector to another one in the same manner as the `ISR_ALIASOF` attribute for the `ISR()` macro. Unlike the `ISR_ALIASOF` attribute macro however, this is compatible for all versions of GCC rather than just GCC version 4.2 onwards.

#### Note

This macro creates a trampoline function for the aliased macro. This will result in a two cycle penalty for the aliased vector compared to the `ISR` the vector is aliased to, due to the `JMP/RJMP` opcode used.

**Deprecated** For new code, the use of `ISR(..., ISR_ALIASOF(...))` is recommended.

#### Example:

```
ISR(INT0_vect)
{
    PORTB = 42;
}
```

```
ISR_ALIAS(INT1_vect, INT0_vect);
```

**21.17.2.6 ISR\_ALIASOF** `#define ISR_ALIASOF(  
    target_vector )`

The ISR is linked to another ISR, specified by the `vect` parameter. This is compatible with GCC 4.2 and greater only.

Use this attribute in the attributes parameter of the `ISR` macro. Example:

```
ISR (INT0_vect)  
{  
    PORTB = 42;  
}  
  
ISR (INT1_vect, ISR_ALIASOF (INT0_vect));
```

**21.17.2.7 ISR\_BLOCK** `#define ISR_BLOCK`

Identical to an ISR with no attributes specified. Global interrupts are initially disabled by the AVR hardware when entering the ISR, without the compiler modifying this state.

Use this attribute in the attributes parameter of the `ISR` macro.

**21.17.2.8 ISR\_FLATTEN** `#define ISR_FLATTEN`

The compiler will try to inline all called function into the ISR. This has an effect with GCC 4.6 and newer only.

Use this attribute in the attributes parameter of the `ISR` macro.

**21.17.2.9 ISR\_NAKED** `#define ISR_NAKED`

ISR is created with no prologue or epilogue code. The user code is responsible for preservation of the machine state including the SREG register, as well as placing a `reti()` at the end of the interrupt routine.

Use this attribute in the attributes parameter of the `ISR` macro.

#### Note

According to GCC documentation, the only code supported in naked functions is [inline assembly](#).

**21.17.2.10 ISR\_NOBLOCK** `#define ISR_NOBLOCK`

ISR runs with global interrupts initially enabled. The interrupt enable flag is activated by the compiler as early as possible within the ISR to ensure minimal processing delay for nested interrupts.

This may be used to create nested ISRs, however care should be taken to avoid stack overflows, or to avoid infinitely entering the ISR for those cases where the AVR hardware does not clear the respective interrupt flag before entering the ISR.

Use this attribute in the attributes parameter of the `ISR` macro.

**21.17.2.11 ISR\_NOGCCISR** `#define ISR_NOGCCISR`

Do not generate `__gcc_isr pseudo instructions` for this ISR. This has an effect with `GCC 8` and newer only.

Use this attribute in the attributes parameter of the `ISR` macro.

**21.17.2.12 ISR\_NOICF** `#define ISR_NOICF`

Avoid identical-code-folding optimization against this ISR. This has an effect with `GCC 5` and newer only.

Use this attribute in the attributes parameter of the `ISR` macro.

**21.17.2.13 reti** `#define reti( ) __asm__ __volatile__ ("reti" ::: "memory")`

Returns from an interrupt routine, enabling global interrupts. This should be the last command executed before leaving an `ISR` defined with the `ISR_NAKED` attribute.

This macro actually compiles into a single line of assembly, so there is no function call overhead.

**Note**

According to the GCC documentation, the only code supported in naked functions is [inline assembly](#).

**21.17.2.14 sei** `#define sei( ) __asm__ __volatile__ ("sei" ::: "memory")`

Enables interrupts by setting the global interrupt mask. This function actually compiles into a single line of assembly, so there is no function call overhead. However, the macro also implies a *memory barrier* which can cause additional loss of optimization.

In order to implement atomic access to multi-byte objects, consider using the macros from `<util/atomic.h>`, rather than implementing them manually with `cli()` and `sei()`.

**21.17.2.15 SIGNAL** `#define SIGNAL(  
vector )`

Introduces an interrupt handler function that runs with global interrupts initially disabled.

This is the same as the `ISR` macro without optional attributes.

**Deprecated** Do not use `SIGNAL()` in new code. Use `ISR()` instead.

**21.18 <avr/io.h>: AVR device-specific IO definitions****Macros**

- `#define _PROTECTED_WRITE(reg, value)`
- `#define _PROTECTED_WRITE_SPM(reg, value)`

### 21.18.1 Detailed Description

```
#include <avr/io.h>
```

This header file includes the appropriate IO definitions for the device that has been specified by the `-mmcu=` compiler command-line switch. This is done by diverting to the appropriate file `<avr/ioXXXX.h>` which should never be included directly. Some register names common to all AVR devices are defined directly within `<avr/common.h>`, which is included in `<avr/io.h>`, but most of the details come from the respective include file.

Note that this file always includes the following files:

```
#include <avr/sfr_defs.h>
#include <avr/portpins.h>
#include <avr/common.h>
#include <avr/version.h>
```

See `<avr/sfr_defs.h>`: [Special function registers](#) for more details about that header file.

Included are definitions of the IO register set and their respective bit values as specified in the Atmel documentation. Note that inconsistencies in naming conventions, so even identical functions sometimes get different names on different devices.

Also included are the specific names useable for interrupt function definitions as documented [here](#).

Finally, the following macros are defined:

- **RAMEND**  
The last on-chip RAM address.
- **XRAMEND**  
The last possible RAM location that is addressable. This is equal to RAMEND for devices that do not allow for external RAM. For devices that allow external RAM, this will be larger than RAMEND.
- **E2END**  
The last EEPROM address.
- **FLASHEND**  
The last byte address in the Flash program space.
- **SPM\_PAGESIZE**  
For devices with bootloader support, the flash pagesize (in bytes) to be used for the SPM instruction.
- **E2PAGESIZE**  
The size of the EEPROM page.

### 21.18.2 Macro Definition Documentation

```
21.18.2.1 _PROTECTED_WRITE #define _PROTECTED_WRITE(  
    reg,  
    value )
```

Write value `value` to IO register `reg` that is protected through the Xmega configuration change protection (CCP) mechanism. This implements the timed sequence that is required for CCP.

Example to modify the CPU clock:

```
#include <avr/io.h>

_PROTECTED_WRITE(CLK_PSCTRL, CLK_PSADIV0_bm);
_PROTECTED_WRITE(CLK_CTRL, CLK_SCLKSEL0_bm);
```



```

21.18.2.2 _PROTECTED_WRITE_SPM #define _PROTECTED_WRITE_SPM(
    reg,
    value )

```

Write value `value` to register `reg` that is protected through the Xmega configuration change protection (CCP) key for self programming (SPM). This implements the timed sequence that is required for CCP.

Example to modify the CPU clock:

```

#include <avr/io.h>

_PROTECTED_WRITE_SPM(NVMCTRL_CTRLA, NVMCTRL_CMD_PAGEERASEWRITE_gc);

```

## 21.19 <avr/lock.h>: Lockbit Support

### Introduction

The Lockbit API allows a user to specify the lockbit settings for the specific AVR device they are compiling for. These lockbit settings will be placed in a special section in the ELF output file, after linking.

Programming tools can take advantage of the lockbit information embedded in the ELF file, by extracting this information and determining if the lockbits need to be programmed after programming the Flash and EEPROM memories. This also allows a single ELF file to contain all the information needed to program an AVR.

To use the Lockbit API, include the <avr/io.h> header file, which in turn automatically includes the individual I/O header file and the <avr/lock.h> file. These other two files provides everything necessary to set the AVR lockbits.

### Lockbit API

Each I/O header file may define up to 3 macros that controls what kinds of lockbits are available to the user.

If `__LOCK_BITS_EXIST` is defined, then two lock bits are available to the user and 3 mode settings are defined for these two bits.

If `__BOOT_LOCK_BITS_0_EXIST` is defined, then the two BLB0 lock bits are available to the user and 4 mode settings are defined for these two bits.

If `__BOOT_LOCK_BITS_1_EXIST` is defined, then the two BLB1 lock bits are available to the user and 4 mode settings are defined for these two bits.

If `__BOOT_LOCK_APPLICATION_TABLE_BITS_EXIST` is defined then two lock bits are available to set the locking mode for the Application Table Section (which is used in the XMEGA family).

If `__BOOT_LOCK_APPLICATION_BITS_EXIST` is defined then two lock bits are available to set the locking mode for the Application Section (which is used in the XMEGA family).

If `__BOOT_LOCK_BOOT_BITS_EXIST` is defined then two lock bits are available to set the locking mode for the Boot Loader Section (which is used in the XMEGA family).

The AVR lockbit modes have inverted values, logical 1 for an unprogrammed (disabled) bit and logical 0 for a programmed (enabled) bit. The defined macros for each individual lock bit represent this in their definition by a bit-wise inversion of a mask. For example, the `LB_MODE_3` macro is defined as:

```

#define LB_MODE_3 (0xFC)

```

To combine the lockbit mode macros together to represent a whole byte, use the bitwise AND operator, like so:

```

(LB_MODE_3 & BLB0_MODE_2)

```

<avr/lock.h> also defines a macro that provides a default lockbit value: `LOCKBITS_DEFAULT` which is defined to be `0xFF`.

See the AVR device specific datasheet for more details about these lock bits and the available mode settings.

A convenience macro, `LOCKMEM`, is defined as a GCC attribute for a custom-named section of ".lock".

A convenience macro, `LOCKBITS`, is defined that declares a variable, `__lock`, of type unsigned char with the attribute defined by `LOCKMEM`. This variable allows the end user to easily set the lockbit data.

## Note

If a device-specific I/O header file has previously defined LOCKMEM, then LOCKMEM is not redefined. If a device-specific I/O header file has previously defined LOCKBITS, then LOCKBITS is not redefined. LOCKBITS is currently known to be defined in the I/O header files for the XMEGA devices.

## API Usage Example

Putting all of this together is easy:

```
#include <avr/io.h>

LOCKBITS = (LB_MODE_1 & BLB0_MODE_3 & BLB1_MODE_4);

int main(void)
{
    return 0;
}
```

Or:

```
#include <avr/io.h>

unsigned char __lock __attribute__((section (".lock"))) =
    (LB_MODE_1 & BLB0_MODE_3 & BLB1_MODE_4);

int main(void)
{
    return 0;
}
```

However there are a number of caveats that you need to be aware of to use this API properly.

Be sure to include <avr/io.h> to get all of the definitions for the API. The LOCKBITS macro defines a global variable to store the lockbit data. This variable is assigned to its own linker section. Assign the desired lockbit values immediately in the variable initialization.

The .lock section in the ELF file will get its values from the initial variable assignment ONLY. This means that you can NOT assign values to this variable in functions and the new values will not be put into the ELF .lock section.

The global variable is declared in the LOCKBITS macro has two leading underscores, which means that it is reserved for the "implementation", meaning the library, so it will not conflict with a user-named variable.

You must initialize the lockbit variable to some meaningful value, even if it is the default value. This is because the lockbits default to a logical 1, meaning unprogrammed. Normal uninitialized data defaults to all logical zeros. So it is vital that all lockbits are initialized, even with default data. If they are not, then the lockbits may not be programmed to the desired settings and can possibly put your device into an unrecoverable state.

Be sure to have the `-mmcu=device` flag in your compile command line and your linker command line to have the correct device selected and to have the correct I/O header file included when you include <avr/io.h>.

You can print out the contents of the .lock section in the ELF file by using this command line:

```
avr-objdump -s -j .lock <ELF file>
```

## 21.20 <avr/pgmspace.h>: Program Space Utilities

### Macros

- #define `PROGMEM_FAR` \_\_attribute\_\_((\_\_section\_\_(".progmemx.data")))
- #define `PROGMEM` \_\_attribute\_\_((\_\_progmem\_\_))
- #define `PSTR`(str) ({ static const `PROGMEM` char c[] = (str); &c[0]; })
- #define `PSTR_FAR`(str) ({ static const `PROGMEM_FAR` char c[] = (str); `pgm_get_far_address`(c[0]); })
- #define `pgm_read_byte_near`(\_\_addr) \_\_LPM ((uint16\_t)(\_\_addr))
- #define `pgm_read_word_near`(\_\_addr) \_\_LPM\_word ((uint16\_t)(\_\_addr))
- #define `pgm_read_dword_near`(\_\_addr) \_\_LPM\_dword ((uint16\_t)(\_\_addr))
- #define `pgm_read_qword_near`(\_\_addr) \_\_LPM\_qword ((uint16\_t)(\_\_addr))
- #define `pgm_read_float_near`(addr) `pgm_read_float` (addr)
- #define `pgm_read_ptr_near`(\_\_addr) ((void\*) \_\_LPM\_word ((uint16\_t)(\_\_addr)))
- #define `pgm_read_byte_far`(\_\_addr) \_\_ELPM (\_\_addr)
- #define `pgm_read_word_far`(\_\_addr) \_\_ELPM\_word (\_\_addr)
- #define `pgm_read_dword_far`(\_\_addr) \_\_ELPM\_dword (\_\_addr)
- #define `pgm_read_qword_far`(\_\_addr) \_\_ELPM\_qword (\_\_addr)
- #define `pgm_read_ptr_far`(\_\_addr) ((void\*) \_\_ELPM\_word (\_\_addr))
- #define `pgm_read_byte`(\_\_addr) `pgm_read_byte_near`(\_\_addr)
- #define `pgm_read_word`(\_\_addr) `pgm_read_word_near`(\_\_addr)
- #define `pgm_read_dword`(\_\_addr) `pgm_read_dword_near`(\_\_addr)
- #define `pgm_read_qword`(\_\_addr) `pgm_read_qword_near`(\_\_addr)
- #define `pgm_read_ptr`(\_\_addr) `pgm_read_ptr_near`(\_\_addr)
- #define `pgm_get_far_address`(var)

### Functions

- static char `pgm_read_char` (const char \*\_\_addr)
- static unsigned char `pgm_read_unsigned_char` (const unsigned char \*\_\_addr)
- static signed char `pgm_read_signed_char` (const signed char \*\_\_addr)
- static `uint8_t` `pgm_read_u8` (const `uint8_t` \*\_\_addr)
- static `int8_t` `pgm_read_i8` (const `int8_t` \*\_\_addr)
- static short `pgm_read_short` (const short \*\_\_addr)
- static unsigned short `pgm_read_unsigned_short` (const unsigned short \*\_\_addr)
- static `uint16_t` `pgm_read_u16` (const `uint16_t` \*\_\_addr)
- static `int16_t` `pgm_read_i16` (const `int16_t` \*\_\_addr)
- static int `pgm_read_int` (const int \*\_\_addr)
- static signed `pgm_read_signed` (const signed \*\_\_addr)
- static unsigned `pgm_read_unsigned` (const unsigned \*\_\_addr)
- static signed int `pgm_read_signed_int` (const signed int \*\_\_addr)
- static unsigned int `pgm_read_unsigned_int` (const unsigned int \*\_\_addr)
- static `__int24` `pgm_read_i24` (const `__int24` \*\_\_addr)
- static `__uint24` `pgm_read_u24` (const `__uint24` \*\_\_addr)
- static `uint32_t` `pgm_read_u32` (const `uint32_t` \*\_\_addr)
- static `int32_t` `pgm_read_i32` (const `int32_t` \*\_\_addr)
- static long `pgm_read_long` (const long \*\_\_addr)
- static unsigned long `pgm_read_unsigned_long` (const unsigned long \*\_\_addr)
- static long long `pgm_read_long_long` (const long long \*\_\_addr)
- static unsigned long long `pgm_read_unsigned_long_long` (const unsigned long long \*\_\_addr)
- static `uint64_t` `pgm_read_u64` (const `uint64_t` \*\_\_addr)
- static `int64_t` `pgm_read_i64` (const `int64_t` \*\_\_addr)
- static float `pgm_read_float` (const float \*\_\_addr)

- static double `pgm_read_double` (const double \*\_\_addr)
- static long double `pgm_read_long_double` (const long double \*\_\_addr)
- static char `pgm_read_char_far` (uint\_farptr\_t \_\_addr)
- static unsigned char `pgm_read_unsigned_char_far` (uint\_farptr\_t \_\_addr)
- static signed char `pgm_read_signed_char_far` (uint\_farptr\_t \_\_addr)
- static uint8\_t `pgm_read_u8_far` (uint\_farptr\_t \_\_addr)
- static int8\_t `pgm_read_i8_far` (uint\_farptr\_t \_\_addr)
- static short `pgm_read_short_far` (uint\_farptr\_t \_\_addr)
- static unsigned short `pgm_read_unsigned_short_far` (uint\_farptr\_t \_\_addr)
- static uint16\_t `pgm_read_u16_far` (uint\_farptr\_t \_\_addr)
- static int16\_t `pgm_read_i16_far` (uint\_farptr\_t \_\_addr)
- static int `pgm_read_int_far` (uint\_farptr\_t \_\_addr)
- static unsigned `pgm_read_unsigned_far` (uint\_farptr\_t \_\_addr)
- static unsigned int `pgm_read_unsigned_int_far` (uint\_farptr\_t \_\_addr)
- static signed `pgm_read_signed_far` (uint\_farptr\_t \_\_addr)
- static signed int `pgm_read_signed_int_far` (uint\_farptr\_t \_\_addr)
- static long `pgm_read_long_far` (uint\_farptr\_t \_\_addr)
- static unsigned long `pgm_read_unsigned_long_far` (uint\_farptr\_t \_\_addr)
- static \_\_int24 `pgm_read_i24_far` (uint\_farptr\_t \_\_addr)
- static \_\_uint24 `pgm_read_u24_far` (uint\_farptr\_t \_\_addr)
- static uint32\_t `pgm_read_u32_far` (uint\_farptr\_t \_\_addr)
- static int32\_t `pgm_read_i32_far` (uint\_farptr\_t \_\_addr)
- static long long `pgm_read_long_long_far` (uint\_farptr\_t \_\_addr)
- static unsigned long long `pgm_read_unsigned_long_long_far` (uint\_farptr\_t \_\_addr)
- static uint64\_t `pgm_read_u64_far` (uint\_farptr\_t \_\_addr)
- static int64\_t `pgm_read_i64_far` (uint\_farptr\_t \_\_addr)
- static float `pgm_read_float_far` (uint\_farptr\_t \_\_addr)
- static double `pgm_read_double_far` (uint\_farptr\_t \_\_addr)
- static long double `pgm_read_long_double_far` (uint\_farptr\_t \_\_addr)
- const void \* `memchr_P` (const void \*, int \_\_val, size\_t \_\_len)
- int `memcmp_P` (const void \*, const void \*, size\_t)
- void \* `memcpy_P` (void \*, const void \*, int \_\_val, size\_t)
- void \* `memcpy_P` (void \*, const void \*, size\_t)
- void \* `memmem_P` (const void \*, size\_t, const void \*, size\_t)
- const void \* `memrchr_P` (const void \*, int \_\_val, size\_t \_\_len)
- char \* `strcat_P` (char \*, const char \*)
- const char \* `strchr_P` (const char \*, int \_\_val)
- const char \* `strchrnul_P` (const char \*, int \_\_val)
- int `strcmp_P` (const char \*, const char \*)
- char \* `strcpy_P` (char \*, const char \*)
- int `strcasecmp_P` (const char \*, const char \*)
- char \* `strcasestr_P` (const char \*, const char \*)
- size\_t `strcspn_P` (const char \*\_\_s, const char \*\_\_reject)
- size\_t `strlcat_P` (char \*, const char \*, size\_t)
- size\_t `strncpy_P` (char \*, const char \*, size\_t)
- size\_t `strlen_P` (const char \*, size\_t)
- int `strncmp_P` (const char \*, const char \*, size\_t)
- int `strncasecmp_P` (const char \*, const char \*, size\_t)
- char \* `strncat_P` (char \*, const char \*, size\_t)
- char \* `strncpy_P` (char \*, const char \*, size\_t)
- char \* `strpbrk_P` (const char \*\_\_s, const char \*\_\_accept)
- const char \* `strrchr_P` (const char \*, int \_\_val)
- char \* `strsep_P` (char \*\*\_\_sp, const char \*\_\_delim)
- size\_t `strspn_P` (const char \*\_\_s, const char \*\_\_accept)
- char \* `strstr_P` (const char \*, const char \*)

- `char * strtok_P (char *__s, const char *__delim)`
- `char * strtok_rP (char *__s, const char *__delim, char **__last)`
- `size_t strlen_PF (uint_farptr_t src)`
- `size_t strlen_PF (uint_farptr_t src, size_t len)`
- `void * memcpy_PF (void *dest, uint_farptr_t src, size_t len)`
- `char * strcpy_PF (char *dest, uint_farptr_t src)`
- `char * strncpy_PF (char *dest, uint_farptr_t src, size_t len)`
- `char * strcat_PF (char *dest, uint_farptr_t src)`
- `size_t strlcat_PF (char *dst, uint_farptr_t src, size_t siz)`
- `char * strncat_PF (char *dest, uint_farptr_t src, size_t len)`
- `int strcmp_PF (const char *s1, uint_farptr_t s2)`
- `int strncmp_PF (const char *s1, uint_farptr_t s2, size_t n)`
- `int strcasecmp_PF (const char *s1, uint_farptr_t s2)`
- `int strncasecmp_PF (const char *s1, uint_farptr_t s2, size_t n)`
- `uint_farptr_t strchr_PF (uint_farptr_t, int __val)`
- `char * strstr_PF (const char *s1, uint_farptr_t s2)`
- `size_t strlcpy_PF (char *dst, uint_farptr_t src, size_t siz)`
- `int memcmp_PF (const void *, uint_farptr_t, size_t)`
- `static size_t strlen_P (const char *s)`
- `template<typename T >`  
`T pgm_read< T > (const T *addr)`
- `template<typename T >`  
`T pgm_read_far< T > (uint_farptr_t addr)`

### 21.20.1 Detailed Description

```
#include <avr/io.h>
#include <avr/pgmspace.h>
```

The functions in this module provide interfaces for a program to access data stored in program space (flash memory) of the device.

#### Note

These functions are an attempt to provide some compatibility with header files that come with IAR C, to make porting applications between different compilers easier. This is not 100% compatibility though (GCC does not have full support for multiple address spaces yet).

If you are working with strings which are completely based in RAM, use the standard string functions described in `<string.h>`: [Strings](#).

If possible, put your constant tables in the lower 64 KB and use `pgm_read_byte_near()` or `pgm_read_word_near()` instead of `pgm_read_byte_far()` or `pgm_read_word_far()` since it is more efficient that way, and you can still use the upper 64K for executable code. All functions that are suffixed with a `_P` *require* their arguments to be in the lower 64 KB of the flash ROM, as they do not use ELPM instructions. This is normally not a big concern as the linker setup arranges any program space constants declared using the macros from this header file so they are placed right after the interrupt vectors, and in front of any executable code. However, it can become a problem if there are too many of these constants, or for bootloaders on devices with more than 64 KB of ROM. *All these functions will not work in that situation.*

For **Xmega** devices, make sure the NVM controller command register (`NVM_CMD` or `NVM_CMD`) is set to 0x00 (NOP) before using any of these functions.

### 21.20.2 Macro Definition Documentation

**21.20.2.1 pgm\_get\_far\_address** #define pgm\_get\_far\_address(  
     var )

**Value:**

```
(( {
    uint_farptr_t __tmp;

    __asm__ __volatile__ (
        "ldi    %A0, lo8(%1)"      "\n\t"
        "ldi    %B0, hi8(%1)"      "\n\t"
        "ldi    %C0, hh8(%1)"      "\n\t"
        "clr    %D0"
        : "=d" (__tmp)
        : "i" (&(var))
    );
    __tmp;
}))
```

This macro evaluates to a `uint_farptr_t` 32-bit "far" pointer (only 24 bits used) to data even beyond the 64 KiB limit for the 16-bit ordinary pointer. It is similar to the '&' operator, with some limitations. Example:

```
#include <avr/pgmspace.h>

// Section .progmemx.data is located after all the code sections.
__attribute__((section(".progmemx.data")))
const int data[] = { 2, 3, 5, 7, 9, 11 };

int get_data (uint8_t idx)
{
    uint_farptr_t pdata = pgm_get_far_address (data[0]);
    return pgm_read_int_far (pdata + idx * sizeof(int));
}
```

**Comments:**

- The overhead is minimal and it's mainly due to the 32-bit size operation.
- 24 bit sizes guarantees the code compatibility for use in future devices.
- `var` has to be resolved at link-time as an existing symbol, i.e. a simple variable name, an array name, or an array or structure element provided the offset is known at compile-time, and `var` is located in static storage, etc.
- The returned value is the symbol's **VMA** (virtual memory address) determined by the linker and falls in the corresponding memory region. The AVR Harvard architecture requires non-overlapping VMA areas for the multiple **memory regions** in the processor: Flash ROM, RAM, and EEPROM. Typical offset for these are `0x0`, `0x800xx0`, and `0x810000` respectively, derived from the linker script used and linker options.

**21.20.2.2 pgm\_read\_byte** #define pgm\_read\_byte(  
     \_\_addr ) pgm\_read\_byte\_near(\_\_addr)

Read a byte from the program space with a 16-bit (near) nyte-address.

**21.20.2.3 pgm\_read\_byte\_far** #define pgm\_read\_byte\_far(  
     \_\_addr ) \_\_ELPM (\_\_addr)

Read a byte from the program space with a 32-bit (far) byte-address.

**21.20.2.4 pgm\_read\_byte\_near** #define pgm\_read\_byte\_near(  
     \_\_addr ) \_\_LPM ((uint16\_t)(\_\_addr))

Read a byte from the program space with a 16-bit (near) byte-address.

**21.20.2.5 `pgm_read_dword`** `#define pgm_read_dword(  
__addr ) pgm\_read\_dword\_near(__addr)`

Read a double word from the program space with a 16-bit (near) byte-address.

**21.20.2.6 `pgm_read_dword_far`** `#define pgm_read_dword_far(  
__addr ) \_\_ELPM\_dword (__addr)`

Read a double word from the program space with a 32-bit (far) byte-address.

**21.20.2.7 `pgm_read_dword_near`** `#define pgm_read_dword_near(  
__addr ) \_\_LPM\_dword ((uint16\_t)(__addr))`

Read a double word from the program space with a 16-bit (near) byte-address.

**21.20.2.8 `pgm_read_float_near`** `#define pgm_read_float_near(  
addr ) pgm\_read\_float (addr)`

Read a `float` from the program space with a 16-bit (near) byte-address.

**21.20.2.9 `pgm_read_ptr`** `#define pgm_read_ptr(  
__addr ) pgm\_read\_ptr\_near(__addr)`

Read a pointer from the program space with a 16-bit (near) byte-address.

**21.20.2.10 `pgm_read_ptr_far`** `#define pgm_read_ptr_far(  
__addr ) ((void*) \_\_ELPM\_word (__addr))`

Read a pointer from the program space with a 32-bit (far) byte-address.

**21.20.2.11 `pgm_read_ptr_near`** `#define pgm_read_ptr_near(  
__addr ) ((void*) \_\_LPM\_word ((uint16\_t)(__addr)))`

Read a pointer from the program space with a 16-bit (near) byte-address.

**21.20.2.12 `pgm_read_qword`** `#define pgm_read_qword(  
__addr ) pgm\_read\_qword\_near(__addr)`

Read a quad-word from the program space with a 16-bit (near) byte-address.

**21.20.2.13 `pgm_read_qword_far`** `#define pgm_read_qword_far(  
__addr ) \_\_ELPM\_qword (__addr)`

Read a quad-word from the program space with a 32-bit (far) byte-address.

**21.20.2.14 `pgm_read_qword_near`** `#define pgm_read_qword_near(  
__addr ) \_\_LPM\_qword ((uint16\_t)(__addr))`

Read a quad-word from the program space with a 16-bit (near) byte-address.

**21.20.2.15** `pgm_read_word` `#define pgm_read_word(`  
`__addr ) pgm_read_word_near(__addr)`

Read a word from the program space with a 16-bit (near) byte-address.

**21.20.2.16** `pgm_read_word_far` `#define pgm_read_word_far(`  
`__addr ) __ELPM_word (__addr)`

Read a word from the program space with a 32-bit (far) byte-address.

**21.20.2.17** `pgm_read_word_near` `#define pgm_read_word_near(`  
`__addr ) __LPM_word ((uint16_t)(__addr))`

Read a word from the program space with a 16-bit (near) byte-address.

**21.20.2.18** `PROGMEM` `#define PROGMEM __attribute__((__progmem__))`

Attribute to use in order to declare an object being located in flash ROM.

**21.20.2.19** `PROGMEM_FAR` `#define PROGMEM_FAR __attribute__((__section__(".progmemx.data")))`

Attribute to use in order to declare an object being located in far flash ROM. This is similar to `PROGMEM`, except that it puts the static storage object in section `.progmemx.data`. In order to access the object, the `pgm_read_*_far` and `_PF` functions declare in this header can be used. In order to get its address, see `pgm_get_far_address()`.

It only makes sense to put read-only objects in this section, though the compiler does not diagnose when this is not the case.

**21.20.2.20** `PSTR` `#define PSTR(`  
`str ) ({ static const PROGMEM char c[] = (str); &c[0]; })`

Used to declare a static pointer to a string in program space.

**21.20.2.21** `PSTR_FAR` `#define PSTR_FAR(`  
`str ) ({ static const PROGMEM_FAR char c[] = (str); pgm_get_far_address(c[0]);`  
`})`

Used to define a string literal in far program space, and to return its address of type `uint_farptr_t`.

## 21.20.3 Function Documentation

**21.20.3.1** `memccpy_P()` `void * memccpy_P (`  
`void * dest,`  
`const void * src,`  
`int val,`  
`size_t len )`

This function is similar to `memccpy()` except that `src` is pointer to a string in program space.



```
21.20.3.2 memchr_P() const void * memchr_P (  
    const void * s,  
    int val,  
    size_t len )
```

Scan flash memory for a character.

The [memchr\\_P\(\)](#) function scans the first `len` bytes of the flash memory area pointed to by `s` for the character `val`. The first byte to match `val` (interpreted as an unsigned character) stops the operation.

#### Returns

The [memchr\\_P\(\)](#) function returns a pointer to the matching byte or `NULL` if the character does not occur in the given memory area.

```
21.20.3.3 memcmp_P() int memcmp_P (  
    const void * s1,  
    const void * s2,  
    size_t len )
```

Compare memory areas.

The [memcmp\\_P\(\)](#) function compares the first `len` bytes of the memory areas `s1` and flash `s2`. The comparison is performed using unsigned char operations.

#### Returns

The [memcmp\\_P\(\)](#) function returns an integer less than, equal to, or greater than zero if the first `len` bytes of `s1` is found, respectively, to be less than, to match, or be greater than the first `len` bytes of `s2`.

```
21.20.3.4 memcmp_PF() int memcmp_PF (  
    const void * s1,  
    uint_farptr_t s2,  
    size_t len )
```

Compare memory areas.

The [memcmp\\_PF\(\)](#) function compares the first `len` bytes of the memory areas `s1` and flash `s2`. The comparison is performed using unsigned char operations. It is an equivalent of [memcmp\\_P\(\)](#) function, except that it is capable working on all FLASH including the extended area above 64kB.

#### Returns

The [memcmp\\_PF\(\)](#) function returns an integer less than, equal to, or greater than zero if the first `len` bytes of `s1` is found, respectively, to be less than, to match, or be greater than the first `len` bytes of `s2`.

```
21.20.3.5 memcpy_P() void * memcpy_P (
    void * dest,
    const void * src,
    size_t n )
```

The `memcpy_P()` function is similar to `memcpy()`, except the `src` string resides in program space.

#### Returns

The `memcpy_P()` function returns a pointer to `dest`.

```
21.20.3.6 memcpy_PF() void * memcpy_PF (
    void * dest,
    uint_farptr_t src,
    size_t n )
```

Copy a memory block from flash to SRAM.

The `memcpy_PF()` function is similar to `memcpy()`, except the data is copied from the program space and is addressed using a far pointer.

#### Parameters

<i>dest</i>	A pointer to the destination buffer
<i>src</i>	A far pointer to the origin of data in flash memory
<i>n</i>	The number of bytes to be copied

#### Returns

The `memcpy_PF()` function returns a pointer to `dst`. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.7 memmem_P() void * memmem_P (
    const void * s1,
    size_t len1,
    const void * s2,
    size_t len2 )
```

The `memmem_P()` function is similar to `memmem()` except that `s2` is pointer to a string in program space.

```
21.20.3.8 memrchr_P() const void memrchr_P (
    const void * src,
    int val,
    size_t len )
```

The `memrchr_P()` function is like the `memchr_P()` function, except that it searches backwards from the end of the `len` bytes pointed to by `src` instead of forwards from the front. (Glibc, GNU extension.)

### Returns

The `memchr_P()` function returns a pointer to the matching byte or `NULL` if the character does not occur in the given memory area.

```
21.20.3.9 pgm_read< T >() template<typename T >  
T pgm_read< T > (  
    const T * addr )
```

Read an object of type `T` from program memory address `addr` and return it. This template is only available when macro `__pgm_read_template__` is defined.

```
21.20.3.10 pgm_read_char() char pgm_read_char (  
    const char * __addr ) [inline], [static]
```

Read a `char` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

```
21.20.3.11 pgm_read_char_far() char pgm_read_char_far (  
    uint_farptr_t __addr ) [inline], [static]
```

Read a `char` from far byte-address `__addr`. The address is in the program memory.

```
21.20.3.12 pgm_read_double() double pgm_read_double (  
    const double * __addr ) [inline], [static]
```

Read a `double` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

```
21.20.3.13 pgm_read_double_far() double pgm_read_double_far (  
    uint_farptr_t __addr ) [inline], [static]
```

Read a `double` from far byte-address `__addr`. The address is in the program memory.

```
21.20.3.14 pgm_read_far< T >() template<typename T >  
T pgm_read_far< T > (  
    uint_farptr_t addr )
```

Read an object of type `T` from program memory address `addr` and return it. This template is only available when macro `__pgm_read_template__` is defined.

```
21.20.3.15 pgm_read_float() float pgm_read_float (  
    const float * __addr ) [inline], [static]
```

Read a `float` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.16** `pgm_read_float_far()` `float` `pgm_read_float_far` (  
    `uint_farptr_t` `__addr`) [`inline`], [`static`]

Read a `float` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.17** `pgm_read_i16()` `int16_t` `pgm_read_i16` (  
    const `int16_t` \* `__addr`) [`inline`], [`static`]

Read an `int16_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.18** `pgm_read_i16_far()` `int16_t` `pgm_read_i16_far` (  
    `uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `int16_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.19** `pgm_read_i24()` `__int24` `pgm_read_i24` (  
    const `__int24` \* `__addr`) [`inline`], [`static`]

Read an `__int24` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.20** `pgm_read_i24_far()` `__int24` `pgm_read_i24_far` (  
    `uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `__int24` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.21** `pgm_read_i32()` `int32_t` `pgm_read_i32` (  
    const `int32_t` \* `__addr`) [`inline`], [`static`]

Read an `int32_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.22** `pgm_read_i32_far()` `int32_t` `pgm_read_i32_far` (  
    `uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `int32_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.23** `pgm_read_i64()` `int64_t` `pgm_read_i64` (  
    const `int64_t` \* `__addr`) [`inline`], [`static`]

Read an `int64_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.24** `pgm_read_i64_far()` `int64_t` `pgm_read_i64_far` (  
    `uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `int64_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.25** `pgm_read_i8()` `int8_t` `pgm_read_i8` (  
    const `int8_t` \* `__addr` ) [inline], [static]

Read an `int8_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.26** `pgm_read_i8_far()` `int8_t` `pgm_read_i8_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read an `int8_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.27** `pgm_read_int()` `int` `pgm_read_int` (  
    const `int` \* `__addr` ) [inline], [static]

Read an `int` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.28** `pgm_read_int_far()` `int` `pgm_read_int_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read an `int` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.29** `pgm_read_long()` `long` `pgm_read_long` (  
    const `long` \* `__addr` ) [inline], [static]

Read a `long` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.30** `pgm_read_long_double()` `long double` `pgm_read_long_double` (  
    const `long double` \* `__addr` ) [inline], [static]

Read a `long double` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.31** `pgm_read_long_double_far()` `long double` `pgm_read_long_double_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read a `long double` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.32** `pgm_read_long_far()` `long` `pgm_read_long_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read a `long` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.33** `pgm_read_long_long()` `long long` `pgm_read_long_long` (  
    const `long long` \* `__addr` ) [inline], [static]

Read a `long long` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.334** `pgm_read_long_long_far()` `long long pgm_read_long_long_far (`  
`uint_farptr_t __addr ) [inline], [static]`

Read a `long long` from far byte-address `__addr`. The address is in the program memory.

**21.20.335** `pgm_read_short()` `short pgm_read_short (`  
`const short * __addr ) [inline], [static]`

Read a `short` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.336** `pgm_read_short_far()` `short pgm_read_short_far (`  
`uint_farptr_t __addr ) [inline], [static]`

Read a `short` from far byte-address `__addr`. The address is in the program memory.

**21.20.337** `pgm_read_signed()` `signed pgm_read_signed (`  
`const signed * __addr ) [inline], [static]`

Read a `signed` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.338** `pgm_read_signed_char()` `signed char pgm_read_signed_char (`  
`const signed char * __addr ) [inline], [static]`

Read a `signed char` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.339** `pgm_read_signed_char_far()` `signed char pgm_read_signed_char_far (`  
`uint_farptr_t __addr ) [inline], [static]`

Read a `signed char` from far byte-address `__addr`. The address is in the program memory.

**21.20.340** `pgm_read_signed_far()` `signed pgm_read_signed_far (`  
`uint_farptr_t __addr ) [inline], [static]`

Read a `signed` from far byte-address `__addr`. The address is in the program memory.

**21.20.341** `pgm_read_signed_int()` `signed int pgm_read_signed_int (`  
`const signed int * __addr ) [inline], [static]`

Read a `signed int` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.342** `pgm_read_signed_int_far()` `signed int pgm_read_signed_int_far (`  
`uint_farptr_t __addr ) [inline], [static]`

Read a `signed int` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.43** `pgm_read_u16()` `uint16_t` `pgm_read_u16` (  
    const `uint16_t` \* `__addr` ) [inline], [static]

Read an `uint16_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.44** `pgm_read_u16_far()` `uint16_t` `pgm_read_u16_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read an `uint16_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.45** `pgm_read_u24()` `__uint24` `pgm_read_u24` (  
    const `__uint24` \* `__addr` ) [inline], [static]

Read an `__uint24` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.46** `pgm_read_u24_far()` `__uint24` `pgm_read_u24_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read an `__uint24` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.47** `pgm_read_u32()` `uint32_t` `pgm_read_u32` (  
    const `uint32_t` \* `__addr` ) [inline], [static]

Read an `uint32_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.48** `pgm_read_u32_far()` `uint32_t` `pgm_read_u32_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read an `uint32_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.49** `pgm_read_u64()` `uint64_t` `pgm_read_u64` (  
    const `uint64_t` \* `__addr` ) [inline], [static]

Read an `uint64_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.50** `pgm_read_u64_far()` `uint64_t` `pgm_read_u64_far` (  
    `uint_farptr_t` `__addr` ) [inline], [static]

Read an `uint64_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.51** `pgm_read_u8()` `uint8_t` `pgm_read_u8` (  
    const `uint8_t` \* `__addr` ) [inline], [static]

Read an `uint8_t` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.52** `pgm_read_u8_far()` `uint8_t` `pgm_read_u8_far` (  
`uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `uint8_t` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.53** `pgm_read_unsigned()` `unsigned` `pgm_read_unsigned` (  
`const unsigned *` `__addr`) [`inline`], [`static`]

Read an `unsigned` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.54** `pgm_read_unsigned_char()` `unsigned char` `pgm_read_unsigned_char` (  
`const unsigned char *` `__addr`) [`inline`], [`static`]

Read an `unsigned char` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.55** `pgm_read_unsigned_char_far()` `unsigned char` `pgm_read_unsigned_char_far` (  
`uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `unsigned char` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.56** `pgm_read_unsigned_far()` `unsigned` `pgm_read_unsigned_far` (  
`uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `unsigned` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.57** `pgm_read_unsigned_int()` `unsigned int` `pgm_read_unsigned_int` (  
`const unsigned int *` `__addr`) [`inline`], [`static`]

Read an `unsigned int` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.58** `pgm_read_unsigned_int_far()` `unsigned int` `pgm_read_unsigned_int_far` (  
`uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `unsigned int` from far byte-address `__addr`. The address is in the program memory.

**21.20.3.59** `pgm_read_unsigned_long()` `unsigned long` `pgm_read_unsigned_long` (  
`const unsigned long *` `__addr`) [`inline`], [`static`]

Read an `unsigned long` from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.60** `pgm_read_unsigned_long_far()` `unsigned long` `pgm_read_unsigned_long_far` (  
`uint_farptr_t` `__addr`) [`inline`], [`static`]

Read an `unsigned long` from far byte-address `__addr`. The address is in the program memory.



**21.20.3.61** `pgm_read_unsigned_long_long()` `unsigned long long pgm_read_unsigned_long_long ( const unsigned long long * __addr ) [inline], [static]`

Read an unsigned long long from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.62** `pgm_read_unsigned_long_long_far()` `unsigned long long pgm_read_unsigned_long_long_far ( uint_farptr_t __addr ) [inline], [static]`

Read an unsigned long long from far byte-address `__addr`. The address is in the program memory.

**21.20.3.63** `pgm_read_unsigned_short()` `unsigned short pgm_read_unsigned_short ( const unsigned short * __addr ) [inline], [static]`

Read an unsigned short from 16-bit (near) byte-address `__addr`. The address is in the lower 64 KiB of program memory.

**21.20.3.64** `pgm_read_unsigned_short_far()` `unsigned short pgm_read_unsigned_short_far ( uint_farptr_t __addr ) [inline], [static]`

Read an unsigned short from far byte-address `__addr`. The address is in the program memory.

**21.20.3.65** `strcasecmp_P()` `int strcasecmp_P ( const char * s1, const char * s2 )`

Compare two strings ignoring case.

The `strcasecmp_P()` function compares the two strings `s1` and `s2`, ignoring the case of the characters.

#### Parameters

<code>s1</code>	A pointer to a string in the devices SRAM.
<code>s2</code>	A pointer to a string in the devices Flash.

#### Returns

The `strcasecmp_P()` function returns an integer less than, equal to, or greater than zero if `s1` is found, respectively, to be less than, to match, or be greater than `s2`. A consequence of the ordering used by `strcasecmp_P()` is that if `s1` is an initial substring of `s2`, then `s1` is considered to be "less than" `s2`.

**21.20.3.66** `strcasecmp_PF()` `int strcasecmp_PF ( const char * s1, uint_farptr_t s2 )`

Compare two strings ignoring case.

The `strcasecmp_PF()` function compares the two strings `s1` and `s2`, ignoring the case of the characters.

## Parameters

<i>s1</i>	A pointer to the first string in SRAM
<i>s2</i>	A far pointer to the second string in Flash

## Returns

The [strcasecmp\\_PF\(\)](#) function returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.67 strcasestr_P() char * strcasestr_P (
    const char * s1,
    const char * s2 )
```

This function is similar to [strcasestr\(\)](#) except that *s2* is pointer to a string in program space.

```
21.20.3.68 strcat_P() char * strcat_P (
    char * dest,
    const char * src )
```

The [strcat\\_P\(\)](#) function is similar to [strcat\(\)](#) except that the *src* string must be located in program space (flash).

## Returns

The [strcat\(\)](#) function returns a pointer to the resulting string *dest*.

```
21.20.3.69 strcat_PF() char * strcat_PF (
    char * dst,
    uint_farptr_t src )
```

Concatenates two strings.

The [strcat\\_PF\(\)](#) function is similar to [strcat\(\)](#) except that the *src* string must be located in program space (flash) and is addressed using a far pointer

## Parameters

<i>dst</i>	A pointer to the destination string in SRAM
<i>src</i>	A far pointer to the string to be appended in Flash

## Returns

The [strcat\\_PF\(\)](#) function returns a pointer to the resulting string *dst*. The contents of RAMPZ SFR are undefined when the function returns

```
21.20.3.70 strchr_P() const char * strchr_P (  
    const char * s,  
    int val )
```

Locate character in program space string.

The [strchr\\_P\(\)](#) function locates the first occurrence of `val` (converted to a char) in the string pointed to by `s` in program space. The terminating null character is considered to be part of the string.

The [strchr\\_P\(\)](#) function is similar to [strchr\(\)](#) except that `s` is pointer to a string in program space.

#### Returns

The [strchr\\_P\(\)](#) function returns a pointer to the matched character or `NULL` if the character is not found.

```
21.20.3.71 strchr_PF() uint_farptr_t strchr_PF (  
    uint_farptr_t s,  
    int val )
```

Locate character in far program space string.

The [strchr\\_PF\(\)](#) function locates the first occurrence of `val` (converted to a char) in the string pointed to by `s` in far program space. The terminating null character is considered to be part of the string.

The [strchr\\_PF\(\)](#) function is similar to [strchr\(\)](#) except that `s` is a far pointer to a string in program space that's *not required* to be located in the lower 64 KiB block like it is the case for [strchr\\_P\(\)](#).

#### Returns

The [strchr\\_PF\(\)](#) function returns a far pointer to the matched character or 0 if the character is not found.

```
21.20.3.72 strchrnul_P() const char * strchrnul_P (  
    const char * s,  
    int c )
```

The [strchrnul\\_P\(\)](#) function is like [strchr\\_P\(\)](#) except that if `c` is not found in `s`, then it returns a pointer to the null byte at the end of `s`, rather than `NULL`. (Glibc, GNU extension.)

#### Returns

The [strchrnul\\_P\(\)](#) function returns a pointer to the matched character, or a pointer to the null byte at the end of `s` (i.e., `s+strlen(s)`) if the character is not found.

```
21.20.3.73 strcmp_P() int strcmp_P (
    const char * s1,
    const char * s2 )
```

The `strcmp_P()` function is similar to `strcmp()` except that `s2` is pointer to a string in program space.

#### Returns

The `strcmp_P()` function returns an integer less than, equal to, or greater than zero if `s1` is found, respectively, to be less than, to match, or be greater than `s2`. A consequence of the ordering used by `strcmp_P()` is that if `s1` is an initial substring of `s2`, then `s1` is considered to be "less than" `s2`.

```
21.20.3.74 strcmp_PF() int strcmp_PF (
    const char * s1,
    uint_farptr_t s2 )
```

Compares two strings.

The `strcmp_PF()` function is similar to `strcmp()` except that `s2` is a far pointer to a string in program space.

#### Parameters

<code>s1</code>	A pointer to the first string in SRAM
<code>s2</code>	A far pointer to the second string in Flash

#### Returns

The `strcmp_PF()` function returns an integer less than, equal to, or greater than zero if `s1` is found, respectively, to be less than, to match, or be greater than `s2`. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.75 strcpy_P() char * strcpy_P (
    char * dest,
    const char * src )
```

The `strcpy_P()` function is similar to `strcpy()` except that `src` is a pointer to a string in program space.

#### Returns

The `strcpy_P()` function returns a pointer to the destination string `dest`.

```
21.20.3.76 strcpy_PF() char * strcpy_PF (
    char * dst,
    uint_farptr_t src )
```

Duplicate a string.

The `strcpy_PF()` function is similar to `strcpy()` except that `src` is a far pointer to a string in program space.

## Parameters

<i>dst</i>	A pointer to the destination string in SRAM
<i>src</i>	A far pointer to the source string in Flash

## Returns

The [strcpy\\_PF\(\)](#) function returns a pointer to the destination string *dst*. The contents of RAMPZ SFR are undefined when the function returns.

**21.20.3.77 strcspn\_P()** `size_t strcspn_P (`  
`const char * s,`  
`const char * reject )`

The [strcspn\\_P\(\)](#) function calculates the length of the initial segment of *s* which consists entirely of characters not in *reject*. This function is similar to [strcspn\(\)](#) except that *reject* is a pointer to a string in program space.

## Returns

The [strcspn\\_P\(\)](#) function returns the number of characters in the initial segment of *s* which are not in the string *reject*. The terminating zero is not considered as a part of string.

**21.20.3.78 strlcat\_P()** `size_t strlcat_P (`  
`char * dst,`  
`const char * src,`  
`size_t siz )`

Concatenate two strings.

The [strlcat\\_P\(\)](#) function is similar to [strlcat\(\)](#), except that the *src* string must be located in program space (flash).

Appends *src* to string *dst* of size *siz* (unlike [strncat\(\)](#), *siz* is the full size of *dst*, not space left). At most *siz*-1 characters will be copied. Always NULL terminates (unless *siz* <= `strlen(dst)`).

## Returns

The [strlcat\\_P\(\)](#) function returns `strlen(src) + MIN(siz, strlen(initial dst))`. If `retval` >= *siz*, truncation occurred.

**21.20.3.79 strlcat\_PF()** `size_t strlcat_PF (`  
`char * dst,`  
`uint_farptr_t src,`  
`size_t n )`

Concatenate two strings.

The [strlcat\\_PF\(\)](#) function is similar to [strlcat\(\)](#), except that the *src* string must be located in program space (flash) and is addressed using a far pointer.

Appends *src* to string *dst* of size *n* (unlike [strncat\(\)](#), *n* is the full size of *dst*, not space left). At most *n*-1 characters will be copied. Always NULL terminates (unless *n* <= `strlen(dst)`).

## Parameters

<i>dst</i>	A pointer to the destination string in SRAM
<i>src</i>	A far pointer to the source string in Flash
<i>n</i>	The total number of bytes allocated to the destination string

## Returns

The `strlcat_PF()` function returns `strlen(src) + MIN(n, strlen(initial dst))`. If `retval >= n`, truncation occurred. The contents of RAMPZ SFR are undefined when the function returns.

**21.20.3.80 `strlcpy_P()`** `size_t strlcpy_P (`  
`char * dst,`  
`const char * src,`  
`size_t siz )`

Copy a string from progmem to RAM.

Copy `src` to string `dst` of size `siz`. At most `siz-1` characters will be copied. Always NULL terminates (unless `siz == 0`). The `strlcpy_P()` function is similar to `strlcpy()` except that the `src` is pointer to a string in memory space.

## Returns

The `strlcpy_P()` function returns `strlen(src)`. If `retval >= siz`, truncation occurred.

**21.20.3.81 `strlcpy_PPF()`** `size_t strlcpy_PPF (`  
`char * dst,`  
`uint_farptr_t src,`  
`size_t siz )`

Copy a string from progmem to RAM.

Copy `src` to string `dst` of size `siz`. At most `siz-1` characters will be copied. Always NULL terminates (unless `siz == 0`).

## Returns

The `strlcpy_PPF()` function returns `strlen(src)`. If `retval >= siz`, truncation occurred. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.82 strlen_P() size_t strlen_P (  
    const char * src ) [inline], [static]
```

The `strlen_P()` function is similar to `strlen()`, except that `src` is a pointer to a string in program space.

#### Returns

The `strlen_P()` function returns the number of characters in `src`.

#### Note

`strlen_P()` is implemented as an inline function in the `avr/pgmspace.h` header file, which will check if the length of the string is a constant and known at compile time. If it is not known at compile time, the macro will issue a call to `__strlen_P()` which will then calculate the length of the string as normal.

```
21.20.3.83 strlen_PF() size_t strlen_PF (  
    uint_farptr_t s )
```

Obtain the length of a string.

The `strlen_PF()` function is similar to `strlen()`, except that `s` is a far pointer to a string in program space.

#### Parameters

<code>s</code>	A far pointer to the string in flash
----------------	--------------------------------------

#### Returns

The `strlen_PF()` function returns the number of characters in `s`. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.84 strncasecmp_P() int strncasecmp_P (  
    const char * s1,  
    const char * s2,  
    size_t n )
```

Compare two strings ignoring case.

The `strncasecmp_P()` function is similar to `strcasecmp_P()`, except it only compares the first `n` characters of `s1`.

#### Parameters

<code>s1</code>	A pointer to a string in the devices SRAM.
<code>s2</code>	A pointer to a string in the devices Flash.
<code>n</code>	The maximum number of bytes to compare.

**Returns**

The `strncasecmp_P()` function returns an integer less than, equal to, or greater than zero if `s1` (or the first `n` bytes thereof) is found, respectively, to be less than, to match, or be greater than `s2`. A consequence of the ordering used by `strncasecmp_P()` is that if `s1` is an initial substring of `s2`, then `s1` is considered to be "less than" `s2`.

```
21.20.3.85 strncasecmp_PF() int strncasecmp_PF (
    const char * s1,
    uint_farptr_t s2,
    size_t n )
```

Compare two strings ignoring case.

The `strncasecmp_PF()` function is similar to `strcasecmp_PF()`, except it only compares the first `n` characters of `s1` and the string in flash is addressed using a far pointer.

**Parameters**

<code>s1</code>	A pointer to a string in SRAM
<code>s2</code>	A far pointer to a string in Flash
<code>n</code>	The maximum number of bytes to compare

**Returns**

The `strncasecmp_PF()` function returns an integer less than, equal to, or greater than zero if `s1` (or the first `n` bytes thereof) is found, respectively, to be less than, to match, or be greater than `s2`. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.86 strncat_P() char * strncat_P (
    char * dest,
    const char * src,
    size_t len )
```

Concatenate two strings.

The `strncat_P()` function is similar to `strncat()`, except that the `src` string must be located in program space (flash).

**Returns**

The `strncat_P()` function returns a pointer to the resulting string `dest`.

```
21.20.3.87 strncat_PF() char * strncat_PF (
    char * dst,
    uint_farptr_t src,
    size_t n )
```

Concatenate two strings.

The `strncat_PF()` function is similar to `strncat()`, except that the `src` string must be located in program space (flash) and is addressed using a far pointer.



## Parameters

<i>dst</i>	A pointer to the destination string in SRAM
<i>src</i>	A far pointer to the source string in Flash
<i>n</i>	The maximum number of bytes to append

## Returns

The [strncat\\_PF\(\)](#) function returns a pointer to the resulting string *dst*. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.88 strncmp_P() int strncmp_P (
    const char * s1,
    const char * s2,
    size_t n )
```

The [strncmp\\_P\(\)](#) function is similar to [strcmp\\_P\(\)](#) except it only compares the first (at most) *n* characters of *s1* and *s2*.

## Returns

The [strncmp\\_P\(\)](#) function returns an integer less than, equal to, or greater than zero if *s1* (or the first *n* bytes thereof) is found, respectively, to be less than, to match, or be greater than *s2*.

```
21.20.3.89 strncmp_PF() int strncmp_PF (
    const char * s1,
    uint_farptr_t s2,
    size_t n )
```

Compare two strings with limited length.

The [strncmp\\_PF\(\)](#) function is similar to [strcmp\\_PF\(\)](#) except it only compares the first (at most) *n* characters of *s1* and *s2*.

## Parameters

<i>s1</i>	A pointer to the first string in SRAM
<i>s2</i>	A far pointer to the second string in Flash
<i>n</i>	The maximum number of bytes to compare

## Returns

The [strncmp\\_PF\(\)](#) function returns an integer less than, equal to, or greater than zero if *s1* (or the first *n* bytes thereof) is found, respectively, to be less than, to match, or be greater than *s2*. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.90 strncpy_P() char * strncpy_P (
    char * dest,
    const char * src,
    size_t n )
```

The `strncpy_P()` function is similar to `strcpy_P()` except that not more than `n` bytes of `src` are copied. Thus, if there is no null byte among the first `n` bytes of `src`, the result will not be null-terminated.

In the case where the length of `src` is less than that of `n`, the remainder of `dest` will be padded with nulls.

#### Returns

The `strncpy_P()` function returns a pointer to the destination string `dest`.

```
21.20.3.91 strncpy_PF() char * strncpy_PF (
    char * dst,
    uint_farptr_t src,
    size_t n )
```

Duplicate a string until a limited length.

The `strncpy_PF()` function is similar to `strcpy_PF()` except that not more than `n` bytes of `src` are copied. Thus, if there is no null byte among the first `n` bytes of `src`, the result will not be null-terminated.

In the case where the length of `src` is less than that of `n`, the remainder of `dst` will be padded with nulls.

#### Parameters

<i>dst</i>	A pointer to the destination string in SRAM
<i>src</i>	A far pointer to the source string in Flash
<i>n</i>	The maximum number of bytes to copy

#### Returns

The `strncpy_PF()` function returns a pointer to the destination string `dst`. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.92 strlen_P() size_t strlen_P (
    const char * src,
    size_t len )
```

Determine the length of a fixed-size string.

The `strlen_P()` function is similar to `strlen()`, except that `src` is a pointer to a string in program space.

#### Returns

The `strlen_P` function returns `strlen_P(src)`, if that is less than `len`, or `len` if there is no `'\0'` character among the first `len` characters pointed to by `src`.

```
21.20.3.93 strlen_PF() size_t strlen_PF (
    uint_farptr_t s,
    size_t len )
```

Determine the length of a fixed-size string.

The [strlen\\_PF\(\)](#) function is similar to [strlen\(\)](#), except that *s* is a far pointer to a string in program space.

#### Parameters

<i>s</i>	A far pointer to the string in Flash
<i>len</i>	The maximum number of length to return

#### Returns

The [strlen\\_PF](#) function returns [strlen\\_P\(s\)](#), if that is less than *len*, or *len* if there is no '\0' character among the first *len* characters pointed to by *s*. The contents of RAMPZ SFR are undefined when the function returns.

```
21.20.3.94 strpbrk_P() char * strpbrk_P (
    const char * s,
    const char * accept )
```

The [strpbrk\\_P\(\)](#) function locates the first occurrence in the string *s* of any of the characters in the flash string *accept*. This function is similar to [strpbrk\(\)](#) except that *accept* is a pointer to a string in program space.

#### Returns

The [strpbrk\\_P\(\)](#) function returns a pointer to the character in *s* that matches one of the characters in *accept*, or `NULL` if no such character is found. The terminating zero is not considered as a part of string: if one or both args are empty, the result will `NULL`.

```
21.20.3.95 strrchr_P() const char * strrchr_P (
    const char * s,
    int val )
```

Locate character in string.

The [strrchr\\_P\(\)](#) function returns a pointer to the last occurrence of the character *val* in the flash string *s*.

#### Returns

The [strrchr\\_P\(\)](#) function returns a pointer to the matched character or `NULL` if the character is not found.

```
21.20.3.96 strsep_P() char * strsep_P (  
    char ** sp,  
    const char * delim )
```

Parse a string into tokens.

The [strsep\\_P\(\)](#) function locates, in the string referenced by *\*sp*, the first occurrence of any character in the string *delim* (or the terminating '\0' character) and replaces it with a '\0'. The location of the next character after the delimiter character (or NULL, if the end of the string was reached) is stored in *\*sp*. An "empty" field, i.e. one caused by two adjacent delimiter characters, can be detected by comparing the location referenced by the pointer returned in *\*sp* to '\0'. This function is similar to [strsep\(\)](#) except that *delim* is a pointer to a string in program space.

#### Returns

The [strsep\\_P\(\)](#) function returns a pointer to the original value of *\*sp*. If *\*sp* is initially NULL, [strsep\\_P\(\)](#) returns NULL.

```
21.20.3.97 strspn_P() size_t strspn_P (  
    const char * s,  
    const char * accept )
```

The [strspn\\_P\(\)](#) function calculates the length of the initial segment of *s* which consists entirely of characters in *accept*. This function is similar to [strspn\(\)](#) except that *accept* is a pointer to a string in program space.

#### Returns

The [strspn\\_P\(\)](#) function returns the number of characters in the initial segment of *s* which consist only of characters from *accept*. The terminating zero is not considered as a part of string.

```
21.20.3.98 strstr_P() char * strstr_P (  
    const char * s1,  
    const char * s2 )
```

Locate a substring.

The [strstr\\_P\(\)](#) function finds the first occurrence of the substring *s2* in the string *s1*. The terminating '\0' characters are not compared. The [strstr\\_P\(\)](#) function is similar to [strstr\(\)](#) except that *s2* is pointer to a string in program space.

#### Returns

The [strstr\\_P\(\)](#) function returns a pointer to the beginning of the substring, or NULL if the substring is not found. If *s2* points to a string of zero length, the function returns *s1*.

**21.20.3.99 strstr\_PF()** `char * strstr_PF (`  
    `const char * s1,`  
    `uint_farptr_t s2 )`

Locate a substring.

The [strstr\\_PF\(\)](#) function finds the first occurrence of the substring `s2` in the string `s1`. The terminating `'\0'` characters are not compared. The [strstr\\_PF\(\)](#) function is similar to [strstr\(\)](#) except that `s2` is a far pointer to a string in program space.

#### Returns

The [strstr\\_PF\(\)](#) function returns a pointer to the beginning of the substring, or NULL if the substring is not found. If `s2` points to a string of zero length, the function returns `s1`. The contents of RAMPZ SFR are undefined when the function returns.

**21.20.3.100 strtok\_P()** `char * strtok_P (`  
    `char * s,`  
    `const char * delim )`

Parses the string into tokens.

[strtok\\_P\(\)](#) parses the string `s` into tokens. The first call to [strtok\\_P\(\)](#) should have `s` as its first argument. Subsequent calls should have the first argument set to NULL. If a token ends with a delimiter, this delimiting character is overwritten with a `'\0'` and a pointer to the next character is saved for the next call to [strtok\\_P\(\)](#). The delimiter string `delim` may be different for each call.

The [strtok\\_P\(\)](#) function is similar to [strtok\(\)](#) except that `delim` is pointer to a string in program space.

#### Returns

The [strtok\\_P\(\)](#) function returns a pointer to the next token or NULL when no more tokens are found.

#### Note

[strtok\\_P\(\)](#) is NOT reentrant. For a reentrant version of this function see [strtok\\_rP\(\)](#).

**21.20.3.101 strtok\_rP()** `char * strtok_rP (`  
    `char * string,`  
    `const char * delim,`  
    `char ** last )`

Parses string into tokens.

The [strtok\\_rP\(\)](#) function parses `string` into tokens. The first call to [strtok\\_rP\(\)](#) should have `string` as its first argument. Subsequent calls should have the first argument set to NULL. If a token ends with a delimiter, this delimiting character is overwritten with a `'\0'` and a pointer to the next character is saved for the next call to [strtok\\_rP\(\)](#). The delimiter string `delim` may be different for each call. `last` is a user allocated `char*` pointer. It must be the same while parsing the same string. [strtok\\_rP\(\)](#) is a reentrant version of [strtok\\_P\(\)](#).

The [strtok\\_rP\(\)](#) function is similar to [strtok\\_r\(\)](#) except that `delim` is pointer to a string in program space.

#### Returns

The [strtok\\_rP\(\)](#) function returns a pointer to the next token or NULL when no more tokens are found.

## 21.21 <avr/power.h>: Power Reduction Management

### Macros

- #define `clock_prescale_get()` (clock\_div\_t)(CLKPR & (uint8\_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))

### Functions

- static void `power_all_enable()`
- static void `power_all_disable()`
- void `clock_prescale_set` (clock\_div\_t \_\_x)

#### 21.21.1 Detailed Description

```
#include <avr/power.h>
```

Many AVR devices contain a Power Reduction Register (PRR) or Registers (PRRx) that allow you to reduce power consumption by disabling or enabling various on-board peripherals as needed. Some devices have the XTAL Divide Control Register (XDIV) which offer similar functionality as System Clock Prescale Register (CLKPR).

There are many macros in this header file that provide an easy interface to enable or disable on-board peripherals to reduce power. See the table below.

#### Note

Not all AVR devices have a Power Reduction Register (for example the ATmega8). On those devices without a Power Reduction Register, the power reduction macros are not available..

Not all AVR devices contain the same peripherals (for example, the LCD interface), or they will be named differently (for example, USART and USART0). Please consult your device's datasheet, or the header file, to find out which macros are applicable to your device.

For device using the XTAL Divide Control Register (XDIV), when prescaler is used, Timer/Counter0 can only be used in asynchronous mode. Keep in mind that Timer/Counter0 source shall be less than 1/4th of peripheral clock. Therefore, when using a typical 32.768 kHz crystal, one shall not scale the clock below 131.072 kHz.

**Table 39 Power Macros**

Power Macro	Description
<code>power_aca_disable()</code>	Disable the Analog Comparator on PortA
<code>power_aca_enable()</code>	Enable the Analog Comparator on PortA
<code>power_adc_enable()</code>	Enable the Analog to Digital Converter module
<code>power_adc_disable()</code>	Disable the Analog to Digital Converter module
<code>power_adca_disable()</code>	Disable the Analog to Digital Converter module on PortA
<code>power_adca_enable()</code>	Enable the Analog to Digital Converter module on PortA
<code>power_evsys_disable()</code>	Disable the EVSYS module
<code>power_evsys_enable()</code>	Enable the EVSYS module
<code>power_hiresc_disable()</code>	Disable the HIRES module on PortC
<code>power_hiresc_enable()</code>	Enable the HIRES module on PortC
<code>power_lcd_enable()</code>	Enable the LCD module
<code>power_lcd_disable()</code>	Disable the LCD module
<code>power_pga_enable()</code>	Enable the Programmable Gain Amplifier module
<code>power_pga_disable()</code>	Disable the Programmable Gain Amplifier module
<code>power_pscr_enable()</code>	Enable the Reduced Power Stage Controller module
<code>power_pscr_disable()</code>	Disable the Reduced Power Stage Controller module
<code>power_psc0_enable()</code>	Enable the Power Stage Controller 0 module
<code>power_psc0_disable()</code>	Disable the Power Stage Controller 0 module

Power Macro	Description
power_psc1_enable()	Enable the Power Stage Controller 1 module
power_psc1_disable()	Disable the Power Stage Controller 1 module
power_psc2_enable()	Enable the Power Stage Controller 2 module
power_psc2_disable()	Disable the Power Stage Controller 2 module
power_ram0_enable()	Enable the SRAM block 0
power_ram0_disable()	Disable the SRAM block 0
power_ram1_enable()	Enable the SRAM block 1
power_ram1_disable()	Disable the SRAM block 1
power_ram2_enable()	Enable the SRAM block 2
power_ram2_disable()	Disable the SRAM block 2
power_ram3_enable()	Enable the SRAM block 3
power_ram3_disable()	Disable the SRAM block 3
power_rtc_disable()	Disable the RTC module
power_rtc_enable()	Enable the RTC module
power_spi_enable()	Enable the Serial Peripheral Interface module
power_spi_disable()	Disable the Serial Peripheral Interface module
power_spic_disable()	Disable the SPI module on PortC
power_spic_enable()	Enable the SPI module on PortC
power_spid_disable()	Disable the SPI module on PortD
power_spid_enable()	Enable the SPI module on PortD
power_tc0c_disable()	Disable the TC0 module on PortC
power_tc0c_enable()	Enable the TC0 module on PortC
power_tc0d_disable()	Disable the TC0 module on PortD
power_tc0d_enable()	Enable the TC0 module on PortD
power_tc0e_disable()	Disable the TC0 module on PortE
power_tc0e_enable()	Enable the TC0 module on PortE
power_tc0f_disable()	Disable the TC0 module on PortF
power_tc0f_enable()	Enable the TC0 module on PortF
power_tclc_disable()	Disable the TC1 module on PortC
power_tclc_enable()	Enable the TC1 module on PortC
power_twic_disable()	Disable the Two Wire Interface module on PortC
power_twic_enable()	Enable the Two Wire Interface module on PortC
power_twie_disable()	Disable the Two Wire Interface module on PortE
power_twie_enable()	Enable the Two Wire Interface module on PortE
power_timer0_enable()	Enable the Timer 0 module
power_timer0_disable()	Disable the Timer 0 module
power_timer1_enable()	Enable the Timer 1 module
power_timer1_disable()	Disable the Timer 1 module
power_timer2_enable()	Enable the Timer 2 module
power_timer2_disable()	Disable the Timer 2 module
power_timer3_enable()	Enable the Timer 3 module
power_timer3_disable()	Disable the Timer 3 module
power_timer4_enable()	Enable the Timer 4 module
power_timer4_disable()	Disable the Timer 4 module
power_timer5_enable()	Enable the Timer 5 module
power_timer5_disable()	Disable the Timer 5 module
power_twi_enable()	Enable the Two Wire Interface module
power_twi_disable()	Disable the Two Wire Interface module
power_usart_enable()	Enable the USART module
power_usart_disable()	Disable the USART module
power_usart0_enable()	Enable the USART 0 module
power_usart0_disable()	Disable the USART 0 module
power_usart1_enable()	Enable the USART 1 module
power_usart1_disable()	Disable the USART 1 module
power_usart2_enable()	Enable the USART 2 module
power_usart2_disable()	Disable the USART 2 module

Power Macro	Description
<code>power_usart3_enable()</code>	Enable the USART 3 module
<code>power_usart3_disable()</code>	Disable the USART 3 module
<code>power_usartc0_disable()</code>	Disable the USART0 module on PortC
<code>power_usartc0_enable()</code>	Enable the USART0 module on PortC
<code>power_usartd0_disable()</code>	Disable the USART0 module on PortD
<code>power_usartd0_enable()</code>	Enable the USART0 module on PortD
<code>power_usarte0_disable()</code>	Disable the USART0 module on PortE
<code>power_usarte0_enable()</code>	Enable the USART0 module on PortE
<code>power_usartf0_disable()</code>	Disable the USART0 module on PortF
<code>power_usartf0_enable()</code>	Enable the USART0 module on PortF
<code>power_usb_enable()</code>	Enable the USB module
<code>power_usb_disable()</code>	Disable the USB module
<code>power_usi_enable()</code>	Enable the Universal Serial Interface module
<code>power_usi_disable()</code>	Disable the Universal Serial Interface module
<code>power_vadc_enable()</code>	Enable the Voltage ADC module
<code>power_vadc_disable()</code>	Disable the Voltage ADC module
<code>power_all_enable()</code>	Enable all modules
<code>power_all_disable()</code>	Disable all modules

Some of the newer AVR's contain a System Clock Prescale Register (CLKPR) that allows you to decrease the system clock frequency and the power consumption when the need for processing power is low. On some earlier AVR's (ATmega103, ATmega64, ATmega128), similar functionality can be achieved through the XTAL Divide Control Register. Below are two macros and an enumerated type that can be used to interface to the Clock Prescale Register or XTAL Divide Control Register.

#### Note

Not all AVR devices have a clock prescaler. On those devices without a Clock Prescale Register or XTAL Divide Control Register, these macros are not available.

```
typedef enum
{
    clock_div_1 = 0,
    clock_div_2 = 1,
    clock_div_4 = 2,
    clock_div_8 = 3,
    clock_div_16 = 4,
    clock_div_32 = 5,
    clock_div_64 = 6,
    clock_div_128 = 7,
    clock_div_256 = 8,
    clock_div_1_rc = 15, // ATmega128RFA1 only
} clock_div_t;
```

Clock prescaler setting enumerations for device using System Clock Prescale Register.

```
typedef enum
{
    clock_div_1 = 1,
    clock_div_2 = 2,
    clock_div_4 = 4,
    clock_div_8 = 8,
    clock_div_16 = 16,
    clock_div_32 = 32,
    clock_div_64 = 64,
    clock_div_128 = 128
} clock_div_t;
```

Clock prescaler setting enumerations for device using XTAL Divide Control Register.

### 21.21.2 Macro Definition Documentation



**21.21.2.1 clock\_prescale\_get** `#define clock_prescale_get ( ) (clock_div_t) (CLKPR & (uint8_t) ((1<<CLKPS0)|(1<<CL`

Gets and returns the clock prescaler register setting. The return type is `clock_div_t`.

**Note**

For device with XTAL Divide Control Register (XDIV), return can actually range from 1 to 129. Care should be taken has the return value could differ from the typedef enum `clock_div_t`. This should only happen if `clock_prescale_set` was previously called with a value other than those defined by `clock_div_t`.

### 21.21.3 Function Documentation

**21.21.3.1 clock\_prescale\_set()** `clock_prescale_set (`  
`clock_div_t x )`

Set the clock prescaler register select bits, selecting a system clock division setting. This function is inlined, even if compiler optimizations are disabled.

The type of `x` is `clock_div_t`.

**Note**

For device with XTAL Divide Control Register (XDIV), `x` can actually range from 1 to 129. Thus, one does not need to use `clock_div_t` type as argument.

**21.21.3.2 power\_all\_disable()** `void power_all_disable ( ) [inline], [static]`

Disable all modules.

**21.21.3.3 power\_all\_enable()** `void power_all_enable ( ) [inline], [static]`

Enable all modules.

## 21.22 Additional notes from <avr/sfr\_defs.h>

The <avr/sfr\_defs.h> file is included by all of the <avr/ioXXXX.h> files, which use macros defined here to make the special function register definitions look like C variables or simple constants, depending on the `__SFR_ASM_COMPAT` define. Some examples from <avr/iocanxx.h> to show how to define such macros:

```
#define PORTA    _SFR_IO8(0x02)
#define EEAR     _SFR_IO16(0x21)
#define UDR0     _SFR_MEM8(0xC6)
#define TCNT3    _SFR_MEM16(0x94)
#define CANIDT   _SFR_MEM32(0xF0)
```

If `__SFR_ASM_COMPAT` is not defined, C programs can use names like `PORTA` directly in C expressions (also on the left side of assignment operators) and GCC will do the right thing (use short I/O instructions if possible). The `__SFR_OFFSET` definition is not used in any way in this case.

Define `__SFR_ASM_COMPAT` as 1 to make these names work as simple constants (addresses of the I/O registers). This is necessary when included in preprocessed assembler (\*.S) source files, so it is done automatically if `__ASSEMBLER__` is defined. By default, all addresses are defined as if they were memory addresses (used in `lds/sts` instructions). To use these addresses in `in/out` instructions, you must subtract 0x20 from them.

For more backwards compatibility, insert the following at the start of your old assembler source file:

```
#define __SFR_OFFSET 0
```

This automatically subtracts 0x20 from I/O space addresses, but it's a hack, so it is recommended to change your source: wrap such addresses in macros defined here, as shown below. After this is done, the `__SFR_OFFSET` definition is no longer necessary and can be removed.

Real example - this code could be used in a boot loader that is portable between devices with `SPMCR` at different addresses.

```
<avr/iom163.h>: #define SPMCR _SFR_IO8(0x37)
<avr/iom128.h>: #define SPMCR _SFR_MEM8(0x68)

#if __SFR_IO_REG_P(SPMCR)
    out _SFR_IO_ADDR(SPMCR), r24
#else
    sts _SFR_MEM_ADDR(SPMCR), r24
#endif
```

You can use the `in/out/cbi/sbi/sbic/sbis` instructions, without the `__SFR_IO_REG_P` test, if you know that the register is in the I/O space (as with `SREG`, for example). If it isn't, the assembler will complain (I/O address out of range 0...0x3f), so this should be fairly safe.

If you do not define `__SFR_OFFSET` (so it will be 0x20 by default), all special register addresses are defined as memory addresses (so `SREG` is 0x5f), and (if code size and speed are not important, and you don't like the ugly `#if` above) you can always use `lds/sts` to access them. But, this will not work if `__SFR_OFFSET != 0x20`, so use a different macro (defined only if `__SFR_OFFSET == 0x20`) for safety:

```
sts _SFR_ADDR(SPMCR), r24
```

In C programs, all 3 combinations of `__SFR_ASM_COMPAT` and `__SFR_OFFSET` are supported - the `__SFR_↔ADDR(SPMCR)` macro can be used to get the address of the `SPMCR` register (0x57 or 0x68 depending on device).

## 21.23 <avr/sfr\_defs.h>: Special function registers

### Modules

- [Additional notes from <avr/sfr\\_defs.h>](#)

## Bit manipulation

- `#define _BV(bit) (1 << (bit))`

## IO register bit manipulation

- `#define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))`
- `#define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))`
- `#define loop_until_bit_is_set(sfr, bit) do { } while (bit_is_clear(sfr, bit))`
- `#define loop_until_bit_is_clear(sfr, bit) do { } while (bit_is_set(sfr, bit))`

### 21.23.1 Detailed Description

When working with microcontrollers, many tasks usually consist of controlling internal peripherals, or external peripherals that are connected to the device. The entire IO address space is made available as *memory-mapped IO*, i.e. it can be accessed using all the MCU instructions that are applicable to normal data memory. For most AVR devices, the IO register space is mapped into the data memory address space with an offset of 0x20 since the bottom of this space is reserved for direct access to the MCU registers. (Actual SRAM is available only behind the IO register area, starting at some specific address depending on the device.)

For example the user can access memory-mapped IO registers as if they were globally defined variables like this:

```
PORTA = 0x33;
unsigned char foo = PINA;
```

The compiler will choose the correct instruction sequence to generate based on the address of the register being accessed.

The advantage of using the memory-mapped registers in C programs is that it makes the programs more portable to other C compilers for the AVR platform.

Note that special care must be taken when accessing some of the 16-bit timer IO registers where access from both the main program and within an interrupt context can happen. See [Why do some 16-bit timer registers sometimes get trashed?](#).

#### Porting programs that use the deprecated sbi/cbi macros

Access to the AVR single bit set and clear instructions are provided via the standard C bit manipulation commands. The sbi and cbi macros are no longer directly supported. sbi (sfr,bit) can be replaced by `sfr |= _BV(bit)`.

i.e.: `sbi(PORTB, PB1);` is now `PORTB |= _BV(PB1);`

This actually is more flexible than having sbi directly, as the optimizer will use a hardware sbi if appropriate, or a read/or/write operation if not appropriate. You do not need to keep track of which registers sbi/cbi will operate on.

Likewise, cbi (sfr,bit) is now `sfr &= ~(_BV(bit));`

### 21.23.2 Macro Definition Documentation

```
21.23.2.1 _BV #define _BV(  
    bit ) (1 << (bit))  
#include <avr/io.h>
```

Converts a bit number into a byte value.

#### Note

The bit shift is performed by the compiler which then inserts the result into the code. Thus, there is no run-time overhead when using `_BV()`.

```
21.23.2.2 bit_is_clear #define bit_is_clear(  
    sfr,  
    bit ) (!(_SFR_BYTE(sfr) & _BV(bit)))  
#include <avr/io.h>
```

Test whether bit `bit` in IO register `sfr` is clear. This will return non-zero if the bit is clear, and a 0 if the bit is set.

```
21.23.2.3 bit_is_set #define bit_is_set(  
    sfr,  
    bit ) (_SFR_BYTE(sfr) & _BV(bit))  
#include <avr/io.h>
```

Test whether bit `bit` in IO register `sfr` is set. This will return a 0 if the bit is clear, and non-zero if the bit is set.

```
21.23.2.4 loop_until_bit_is_clear #define loop_until_bit_is_clear(  
    sfr,  
    bit ) do { } while (bit_is_set(sfr, bit))  
#include <avr/io.h>
```

Wait until bit `bit` in IO register `sfr` is clear.

```
21.23.2.5 loop_until_bit_is_set #define loop_until_bit_is_set(  
    sfr,  
    bit ) do { } while (bit_is_clear(sfr, bit))  
#include <avr/io.h>
```

Wait until bit `bit` in IO register `sfr` is set.

## 21.24 <avr/signature.h>: Signature Support

### Introduction

The `<avr/signature.h>` header file allows the user to automatically and easily include the device's signature data in a special section of the final linked ELF file.

This value can then be used by programming software to compare the on-device signature with the signature recorded in the ELF file to look for a match before programming the device.

## API Usage Example

Usage is very simple; just include the header file:

```
#include <avr/signature.h>
```

This will declare a constant unsigned char array and it is initialized with the three signature bytes, MSB first, that are defined in the device I/O header file. This array is then placed in the .signature section in the resulting linked ELF file.

The three signature bytes that are used to initialize the array are these defined macros in the device I/O header file, from MSB to LSB: SIGNATURE\_2, SIGNATURE\_1, SIGNATURE\_0.

This header file should only be included once in an application.

## 21.25 <avr/sleep.h>: Power Management and Sleep Modes

### Functions

- void [sleep\\_enable](#) (void)
- void [sleep\\_disable](#) (void)
- void [sleep\\_cpu](#) (void)
- void [sleep\\_mode](#) (void)
- void [sleep\\_bod\\_disable](#) (void)

### 21.25.1 Detailed Description

### 21.25.2 Function Documentation

**21.25.2.1 [sleep\\_bod\\_disable\(\)](#)** void [sleep\\_bod\\_disable](#) (  
void )

Disable BOD before going to sleep. Not available on all devices.

**21.25.2.2 [sleep\\_cpu\(\)](#)** void [sleep\\_cpu](#) (  
void )

Put the device into sleep mode. The SE bit must be set beforehand, and it is recommended to clear it afterwards.

**21.25.2.3 [sleep\\_disable\(\)](#)** void [sleep\\_disable](#) (  
void )

Clear the SE (sleep enable) bit.

```

21.25.2.4 sleep_enable() void sleep_enable (
    void )
#include <avr/sleep.h>

```

Use of the `SLEEP` instruction can allow an application to reduce its power consumption considerably. AVR devices can be put into different sleep modes. Refer to the datasheet for the details relating to the device you are using.

There are several macros provided in this header file to actually put the device into sleep mode. The simplest way is to optionally set the desired sleep mode using `set_sleep_mode()` (it usually defaults to idle mode where the CPU is put on sleep but all peripheral clocks are still running), and then call `sleep_mode()`. This macro automatically sets the sleep enable bit, goes to sleep, and clears the sleep enable bit.

Example:

```

#include <avr/sleep.h>

...
set_sleep_mode(<mode>);
sleep_mode();

```

Note that unless your purpose is to completely lock the CPU (until a hardware reset), interrupts need to be enabled before going to sleep.

As the `sleep_mode()` macro might cause race conditions in some situations, the individual steps of manipulating the sleep enable (SE) bit, and actually issuing the `SLEEP` instruction, are provided in the macros `sleep_enable()`, `sleep_disable()`, and `sleep_cpu()`. This also allows for test-and-sleep scenarios that take care of not missing the interrupt that will awake the device from sleep.

Example:

```

#include <avr/interrupt.h>
#include <avr/sleep.h>

...
set_sleep_mode(<mode>);
cli();
if (some_condition)
{
    sleep_enable();
    sei();
    sleep_cpu();
    sleep_disable();
}
sei();

```

This sequence ensures an atomic test of `some_condition` with interrupts being disabled. If the condition is met, sleep mode will be prepared, and the `SLEEP` instruction will be scheduled immediately after an `SEI` instruction. As the instruction right after the `SEI` is guaranteed to be executed before an interrupt could trigger, it is sure the device will really be put to sleep.

Some devices have the ability to disable the Brown Out Detector (BOD) before going to sleep. This will also reduce power while sleeping. If the specific AVR device has this ability then an additional macro is defined: `sleep_bod_disable()`. This macro generates inlined assembly code that will correctly implement the timed sequence for disabling the BOD before sleeping. However, there is a limited number of cycles after the BOD has been disabled that the device can be put into sleep mode, otherwise the BOD will not truly be disabled. Recommended practice is to disable the BOD (`sleep_bod_disable()`), set the interrupts (`sei()`), and then put the device to sleep (`sleep_cpu()`), like so:

```

#include <avr/interrupt.h>
#include <avr/sleep.h>

...
set_sleep_mode(<mode>);
cli();
if (some_condition)
{
    sleep_enable();
    sleep_bod_disable();
    sei();
    sleep_cpu();
    sleep_disable();
}
sei();

```

Put the device in sleep mode. How the device is brought out of sleep mode depends on the specific mode selected with the `set_sleep_mode()` function. See the data sheet for your device for more details.

Set the SE (sleep enable) bit.

**21.25.2.5 sleep\_mode()** `void sleep_mode (`  
`void )`

Put the device into sleep mode, taking care of setting the SE bit before, and clearing it afterwards.

## 21.26 <avr/version.h>: avr-libc version macros

### Macros

- `#define __AVR_LIBC_VERSION_STRING__ "2.2.0"`
- `#define __AVR_LIBC_VERSION__ 20200UL`
- `#define __AVR_LIBC_DATE_STRING__ "20240608"`
- `#define __AVR_LIBC_DATE__ 20240608UL`
- `#define __AVR_LIBC_MAJOR__ 2`
- `#define __AVR_LIBC_MINOR__ 2`
- `#define __AVR_LIBC_REVISION__ 0`

### 21.26.1 Detailed Description

```
#include <avr/version.h>
```

This header file defines macros that contain version numbers and strings describing the current version of avr-libc.

The version number itself basically consists of three pieces that are separated by a dot: the major number, the minor number, and the revision number. For development versions (which use an odd minor number), the string representation additionally gets the date code (YYYYMMDD) appended.

This file will also be included by <avr/io.h>. That way, portable tests can be implemented using <avr/io.h> that can be used in code that wants to remain backwards-compatible to library versions prior to the date when the library version API had been added, as referenced but undefined C preprocessor macros automatically evaluate to 0.

### 21.26.2 Macro Definition Documentation

**21.26.2.1 \_\_AVR\_LIBC\_DATE\_\_** `#define __AVR_LIBC_DATE__ 20240608UL`

Numerical representation of the release date.

**21.26.2.2 \_\_AVR\_LIBC\_DATE\_STRING\_\_** `#define __AVR_LIBC_DATE_STRING__ "20240608"`

String literal representation of the release date.

**21.26.2.3 \_\_AVR\_LIBC\_MAJOR\_\_** `#define __AVR_LIBC_MAJOR__ 2`

Library major version number.

**21.26.2.4** `__AVR_LIBC_MINOR__` `#define __AVR_LIBC_MINOR__ 2`

Library minor version number.

**21.26.2.5** `__AVR_LIBC_REVISION__` `#define __AVR_LIBC_REVISION__ 0`

Library revision number.

**21.26.2.6** `__AVR_LIBC_VERSION__` `#define __AVR_LIBC_VERSION__ 20200UL`

Numerical representation of the current library version.

In the numerical representation, the major number is multiplied by 10000, the minor number by 100, and all three parts are then added. It is intended to provide a monotonically increasing numerical value that can easily be used in numerical checks.

**21.26.2.7** `__AVR_LIBC_VERSION_STRING__` `#define __AVR_LIBC_VERSION_STRING__ "2.2.0"`

String literal representation of the current library version.

## 21.27 <avr/builtins.h>: avr-gcc builtins documentation

### Functions

- void `__builtin_avr_sei` (void)
- void `__builtin_avr_cli` (void)
- void `__builtin_avr_sleep` (void)
- void `__builtin_avr_wdr` (void)
- `uint8_t` `__builtin_avr_swap` (`uint8_t` \_\_b)
- `uint16_t` `__builtin_avr_fmul` (`uint8_t` \_\_a, `uint8_t` \_\_b)
- `int16_t` `__builtin_avr_fmuls` (`int8_t` \_\_a, `int8_t` \_\_b)
- `int16_t` `__builtin_avr_fmulsu` (`int8_t` \_\_a, `uint8_t` \_\_b)

### 21.27.1 Detailed Description

```
#include <avr/builtins.h>
```

#### Note

This file only documents some avr-gcc builtins. For functions built-in in the compiler, there should be no prototype declarations.

See also the [GCC documentation](#) for a full list of avr-gcc builtins.

### 21.27.2 Function Documentation



**21.27.2.1** `__builtin_avr_cli()` `void __builtin_avr_cli (`  
`void )`

Disables all interrupts by clearing the global interrupt mask.

**21.27.2.2** `__builtin_avr_fmulsu()` `uint16_t __builtin_avr_fmulsu (`  
`uint8_t __a,`  
`uint8_t __b )`

Emits an FMUL (fractional multiply unsigned) instruction.

**21.27.2.3** `__builtin_avr_fmuls()` `int16_t __builtin_avr_fmuls (`  
`int8_t __a,`  
`int8_t __b )`

Emits an FMUL (fractional multiply signed) instruction.

**21.27.2.4** `__builtin_avr_fmulsu()` `int16_t __builtin_avr_fmulsu (`  
`int8_t __a,`  
`uint8_t __b )`

Emits an FMUL (fractional multiply signed with unsigned) instruction.

**21.27.2.5** `__builtin_avr_sei()` `void __builtin_avr_sei (`  
`void )`

Enables interrupts by setting the global interrupt mask.

**21.27.2.6** `__builtin_avr_sleep()` `void __builtin_avr_sleep (`  
`void )`

Emits a SLEEP instruction.

**21.27.2.7** `__builtin_avr_swap()` `uint8_t __builtin_avr_swap (`  
`uint8_t __b )`

Emits a SWAP (nibble swap) instruction on `__b`.

**21.27.2.8** `__builtin_avr_wdr()` `void __builtin_avr_wdr (`  
`void )`

Emits a WDR (watchdog reset) instruction.

## 21.28 <avr/wdt.h>: Watchdog timer handling

### Macros

- #define `wdt_reset()` `__asm__ __volatile__ ("wdr")`
- #define `wdt_enable(timeout)`
- #define `WDTO_15MS` 0
- #define `WDTO_30MS` 1
- #define `WDTO_60MS` 2
- #define `WDTO_120MS` 3
- #define `WDTO_250MS` 4
- #define `WDTO_500MS` 5
- #define `WDTO_1S` 6
- #define `WDTO_2S` 7
- #define `WDTO_4S` 8
- #define `WDTO_8S` 9

### 21.28.1 Detailed Description

```
#include <avr/wdt.h>
```

This header file declares the interface to some inline macros handling the watchdog timer present in many AVR devices. In order to prevent the watchdog timer configuration from being accidentally altered by a crashing application, a special timed sequence is required in order to change it. The macros within this header file handle the required sequence automatically before changing any value. Interrupts will be disabled during the manipulation.

#### Note

Depending on the fuse configuration of the particular device, further restrictions might apply, in particular it might be disallowed to turn off the watchdog timer.

Note that for newer devices (ATmega88 and newer, effectively any AVR that has the option to also generate interrupts), the watchdog timer remains active even after a system reset (except a power-on condition), using the fastest prescaler value (approximately 15 ms). It is therefore required to turn off the watchdog early during program startup, the datasheet recommends a sequence like the following:

```
#include <stdint.h>
#include <avr/wdt.h>

uint8_t mcusr_mirror __attribute__((section(".noinit")));

__attribute__((used, unused, naked, section(".init3")))
static void get_mcusr (void);

void get_mcusr (void)
{
    mcusr_mirror = MCUSR;
    MCUSR = 0;
    wdt_disable();
}
```

Saving the value of MCUSR in `mcusr_mirror` is only needed if the application later wants to examine the reset source, but in particular, clearing the watchdog reset flag before disabling the watchdog is required, according to the datasheet.

### 21.28.2 Macro Definition Documentation

**21.28.2.1 wdt\_enable** `#define wdt_enable(  
 timeout )`

Enable the watchdog timer, configuring it for expiry after `timeout` (which is a combination of the `WDP0` through `WDP2` bits to write into the `WDTCSR` register; For those devices that have a `WDTCSR` register, it uses the combination of the `WDP0` through `WDP3` bits).

See also the symbolic constants `WDTO_15MS` et al.

**21.28.2.2 wdt\_reset** `#define wdt_reset( ) __asm__ __volatile__ ("wdr")`

Reset the watchdog timer. When the watchdog timer is enabled, a call to this instruction is required before the timer expires, otherwise a watchdog-initiated device reset will occur.

**21.28.2.3 WDTO\_120MS** `#define WDTO_120MS 3`

See `WDTO_15MS`

**21.28.2.4 WDTO\_15MS** `#define WDTO_15MS 0`

Symbolic constants for the watchdog timeout. Since the watchdog timer is based on a free-running RC oscillator, the times are approximate only and apply to a supply voltage of 5 V. At lower supply voltages, the times will increase. For older devices, the times will be as large as three times when operating at  $V_{cc} = 3$  V, while the newer devices (e. g. ATmega128, ATmega8) only experience a negligible change.

Possible timeout values are: 15 ms, 30 ms, 60 ms, 120 ms, 250 ms, 500 ms, 1 s, 2 s. (Some devices also allow for 4 s and 8 s.) Symbolic constants are formed by the prefix `WDTO_`, followed by the time.

Example that would select a watchdog timer expiry of approximately 500 ms:  
`wdt_enable(WDTO_500MS);`

**21.28.2.5 WDTO\_1S** `#define WDTO_1S 6`

See `WDTO_15MS`

**21.28.2.6 WDTO\_250MS** `#define WDTO_250MS 4`

See `WDTO_15MS`

**21.28.2.7 WDTO\_2S** `#define WDTO_2S 7`

See `WDTO_15MS`

**21.28.2.8 WDTO\_30MS** `#define WDTO_30MS 1`

See `WDTO_15MS`

**21.28.2.9 WDTO\_4S** `#define WDTO_4S 8`

See `WDTO_15MS` Note: This is only available on the ATtiny2313, ATtiny24, ATtiny44, ATtiny84, ATtiny84A, ATtiny25, ATtiny45, ATtiny85, ATtiny261, ATtiny461, ATtiny861, ATmega48\*, ATmega88\*, ATmega168\*, ATmega328\*, ATmega164P, ATmega324P, ATmega324PB, ATmega644P, ATmega644, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega8HVA, ATmega16HVA, ATmega32HVB, ATmega406, ATmega1284P, AT90PWM1, AT90PWM2, AT90PWM2B, AT90PWM3, AT90PWM3B, AT90PWM216, AT90PWM316, AT90PWM81, AT90PWM161, AT90USB82, AT90USB162, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287, ATtiny48, ATtiny88.

Note: This value does *not* match the bit pattern of the respective control register. It is solely meant to be used together with `wdt_enable()`.

**21.28.2.10 WDTO\_500MS** `#define WDTO_500MS 5`

See `WDTO_15MS`

**21.28.2.11 WDTO\_60MS** `#define WDTO_60MS 2`

See `WDTO_15MS`

**21.28.2.12 WDTO\_8S** `#define WDTO_8S 9`

See `WDTO_15MS` Note: This is only available on the ATtiny2313, ATtiny24, ATtiny44, ATtiny84, ATtiny84A, ATtiny25, ATtiny45, ATtiny85, ATtiny261, ATtiny461, ATtiny861, ATmega48\*, ATmega88\*, ATmega168\*, ATmega328\*, ATmega164P, ATmega324P, ATmega324PB, ATmega644P, ATmega644, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega8HVA, ATmega16HVA, ATmega32HVB, ATmega406, ATmega1284P, ATmega2564RFR2, ATmega256RFR2, ATmega1284RFR2, ATmega128RFR2, ATmega644RFR2, ATmega64RFR2, AT90PWM1, AT90PWM2, AT90PWM2B, AT90PWM3, AT90PWM3B, AT90PWM216, AT90PWM316, AT90PWM81, AT90PWM161, AT90USB82, AT90USB162, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287, ATtiny48, ATtiny88, ATxmega16a4u, ATxmega32a4u, ATxmega16c4, ATxmega32c4, ATxmega128c3, ATxmega192c3, ATxmega256c3.

Note: This value does *not* match the bit pattern of the respective control register. It is solely meant to be used together with `wdt_enable()`.

**21.29 <util/delay.h>: Convenience functions for busy-wait delay loops****Macros**

- `#define F_CPU 1000000UL`

**Functions**

- static void `_delay_ms` (double `__ms`)
- static void `_delay_us` (double `__us`)

### 21.29.1 Detailed Description

```
#define F_CPU 1000000UL // 1 MHz
//#define F_CPU 14.7456e6
#include <util/delay.h>
```

#### Note

As an alternative method, it is possible to pass the `F_CPU` macro down to the compiler from the Makefile. Obviously, in that case, no `#define` statement should be used.

The functions in this header file are wrappers around the basic busy-wait functions from `<util/delay_basic.h>`. They are meant as convenience functions where actual time values can be specified rather than a number of cycles to wait for. The idea behind is that compile-time constant expressions will be eliminated by compiler optimization so floating-point expressions can be used to calculate the number of delay cycles needed based on the CPU frequency passed by the macro `F_CPU`.

#### Note

In order for these functions to work as intended, compiler optimizations *must* be enabled, and the delay time *must* be an expression that is a known constant at compile-time. If these requirements are not met, the resulting delay will be much longer (and basically unpredictable), and applications that otherwise do not use floating-point calculations will experience severe code bloat by the floating-point library routines linked into the application.

The functions available allow the specification of microsecond, and millisecond delays directly, using the application-supplied macro `F_CPU` as the CPU clock frequency (in Hertz).

### 21.29.2 Macro Definition Documentation

#### 21.29.2.1 `F_CPU` `#define F_CPU 1000000UL`

CPU frequency in Hz.

The macro `F_CPU` specifies the CPU frequency to be considered by the delay macros. This macro is normally supplied by the environment (e.g. from within a project header, or the project's Makefile). The value 1 MHz here is only provided as a "vanilla" fallback if no such user-provided definition could be found.

In terms of the delay functions, the CPU frequency can be given as a floating-point constant (e.g. 3.6864e6 for 3.6864 MHz). However, the macros in `<util/setbaud.h>` require it to be an integer value.

### 21.29.3 Function Documentation

```
21.29.3.1 _delay_ms() void _delay_ms (  
    double __ms ) [inline], [static]
```

Perform a delay of `__ms` milliseconds, using `_delay_loop_2()`.

The macro `F_CPU` is supposed to be defined to a constant defining the CPU clock frequency (in Hertz).

The maximal possible delay is  $262.14 \text{ ms} / F\_CPU$  in MHz.

When the user request delay which exceed the maximum possible one, `_delay_ms()` provides a decreased resolution functionality. In this mode `_delay_ms()` will work with a resolution of 1/10 ms, providing delays up to 6.5535 seconds (independent from CPU frequency). The user will not be informed about decreased resolution.

If the avr-gcc toolchain has `__builtin_avr_delay_cycles()` support, maximal possible delay is  $4294967.295 \text{ ms} / F\_CPU$  in MHz. For values greater than the maximal possible delay, overflow may result in no delay i.e., 0 ms.

Conversion of `__ms` into clock cycles may not always result in an integral value. By default, the clock cycles are rounded up to the next integer. This ensures that the user gets at least `__ms` microseconds of delay.

Alternatively, by defining the macro `__DELAY_ROUND_DOWN__`, or `__DELAY_ROUND_CLOSEST__`, before including this header file, the algorithm can be made to round down, or round to closest integer, respectively.

#### Note

The implementation of `_delay_ms()` based on `__builtin_avr_delay_cycles()` is not backward compatible with older implementations. In order to get functionality backward compatible with previous versions, the macro `__DELAY_BACKWARD_COMPATIBLE__` must be defined before including this header file.

```
21.29.3.2 _delay_us() void _delay_us (  
    double __us ) [inline], [static]
```

Perform a delay of `__us` microseconds, using `_delay_loop_1()`.

The macro `F_CPU` is supposed to be defined to a constant defining the CPU clock frequency (in Hertz).

The maximal possible delay is  $768 \mu\text{s} / F\_CPU$  in MHz.

If the user requests a delay greater than the maximal possible one, `_delay_us()` will automatically call `_delay_ms()` instead. The user will not be informed about this case.

If the avr-gcc toolchain has `__builtin_avr_delay_cycles()` support, maximal possible delay is  $4294967.295 \mu\text{s} / F\_CPU$  in MHz. For values greater than the maximal possible delay, overflow may result in no delay i.e., 0  $\mu\text{s}$ .

Conversion of `__us` into clock cycles may not always result in integer. By default, the clock cycles are rounded up to next integer. This ensures that the user gets at least `__us` microseconds of delay.

Alternatively, by defining the macro `__DELAY_ROUND_DOWN__`, or `__DELAY_ROUND_CLOSEST__`, before including this header file, the algorithm can be made to round down, or round to closest integer, respectively.

#### Note

The implementation of `_delay_us()` based on `__builtin_avr_delay_cycles()` is not backward compatible with older implementations. In order to get functionality backward compatible with previous versions, the macro `__DELAY_BACKWARD_COMPATIBLE__` must be defined before including this header file.

## 21.30 <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks

### Macros

- #define `ATOMIC_BLOCK`(type)
- #define `NONATOMIC_BLOCK`(type)
- #define `ATOMIC_RESTORESTATE`
- #define `ATOMIC_FORCEON`
- #define `NONATOMIC_RESTORESTATE`
- #define `NONATOMIC_FORCEOFF`

### 21.30.1 Detailed Description

```
#include <util/atomic.h>
```

#### Note

The macros in this header file require the ISO/IEC 9899:1999 ("ISO C99") feature of for loop variables that are declared inside the for loop itself. For that reason, this header file can only be used if the standard level of the compiler (option `-std=`) is set to either `c99`, `gnu99` or higher.

The macros in this header file deal with code blocks that are guaranteed to be executed Atomically or Non-Atomically. The term "Atomic" in this context refers to the inability of the respective code to be interrupted.

These macros operate via automatic manipulation of the Global Interrupt Status (I) bit of the SREG register. Exit paths from both block types are all managed automatically without the need for special considerations, i.e. the interrupt status will be restored to the same value it had when entering the respective block (unless `ATOMIC_FORCEON` or `NONATOMIC_FORCEOFF` are used).

#### Warning

The features in this header are implemented by means of a for loop. This means that commands like `break` and `continue` that are located in an atomic block refer to the atomic for loop, not to a loop construct that hosts the atomic block.

A typical example that requires atomic access is a 16 (or more) bit variable that is shared between the main execution path and an ISR. While declaring such a variable as `volatile` ensures that the compiler will not optimize accesses to it away, it does not guarantee atomic access to it. Assuming the following example:

```
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/io.h>

volatile uint16_t ctr;

ISR(TIMER1_OVF_vect)
{
    ctr--;
}

...
int
main(void)
{
    ...
    ctr = 0x200;
    start_timer();
    while (ctr != 0)
        // wait
        ;
    ...
}
```

There is a chance where the main context will exit its wait loop when the variable `ctr` just reached the value `0xFF`. This happens because the compiler cannot natively access a 16-bit variable atomically in an 8-bit CPU. So the variable is for example at `0x100`, the compiler then tests the low byte for 0, which succeeds. It then proceeds to test the high byte, but that moment the ISR triggers, and the main context is interrupted. The ISR will decrement the variable from `0x100` to `0xFF`, and the main context proceeds. It now tests the high byte of the variable which is (now) also 0, so it concludes the variable has reached 0, and terminates the loop.

Using the macros from this header file, the above code can be rewritten like:

```
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/atomic.h>

volatile uint16_t ctr;

ISR(TIMER1_OVF_vect)
{
    ctr--;
}

...
int
main(void)
{
    ...
    ctr = 0x200;
    start_timer();
    sei();
    uint16_t ctr_copy;
    do
    {
        ATOMIC_BLOCK(ATOMIC_FORCEON)
        {
            ctr_copy = ctr;
        }
    }
    while (ctr_copy != 0);
    ...
}
```

This will install the appropriate interrupt protection before accessing variable `ctr`, so it is guaranteed to be consistently tested. If the global interrupt state were uncertain before entering the `ATOMIC_BLOCK`, it should be executed with the parameter `ATOMIC_RESTORESTATE` rather than `ATOMIC_FORCEON`.

See [Problems with reordering code](#) for things to be taken into account with respect to compiler optimizations.

## 21.30.2 Macro Definition Documentation

### 21.30.2.1 ATOMIC\_BLOCK `#define ATOMIC_BLOCK( type )`

Creates a block of code that is guaranteed to be executed atomically. Upon entering the block the Global Interrupt Status flag in SREG is disabled, and re-enabled upon exiting the block from any exit path.

Two possible macro parameters are permitted, `ATOMIC_RESTORESTATE` and `ATOMIC_FORCEON`.

### 21.30.2.2 ATOMIC\_FORCEON `#define ATOMIC_FORCEON`

This is a possible parameter for `ATOMIC_BLOCK`. When used, it will cause the `ATOMIC_BLOCK` to force the state of the SREG register on exit, enabling the Global Interrupt Status flag bit. This saves a small amount of flash space, a register, and one or more processor cycles, since the previous value of the SREG register does not need to be saved at the start of the block.

Care should be taken that `ATOMIC_FORCEON` is only used when it is known that interrupts are enabled before the block's execution or when the side effects of enabling global interrupts at the block's completion are known and understood.



### 21.30.2.3 **ATOMIC\_RESTORESTATE** `#define ATOMIC_RESTORESTATE`

This is a possible parameter for [ATOMIC\\_BLOCK](#). When used, it will cause the `ATOMIC_BLOCK` to restore the previous state of the `SREG` register, saved before the Global Interrupt Status flag bit was disabled. The net effect of this is to make the `ATOMIC_BLOCK`'s contents guaranteed atomic, without changing the state of the Global Interrupt Status flag when execution of the block completes.

### 21.30.2.4 **NONATOMIC\_BLOCK** `#define NONATOMIC_BLOCK( type )`

Creates a block of code that is executed non-atomically. Upon entering the block the Global Interrupt Status flag in `SREG` is enabled, and disabled upon exiting the block from any exit path. This is useful when nested inside `ATOMIC_BLOCK` sections, allowing for non-atomic execution of small blocks of code while maintaining the atomic access of the other sections of the parent `ATOMIC_BLOCK`.

Two possible macro parameters are permitted, [NONATOMIC\\_RESTORESTATE](#) and [NONATOMIC\\_FORCEOFF](#).

### 21.30.2.5 **NONATOMIC\_FORCEOFF** `#define NONATOMIC_FORCEOFF`

This is a possible parameter for [NONATOMIC\\_BLOCK](#). When used, it will cause the `NONATOMIC_BLOCK` to force the state of the `SREG` register on exit, disabling the Global Interrupt Status flag bit. This saves a small amount of flash space, a register, and one or more processor cycles, since the previous value of the `SREG` register does not need to be saved at the start of the block.

Care should be taken that `NONATOMIC_FORCEOFF` is only used when it is known that interrupts are disabled before the block's execution or when the side effects of disabling global interrupts at the block's completion are known and understood.

### 21.30.2.6 **NONATOMIC\_RESTORESTATE** `#define NONATOMIC_RESTORESTATE`

This is a possible parameter for [NONATOMIC\\_BLOCK](#). When used, it will cause the `NONATOMIC_BLOCK` to restore the previous state of the `SREG` register, saved before the Global Interrupt Status flag bit was enabled. The net effect of this is to make the `NONATOMIC_BLOCK`'s contents guaranteed non-atomic, without changing the state of the Global Interrupt Status flag when execution of the block completes.

## 21.31 `<util/crc16.h>`: CRC Computations

### Functions

- `static uint16_t _crc16_update (uint16_t __crc, uint8_t __data)`
- `static uint16_t _crc_xmodem_update (uint16_t __crc, uint8_t __data)`
- `static uint16_t _crc_ccitt_update (uint16_t __crc, uint8_t __data)`
- `static uint8_t _crc_ibutton_update (uint8_t __crc, uint8_t __data)`
- `static uint8_t _crc8_ccitt_update (uint8_t __crc, uint8_t __data)`

### 21.31.1 Detailed Description

```
#include <util/crc16.h>
```

This header file provides a optimized inline functions for calculating cyclic redundancy checks (CRC) using common polynomials.

#### References:

See the Dallas Semiconductor app note 27 for 8051 assembler example and general CRC optimization suggestions. The table on the last page of the app note is the key to understanding these implementations.

Jack Crenshaw's "Implementing CRCs" article in the January 1992 issue of *Embedded Systems Programming*. This may be difficult to find, but it explains CRC's in very clear and concise terms. Well worth the effort to obtain a copy.

A typical application would look like:

```
// Dallas iButton test vector.
uint8_t serno[] = { 0x02, 0x1c, 0xb8, 0x01, 0, 0, 0, 0xa2 };

int
checkcrc (void)
{
    uint8_t crc = 0, i;

    for (i = 0; i < sizeof serno / sizeof serno[0]; i++)
        crc = _crc_ibutton_update (crc, serno[i]);

    return crc; // must be 0
}
```

### 21.31.2 Function Documentation

**21.31.2.1 \_crc16\_update()** static uint16\_t \_crc16\_update (
 uint16\_t \_\_crc,
 uint8\_t \_\_data ) [inline], [static]

Optimized CRC-16 calculation.

Polynomial:  $x^{16} + x^{15} + x^2 + 1$  (0xa001)

Initial value: 0xffff

This CRC is normally used in disk-drive controllers.

The following is the equivalent functionality written in C.

```
uint16_t
crc16_update (uint16_t crc, uint8_t a)
{
    crc ^= a;
    for (int i = 0; i < 8; ++i)
    {
        if (crc & 1)
            crc = (crc >> 1) ^ 0xA001;
        else
            crc = crc >> 1;
    }

    return crc;
}
```

```
21.31.2.2 _crc8_ccitt_update() static uint8_t _crc8_ccitt_update (
    uint8_t __crc,
    uint8_t __data ) [inline], [static]
```

Optimized CRC-8-CCITT calculation.

Polynomial:  $x^8 + x^2 + x + 1$  (0xE0)

For use with simple CRC-8  
Initial value: 0x0

For use with CRC-8-ROHC  
Initial value: 0xff

Reference: <http://tools.ietf.org/html/rfc3095#section-5.9.1>

For use with CRC-8-ATM/ITU  
Initial value: 0xff

Final XOR value: 0x55

Reference: <http://www.itu.int/rec/T-REC-I.432.1-199902-I/en>

The C equivalent has been originally written by Dave Hylands. Assembly code is based on `_crc_ibutton_update` optimization.

The following is the equivalent functionality written in C.

```
uint8_t
_crc8_ccitt_update (uint8_t inCrc, uint8_t inData)
{
    uint8_t data = inCrc ^ inData;

    for (int i = 0; i < 8; i++)
    {
        if ((data & 0x80) != 0)
        {
            data <<= 1;
            data ^= 0x07;
        }
        else
        {
            data <<= 1;
        }
    }
    return data;
}
```

```
21.31.2.3 _crc_ccitt_update() static uint16_t _crc_ccitt_update (
    uint16_t __crc,
    uint8_t __data ) [inline], [static]
```

Optimized CRC-CCITT calculation.

Polynomial:  $x^{16} + x^{12} + x^5 + 1$  (0x8408)

Initial value: 0xffff

This is the CRC used by PPP and IrDA.

See RFC1171 (PPP protocol) and IrDA IrLAP 1.1

**Note**

Although the CCITT polynomial is the same as that used by the Xmodem protocol, they are quite different. The difference is in how the bits are shifted through the algorithm. Xmodem shifts the MSB of the CRC and the input first, while CCITT shifts the LSB of the CRC and the input first.

The following is the equivalent functionality written in C.

```
uint16_t
crc_ccitt_update (uint16_t crc, uint8_t data)
{
    data ^= 108 (crc);
    data ^= data << 4;

    return (((uint16_t)data << 8) | hi8 (crc)) ^ (uint8_t) (data >> 4)
           ^ ((uint16_t)data << 3);
}
```

**21.31.2.4 \_crc\_ibutton\_update()** static uint8\_t \_crc\_ibutton\_update (
 uint8\_t \_\_crc,
 uint8\_t \_\_data ) [inline], [static]

Optimized Dallas (now Maxim) iButton 8-bit CRC calculation.

Polynomial:  $x^8 + x^5 + x^4 + 1$  (0x8C)

Initial value: 0x0

See [http://www.maxim-ic.com/appnotes.cfm/appnote\\_number/27](http://www.maxim-ic.com/appnotes.cfm/appnote_number/27)

The following is the equivalent functionality written in C.

```
uint8_t
_crc_ibutton_update (uint8_t crc, uint8_t data)
{
    crc = crc ^ data;
    for (uint8_t i = 0; i < 8; i++)
    {
        if (crc & 0x01)
            crc = (crc >> 1) ^ 0x8C;
        else
            crc >>= 1;
    }
    return crc;
}
```

**21.31.2.5 \_crc\_xmodem\_update()** static uint16\_t \_crc\_xmodem\_update (
 uint16\_t \_\_crc,
 uint8\_t \_\_data ) [inline], [static]

Optimized CRC-XMODEM calculation.

Polynomial:  $x^{16} + x^{12} + x^5 + 1$  (0x1021)

Initial value: 0x0

This is the CRC used by the Xmodem-CRC protocol.

The following is the equivalent functionality written in C.

```
uint16_t
_crc_xmodem_update (uint16_t crc, uint8_t data)
{
    crc = crc ^ ((uint16_t)data << 8);
    for (int i = 0; i < 8; i++)
    {
        if (crc & 0x8000)
            crc = (crc << 1) ^ 0x1021;
        else
            crc <<= 1;
    }
    return crc;
}
```

## 21.32 <util/delay\_basic.h>: Basic busy-wait delay loops

### Functions

- void `_delay_loop_1` (uint8\_t \_\_count)
- void `_delay_loop_2` (uint16\_t \_\_count)

#### 21.32.1 Detailed Description

```
#include <util/delay_basic.h>
```

The functions in this header file implement simple delay loops that perform a busy-waiting. They are typically used to facilitate short delays in the program execution. They are implemented as count-down loops with a well-known CPU cycle count per loop iteration. As such, no other processing can occur simultaneously. It should be kept in mind that the functions described here do not disable interrupts.

In general, for long delays, the use of hardware timers is much preferable, as they free the CPU, and allow for concurrent processing of other events while the timer is running. However, in particular for very short delays, the overhead of setting up a hardware timer is too much compared to the overall delay time.

Two inline functions are provided for the actual delay algorithms.

#### 21.32.2 Function Documentation

**21.32.2.1 `_delay_loop_1()`** void `_delay_loop_1` (  
    uint8\_t \_\_count )

Delay loop using an 8-bit counter `__count`, so up to 256 iterations are possible. (The value 256 would have to be passed as 0.) The loop executes three CPU cycles per iteration, not including the overhead the compiler needs to setup the counter register.

Thus, at a CPU speed of 1 MHz, delays of up to 768 microseconds can be achieved.

**21.32.2.2 `_delay_loop_2()`** void `_delay_loop_2` (  
    uint16\_t \_\_count )

Delay loop using a 16-bit counter `__count`, so up to 65536 iterations are possible. (The value 65536 would have to be passed as 0.) The loop executes four CPU cycles per iteration, not including the overhead the compiler requires to setup the counter register pair.

Thus, at a CPU speed of 1 MHz, delays of up to about 262.1 milliseconds can be achieved.

## 21.33 <util/eu\_dst.h>: Daylight Saving function for the European Union.

### Functions

- int `eu_dst` (const time\_t \*timer, int32\_t \*z)

### 21.33.1 Detailed Description

```
#include <util/eu_dst.h>
```

Daylight Saving Time for the European Union

### 21.33.2 Function Documentation

**21.33.2.1 eu\_dst()** `int eu_dst (`  
    `const time_t * timer,`  
    `int32_t * z )`

To utilize this function, call  
`set_dst(eu_dst);`

Given the time stamp and time zone parameters provided, the Daylight Saving function must return a value appropriate for the tm structures' tm\_isdst element. That is:

- 0 : If Daylight Saving is not in effect.
- -1 : If it cannot be determined if Daylight Saving is in effect.
- A positive integer: Represents the number of seconds a clock is advanced for Daylight Saving. This will typically be ONE\_HOUR.

Daylight Saving 'rules' are subject to frequent change. For production applications it is recommended to write your own DST function, which uses 'rules' obtained from, and modifiable by, the end user (perhaps stored in EEPROM).

## 21.34 <util/parity.h>: Parity bit generation

### Functions

- static `uint8_t parity_even_bit (uint8_t __val)`

### 21.34.1 Detailed Description

```
#include <util/parity.h>
```

This header file contains optimized assembler code to calculate the parity bit for a byte.

### 21.34.2 Function Documentation

**21.34.2.1 parity\_even\_bit()** `uint8_t parity_even_bit (`  
    `uint8_t val ) [inline], [static]`

#### Returns

1 if `val` has an odd number of bits set, and 0 otherwise.

## 21.35 <util/setbaud.h>: Helper macros for baud rate calculations

### Macros

- #define BAUD\_TOL 2
- #define UBRR\_VALUE
- #define UBRRL\_VALUE
- #define UBRRH\_VALUE
- #define USE\_2X 0

### 21.35.1 Detailed Description

```
#define F_CPU 11059200
#define BAUD 38400
#include <util/setbaud.h>
```

This header file requires that on entry values are already defined for F\_CPU and BAUD. In addition, the macro BAUD\_TOL will define the baud rate tolerance (in percent) that is acceptable during the calculations. The value of BAUD\_TOL will default to 2 %.

This header file defines macros suitable to setup the UART baud rate prescaler registers of an AVR. All calculations are done using the C preprocessor. Including this header file causes no other side effects so it is possible to include this file more than once (supposedly, with different values for the BAUD parameter), possibly even within the same function.

Assuming that the requested BAUD is valid for the given F\_CPU then the macro UBRR\_VALUE is set to the required prescaler value. Two additional macros are provided for the low and high bytes of the prescaler, respectively↔: UBRRL\_VALUE is set to the lower byte of the UBRR\_VALUE and UBRRH\_VALUE is set to the upper byte. An additional macro USE\_2X will be defined. Its value is set to 1 if the desired BAUD rate within the given tolerance could only be achieved by setting the U2X bit in the UART configuration. It will be defined to 0 if U2X is not needed.

#### Example usage:

```
#include <avr/io.h>

#define F_CPU 4000000

static void
uart_9600(void)
{
#define BAUD 9600
#include <util/setbaud.h>
UBRRH = UBRRH_VALUE;
UBRRL = UBRRL_VALUE;
#if USE_2X
UCSRA |= (1 << U2X);
#else
UCSRA &= ~(1 << U2X);
#endif
}

static void
uart_38400(void)
{
#undef BAUD // avoid compiler warning
#define BAUD 38400
#include <util/setbaud.h>
UBRRH = UBRRH_VALUE;
UBRRL = UBRRL_VALUE;
#if USE_2X
UCSRA |= (1 << U2X);
#else
UCSRA &= ~(1 << U2X);
#endif
}
```

In this example, two functions are defined to setup the UART to run at 9600 Bd, and 38400 Bd, respectively. Using a CPU clock of 4 MHz, 9600 Bd can be achieved with an acceptable tolerance without setting U2X (prescaler 25), while 38400 Bd require U2X to be set (prescaler 12).

## 21.35.2 Macro Definition Documentation

### 21.35.2.1 BAUD\_TOL `#define BAUD_TOL 2`

Input and output macro for <util/setbaud.h>

Define the acceptable baud rate tolerance in percent. If not set on entry, it will be set to its default value of 2.

### 21.35.2.2 UBRR\_VALUE `#define UBRR_VALUE`

Output macro from <util/setbaud.h>

Contains the calculated baud rate prescaler value for the UBRR register.

### 21.35.2.3 UBRRH\_VALUE `#define UBRRH_VALUE`

Output macro from <util/setbaud.h>

Contains the upper byte of the calculated prescaler value (UBRR\_VALUE).

### 21.35.2.4 UBRRL\_VALUE `#define UBRRL_VALUE`

Output macro from <util/setbaud.h>

Contains the lower byte of the calculated prescaler value (UBRR\_VALUE).

### 21.35.2.5 USE\_2X `#define USE_2X 0`

Output macro from <util/setbaud.h>

Contains the value 1 if the desired baud rate tolerance could only be achieved by setting the U2X bit in the UART configuration. Contains 0 otherwise.



## 21.36 <util/twi.h>: TWI bit mask definitions

### TWSR values

Mnemonics:

TW\_MT\_xxx - master transmitter

TW\_MR\_xxx - master receiver

TW\_ST\_xxx - slave transmitter

TW\_SR\_xxx - slave receiver

- #define TW\_START 0x08
- #define TW\_REP\_START 0x10
- #define TW\_MT\_SLA\_ACK 0x18
- #define TW\_MT\_SLA\_NACK 0x20
- #define TW\_MT\_DATA\_ACK 0x28
- #define TW\_MT\_DATA\_NACK 0x30
- #define TW\_MT\_ARB\_LOST 0x38
- #define TW\_MR\_ARB\_LOST 0x38
- #define TW\_MR\_SLA\_ACK 0x40
- #define TW\_MR\_SLA\_NACK 0x48
- #define TW\_MR\_DATA\_ACK 0x50
- #define TW\_MR\_DATA\_NACK 0x58
- #define TW\_ST\_SLA\_ACK 0xA8
- #define TW\_ST\_ARB\_LOST\_SLA\_ACK 0xB0
- #define TW\_ST\_DATA\_ACK 0xB8
- #define TW\_ST\_DATA\_NACK 0xC0
- #define TW\_ST\_LAST\_DATA 0xC8
- #define TW\_SR\_SLA\_ACK 0x60
- #define TW\_SR\_ARB\_LOST\_SLA\_ACK 0x68
- #define TW\_SR\_GCALL\_ACK 0x70
- #define TW\_SR\_ARB\_LOST\_GCALL\_ACK 0x78
- #define TW\_SR\_DATA\_ACK 0x80
- #define TW\_SR\_DATA\_NACK 0x88
- #define TW\_SR\_GCALL\_DATA\_ACK 0x90
- #define TW\_SR\_GCALL\_DATA\_NACK 0x98
- #define TW\_SR\_STOP 0xA0
- #define TW\_NO\_INFO 0xF8
- #define TW\_BUS\_ERROR 0x00
- #define TW\_STATUS\_MASK
- #define TW\_STATUS (TWSR & TW\_STATUS\_MASK)

### R/~W bit in SLA+R/W address field.

- #define TW\_READ 1
- #define TW\_WRITE 0

#### 21.36.1 Detailed Description

```
#include <util/twi.h>
```

This header file contains bit mask definitions for use with the AVR TWI interface.

## 21.36.2 Macro Definition Documentation

**21.36.2.1 TW\_BUS\_ERROR** `#define TW_BUS_ERROR 0x00`

illegal start or stop condition

**21.36.2.2 TW\_MR\_ARB\_LOST** `#define TW_MR_ARB_LOST 0x38`

arbitration lost in SLA+R or NACK

**21.36.2.3 TW\_MR\_DATA\_ACK** `#define TW_MR_DATA_ACK 0x50`

data received, ACK returned

**21.36.2.4 TW\_MR\_DATA\_NACK** `#define TW_MR_DATA_NACK 0x58`

data received, NACK returned

**21.36.2.5 TW\_MR\_SLA\_ACK** `#define TW_MR_SLA_ACK 0x40`

SLA+R transmitted, ACK received

**21.36.2.6 TW\_MR\_SLA\_NACK** `#define TW_MR_SLA_NACK 0x48`

SLA+R transmitted, NACK received

**21.36.2.7 TW\_MT\_ARB\_LOST** `#define TW_MT_ARB_LOST 0x38`

arbitration lost in SLA+W or data

**21.36.2.8 TW\_MT\_DATA\_ACK** `#define TW_MT_DATA_ACK 0x28`

data transmitted, ACK received

**21.36.2.9 TW\_MT\_DATA\_NACK** `#define TW_MT_DATA_NACK 0x30`

data transmitted, NACK received

**21.36.2.10 TW\_MT\_SLA\_ACK** `#define TW_MT_SLA_ACK 0x18`

SLA+W transmitted, ACK received

**21.36.2.11 TW\_MT\_SLA\_NACK** #define TW\_MT\_SLA\_NACK 0x20

SLA+W transmitted, NACK received

**21.36.2.12 TW\_NO\_INFO** #define TW\_NO\_INFO 0xF8

no state information available

**21.36.2.13 TW\_READ** #define TW\_READ 1

SLA+R address

**21.36.2.14 TW\_REP\_START** #define TW\_REP\_START 0x10

repeated start condition transmitted

**21.36.2.15 TW\_SR\_ARB\_LOST\_GCALL\_ACK** #define TW\_SR\_ARB\_LOST\_GCALL\_ACK 0x78

arbitration lost in SLA+RW, general call received, ACK returned

**21.36.2.16 TW\_SR\_ARB\_LOST\_SLA\_ACK** #define TW\_SR\_ARB\_LOST\_SLA\_ACK 0x68

arbitration lost in SLA+RW, SLA+W received, ACK returned

**21.36.2.17 TW\_SR\_DATA\_ACK** #define TW\_SR\_DATA\_ACK 0x80

data received, ACK returned

**21.36.2.18 TW\_SR\_DATA\_NACK** #define TW\_SR\_DATA\_NACK 0x88

data received, NACK returned

**21.36.2.19 TW\_SR\_GCALL\_ACK** #define TW\_SR\_GCALL\_ACK 0x70

general call received, ACK returned

**21.36.2.20 TW\_SR\_GCALL\_DATA\_ACK** #define TW\_SR\_GCALL\_DATA\_ACK 0x90

general call data received, ACK returned

**21.36.2.21 TW\_SR\_GCALL\_DATA\_NACK** #define TW\_SR\_GCALL\_DATA\_NACK 0x98

general call data received, NACK returned

**21.36.2.22 TW\_SR\_SLA\_ACK** #define TW\_SR\_SLA\_ACK 0x60

SLA+W received, ACK returned

**21.36.2.23 TW\_SR\_STOP** #define TW\_SR\_STOP 0xA0

stop or repeated start condition received while selected

**21.36.2.24 TW\_ST\_ARB\_LOST\_SLA\_ACK** #define TW\_ST\_ARB\_LOST\_SLA\_ACK 0xB0

arbitration lost in SLA+RW, SLA+R received, ACK returned

**21.36.2.25 TW\_ST\_DATA\_ACK** #define TW\_ST\_DATA\_ACK 0xB8

data transmitted, ACK received

**21.36.2.26 TW\_ST\_DATA\_NACK** #define TW\_ST\_DATA\_NACK 0xC0

data transmitted, NACK received

**21.36.2.27 TW\_ST\_LAST\_DATA** #define TW\_ST\_LAST\_DATA 0xC8

last data byte transmitted, ACK received

**21.36.2.28 TW\_ST\_SLA\_ACK** #define TW\_ST\_SLA\_ACK 0xA8

SLA+R received, ACK returned

**21.36.2.29 TW\_START** #define TW\_START 0x08

start condition transmitted

**21.36.2.30 TW\_STATUS** #define TW\_STATUS (TWSR & TW\_STATUS\_MASK)

TWSR, masked by TW\_STATUS\_MASK

**21.36.2.31 TW\_STATUS\_MASK** #define TW\_STATUS\_MASK

**Value:**

```
(_BV(TWS7) | _BV(TWS6) | _BV(TWS5) | _BV(TWS4) | \
_BV(TWS3))
```

The lower 3 bits of TWSR are reserved on the ATmega163. The 2 LSB carry the prescaler bits on the newer ATmegas.

**21.36.2.32 TW\_WRITE** `#define TW_WRITE 0`

SLA+W address

## 21.37 <util/usa\_dst.h>: Daylight Saving function for the USA.

### Functions

- int `usa_dst` (const `time_t` \*timer, `int32_t` \*z)

#### 21.37.1 Detailed Description

```
#include <util/usa_dst.h>
```

Daylight Saving function for the USA.

#### 21.37.2 Function Documentation

**21.37.2.1 `usa_dst()`** `int usa_dst (`  
    `const time_t * timer,`  
    `int32_t * z )`

To utilize this function, call  
`set_dst(usa_dst);`

Given the time stamp and time zone parameters provided, the Daylight Saving function must return a value appropriate for the tm structures' tm\_isdst element. That is:

- 0 : If Daylight Saving is not in effect.
- -1 : If it cannot be determined if Daylight Saving is in effect.
- A positive integer : Represents the number of seconds a clock is advanced for Daylight Saving. This will typically be ONE\_HOUR.

Daylight Saving 'rules' are subject to frequent change. For production applications it is recommended to write your own DST function, which uses 'rules' obtained from, and modifiable by, the end user (perhaps stored in EEPROM).

## 21.38 <compat/deprecated.h>: Deprecated items

### Allowing specific system-wide interrupts

In addition to globally enabling interrupts, each device's particular interrupt needs to be enabled separately if interrupts for this device are desired. While some devices maintain their interrupt enable bit inside the device's register set, external and timer interrupts have system-wide configuration registers.

Example:

```
// Enable timer 1 overflow interrupts.
timer_enable_int(_BV(TOIE1));

// Do some work...

// Disable all timer interrupts.
timer_enable_int(0);
```

#### Note

Be careful when you use these functions. If you already have a different interrupt enabled, you could inadvertently disable it by enabling another interrupt.

- static void `timer_enable_int` (unsigned char ints)
- #define `enable_external_int`(mask) (`__EICR = mask`)
- #define `INTERRUPT`(signame)
- #define `__INTR_ATTRS __used__`

### Obsolete IO macros

Back in a time when AVR-GCC and AVR-LibC could not handle IO port access in the direct assignment form as they are handled now, all IO port access had to be done through specific macros that eventually resulted in inline assembly instructions performing the desired action.

These macros became obsolete, as reading and writing IO ports can be done by simply using the IO port name in an expression, and all bit manipulation (including those on IO ports) can be done using generic C bit manipulation operators.

The macros in this group simulate the historical behaviour. While they are supposed to be applied to IO ports, the emulation actually uses standard C methods, so they could be applied to arbitrary memory locations as well.

- #define `inp`(port) (port)
- #define `outp`(val, port) (port) = (val)
- #define `inb`(port) (port)
- #define `outb`(port, val) (port) = (val)
- #define `sbi`(port, bit) (port) |= (1 << (bit))
- #define `cbi`(port, bit) (port) &= ~(1 << (bit))

#### 21.38.1 Detailed Description

This header file contains several items that used to be available in previous versions of this library, but have eventually been deprecated over time.

```
#include <compat/deprecated.h>
```

These items are supplied within that header file for backward compatibility reasons only, so old source code that has been written for previous library versions could easily be maintained until its end-of-life. Use of any of these items in new code is strongly discouraged.

## 21.38.2 Macro Definition Documentation

**21.38.2.1 cbi** `#define cbi(  
port,  
bit ) (port) &= ~(1 << (bit))`

### Deprecated

Clear bit in IO port `port`.

**21.38.2.2 enable\_external\_int** `#define enable_external_int(  
mask ) (__EICR = mask)`

### Deprecated

This macro gives access to the `GIMSK` register (or `EIMSK` register if using an AVR Mega device or `GICR` register for others). Although this macro is essentially the same as assigning to the register, it does adapt slightly to the type of device being used. This macro is unavailable if none of the registers listed above are defined.

**21.38.2.3 inb** `#define inb(  
port ) (port)`

### Deprecated

Read a value from an IO port `port`.

**21.38.2.4 inp** `#define inp(  
port ) (port)`

### Deprecated

Read a value from an IO port `port`.

**21.38.2.5 INTERRUPT** `#define INTERRUPT(  
signame )`

#### Value:

```
void signame (void) __attribute__ ((__interrupt__, __INTR_ATTRS)); \
void signame (void)
```

### Deprecated

Introduces an interrupt handler function that runs with global interrupts initially enabled. This allows interrupt handlers to be interrupted.

As this macro has been used by too many unsuspecting people in the past, it has been deprecated, and will be removed in a future version of the library. Users who want to legitimately re-enable interrupts in their interrupt handlers as quickly as possible are encouraged to explicitly declare their handlers as described [above](#).

```
21.38.2.6 outb #define outb(  
    port,  
    val ) (port) = (val)
```

### Deprecated

Write `val` to IO port `port`.

```
21.38.2.7 outp #define outp(  
    val,  
    port ) (port) = (val)
```

### Deprecated

Write `val` to IO port `port`.

```
21.38.2.8 sbi #define sbi(  
    port,  
    bit ) (port) |= (1 << (bit))
```

### Deprecated

Set `bit` in IO port `port`.

## 21.38.3 Function Documentation

```
21.38.3.1 timer_enable_int() static void timer_enable_int (  
    unsigned char ints ) [inline], [static]
```

### Deprecated

This function modifies the `timsk` register. The value you pass via `ints` is device specific.

## 21.39 <compat/ina90.h>: Compatibility with IAR EWB 3.x

```
#include <compat/ina90.h>
```

This is an attempt to provide some compatibility with header files that come with IAR C, to make porting applications between different compilers easier. No 100% compatibility though.

### Note

For actual documentation, please see the IAR manual.



## 21.40 Demo projects

### Modules

- [Combining C and assembly source files](#)
- [A simple project](#)
- [A more sophisticated project](#)
- [Using the standard IO facilities](#)
- [Example using the two-wire interface \(TWI\)](#)

### 21.40.1 Detailed Description

Various small demo projects are provided to illustrate several aspects of using the opensource utilities for the AVR controller series. It should be kept in mind that these demos serve mainly educational purposes, and are normally not directly suitable for use in any production environment. Usually, they have been kept as simple as sufficient to demonstrate one particular feature.

The [simple project](#) is somewhat like the "Hello world!" application for a microcontroller, about the most simple project that can be done. It is explained in good detail, to allow the reader to understand the basic concepts behind using the tools on an AVR microcontroller.

The [more sophisticated demo project](#) builds on top of that simple project, and adds some controls to it. It touches a number of AVR-LibC's basic concepts on its way.

A [comprehensive example on using the standard IO facilities](#) intends to explain that complex topic, using a practical microcontroller peripheral setup with one RS-232 connection, and an HD44780-compatible industry-standard LCD display.

The [Example using the two-wire interface \(TWI\)](#) project explains the use of the two-wire hardware interface (also known as "I2C") that is present on many AVR controllers.

Finally, the [Combining C and assembly source files](#) demo shows how C and assembly language source files can collaborate within one project. While the overall project is managed by a C program part for easy maintenance, time-critical parts are written directly in manually optimized assembly language for shortest execution times possible. Naturally, this kind of project is very closely tied to the hardware design, thus it is custom-tailored to a particular controller type and peripheral setup. As an alternative to the assembly-language solution, this project also offers a C-only implementation (deploying the exact same peripheral setup) based on a more sophisticated (and thus more expensive) but pin-compatible controller.

While the simple demo is meant to run on about any AVR setup possible where a LED could be connected to the OCR1[A] output, the [large](#) and [stdio](#) demos are mainly targeted to the Atmel STK500 starter kit, and the [TWI](#) example requires a controller where some 24Cxx two-wire EEPROM can be connected to. For the STK500 demos, the default CPU (either an AT90S8515 or an ATmega8515) should be removed from its socket, and the ATmega16 that ships with the kit should be inserted into socket SCKT3100A3. The ATmega16 offers an on-board ADC that is used in the [large](#) demo, and all AVRs with an ADC feature a different pinout than the industry-standard compatible devices.

In order to fully utilize the [large](#) demo, a female 10-pin header with cable, connecting to a 10 kOhm potentiometer will be useful.

For the [stdio](#) demo, an industry-standard HD44780-compatible LCD display of at least 16x1 characters will be needed. Among other things, the [LCD4Linux](#) project page describes many things around these displays, including common pinouts.

## 21.41 Combining C and assembly source files

For time- or space-critical applications, it can often be desirable to combine C code (for easy maintenance) and assembly code (for maximal speed or minimal code size) together. This demo provides an example of how to do that.

The objective of the demo is to decode radio-controlled model PWM signals, and control an output PWM based on the current input signal's value. The incoming PWM pulses follow a standard encoding scheme where a pulse width of 920 microseconds denotes one end of the scale (represented as 0 % pulse width on output), and 2120 microseconds mark the other end (100 % output PWM). Normally, multiple channels would be encoded that way in subsequent pulses, followed by a larger gap, so the entire frame will repeat each 14 through 20 ms, but this is ignored for the purpose of the demo, so only a single input PWM channel is assumed.

The basic challenge is to use the cheapest controller available for the task, an ATtiny13 that has only a single timer channel. As this timer channel is required to run the outgoing PWM signal generation, the incoming PWM decoding had to be adjusted to the constraints set by the outgoing PWM.

As PWM generation toggles the counting direction of timer 0 between up and down after each 256 timer cycles, the current time cannot be deduced by reading TCNT0 only, but the current counting direction of the timer needs to be considered as well. This requires servicing interrupts whenever the timer hits *TOP* (255) and *BOTTOM* (0) to learn about each change of the counting direction. For PWM generation, it is usually desired to run it at the highest possible speed so filtering the PWM frequency from the modulated output signal is made easy. Thus, the PWM timer runs at full CPU speed. This causes the overflow and compare match interrupts to be triggered each 256 CPU clocks, so they must run with the minimal number of processor cycles possible in order to not impose a too high CPU load by these interrupt service routines. This is the main reason to implement the entire interrupt handling in fine-tuned assembly code rather than in C.

In order to verify parts of the algorithm, and the underlying hardware, the demo has been set up in a way so the pin-compatible but more expensive ATtiny45 (or its siblings ATtiny25 and ATtiny85) could be used as well. In that case, no separate assembly code is required, as two timer channels are available.

### 21.41.1 Hardware setup

The incoming PWM pulse train is fed into PB4. It will generate a pin change interrupt there on each edge of the incoming signal.

The outgoing PWM is generated through OC0B of timer channel 0 (PB1). For demonstration purposes, a LED should be connected to that pin (like, one of the LEDs of an STK500).

The controllers run on their internal calibrated RC oscillators, 1.2 MHz on the ATtiny13, and 1.0 MHz on the ATtiny45.

### 21.41.2 A code walkthrough

**21.41.2.1 `asmdemo.c`** After the usual include files, two variables are defined. The first one, `pwm_incoming` is used to communicate the most recent pulse width detected by the incoming PWM decoder up to the main loop.

The second variable actually only constitutes of a single bit, `intbits.pwm_received`. This bit will be set whenever the incoming PWM decoder has updated `pwm_incoming`.

Both variables are marked *volatile* to ensure their readers will always pick up an updated value, as both variables will be set by interrupt service routines.

The function `ioinit()` initializes the microcontroller peripheral devices. In particular, it starts timer 0 to generate the outgoing PWM signal on OC0B. Setting OCR0A to 255 (which is the *TOP* value of timer 0) is used to generate a timer 0 overflow A interrupt on the ATtiny13. This interrupt is used to inform the incoming PWM decoder that the

counting direction of channel 0 is just changing from up to down. Likewise, an overflow interrupt will be generated whenever the countdown reached *BOTTOM* (value 0), where the counter will again alter its counting direction to upwards. This information is needed in order to know whether the current counter value of `TCNT0` is to be evaluated from bottom or top.

Further, `ioinit()` activates the pin-change interrupt `PCINT0` on any edge of PB4. Finally, PB1 (`OC0B`) will be activated as an output pin, and global interrupts are being enabled.

In the ATtiny45 setup, the C code contains an ISR for `PCINT0`. At each pin-change interrupt, it will first be analyzed whether the interrupt was caused by a rising or a falling edge. In case of the rising edge, timer 1 will be started with a prescaler of 16 after clearing the current timer value. Then, at the falling edge, the current timer value will be recorded (and timer 1 stopped), the pin-change interrupt will be suspended, and the upper layer will be notified that the incoming PWM measurement data is available.

Function `main()` first initializes the hardware by calling `ioinit()`, and then waits until some incoming PWM value is available. If it is, the output PWM will be adjusted by computing the relative value of the incoming PWM. Finally, the pin-change interrupt is re-enabled, and the CPU is put to sleep.

**21.41.2.2 project.h** In order for the interrupt service routines to be as fast as possible, some of the CPU registers are set aside completely for use by these routines, so the compiler would not use them for C code. This is arranged for in [project.h](#).

The file is divided into one section that will be used by the assembly source code, and another one to be used by C code. The assembly part is distinguished by the preprocessing macro `__ASSEMBLER__` (which will be automatically set by the compiler front-end when preprocessing an assembly-language file), and it contains just macros that give symbolic names to a number of CPU registers. The preprocessor will then replace the symbolic names by their right-hand side definitions before calling the assembler.

In C code, the compiler needs to see variable declarations for these objects. This is done by using declarations that bind a variable permanently to a CPU register (see [How to permanently bind a variable to a register?](#)). Even in case the C code never has a need to access these variables, declaring the register binding that way causes the compiler to not use these registers in C code at all.

The `flags` variable needs to be in the range of r16 through r31 as it is the target of a *load immediate* (or *SER*) instruction that is not applicable to the entire register file.

**21.41.2.3 isrs.S** This file is a preprocessed assembly source file. The C preprocessor will be run by the compiler front-end first, resolving all `#include`, `#define` etc. directives. The resulting program text will then be passed on to the assembler.

As the C preprocessor strips all C-style comments, preprocessed assembly source files can have both, C-style (`/* ... */`, `// ...`) as well as assembly-style (`;` ...) comments.

At the top, the IO register definition file `avr/io.h` and the project declaration file [project.h](#) are included. The remainder of the file is conditionally assembled only if the target MCU type is an ATtiny13, so it will be completely ignored for the ATtiny45 option.

Next are the two interrupt service routines for timer 0 compare A match (timer 0 hits *TOP*, as `OCR0A` is set to 255) and timer 0 overflow (timer 0 hits *BOTTOM*). As discussed above, these are kept as short as possible. They only save `SREG` (as the flags will be modified by the `INC` instruction), increment the `counter_hi` variable which forms the high part of the current time counter (the low part is formed by querying `TCNT0` directly), and clear or set the variable `flags`, respectively, in order to note the current counting direction. The `RETI` instruction terminates these interrupt service routines. Total cycle count is 8 CPU cycles, so together with the 4 CPU cycles needed for interrupt setup, and the 2 cycles for the `RJMP` from the interrupt vector to the handler, these routines will require 14 out of each 256 CPU cycles, or about 5 % of the overall CPU time.

The pin-change interrupt `PCINT0` will be handled in the final part of this file. The basic algorithm is to quickly evaluate the current system time by fetching the current timer value of `TCNT0`, and combining it with the overflow part in `counter_hi`. If the counter is currently counting down rather than up, the value fetched from `TCNT0` must be negated. Finally, if this pin-change interrupt was triggered by a rising edge, the time computed will be recorded as the start time only. Then, at the falling edge, this start time will be subtracted from the current time to compute the actual pulse width seen (left in `pwm_incoming`), and the upper layers are informed of the new value by setting bit 0 in the `intbits` flags. At the same time, this pin-change interrupt will be disabled so no new measurement can be performed until the upper layer had a chance to process the current value.

### 21.41.3 The source code

The source code is installed under

```
$prefix/share/doc/avr-libc/examples/asmdemo/,
```

where `$prefix` is a configuration option. For Unix systems, it is usually set to either `/usr` or `/usr/local`.

## 21.42 A simple project

At this point, you should have the GNU tools configured, built, and installed on your system. In this chapter, we present a simple example of using the GNU tools in an AVR project. After reading this chapter, you should have a better feel as to how the tools are used and how a `Makefile` can be configured.

### 21.42.1 The Project

This project will use the pulse-width modulator (PWM) to ramp an LED on and off every two seconds. An AT90S2313 processor will be used as the controller. The circuit for this demonstration is shown in the [schematic diagram](#). If you have a development kit, you should be able to use it, rather than build the circuit, for this project.

#### Note

Meanwhile, the AT90S2313 became obsolete. Either use its successor, the (pin-compatible) ATtiny2313 for the project, or perhaps the ATmega8 or one of its successors (ATmega48/88/168) which have become quite popular since the original demo project had been established. For all these more modern devices, it is no longer necessary to use an external crystal for clocking as they ship with the internal 1 MHz oscillator enabled, so C1, C2, and Q1 can be omitted. Normally, for this experiment, the external circuitry on /RESET (R1, C3) can be omitted as well, leaving only the AVR, the LED, the bypass capacitor C4, and perhaps R2. For the ATmega8/48/88/168, use PB1 (pin 15 at the DIP-28 package) to connect the LED to. Additionally, this demo has been ported to many different other AVRs. The location of the respective OC pin varies between different AVRs, and it is mandated by the AVR hardware.

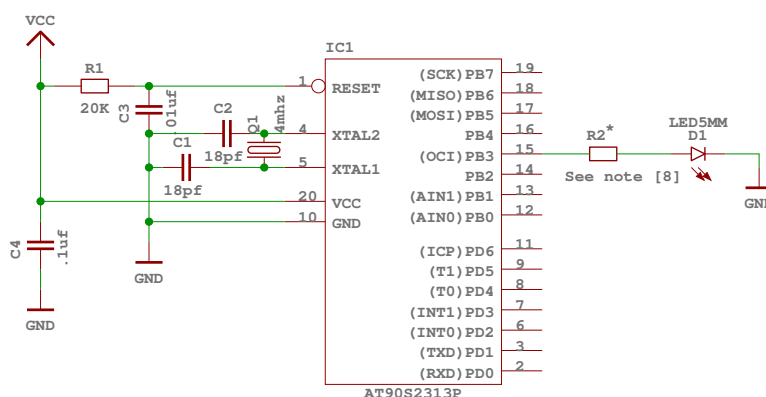


Figure 4 Schematic of circuit for demo project

The source code is given in [demo.c](#). For the sake of this example, create a file called `demo.c` containing this source code. Some of the more important parts of the code are:

**Note [1]:**

As the AVR microcontroller series has been developed during the past years, new features have been added over time. Even though the basic concepts of the timer/counter1 are still the same as they used to be back in early 2001 when this simple demo was written initially, the names of registers and bits have been changed slightly to reflect the new features. Also, the port and pin mapping of the output compare match 1A (or 1 for older devices) pin which is used to control the LED varies between different AVRs. The file `iocompat.h` tries to abstract between all this differences using some preprocessor `#ifdef` statements, so the actual program itself can operate on a common set of symbolic names. The macros defined by that file are:

- `OCR` the name of the OCR register used to control the PWM (usually either `OCR1` or `OCR1A`)
- `DDROC` the name of the DDR (data direction register) for the OC output
- `OC1` the pin number of the OC1[A] output within its port
- `TIMER1_TOP` the TOP value of the timer used for the PWM (1023 for 10-bit PWMs, 255 for devices that can only handle an 8-bit PWM)
- `TIMER1_PWM_INIT` the initialization bits to be set into control register 1A in order to setup 10-bit (or 8-bit) phase and frequency correct PWM mode
- `TIMER1_CLOCKSOURCE` the clock bits to set in the respective control register to start the PWM timer; usually the timer runs at full CPU clock for 10-bit PWMs, while it runs on a prescaled clock for 8-bit PWMs

**Note [2]:**

`ISR()` is a macro that marks the function as an interrupt routine. In this case, the function will get called when timer 1 overflows. Setting up interrupts is explained in greater detail in [<avr/interrupt.h>: Interrupts](#).

**Note [3]:**

The `PWM` is being used in 10-bit mode, so we need a 16-bit variable to remember the current value.

**Note [4]:**

This section determines the new value of the `PWM`.

**Note [5]:**

Here's where the newly computed value is loaded into the `PWM` register. Since we are in an interrupt routine, it is safe to use a 16-bit assignment to the register. Outside of an interrupt, the assignment should only be performed with interrupts disabled if there's a chance that an interrupt routine could also access this register (or another register that uses `TEMP`), see the appropriate [FAQ entry](#).

**Note [6]:**

This routine gets called after a reset. It initializes the `PWM` and enables interrupts.

**Note [7]:**

The main loop of the program does nothing – all the work is done by the interrupt routine! The `sleep_mode()` puts the processor on sleep until the next interrupt, to conserve power. Of course, that probably won't be noticeable as we are still driving a LED, it is merely mentioned here to demonstrate the basic principle.

**Note [8]:**

Early AVR devices saturate their outputs at rather low currents when sourcing current, so the LED can be connected directly, the resulting current through the LED will be about 15 mA. For modern parts (at least for the ATmega 128), however Atmel has drastically increased the IO source capability, so when operating at 5 V `Vcc`, `R2` is needed. Its value should be about 150 Ohms. When operating the circuit at 3 V, it can still be omitted though.

## 21.42.2 The Source Code

```

/*
 * -----
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <joerg@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
 * can do whatever you want with this stuff.  If we meet some day, and you think
 * this stuff is worth it, you can buy me a beer in return.          Joerg Wunsch
 * -----
 *
 * Simple AVR demonstration.  Controls a LED that can be directly
 * connected from OC1/OC1A to GND.  The brightness of the LED is
 * controlled with the PWM.  After each period of the PWM, the PWM
 * value is either incremented or decremented, that's all.
 *
 * $Id$
 */

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#include "iocompat.h"      /* Note [1] */

enum { UP, DOWN };

ISR (TIMER1_OVF_vect)     /* Note [2] */
{
    static uint16_t pwm;   /* Note [3] */
    static uint8_t direction;

    switch (direction)    /* Note [4] */
    {
        case UP:
            if (++pwm == TIMER1_TOP)
                direction = DOWN;
            break;

        case DOWN:
            if (--pwm == 0)
                direction = UP;
            break;
    }

    OCR = pwm;           /* Note [5] */
}

void
ioinit (void)           /* Note [6] */
{
    /* Timer 1 is 10-bit PWM (8-bit PWM on some ATtinys). */
    TCCR1A = TIMER1_PWM_INIT;
    /*
     * Start timer 1.
     *
     * NB: TCCR1A and TCCR1B could actually be the same register, so
     * take care to not clobber it.
     */
    TCCR1B |= TIMER1_CLOCKSOURCE;
    /*
     * Run any device-dependent timer 1 setup hook if present.
     */
#ifdef TIMER1_SETUP_HOOK
    TIMER1_SETUP_HOOK();
#endif

    /* Set PWM value to 0. */
    OCR = 0;

    /* Enable OC1 as output. */
    DDRC = _BV (OC1);

    /* Enable timer 1 overflow interrupt. */
    TIMSK = _BV (TOIE1);
    sei ();
}

int
main (void)
{
    ioinit ();

    /* loop forever, the interrupts are doing the rest */

    for (;;)            /* Note [7] */

```

```
    sleep_mode();  
    return (0);  
}
```

### 21.42.3 Compiling and Linking

This first thing that needs to be done is compile the source. When compiling, the compiler needs to know the processor type so the `-mmcu` option is specified. The `-Os` option will tell the compiler to optimize the code for efficient space usage (at the possible expense of code execution speed). The `-g` is used to embed debug info. The debug info is useful for disassemblies and doesn't end up in the `.hex` files, so I usually specify it. Finally, the `-c` tells the compiler to compile and stop – don't link. This demo is small enough that we could compile and link in one step. However, real-world projects will have several modules and will typically need to break up the building of the project into several compiles and one link.

```
$ avr-gcc -g -Os -mmcu=atmega8 -c demo.c
```

The compilation will create a `demo.o` file. Next we link it into a binary called `demo.elf`.

```
$ avr-gcc -g -mmcu=atmega8 -o demo.elf demo.o
```

It is important to specify the MCU type when linking. The compiler uses the `-mmcu` option to choose start-up files and run-time libraries that get linked together. If this option isn't specified, the compiler defaults to the 8515 processor environment, which is most certainly what you didn't want.

### 21.42.4 Examining the Object File

Now we have a binary file. Can we do anything useful with it (besides put it into the processor?) The GNU Binutils suite is made up of many useful tools for manipulating object files that get generated. One tool is `avr-objdump`, which takes information from the object file and displays it in many useful ways. Typing the command by itself will cause it to list out its options.

For instance, to get a feel of the application's size, the `-h` option can be used. The output of this option shows how much space is used in each of the sections (the `.stab` and `.stabstr` sections hold the debugging information and won't make it into the ROM file).

An even more useful option is `-S`. This option disassembles the binary file and intersperses the source code in the output! This method is much better, in my opinion, than using the `-S` with the compiler because this listing includes routines from the libraries and the vector table contents. Also, all the "fix-ups" have been satisfied. In other words, the listing generated by this option reflects the actual code that the processor will run.

```
$ avr-objdump -h -S demo.elf > demo.lst
```

Here's the output as saved in the `demo.lst` file:

```
demo.elf:      file format elf32-avr
```

## Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	000000dc	00000000	00000000	00000094	2**1
		CONTENTS, ALLOC, LOAD, READONLY, CODE				
1	.data	00000000	00800060	000000dc	00000170	2**0
		CONTENTS, ALLOC, LOAD, DATA				
2	.bss	00000003	00800060	00800060	00000170	2**0
		ALLOC				
3	.stab	00000234	00000000	00000000	00000170	2**2
		CONTENTS, READONLY, DEBUGGING				
4	.stabstr	0000015e	00000000	00000000	000003a4	2**0
		CONTENTS, READONLY, DEBUGGING				
5	.comment	00000012	00000000	00000000	00000502	2**0
		CONTENTS, READONLY				
6	.note.gnu.avr.deviceinfo	0000003c	00000000	00000000	00000000	00000514 2**2
		CONTENTS, READONLY				
7	.debug_info	0000048c	00000000	00000000	00000550	2**0
		CONTENTS, READONLY, DEBUGGING				
8	.debug_abbrev	0000044e	00000000	00000000	000009dc	2**0
		CONTENTS, READONLY, DEBUGGING				
9	.debug_line	0000001d	00000000	00000000	00000e2a	2**0
		CONTENTS, READONLY, DEBUGGING				
10	.debug_str	0000017a	00000000	00000000	00000e47	2**0
		CONTENTS, READONLY, DEBUGGING				

## Disassembly of section .text:

```
00000000 <__vectors>:
 0: 12 c0      rjmp .+36      ; 0x26 <__ctors_end>
 2: 5e c0      rjmp .+188     ; 0xc0 <__bad_interrupt>
 4: 5d c0      rjmp .+186     ; 0xc0 <__bad_interrupt>
 6: 5c c0      rjmp .+184     ; 0xc0 <__bad_interrupt>
 8: 5b c0      rjmp .+182     ; 0xc0 <__bad_interrupt>
 a: 5a c0      rjmp .+180     ; 0xc0 <__bad_interrupt>
 c: 59 c0      rjmp .+178     ; 0xc0 <__bad_interrupt>
 e: 58 c0      rjmp .+176     ; 0xc0 <__bad_interrupt>
10: 1a c0      rjmp .+52      ; 0x46 <__vector_8>
12: 56 c0      rjmp .+172     ; 0xc0 <__bad_interrupt>
14: 55 c0      rjmp .+170     ; 0xc0 <__bad_interrupt>
16: 54 c0      rjmp .+168     ; 0xc0 <__bad_interrupt>
18: 53 c0      rjmp .+166     ; 0xc0 <__bad_interrupt>
1a: 52 c0      rjmp .+164     ; 0xc0 <__bad_interrupt>
1c: 51 c0      rjmp .+162     ; 0xc0 <__bad_interrupt>
1e: 50 c0      rjmp .+160     ; 0xc0 <__bad_interrupt>
20: 4f c0      rjmp .+158     ; 0xc0 <__bad_interrupt>
22: 4e c0      rjmp .+156     ; 0xc0 <__bad_interrupt>
24: 4d c0      rjmp .+154     ; 0xc0 <__bad_interrupt>

00000026 <__ctors_end>:
26: 11 24      eor r1, r1
28: 1f be      out 0x3f, r1 ; 63
2a: cf e5      ldi r28, 0x5F ; 95
2c: d4 e0      ldi r29, 0x04 ; 4
2e: de bf      out 0x3e, r29 ; 62
30: cd bf      out 0x3d, r28 ; 61

00000032 <__do_clear_bss>:
32: 20 e0      ldi r18, 0x00 ; 0
34: a0 e6      ldi r26, 0x60 ; 96
36: b0 e0      ldi r27, 0x00 ; 0
38: 01 c0      rjmp .+2       ; 0x3c <.do_clear_bss_start>

0000003a <.do_clear_bss_loop>:
3a: 1d 92      st X+, r1

0000003c <.do_clear_bss_start>:
3c: a3 36      cpi r26, 0x63 ; 99
3e: b2 07      cpc r27, r18
40: e1 f7      brne .-8       ; 0x3a <.do_clear_bss_loop>
42: 3f d0      rcall .+126    ; 0xc2 <main>
44: 47 c0      rjmp .+142     ; 0xd4 <exit>
```



```

00000046 <__vector_8>:
#include "iocompat.h" /* Note [1] */

enum { UP, DOWN };

ISR (TIMER1_OVF_vect) /* Note [2] */
{
  46: 1f 92      push r1
  48: 1f b6      in r1, 0x3f ; 63
  4a: 1f 92      push r1
  4c: 11 24      eor r1, r1
  4e: 2f 93      push r18
  50: 8f 93      push r24
  52: 9f 93      push r25
      static uint16_t pwm; /* Note [3] */
      static uint8_t direction;

      switch (direction) /* Note [4] */
  54: 20 91 62 00 lds r18, 0x0062 ; 0x800062 <direction.1>
      {
          case UP:
              if (++pwm == TIMER1_TOP)
  58: 80 91 60 00 lds r24, 0x0060 ; 0x800060 <__DATA_REGION_ORIGIN__>
  5c: 90 91 61 00 lds r25, 0x0061 ; 0x800061 <__DATA_REGION_ORIGIN__+0x1>
              switch (direction) /* Note [4] */
  60: 22 23      and r18, r18
  62: a1 f0      breq .+40      ; 0x8c <__vector_8+0x46>
  64: 21 30      cpi r18, 0x01 ; 1
  66: 49 f4      brne .+18     ; 0x7a <__vector_8+0x34>
                  direction = DOWN;
                  break;

          case DOWN:
              if (--pwm == 0)
  68: 01 97      sbiw r24, 0x01 ; 1
  6a: 90 93 61 00 sts 0x0061, r25 ; 0x800061 <__DATA_REGION_ORIGIN__+0x1>
  6e: 80 93 60 00 sts 0x0060, r24 ; 0x800060 <__DATA_REGION_ORIGIN__>
  72: 00 97      sbiw r24, 0x00 ; 0
  74: 11 f4      brne .+4      ; 0x7a <__vector_8+0x34>
                  direction = UP;
  76: 10 92 62 00 sts 0x0062, r1 ; 0x800062 <direction.1>
                  break;
      }

      OCR = pwm; /* Note [5] */
  7a: 9b bd      out 0x2b, r25 ; 43
  7c: 8a bd      out 0x2a, r24 ; 42
}
  7e: 9f 91      pop r25
  80: 8f 91      pop r24
  82: 2f 91      pop r18
  84: 1f 90      pop r1
  86: 1f be      out 0x3f, r1 ; 63
  88: 1f 90      pop r1
  8a: 18 95      reti

      if (++pwm == TIMER1_TOP)
  8c: 01 96      adiw r24, 0x01 ; 1
  8e: 90 93 61 00 sts 0x0061, r25 ; 0x800061 <__DATA_REGION_ORIGIN__+0x1>
  92: 80 93 60 00 sts 0x0060, r24 ; 0x800060 <__DATA_REGION_ORIGIN__>
  96: 8f 3f      cpi r24, 0xFF ; 255
  98: 23 e0      ldi r18, 0x03 ; 3
  9a: 92 07      cpc r25, r18
  9c: 71 f7      brne .-36     ; 0x7a <__vector_8+0x34>
                  direction = DOWN;
  9e: 21 e0      ldi r18, 0x01 ; 1
  a0: 20 93 62 00 sts 0x0062, r18 ; 0x800062 <direction.1>
  a4: ea cf      rjmp .-44     ; 0x7a <__vector_8+0x34>

000000a6 <iointit>:

void
iointit (void) /* Note [6] */

```

```

{
    /* Timer 1 is 10-bit PWM (8-bit PWM on some ATTinys). */
    TCCR1A = TIMER1_PWM_INIT;
a6: 83 e8          ldi r24, 0x83 ; 131
a8: 8f bd          out 0x2f, r24 ; 47
    * Start timer 1.
    *
    * NB: TCCR1A and TCCR1B could actually be the same register, so
    * take care to not clobber it.
    */
    TCCR1B |= TIMER1_CLOCKSOURCE;
aa: 8e b5          in r24, 0x2e ; 46
ac: 81 60          ori r24, 0x01 ; 1
ae: 8e bd          out 0x2e, r24 ; 46
#if defined(TIMER1_SETUP_HOOK)
    TIMER1_SETUP_HOOK();
#endif

    /* Set PWM value to 0. */
    OCR = 0;
b0: 1b bc          out 0x2b, r1 ; 43
b2: 1a bc          out 0x2a, r1 ; 42

    /* Enable OC1 as output. */
    DDRC = _BV (OC1);
b4: 82 e0          ldi r24, 0x02 ; 2
b6: 87 bb          out 0x17, r24 ; 23

    /* Enable timer 1 overflow interrupt. */
    TIMSK = _BV (TOIE1);
b8: 84 e0          ldi r24, 0x04 ; 4
ba: 89 bf          out 0x39, r24 ; 57
    sei ();
bc: 78 94          sei
}
be: 08 95          ret

000000c0 <__bad_interrupt>:
c0: 9f cf          rjmp .-194      ; 0x0 <__vectors>

000000c2 <main>:

int
main (void)
{

    ioinit ();
c2: f1 df          rcall .-30      ; 0xa6 <ioinit>

    /* loop forever, the interrupts are doing the rest */

    for (;;) /* Note [7] */
        sleep_mode();
c4: 85 b7          in r24, 0x35 ; 53
c6: 80 68          ori r24, 0x80 ; 128
c8: 85 bf          out 0x35, r24 ; 53
ca: 88 95          sleep
cc: 85 b7          in r24, 0x35 ; 53
ce: 8f 77          andi r24, 0x7F ; 127
d0: 85 bf          out 0x35, r24 ; 53
d2: f8 cf          rjmp .-16      ; 0xc4 <main+0x2>

000000d4 <exit>:
d4: f8 94          cli
d6: 00 c0          rjmp .+0      ; 0xd8 <_exit>

000000d8 <_exit>:
d8: f8 94          cli

000000da <__stop_program>:
da: ff cf          rjmp .-2      ; 0xda <__stop_program>

```

## 21.42.5 Linker Map Files

`avr-objdump` is very useful, but sometimes it's necessary to see information about the link that can only be generated by the linker. A map file contains this information. A map file is useful for monitoring the sizes of your code and data. It also shows where modules are loaded and which modules were loaded from libraries. It is yet another view of your application. To get a map file, I usually add `-Wl, -Map, demo.map` to my link command. Relink the application using the following command to generate `demo.map` (a portion of which is shown below).

```
$ avr-gcc -g -mmcu=atmega8 -Wl,-Map,demo.map -o demo.elf demo.o
```

Some points of interest in the `demo.map` file are:

```
.rela.plt
*(.rela.plt)

.text          0x0000000000000000      0xdc
*(.vectors)
.vectors      0x0000000000000000      0x26 /home/joerg/src/avr-libc/avr/devices/atmega8/crtatmega8.o
              0x0000000000000000      __vectors
              0x0000000000000000      __vector_default
*(.vectors)
*(.progmem.gcc*)
              0x0000000000000026      . = ALIGN (0x2)
              0x0000000000000026      __trampolines_start = .
*(.trampolines)
.trampolines  0x0000000000000026      0x0 linker stubs
*(.trampolines*)
              0x0000000000000026      __trampolines_end = .
*libprintf_flt.a:*(.progmem.data)
*libc.a:*(.progmem.data)
*(.progmem.*)
              0x0000000000000026      . = ALIGN (0x2)
*(.lowtext)
*(.lowtext*)
              0x0000000000000026      __ctors_start = .
```

The `.text` segment (where program instructions are stored) starts at location `0x0`.

```
*(.fini2)
*(.fini2)
*(.fini1)
*(.fini1)
*(.fini0)
.fini0        0x00000000000000d8      0x4 /usr/local/lib/gcc/avr/11.2.0/avr4/libgcc.a(_exit.o)
*(.fini0)
*(.hightext)
*(.hightext*)
*(.progmemx.*)
              0x00000000000000dc      . = ALIGN (0x2)
*(.jumptables)
*(.jumptables*)
              0x00000000000000dc      _etext = .

.data         0x0000000000800060      0x0 load address 0x00000000000000dc
              [!provide]
              PROVIDE (__data_start = .)
*(.data)
.data         0x0000000000800060      0x0 demo.o
.data         0x0000000000800060      0x0 /home/joerg/src/avr-libc/avr/lib/avr4/exit.o
.data         0x0000000000800060      0x0 /home/joerg/src/avr-libc/avr/devices/atmega8/crtatmega8.o
.data         0x0000000000800060      0x0 /usr/local/lib/gcc/avr/11.2.0/avr4/libgcc.a(_exit.o)
.data         0x0000000000800060      0x0 /usr/local/lib/gcc/avr/11.2.0/avr4/libgcc.a(_clear_bss.o)
*(.data*)
*(.gnu.linkonce.d*)
*(.rodata)
*(.rodata*)
*(.gnu.linkonce.r*)
              0x0000000000800060      . = ALIGN (0x2)
              0x0000000000800060      _edata = .
              [!provide]
              PROVIDE (__data_end = .)

.bss         0x0000000000800060      0x3
              0x0000000000800060      PROVIDE (__bss_start = .)

*(.bss)
.bss         0x0000000000800060      0x3 demo.o
.bss         0x0000000000800063      0x0 /home/joerg/src/avr-libc/avr/lib/avr4/exit.o
.bss         0x0000000000800063      0x0 /home/joerg/src/avr-libc/avr/devices/atmega8/crtatmega8.o
.bss         0x0000000000800063      0x0 /usr/local/lib/gcc/avr/11.2.0/avr4/libgcc.a(_exit.o)
.bss         0x0000000000800063      0x0 /usr/local/lib/gcc/avr/11.2.0/avr4/libgcc.a(_clear_bss.o)
*(.bss*)
*(COMMON)
              0x0000000000800063      PROVIDE (__bss_end = .)
```

```

                0x00000000000000dc          __data_load_start = LOADADDR (.data)
                0x00000000000000dc          __data_load_end = (__data_load_start + SIZEOF (.data))

.noinit         0x0000000000800063          0x0          PROVIDE (__noinit_start = .)
* (.noinit*)    [!provide]
                [!provide]                PROVIDE (__noinit_end = .)
                0x0000000000800063          _end = .
                [!provide]                PROVIDE (__heap_start = .)

.eeprom        0x0000000000810000          0x0
* (.eeprom*)    0x0000000000810000          __eeprom_end = .

```

The last address in the `.text` segment is location `0x114` (denoted by `_etext`), so the instructions use up 276 bytes of FLASH.

The `.data` segment (where initialized static variables are stored) starts at location `0x60`, which is the first address after the register bank on an ATmega8 processor.

The next available address in the `.data` segment is also location `0x60`, so the application has no initialized data.

The `.bss` segment (where uninitialized data is stored) starts at location `0x60`.

The next available address in the `.bss` segment is location `0x63`, so the application uses 3 bytes of uninitialized data.

The `.eeprom` segment (where EEPROM variables are stored) starts at location `0x0`.

The next available address in the `.eeprom` segment is also location `0x0`, so there aren't any EEPROM variables.

### 21.42.6 Generating Intel Hex Files

We have a binary of the application, but how do we get it into the processor? Most (if not all) programmers will not accept a GNU executable as an input file, so we need to do a little more processing. The next step is to extract portions of the binary and save the information into `.hex` files. The GNU utility that does this is called `avr-objcopy`.

The ROM contents can be pulled from our project's binary and put into the file `demo.hex` using the following command:

```
$ avr-objcopy -j .text -j .data -O ihex demo.elf demo.hex
```

The resulting `demo.hex` file contains:

```

:1000000012C05EC05DC05CC05BC05AC059C058C061
:100010001AC056C055C054C053C052C051C050C081
:100020004FC04EC04DC011241FBECFE5D4E0DEBF8F
:10003000CDBF20E0A0E6B0E001C01D92A336B2071C
:10004000E1F73FD047C01F921FB61F9211242F9394
:100050008F939F9320916200809160009091610046
:100060002223A1F0213049F40197909361008093FD
:100070006000009711F4109262009BBD8ABD9F91B1
:100080008F912F911F901FBE1F90189501969093EE
:100090006100809360008F3F23E0920771F721E0B9
:1000A00020936200EACF83E88FBD8EB581608EBD5C
:1000B0001BBC1ABC82E087BB84E089BF789408959A
:1000C0009FCFF1DF85B7806885BF889585B78F772B
:0C00D00085BFF8CFF89400C0F894FFCF73
:00000001FF

```

The `-j` option indicates that we want the information from the `.text` and `.data` segment extracted. If we specify the EEPROM segment, we can generate a `.hex` file that can be used to program the EEPROM:

```
$ avr-objcopy -j .eeprom --change-section-lma .eeprom=0 -O ihex demo.elf demo_eeprom.hex
```

There is no `demo_eeprom.hex` file written, as that file would be empty.

Starting with version 2.17 of the GNU binutils, the `avr-objcopy` command that used to generate the empty EEPROM files now aborts because of the empty input section `.eeprom`, so these empty files are not generated. It also signals an error to the Makefile which will be caught there, and makes it print a message about the empty file not being generated.

## 21.42.7 Letting Make Build the Project

Rather than type these commands over and over, they can all be placed in a make file. To build the demo project using `make`, save the following in a file called `Makefile`.

### Note

This `Makefile` can only be used as input for the GNU version of `make`.

```

PRG           = demo
OBJ           = demo.o
#MCU_TARGET   = at90s2313
#MCU_TARGET   = at90s2333
#MCU_TARGET   = at90s4414
#MCU_TARGET   = at90s4433
#MCU_TARGET   = at90s4434
#MCU_TARGET   = at90s8515
#MCU_TARGET   = at90s8535
#MCU_TARGET   = atmega128
#MCU_TARGET   = atmega1280
#MCU_TARGET   = atmega1281
#MCU_TARGET   = atmega1284p
#MCU_TARGET   = atmega16
#MCU_TARGET   = atmega163
#MCU_TARGET   = atmega164p
#MCU_TARGET   = atmega165
#MCU_TARGET   = atmega165p
#MCU_TARGET   = atmega168
#MCU_TARGET   = atmega169
#MCU_TARGET   = atmega169p
#MCU_TARGET   = atmega2560
#MCU_TARGET   = atmega2561
#MCU_TARGET   = atmega32
#MCU_TARGET   = atmega324p
#MCU_TARGET   = atmega325
#MCU_TARGET   = atmega3250
#MCU_TARGET   = atmega329
#MCU_TARGET   = atmega3290
#MCU_TARGET   = atmega32u4
#MCU_TARGET   = atmega48
#MCU_TARGET   = atmega64
#MCU_TARGET   = atmega640
#MCU_TARGET   = atmega644
#MCU_TARGET   = atmega644p
#MCU_TARGET   = atmega645
#MCU_TARGET   = atmega6450
#MCU_TARGET   = atmega649
#MCU_TARGET   = atmega6490
MCU_TARGET    = atmega8
#MCU_TARGET   = atmega8515
#MCU_TARGET   = atmega8535
#MCU_TARGET   = atmega88
#MCU_TARGET   = attiny2313
#MCU_TARGET   = attiny24
#MCU_TARGET   = attiny25
#MCU_TARGET   = attiny26
#MCU_TARGET   = attiny261
#MCU_TARGET   = attiny44
#MCU_TARGET   = attiny45
#MCU_TARGET   = attiny461
#MCU_TARGET   = attiny84
#MCU_TARGET   = attiny85
#MCU_TARGET   = attiny861
OPTIMIZE      = -O2

DEFS          =
LIBS          =

# You should not have to change anything below here.

CC            = avr-gcc

# Override is only needed by avr-lib build system.

override CFLAGS      = -g -Wall $(OPTIMIZE) -mmcu=$(MCU_TARGET) $(DEFS)
override LDFLAGS     = -Wl,-Map,$(PRG).map

OBJCOPY       = avr-objcopy
OBJDUMP       = avr-objdump

all: $(PRG).elf lst text eeprom

$(PRG).elf: $(OBJ)

```

```

$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LIBS)

# dependency:
demo.o: demo.c iocompat.h

clean:
    rm -rf *.o $(PRG).elf *.eps *.png *.pdf *.bak
    rm -rf *.lst *.map $(EXTRA_CLEAN_FILES)

lst: $(PRG).lst

%.lst: %.elf
    $(OBJDUMP) -h -S $< > $@

# Rules for building the .text rom images

text: hex bin srec

hex: $(PRG).hex
bin: $(PRG).bin
srec: $(PRG).srec

%.hex: %.elf
    $(OBJCOPY) -j .text -j .data -O ihex $< $@

%.srec: %.elf
    $(OBJCOPY) -j .text -j .data -O srec $< $@

%.bin: %.elf
    $(OBJCOPY) -j .text -j .data -O binary $< $@

# Rules for building the .eeprom rom images

eeprom: ehex ebin esrec

ehex: $(PRG)_eeprom.hex
ebin: $(PRG)_eeprom.bin
esrec: $(PRG)_eeprom.srec

%_eeprom.hex: %.elf
    $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O ihex $< $@ \
    || { echo empty $@ not generated; exit 0; }

%_eeprom.srec: %.elf
    $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O srec $< $@ \
    || { echo empty $@ not generated; exit 0; }

%_eeprom.bin: %.elf
    $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O binary $< $@ \
    || { echo empty $@ not generated; exit 0; }

# Every thing below here is used by avr-libc's build system and can be ignored
# by the casual user.

FIG2DEV          = fig2dev
EXTRA_CLEAN_FILES = *.hex *.bin *.srec

dox: eps png pdf

eps: $(PRG).eps
png: $(PRG).png
pdf: $(PRG).pdf

%.eps: %.fig
    $(FIG2DEV) -L eps $< $@

%.pdf: %.fig
    $(FIG2DEV) -L pdf $< $@

%.png: %.fig
    $(FIG2DEV) -L png $< $@

```

### 21.42.8 Reference to the source code

The source code is installed under

`$prefix/share/doc/avr-libc/examples/demo/`,

where `$prefix` is a configuration option. For Unix systems, it is usually set to either `/usr` or `/usr/local`.

## 21.43 A more sophisticated project

This project extends the basic idea of the [simple project](#) to control a LED with a PWM output, but adds methods to adjust the LED brightness. It employs a lot of the basic concepts of AVR-LibC to achieve that goal.

Understanding this project assumes the simple project has been understood in full, as well as being acquainted with the basic hardware concepts of an AVR microcontroller.

### 21.43.1 Hardware setup

The demo is set up in a way so it can be run on the ATmega16 that ships with the STK500 development kit. The only external part needed is a potentiometer attached to the ADC. It is connected to a 10-pin ribbon cable for port A, both ends of the potentiometer to pins 9 (GND) and 10 (VCC), and the wiper to pin 1 (port A0). A bypass capacitor from pin 1 to pin 9 (like 47 nF) is recommendable.

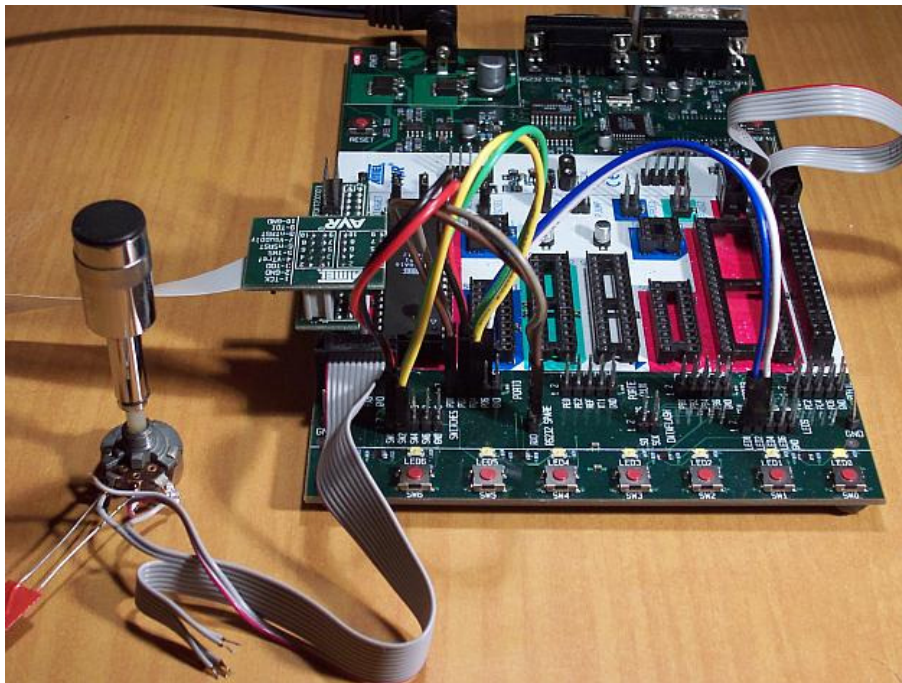


Figure 5 Setup of the STK500

The coloured patch cables are used to provide various interconnections. As there are only four of them in the STK500, there are two options to connect them for this demo. The second option for the yellow-green cable is shown in parenthesis in the table. Alternatively, the "squid" cable from the JTAG ICE kit can be used if available.

Port	Header	Color	Function	Connect to
D0	1	brown	RxD	RXD of the RS-232 header
D1	2	grey	TxD	TXD of the RS-232 header
D2	3	black	button "down"	SW0 (pin 1 switches header)
D3	4	red	button "up"	SW1 (pin 2 switches header)
D4	5	green	button "ADC"	SW2 (pin 3 switches header)
D5	6	blue	LED	LED0 (pin 1 LEDs header)
D6	7	(green)	clock out	LED1 (pin 2 LEDs header)
D7	8	white	1-second flash	LED2 (pin 3 LEDs header)
GND	9		unused	
VCC	10		unused	

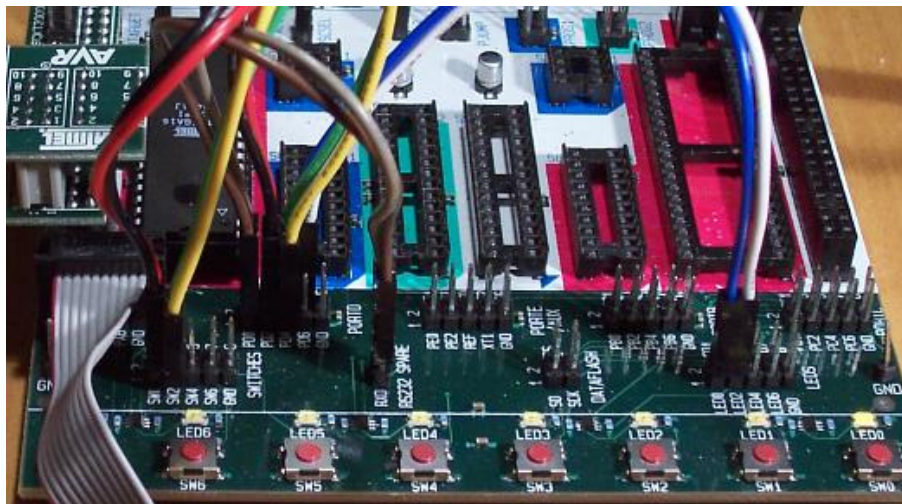


Figure 6 Wiring of the STK500

The following picture shows the alternate wiring where LED1 is connected but SW2 is not:

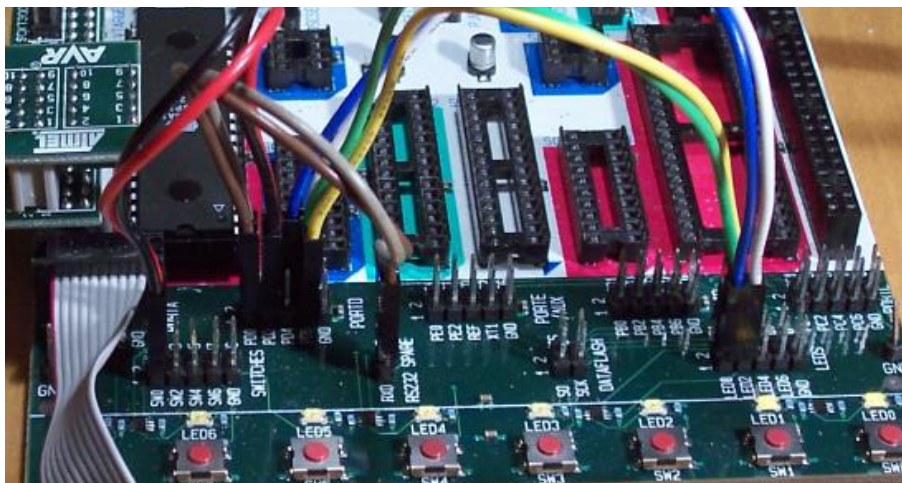


Figure 7 Wiring option #2 of the STK500

As an alternative, this demo can also be run on the popular ATmega8 controller, or its successor ATmega88 as well as the ATmega48 and ATmega168 variants of the latter. These controllers do not have a port named "A", so their ADC inputs are located on port C instead, thus the potentiometer needs to be attached to port C. Likewise, the OC1A output is not on port D pin 5 but on port B pin 1 (PB1). Thus, the above cabling scheme needs to be changed so that PB1 connects to the LED0 pin. (PD6 remains unconnected.) When using the STK500, use one of the jumper cables for this connection. All other port D pins should be connected the same way as described for the ATmega16 above.

When not using an STK500 starter kit, attach the LEDs through some resistor to Vcc (low-active LEDs), and attach pushbuttons from the respective input pins to GND. The internal pull-up resistors are enabled for the pushbutton pins, so no external resistors are needed.

Finally, the demo has been ported to the ATtiny2313 as well. As this AVR does not offer an ADC, everything related to handling the ADC is disabled in the code for that MCU type. Also, port D of this controller type only features 6 pins, so the 1-second flash LED had to be moved from PD6 to PD4. (PD4 is used as the ADC control button on the other MCU types, but that is not needed here.) OC1A is located at PB3 on this device.

The `MCU_TARGET` macro in the Makefile needs to be adjusted appropriately for the alternative controller types.



The flash ROM and RAM consumption of this demo are way below the resources of even an ATmega48, and still well within the capabilities of an ATtiny2313. The major advantage of experimenting with the ATmega16 (in addition that it ships together with an STK500 anyway) is that it can be debugged online via JTAG. Likewise, the ATmega48/88/168 and ATtiny2313 devices can be debugged through debugWire, using the Atmel JTAG ICE mkII or the low-cost AVR Dragon.

Note that in the explanation below, all port/pin names are applicable to the ATmega16 setup.

### 21.43.2 Functional overview

PD6 will be toggled with each internal clock tick (approx. 10 ms). PD7 will flash once per second.

PD0 and PD1 are configured as UART IO, and can be used to connect the demo kit to a PC (9600 Bd, 8N1 frame format). The demo application talks to the serial port, and it can be controlled from the serial port.

PD2 through PD4 are configured as inputs, and control the application unless control has been taken over by the serial port. Shorting PD2 to GND will decrease the current PWM value, shorting PD3 to GND will increase it.

While PD4 is shorted to GND, one ADC conversion for channel 0 (ADC input is on PA0) will be triggered each internal clock tick, and the resulting value will be used as the PWM value. So the brightness of the LED follows the analog input value on PC0. VAREF on the STK500 should be set to the same value as VCC.

When running in serial control mode, the function of the watchdog timer can be demonstrated by typing an `r'. This will make the demo application run in a tight loop without retriggering the watchdog so after some seconds, the watchdog will reset the MCU. This situation can be figured out on startup by reading the MCUCSR register.

The current value of the PWM is backed up in an EEPROM cell after about 3 seconds of idle time after the last change. If that EEPROM cell contains a reasonable (i. e. non-erased) value at startup, it is taken as the initial value for the PWM. This virtually preserves the last value across power cycles. By not updating the EEPROM immediately but only after a timeout, EEPROM wear is reduced considerably compared to immediately writing the value at each change.

### 21.43.3 A code walkthrough

This section explains the ideas behind individual parts of the code. The [source code](#) has been divided into numbered parts, and the following subsections explain each of these parts.

**21.43.3.1 Part 1: Macro definitions** A number of preprocessor macros are defined to improve readability and/or portability of the application.

The first macros describe the IO pins our LEDs and pushbuttons are connected to. This provides some kind of mini-HAL (hardware abstraction layer) so should some of the connections be changed, they don't need to be changed inside the code but only on top. Note that the location of the PWM output itself is mandated by the hardware, so it cannot be easily changed. As the ATmega48/88/168 controllers belong to a more recent generation of AVRs, a number of register and bit names have been changed there, so they are mapped back to their ATmega8/16 equivalents to keep the actual program code portable.

The name `F_CPU` is the conventional name to describe the CPU clock frequency of the controller. This demo project just uses the internal calibrated 1 MHz RC oscillator that is enabled by default. Note that when using the `<util/delay.h>` functions, `F_CPU` needs to be defined before including that file.

The remaining macros have their own comments in the source code. The macro `TMR1_SCALE` shows how to use the preprocessor and the compiler's constant expression computation to calculate the value of timer 1's post-scaler in a way so it only depends on `F_CPU` and the desired software clock frequency. While the formula looks a bit complicated, using a macro offers the advantage that the application will automatically scale to new target softclock or master CPU frequencies without having to manually re-calculate hardcoded constants.

**21.43.3.2 Part 2: Variable definitions** The `intflgs` structure demonstrates a way to allocate bit variables in memory. Each of the interrupt service routines just sets one bit within that structure, and the application's main loop then monitors the bits in order to act appropriately.

Like all variables that are used to communicate values between an interrupt service routine and the main application, it is declared `volatile`.

The variable `ee_pwm` is not a variable in the classical C sense that could be used as an lvalue or within an expression to obtain its value. Instead, the

```
__attribute__((section(".eeprom")))
```

marks it as belonging to the `EEPROM` section. This section is merely used as a placeholder so the compiler can arrange for each individual variable's location in EEPROM. The compiler will also keep track of initial values assigned, and usually the Makefile is arranged to extract these initial values into a separate load file (`largedemo ← _eeprom.*` in this case) that can be used to initialize the EEPROM.

The actual EEPROM IO must be performed manually.

Similarly, the variable `mcucsr` is kept in the `.noinit` section in order to prevent it from being cleared upon application startup.

**21.43.3.3 Part 3: Interrupt service routines** The ISR to handle timer 1's overflow interrupt arranges for the software clock. While timer 1 runs the PWM, it calls its overflow handler rather frequently, so the `TMR1_SCALE` value is used as a postscaler to reduce the internal software clock frequency further. If the software clock triggers, it sets the `tmr_int` bitfield, and defers all further tasks to the main loop.

The ADC ISR just fetches the value from the ADC conversion, disables the ADC interrupt again, and announces the presence of the new value in the `adc_int` bitfield. The interrupt is kept disabled while not needed, because the ADC will also be triggered by executing the `SLEEP` instruction in idle mode (which is the default sleep mode). Another option would be to turn off the ADC completely here, but that increases the ADC's startup time (not that it would matter much for this application).

**21.43.3.4 Part 4: Auxiliary functions** The function `handle_mcucsr()` uses two `__attribute__ ←` declarators to achieve specific goals. First, it will instruct the compiler to place the generated code into the `.init3` section of the output. Thus, it will become part of the application initialization sequence. This is done in order to fetch (and clear) the reason of the last hardware reset from `MCUCSR` as early as possible. There is a short period of time where the next reset could already trigger before the current reason has been evaluated. This also explains why the variable `mcucsr` that mirrors the register's value needs to be placed into the `.noinit` section, because otherwise the default initialization (which happens after `.init3`) would blank the value again.

As the initialization code is not called using `CALL/RET` instructions but rather concatenated together, the compiler needs to be instructed to omit the entire function prologue and epilogue. This is performed by the `naked` attribute. So while syntactically, `handle_mcucsr()` is a function to the compiler, the compiler will just emit the instructions for it without setting up any stack frame, and not even a `RET` instruction at the end.

Function `ioinit()` centralizes all hardware setup. The very last part of that function demonstrates the use of the EEPROM variable `ee_pwm` to obtain an EEPROM address that can in turn be applied as an argument to `eeprom_read_word()`.

The following functions handle UART character and string output. (UART input is handled by an ISR.) There are two string output functions, `printstr()` and `printstr_p()`. The latter function fetches the string from `program memory`. Both functions translate a newline character into a carriage return/newline sequence, so a simple `\n` can be used in the source code.

The function `set_pwm()` propagates the new PWM value to the PWM, performing range checking. When the value has been changed, the new percentage will be announced on the serial link. The current value is mirrored in the variable `pwm` so others can use it in calculations. In order to allow for a simple calculation of a percentage value without requiring floating-point mathematics, the maximal value of the PWM is restricted to 1000 rather than 1023, so a simple division by 10 can be used. Due to the nature of the human eye, the difference in LED brightness between 1000 and 1023 is not noticeable anyway.

**21.43.3.5 Part 5: main()** At the start of `main()`, a variable `mode` is declared to keep the current mode of operation. An enumeration is used to improve the readability. By default, the compiler would allocate a variable of type `int` for an enumeration. The `packed` attribute declarator instructs the compiler to use the smallest possible integer type (which would be an 8-bit type here).

After some initialization actions, the application's main loop follows. In an embedded application, this is normally an infinite loop as there is nothing an application could "exit" into anyway.

At the beginning of the loop, the watchdog timer will be retriggered. If that timer is not triggered for about 2 seconds, it will issue a hardware reset. Care needs to be taken that no code path blocks longer than this, or it needs to frequently perform watchdog resets of its own. An example of such a code path would be the string IO functions: for an overly large string to print (about 2000 characters at 9600 Bd), they might block for too long.

The loop itself then acts on the interrupt indication bitfields as appropriate, and will eventually put the CPU on sleep at its end to conserve power.

The first interrupt bit that is handled is the (software) timer, at a frequency of approximately 100 Hz. The `CLOCKOUT` pin will be toggled here, so e. g. an oscilloscope can be used on that pin to measure the accuracy of our software clock. Then, the LED flasher for LED2 ("We are alive"-LED) is built. It will flash that LED for about 50 ms, and pause it for another 950 ms. Various actions depending on the operation mode follow. Finally, the 3-second backup timer is implemented that will write the PWM value back to EEPROM once it is not changing anymore.

The ADC interrupt will just adjust the PWM value only.

Finally, the UART Rx interrupt will dispatch on the last character received from the UART.

All the string literals that are used as informational messages within `main()` are placed in [program memory](#) so no SRAM needs to be allocated for them. This is done by using the `PSTR` macro, and passing the string to `printstr_p()`.

#### 21.43.4 The source code

The source code is installed under

```
$prefix/share/doc/avr-libc/examples/largedemo/largedemo.c,
```

where `$prefix` is a configuration option. For Unix systems, it is usually set to either `/usr` or `/usr/local`.

## 21.44 Using the standard IO facilities

This project illustrates how to use the standard IO facilities (`stdio`) provided by this library. It assumes a basic knowledge of how the `stdio` subsystem is used in standard C applications, and concentrates on the differences in this library's implementation that mainly result from the differences of the microcontroller environment, compared to a hosted environment of a standard computer.

This demo is meant to supplement the [documentation](#), not to replace it.

### 21.44.1 Hardware setup

The demo is set up in a way so it can be run on the ATmega16 that ships with the STK500 development kit. The UART port needs to be connected to the RS-232 "spare" port by a jumper cable that connects PD0 to RxD and PD1 to TxD. The RS-232 channel is set up as standard input (`stdin`) and standard output (`stdout`), respectively.

In order to have a different device available for a standard error channel (`stderr`), an industry-standard LCD display with an HD44780-compatible LCD controller has been chosen. This display needs to be connected to port A of the STK500 in the following way:

Port	Header	Function
A0	1	LCD D4
A1	2	LCD D5
A2	3	LCD D6
A3	4	LCD D7
A4	5	LCD R/~W
A5	6	LCD E
A6	7	LCD RS
A7	8	unused
GND	9	GND
VCC	10	Vcc

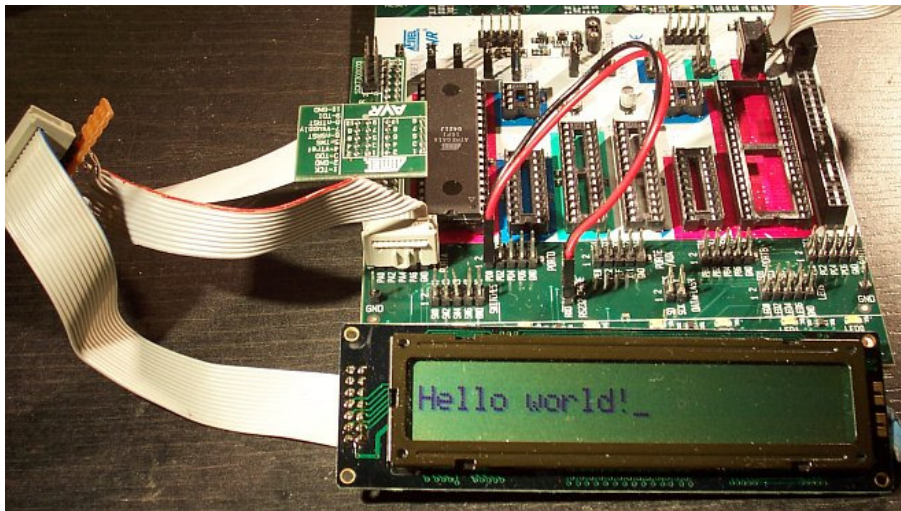


Figure 8 Wiring of the STK500

The LCD controller is used in 4-bit mode, including polling the "busy" flag so the R/~W line from the LCD controller needs to be connected. Note that the LCD controller has yet another supply pin that is used to adjust the LCD's contrast (V5). Typically, that pin connects to a potentiometer between Vcc and GND. Often, it might work to just connect that pin to GND, while leaving it unconnected usually yields an unreadable display.

Port A has been chosen as 7 pins are needed to connect the LCD, yet all other ports are already partially in use: port B has the pins for in-system programming (ISP), port C has the ports for JTAG (can be used for debugging), and port D is used for the UART connection.

### 21.44.2 Functional overview

The project consists of the following files:

- `stdiodemo.c` This is the main example file.
- `defines.h` Contains some global defines, like the LCD wiring
- `hd44780.c` Implementation of an HD44780 LCD display driver
- `hd44780.h` Interface declarations for the HD44780 driver
- `lcd.c` Implementation of LCD character IO on top of the HD44780 driver
- `lcd.h` Interface declarations for the LCD driver
- `uart.c` Implementation of a character IO driver for the internal UART
- `uart.h` Interface declarations for the UART driver

### 21.44.3 A code walkthrough

**21.44.3.1 stdiodemo.c** As usual, include files go first. While conventionally, system header files (those in angular brackets `< ... >`) go before application-specific header files (in double quotes), `defines.h` comes as the first header file here. The main reason is that this file defines the value of `F_CPU` which needs to be known before including `<utils/delay.h>`.

The function `ioinit()` summarizes all hardware initialization tasks. As this function is declared to be module-internal only (`static`), the compiler will notice its simplicity, and with a reasonable optimization level in effect, it will inline that function. That needs to be kept in mind when debugging, because the inlining might cause the debugger to "jump around wildly" at a first glance when single-stepping.

The definitions of `uart_str` and `lcd_str` set up two stdio streams. The initialization is done using the `FDEV_SETUP_STREAM()` initializer template macro, so a static object can be constructed that can be used for IO purposes. This initializer macro takes three arguments, two function macros to connect the corresponding output and input functions, respectively, the third one describes the intent of the stream (read, write, or both). Those functions that are not required by the specified intent (like the input function for `lcd_str` which is specified to only perform output operations) can be given as `NULL`.

The stream `uart_str` corresponds to input and output operations performed over the RS-232 connection to a terminal (e.g. from/to a PC running a terminal program), while the `lcd_str` stream provides a method to display character data on the LCD text display.

The function `delay_1s()` suspends program execution for approximately one second. This is done using the `_delay_ms()` function from `<util/delay.h>` which in turn needs the `F_CPU` macro in order to adjust the cycle counts. As the `_delay_ms()` function has a limited range of allowable argument values (depending on `F_CPU`), a value of 10 ms has been chosen as the base delay which would be safe for CPU frequencies of up to about 26 MHz. This function is then called 100 times to accommodate for the actual one-second delay.

In a practical application, long delays like this one were better be handled by a hardware timer, so the main CPU would be free for other tasks while waiting, or could be put on sleep.

At the beginning of `main()`, after initializing the peripheral devices, the default stdio streams `stdin`, `stdout`, and `stderr` are set up by using the existing static `FILE` stream objects. While this is not mandatory, the availability of `stdin` and `stdout` allows to use the shorthand functions (e.g. `printf()` instead of `fprintf()`), and `stderr` can mnemonically be referred to when sending out diagnostic messages.

Just for demonstration purposes, `stdin` and `stdout` are connected to a stream that will perform UART IO, while `stderr` is arranged to output its data to the LCD text display.

Finally, a main loop follows that accepts simple "commands" entered via the RS-232 connection, and performs a few simple actions based on the commands.

First, a prompt is sent out using `printf_P()` (which takes a [program space string](#)). The string is read into an internal buffer as one line of input, using `fgets()`. While it would be also possible to use `gets()` (which implicitly reads from `stdin`), `gets()` has no control that the user's input does not overflow the input buffer provided so it should never be used at all.

If `fgets()` fails to read anything, the main loop is left. Of course, normally the main loop of a microcontroller application is supposed to never finish, but again, for demonstrational purposes, this explains the error handling of stdio. `fgets()` will return `NULL` in case of an input error or end-of-file condition on input. Both these conditions are in the domain of the function that is used to establish the stream, `uart_putchar()` in this case. In short, this function returns EOF in case of a serial line "break" condition (extended start condition) has been recognized on the serial line. Common PC terminal programs allow to assert this condition as some kind of out-of-band signalling on an RS-232 connection.

When leaving the main loop, a goodbye message is sent to standard error output (i.e. to the LCD), followed by three dots in one-second spacing, followed by a sequence that will clear the LCD. Finally, `main()` will be terminated, and the library will add an infinite loop, so only a CPU reset will be able to restart the application.

There are three "commands" recognized, each determined by the first letter of the line entered (converted to lower case):

- The 'q' (quit) command has the same effect of leaving the main loop.
- The 'l' (LCD) command takes its second argument, and sends it to the LCD.
- The 'u' (UART) command takes its second argument, and sends it back to the UART connection.

Command recognition is done using `sscanf()` where the first format in the format string just skips over the command itself (as the assignment suppression modifier `*` is given).

**21.44.3.2 defines.h** This file just contains a few peripheral definitions.

The `F_CPU` macro defines the CPU clock frequency, to be used in delay loops, as well as in the UART baud rate calculation.

The macro `UART_BAUD` defines the RS-232 baud rate. Depending on the actual CPU frequency, only a limited range of baud rates can be supported.

The remaining macros customize the IO port and pins used for the HD44780 LCD driver. Each definition consists of a letter naming the port this pin is attached to, and a respective bit number. For accessing the data lines, only the first data line gets its own macro (line D4 on the HD44780, lines D0 through D3 are not used in 4-bit mode), all other data lines are expected to be in ascending order next to D4.

**21.44.3.3 hd44780.h** This file describes the public interface of the low-level LCD driver that interfaces to the HD44780 LCD controller. Public functions are available to initialize the controller into 4-bit mode, to wait for the controller's busy bit to be clear, and to read or write one byte from or to the controller.

As there are two different forms of controller IO, one to send a command or receive the controller status (RS signal clear), and one to send or receive data to/from the controller's SRAM (RS asserted), macros are provided that build on the mentioned function primitives.

Finally, macros are provided for all the controller commands to allow them to be used symbolically. The HD44780 datasheet explains these basic functions of the controller in more detail.

**21.44.3.4 hd44780.c** This is the implementation of the low-level HD44780 LCD controller driver.

On top, a few preprocessor glueing tricks are used to establish symbolic access to the hardware port pins the LCD controller is attached to, based on the application's definitions made in [defines.h](#).

The `hd44780_pulse_e()` function asserts a short pulse to the controller's E (enable) pin. Since reading back the data asserted by the LCD controller needs to be performed while E is active, this function reads and returns the input data if the parameter `readback` is true. When called with a compile-time constant parameter that is false, the compiler will completely eliminate the unused readback operation, as well as the return value as part of its optimizations.

As the controller is used in 4-bit interface mode, all byte IO to/from the controller needs to be handled as two nibble IOs. The functions `hd44780_outnibble()` and `hd44780_innibble()` implement this. They do not belong to the public interface, so they are declared static.

Building upon these, the public functions `hd44780_outbyte()` and `hd44780_inbyte()` transfer one byte to/from the controller.

The function `hd44780_wait_ready()` waits for the controller to become ready, by continuously polling the controller's status (which is read by performing a byte read with the RS signal cleared), and examining the BUSY flag within the status byte. This function needs to be called before performing any controller IO.

Finally, `hd44780_init()` initializes the LCD controller into 4-bit mode, based on the initialization sequence mandated by the datasheet. As the BUSY flag cannot be examined yet at this point, this is the only part of this code where timed delays are used. While the controller can perform a power-on reset when certain constraints on the power supply rise time are met, always calling the software initialization routine at startup ensures the controller will be in a known state. This function also puts the interface into 4-bit mode (which would not be done automatically after a power-on reset).

**21.44.3.5 `lcd.h`** This function declares the public interface of the higher-level (character IO) LCD driver.

**21.44.3.6 `lcd.c`** The implementation of the higher-level LCD driver. This driver builds on top of the HD44780 low-level LCD controller driver, and offers a character IO interface suitable for direct use by the standard IO facilities. Where the low-level HD44780 driver deals with setting up controller SRAM addresses, writing data to the controller's SRAM, and controlling display functions like clearing the display, or moving the cursor, this high-level driver allows to just write a character to the LCD, in the assumption this will somehow show up on the display.

Control characters can be handled at this level, and used to perform specific actions on the LCD. Currently, there is only one control character that is being dealt with: a newline character (`\n`) is taken as an indication to clear the display and set the cursor into its initial position upon reception of the next character, so a "new line" of text can be displayed. Therefore, a received newline character is remembered until more characters have been sent by the application, and will only then cause the display to be cleared before continuing. This provides a convenient abstraction where full lines of text can be sent to the driver, and will remain visible at the LCD until the next line is to be displayed.

Further control characters could be implemented, e. g. using a set of escape sequences. That way, it would be possible to implement self-scrolling display lines etc.

The public function `lcd_init()` first calls the initialization entry point of the lower-level HD44780 driver, and then sets up the LCD in a way we'd like to (display cleared, non-blinking cursor enabled, SRAM addresses are increasing so characters will be written left to right).

The public function `lcd_putchar()` takes arguments that make it suitable for being passed as a `put()` function pointer to the stdio stream initialization functions and macros (`fdevopen()`, `FDEV_SETUP_STREAM()` etc.). Thus, it takes two arguments, the character to display itself, and a reference to the underlying stream object, and it is expected to return 0 upon success.

This function remembers the last unprocessed newline character seen in the function-local static variable `nl_↔seen`. If a newline character is encountered, it will simply set this variable to a true value, and return to the caller. As soon as the first non-newline character is to be displayed with `nl_↔seen` still true, the LCD controller is told to clear the display, put the cursor home, and restart at SRAM address 0. All other characters are sent to the display.

The single static function-internal variable `nl_↔seen` works for this purpose. If multiple LCDs should be controlled using the same set of driver functions, that would not work anymore, as a way is needed to distinguish between the various displays. This is where the second parameter can be used, the reference to the stream itself: instead of keeping the state inside a private variable of the function, it can be kept inside a private object that is attached to the stream itself. A reference to that private object can be attached to the stream (e.g. inside the function `lcd_↔init()` that then also needs to be passed a reference to the stream) using `fdev_set_udata()`, and can be accessed inside `lcd_putchar()` using `fdev_get_udata()`.

**21.44.3.7 `uart.h`** Public interface definition for the RS-232 UART driver, much like in `lcd.h` except there is now also a character input function available.

As the RS-232 input is line-buffered in this example, the macro `RX_BUFSIZE` determines the size of that buffer.

**21.44.3.8 `uart.c`** This implements an stdio-compatible RS-232 driver using an AVR's standard UART (or USART in asynchronous operation mode). Both, character output as well as character input operations are implemented. Character output takes care of converting the internal newline `\n` into its external representation carriage return/line feed (`\r\n`).

Character input is organized as a line-buffered operation that allows to minimally edit the current line until it is "sent" to the application when either a carriage return (`\r`) or newline (`\n`) character is received from the terminal. The line editing functions implemented are:

- `\b` (back space) or `\177` (delete) deletes the previous character
- `^u` (control-U, ASCII NAK) deletes the entire input buffer
- `^w` (control-W, ASCII ETB) deletes the previous input word, delimited by white space
- `^r` (control-R, ASCII DC2) sends a `\r`, then reprints the buffer (refresh)
- `\t` (tabulator) will be replaced by a single space

The function `uart_init()` takes care of all hardware initialization that is required to put the UART into a mode with 8 data bits, no parity, one stop bit (commonly referred to as 8N1) at the baud rate configured in `defines.h`. At low CPU clock frequencies, the `U2X` bit in the UART is set, reducing the oversampling from 16x to 8x, which allows for a 9600 Bd rate to be achieved with tolerable error using the default 1 MHz RC oscillator.

The public function `uart_putchar()` again has suitable arguments for direct use by the `stdio` stream interface. It performs the `\n` into `\r\n` translation by recursively calling itself when it sees a `\n` character. Just for demonstration purposes, the `\a` (audible bell, ASCII BEL) character is implemented by sending a string to `stderr`, so it will be displayed on the LCD.

The public function `uart_getchar()` implements the line editor. If there are characters available in the line buffer (variable `rxp` is not `NULL`), the next character will be returned from the buffer without any UART interaction.

If there are no characters inside the line buffer, the input loop will be entered. Characters will be read from the UART, and processed accordingly. If the UART signalled a framing error (`FE` bit set), typically caused by the terminal sending a *line break* condition (start condition held much longer than one character period), the function will return an end-of-file condition using `_FDEV_EOF`. If there was a data overrun condition on input (`DOR` bit set), an error condition will be returned as `_FDEV_ERR`.

Line editing characters are handled inside the loop, potentially modifying the buffer status. If characters are attempted to be entered beyond the size of the line buffer, their reception is refused, and a `\a` character is sent to the terminal. If a `\r` or `\n` character is seen, the variable `rxp` (receive pointer) is set to the beginning of the buffer, the loop is left, and the first character of the buffer will be returned to the application. (If no other characters have been entered, this will just be the newline character, and the buffer is marked as being exhausted immediately again.)

#### 21.44.4 The source code

The source code is installed under

```
$prefix/share/doc/avr-libc/examples/stdiodemo/,
```

where `$prefix` is a configuration option. For Unix systems, it is usually set to either `/usr` or `/usr/local`.

## 21.45 Example using the two-wire interface (TWI)

Some newer devices of the ATmega series contain builtin support for interfacing the microcontroller to a two-wire bus, called TWI. This is essentially the same called I2C by Philips, but that term is avoided in Atmel's documentation due to patenting issues.

For further documentation, see:

[http://www.nxp.com/documents/user\\_manual/UM10204.pdf](http://www.nxp.com/documents/user_manual/UM10204.pdf)



### 21.45.1 Introduction into TWI

The two-wire interface consists of two signal lines named *SDA* (serial data) and *SCL* (serial clock) (plus a ground line, of course). All devices participating in the bus are connected together, using open-drain driver circuitry, so the wires must be terminated using appropriate pullup resistors. The pullups must be small enough to recharge the line capacity in short enough time compared to the desired maximal clock frequency, yet large enough so all drivers will not be overloaded. There are formulas in the datasheet that help selecting the pullups.

Devices can either act as a master to the bus (i. e., they initiate a transfer), or as a slave (they only act when being called by a master). The bus is multi-master capable, and a particular device implementation can act as either master or slave at different times. Devices are addressed using a 7-bit address (coordinated by Philips) transferred as the first byte after the so-called start condition. The LSB of that byte is R/~W, i. e. it determines whether the request to the slave is to read or write data during the next cycles. (There is also an option to have devices using 10-bit addresses but that is not covered by this example.)

### 21.45.2 The TWI example project

The ATmega TWI hardware supports both, master and slave operation. This example will only demonstrate how to use an AVR microcontroller as TWI master. The implementation is kept simple in order to concentrate on the steps that are required to talk to a TWI slave, so all processing is done in polled-mode, waiting for the TWI interface to indicate that the next processing step is due (by setting the TWINT interrupt bit). If it is desired to have the entire TWI communication happen in "background", all this can be implemented in an interrupt-controlled way, where only the start condition needs to be triggered from outside the interrupt routine.

There is a variety of slave devices available that can be connected to a TWI bus. For the purpose of this example, an EEPROM device out of the industry-standard **24Cxx** series has been chosen (where xx can be one of **01**, **02**, **04**, **08**, or **16**) which are available from various vendors. The choice was almost arbitrary, mainly triggered by the fact that an EEPROM device is being talked to in both directions, reading and writing the slave device, so the example will demonstrate the details of both.

Usually, there is probably not much need to add more EEPROM to an ATmega system that way: the smallest possible AVR device that offers hardware TWI support is the ATmega8 which comes with 512 bytes of EEPROM, which is equivalent to an 24C04 device. The ATmega128 already comes with twice as much EEPROM as the 24C16 would offer. One exception might be to use an externally connected EEPROM device that is removable; e. g. SDRAM PC memory comes with an integrated TWI EEPROM that carries the RAM configuration information.

### 21.45.3 The Source Code

The source code is installed under

```
$prefix/share/doc/avr-libc/examples/twitest/twitest.c,
```

where `$prefix` is a configuration option. For Unix systems, it is usually set to either `/usr` or `/usr/local`.

#### Note [1]

The header file `<util/twi.h>` contains some macro definitions for symbolic constants used in the TWI status register. These definitions match the names used in the Atmel datasheet except that all names have been prefixed with `TW_`.

**Note [2]**

The clock is used in timer calculations done by the compiler, for the UART baud rate and the TWI clock rate.

**Note [3]**

The address assigned for the 24Cxx EEPROM consists of 1010 in the upper four bits. The following three bits are normally available as slave sub-addresses, allowing to operate more than one device of the same type on a single bus, where the actual subaddress used for each device is configured by hardware strapping. However, since the next data packet following the device selection only allows for 8 bits that are used as an EEPROM address, devices that require more than 8 address bits (24C04 and above) "steal" subaddress bits and use them for the EEPROM cell address bits 9 to 11 as required. This example simply assumes all subaddress bits are 0 for the smaller devices, so the E0, E1, and E2 inputs of the 24Cxx must be grounded.

**Note [3a]**

EEPROMs of type 24C32 and above cannot be addressed anymore even with the subaddress bit trick. Thus, they require the upper address bits being sent separately on the bus. When activating the `WORD_ADDRESS_16BIT` define, the algorithm implements that auxiliary address byte transmission.

**Note [4]**

For slow clocks, enable the 2 x U[S]ART clock multiplier, to improve the baud rate error. This will allow a 9600 Bd communication using the standard 1 MHz calibrated RC oscillator. See also the Baud rate tables in the datasheets.

**Note [5]**

The datasheet explains why a minimum TWBR value of 10 should be maintained when running in master mode. Thus, for system clocks below 3.6 MHz, we cannot run the bus at the intended clock rate of 100 kHz but have to slow down accordingly.

**Note [6]**

This function is used by the standard output facilities that are utilized in this example for debugging and demonstration purposes.

**Note [7]**

In order to shorten the data to be sent over the TWI bus, the 24Cxx EEPROMs support multiple data bytes transferred within a single request, maintaining an internal address counter that is updated after each data byte transferred successfully. When reading data, one request can read the entire device memory if desired (the counter would wrap around and start back from 0 when reaching the end of the device).

**Note [8]**

When reading the EEPROM, a first device selection must be made with write intent ( $R/\sim W$  bit set to 0 indicating a write operation) in order to transfer the EEPROM address to start reading from. This is called *master transmitter mode*. Each completion of a particular step in TWI communication is indicated by an asserted TWINT bit in TWCR. (An interrupt would be generated if allowed.) After performing any actions that are needed for the next communication step, the interrupt condition must be manually cleared by *setting* the TWINT bit. Unlike with many other interrupt sources, this would even be required when using a true interrupt routine, since as soon as TWINT is re-asserted, the next bus transaction will start.

**Note [9]**

Since the TWI bus is multi-master capable, there is potential for a bus contention when one master starts to access the bus. Normally, the TWI bus interface unit will detect this situation, and will not initiate a start condition while the bus is busy. However, in case two masters were starting at exactly the same time, the way bus arbitration works, there is always a chance that one master could lose arbitration of the bus during any transmit operation. A master that has lost arbitration is required by the protocol to immediately cease talking on the bus; in particular it must not initiate a stop condition in order to not corrupt the ongoing transfer from the active master. In this example, upon detecting a lost arbitration condition, the entire transfer is going to be restarted. This will cause a new start condition to be initiated, which will normally be delayed until the currently active master has released the bus.

**Note [10]**

Next, the device slave is going to be reselected (using a so-called repeated start condition which is meant to guarantee that the bus arbitration will remain at the current master) using the same slave address (SLA), but this time with read intent ( $R/\sim W$  bit set to 1) in order to request the device slave to start transferring data from the slave to the master in the next packet.

**Note [11]**

If the EEPROM device is still busy writing one or more cells after a previous write request, it will simply leave its bus interface drivers at high impedance, and does not respond to a selection in any way at all. The master selecting the device will see the high level at SDA after transferring the SLA+R/W packet as a NACK to its selection request. Thus, the select process is simply started over (effectively causing a *repeated start condition*), until the device will eventually respond. This polling procedure is recommended in the 24Cxx datasheet in order to minimize the busy wait time when writing. Note that in case a device is broken and never responds to a selection (e. g. since it is no longer present at all), this will cause an infinite loop. Thus the maximal number of iterations made until the device is declared to be not responding at all, and an error is returned, will be limited to MAX\_ITER.

**Note [12]**

This is called *master receiver mode*: the bus master still supplies the SCL clock, but the device slave drives the SDA line with the appropriate data. After 8 data bits, the master responds with an ACK bit (SDA driven low) in order to request another data transfer from the slave, or it can leave the SDA line high (NACK), indicating to the slave that it is going to stop the transfer now. Assertion of ACK is handled by setting the TWEA bit in TWCR when starting the current transfer.

**Note [13]**

The control word sent out in order to initiate the transfer of the next data packet is initially set up to assert the TWEA bit. During the last loop iteration, TWEA is de-asserted so the client will get informed that no further transfer is desired.

**Note [14]**

Except in the case of lost arbitration, all bus transactions must properly be terminated by the master initiating a stop condition.

**Note [15]**

Writing to the EEPROM device is simpler than reading, since only a master transmitter mode transfer is needed. Note that the first packet after the SLA+W selection is always considered to be the EEPROM address for the next operation. (This packet is exactly the same as the one above sent before starting to read the device.) In case a master transmitter mode transfer is going to send more than one data packet, all following packets will be considered data bytes to write at the indicated address. The internal address pointer will be incremented after each write operation.

**Note [16]**

24Cxx devices can become write-protected by strapping their  $\sim$ WC pin to logic high. (Leaving it unconnected is explicitly allowed, and constitutes logic low level, i. e. no write protection.) In case of a write protected device, all data transfer attempts will be NACKed by the device. Note that some devices might not implement this.

## 22 Data Structure Documentation

### 22.1 div\_t Struct Reference

```
#include <stdlib.h>
```

#### Data Fields

- int [quot](#)
- int [rem](#)

#### 22.1.1 Detailed Description

Result type for function [div\(\)](#).

## 22.1.2 Field Documentation

### 22.1.2.1 `quot` `int div_t::quot`

The Quotient.

### 22.1.2.2 `rem` `int div_t::rem`

The Remainder.

The documentation for this struct was generated from the following file:

- [stdlib.h](#)

## 22.2 `ldiv_t` Struct Reference

```
#include <stdlib.h>
```

### Data Fields

- long `quot`
- long `rem`

### 22.2.1 Detailed Description

Result type for function `ldiv()`.

### 22.2.2 Field Documentation

#### 22.2.2.1 `quot` `long ldiv_t::quot`

The Quotient.

#### 22.2.2.2 `rem` `long ldiv_t::rem`

The Remainder.

The documentation for this struct was generated from the following file:

- [stdlib.h](#)

## 22.3 tm Struct Reference

```
#include <time.h>
```

### Data Fields

- [int8\\_t tm\\_sec](#)
- [int8\\_t tm\\_min](#)
- [int8\\_t tm\\_hour](#)
- [int8\\_t tm\\_mday](#)
- [int8\\_t tm\\_wday](#)
- [int8\\_t tm\\_mon](#)
- [int16\\_t tm\\_year](#)
- [int16\\_t tm\\_yday](#)
- [int16\\_t tm\\_isdst](#)

### 22.3.1 Detailed Description

The tm structure contains a representation of time 'broken down' into components of the Gregorian calendar.

The value of tm\_isdst is zero if Daylight Saving Time is not in effect, and is negative if the information is not available.

When Daylight Saving Time is in effect, the value represents the number of seconds the clock is advanced.

See the [set\\_dst\(\)](#) function for more information about Daylight Saving.

### 22.3.2 Field Documentation

#### 22.3.2.1 tm\_hour [int8\\_t](#) tm::tm\_hour

hours since midnight - [ 0 to 23 ]

#### 22.3.2.2 tm\_isdst [int16\\_t](#) tm::tm\_isdst

Daylight Saving Time flag

#### 22.3.2.3 tm\_mday [int8\\_t](#) tm::tm\_mday

day of the month - [ 1 to 31 ]

#### 22.3.2.4 tm\_min [int8\\_t](#) tm::tm\_min

minutes after the hour - [ 0 to 59 ]

**22.3.2.5 tm\_mon** `int8_t` `tm::tm_mon`

months since January - [ 0 to 11 ]

**22.3.2.6 tm\_sec** `int8_t` `tm::tm_sec`

seconds after the minute - [ 0 to 59 ]

**22.3.2.7 tm\_wday** `int8_t` `tm::tm_wday`

days since Sunday - [ 0 to 6 ]

**22.3.2.8 tm\_yday** `int16_t` `tm::tm_yday`

days since January 1 - [ 0 to 365 ]

**22.3.2.9 tm\_year** `int16_t` `tm::tm_year`

years since 1900

The documentation for this struct was generated from the following file:

- [time.h](#)

## 22.4 week\_date Struct Reference

```
#include <time.h>
```

### Data Fields

- int [year](#)
- int [week](#)
- int [day](#)

### 22.4.1 Detailed Description

Structure which represents a date as a year, week number of that year, and day of week. See [http://en.wikipedia.org/wiki/ISO\\_week\\_date](http://en.wikipedia.org/wiki/ISO_week_date) for more information.

### 22.4.2 Field Documentation

**22.4.2.1 day** int week\_date::day

day within week

**22.4.2.2 week** int week\_date::week

week number (#1 is where first Thursday is in)

**22.4.2.3 year** int week\_date::year

year number (Gregorian calendar)

The documentation for this struct was generated from the following file:

- [time.h](#)

## 23 File Documentation

### 23.1 project.h

```
00001 /*
00002  * -----
00003  * "THE BEER-WARE LICENSE" (Revision 42):
00004  * Joerg Wunsch wrote this file. As long as you retain this notice you
00005  * can do whatever you want with this stuff. If we meet some day, and you think
00006  * this stuff is worth it, you can buy me a beer in return.      Joerg Wunsch
00007  * -----
00008  *
00009  * Demo combining C and assembly source files.
00010  *
00011  * $Id$
00012  */
00013
00014 /*
00015  * Global register variables.
00016  */
00017 #ifdef __ASSEMBLER__
00018
00019 # define sreg_save r2
00020 # define flags     r16
00021 # define counter_hi r4
00022
00023 #else /* !ASSEMBLER */
00024
00025 #include <stdint.h>
00026
00027 register uint8_t sreg_save asm("r2");
00028 register uint8_t flags     asm("r16");
00029 register uint8_t counter_hi asm("r4");
00030
00031 #endif /* ASSEMBLER */
```

### 23.2 iocompat.h

```
00001 /*
00002  * -----
00003  * "THE BEER-WARE LICENSE" (Revision 42):
00004  * <joerg@FreeBSD.ORG> wrote this file. As long as you retain this notice you
00005  * can do whatever you want with this stuff. If we meet some day, and you think
00006  * this stuff is worth it, you can buy me a beer in return.      Joerg Wunsch
00007  * -----
00008  *
00009  * IO feature compatibility definitions for various AVRs.
00010  *
00011  * $Id$
00012  */
00013
```



```

00014 #if !defined(IOCMPAT_H)
00015 #define IOCMPAT_H 1
00016
00017 /*
00018  * Device-specific adjustments:
00019  *
00020  * Supply definitions for the location of the OCR1[A] port/pin, the
00021  * name of the OCR register controlling the PWM, and adjust interrupt
00022  * vector names that differ from the one used in demo.c
00023  * [TIMER1_OVF_vect].
00024  */
00025 #if defined(__AVR_AT90S2313__)
00026 # define OC1 PB3
00027 # define OCR OCR1
00028 # define DDROC DDRB
00029 # define TIMER1_OVF_vect TIMER1_OVF1_vect
00030 #elif defined(__AVR_AT90S2333__) || defined(__AVR_AT90S4433__)
00031 # define OC1 PB1
00032 # define DDROC DDRB
00033 # define OCR OCR1
00034 #elif defined(__AVR_AT90S4414__) || defined(__AVR_AT90S8515__) || \
00035     defined(__AVR_AT90S4434__) || defined(__AVR_AT90S8535__) || \
00036     defined(__AVR_ATmega163__) || defined(__AVR_ATmega8515__) || \
00037     defined(__AVR_ATmega8535__) || \
00038     defined(__AVR_ATmega164P__) || defined(__AVR_ATmega324P__) || \
00039     defined(__AVR_ATmega644__) || defined(__AVR_ATmega644P__) || \
00040     defined(__AVR_ATmega1284P__)
00041 # define OC1 PD5
00042 # define DDROC DDRD
00043 # define OCR OCR1A
00044 # if !defined(TIMSK) /* new ATmegas */
00045 # define TIMSK TIMSK1
00046 # endif
00047 #elif defined(__AVR_ATmega8__) || defined(__AVR_ATmega48__) || \
00048     defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
00049 # define OC1 PB1
00050 # define DDROC DDRB
00051 # define OCR OCR1A
00052 # if !defined(TIMSK) /* ATmega48/88/168 */
00053 # define TIMSK TIMSK1
00054 # endif /* !defined(TIMSK) */
00055 #elif defined(__AVR_ATtiny2313__)
00056 # define OC1 PB3
00057 # define OCR OCR1A
00058 # define DDROC DDRB
00059 #elif defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || \
00060     defined(__AVR_ATtiny84__)
00061 # define OC1 PA6
00062 # define DDROC DDRA
00063 # if !defined(OCR1A)
00064 # /* work around misspelled name in AVR-LibC 1.4.[0..1] */
00065 # define OCR OCR1A
00066 # else
00067 # define OCR OCR1A
00068 # endif
00069 # define TIMSK TIMSK1
00070 # define TIMER1_OVF_vect TIM1_OVF_vect /* XML and datasheet mismatch */
00071 #elif defined(__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || \
00072     defined(__AVR_ATtiny85__)
00073 /* Timer 1 is only an 8-bit timer on these devices. */
00074 # define OC1 PB1
00075 # define DDROC DDRB
00076 # define OCR OCR1A
00077 # define TCCR1A TCCR1
00078 # define TCCR1B TCCR1
00079 # define TIMER1_OVF_vect TIM1_OVF_vect
00080 # define TIMER1_TOP 255 /* only 8-bit PWM possible */
00081 # define TIMER1_PWM_INIT _BV(PWM1A) | _BV(COM1A1)
00082 # define TIMER1_CLOCKSOURCE _BV(CS12) /* use 1/8 prescaler */
00083 #elif defined(__AVR_ATtiny26__)
00084 /* Rather close to ATtinyX5 but different enough for its own section. */
00085 # define OC1 PB1
00086 # define DDROC DDRB
00087 # define OCR OCR1A
00088 # define TIMER1_OVF_vect TIMER1_OVF1_vect
00089 # define TIMER1_TOP 255 /* only 8-bit PWM possible */
00090 # define TIMER1_PWM_INIT _BV(PWM1A) | _BV(COM1A1)
00091 # define TIMER1_CLOCKSOURCE _BV(CS12) /* use 1/8 prescaler */
00092 /*
00093  * Without setting OCR1C to TOP, the ATtiny26 does not trigger an
00094  * overflow interrupt in PWM mode.
00095  */
00096 # define TIMER1_SETUP_HOOK() OCR1C = 255
00097 #elif defined(__AVR_ATtiny261__) || defined(__AVR_ATtiny461__) || \
00098     defined(__AVR_ATtiny861__)
00099 # define OC1 PB1
00100 # define DDROC DDRB

```

```

00101 # define OCR OCR1A
00102 # define TIMER1_PWM_INIT _BV(WGM10) | _BV(PWM1A) | _BV(COM1A1)
00103 /*
00104 * While timer 1 could be operated in 10-bit mode on these devices,
00105 * the handling of the 10-bit IO registers is more complicated than
00106 * that of the 16-bit registers of other AVR devices (no combined
00107 * 16-bit IO operations possible), so we restrict this demo to 8-bit
00108 * mode which is pretty standard.
00109 */
00110 # define TIMER1_TOP 255
00111 # define TIMER1_CLOCKSOURCE _BV(CS12) /* use 1/8 prescaler */
00112 #elif defined(__AVR_ATmega32__) || defined(__AVR_ATmega16__)
00113 # define OC1 PD5
00114 # define DDROC DDRD
00115 # define OCR OCR1A
00116 #elif defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__) || \
00117     defined(__AVR_ATmega165__) || defined(__AVR_ATmega169__) || \
00118     defined(__AVR_ATmega325__) || defined(__AVR_ATmega3250__) || \
00119     defined(__AVR_ATmega645__) || defined(__AVR_ATmega6450__) || \
00120     defined(__AVR_ATmega329__) || defined(__AVR_ATmega3290__) || \
00121     defined(__AVR_ATmega649__) || defined(__AVR_ATmega6490__) || \
00122     defined(__AVR_ATmega640__) || \
00123     defined(__AVR_ATmega1280__) || defined(__AVR_ATmega1281__) || \
00124     defined(__AVR_ATmega2560__) || defined(__AVR_ATmega2561__) || \
00125     defined(__AVR_ATmega32U4__)
00126 # define OC1 PB5
00127 # define DDROC DDRB
00128 # define OCR OCR1A
00129 # if !defined(PB5) /* work around missing bit definition */
00130 #     define PB5 5
00131 # endif
00132 # if !defined(TIMSK) /* new ATmegas */
00133 #     define TIMSK TIMSK1
00134 # endif
00135 #else
00136 # error "Don't know what kind of MCU you are compiling for"
00137 #endif
00138
00139 /*
00140 * Map register names for older AVRs here.
00141 */
00142 #if !defined(COM1A1)
00143 # define COM1A1 COM1
00144 #endif
00145
00146 #if !defined(WGM10)
00147 # define WGM10 PWM10
00148 # define WGM11 PWM11
00149 #endif
00150
00151 /*
00152 * Provide defaults for device-specific macros unless overridden
00153 * above.
00154 */
00155 #if !defined(TIMER1_TOP)
00156 # define TIMER1_TOP 1023 /* 10-bit PWM */
00157 #endif
00158
00159 #if !defined(TIMER1_PWM_INIT)
00160 # define TIMER1_PWM_INIT _BV(WGM10) | _BV(WGM11) | _BV(COM1A1)
00161 #endif
00162
00163 #if !defined(TIMER1_CLOCKSOURCE)
00164 # define TIMER1_CLOCKSOURCE _BV(CS10) /* full clock */
00165 #endif
00166
00167 #endif /* !defined(IOCMPAT_H) */

```

## 23.3 defines.h

```

00001 /*
00002 * -----
00003 * "THE BEER-WARE LICENSE" (Revision 42):
00004 * <joerg@FreeBSD.ORG> wrote this file. As long as you retain this notice you
00005 * can do whatever you want with this stuff. If we meet some day, and you think
00006 * this stuff is worth it, you can buy me a beer in return.      Joerg Wunsch
00007 * -----
00008 *
00009 * General stdiodemo defines
00010 *
00011 * $Id$
00012 */
00013
00014 /* CPU frequency */

```

```

00015 #define F_CPU 1000000UL
00016
00017 /* UART baud rate */
00018 #define UART_BAUD 9600
00019
00020 /* HD44780 LCD port connections */
00021 #define HD44780_RS A, 6
00022 #define HD44780_RW A, 4
00023 #define HD44780_E A, 5
00024 /* The data bits have to be not only in ascending order but also consecutive. */
00025 #define HD44780_D4 A, 0
00026
00027 /* Whether to read the busy flag, or fall back to
00028    worst-time delays. */
00029 #define USE_BUSY_BIT 1

```

## 23.4 hd44780.h

```

00001 /*
00002 * -----
00003 * "THE BEER-WARE LICENSE" (Revision 42):
00004 * <joerg@FreeBSD.ORG> wrote this file. As long as you retain this notice you
00005 * can do whatever you want with this stuff. If we meet some day, and you think
00006 * this stuff is worth it, you can buy me a beer in return. Joerg Wunsch
00007 * -----
00008 *
00009 * HD44780 LCD display driver
00010 *
00011 * $Id$
00012 */
00013
00014 /*
00015 * Send byte b to the LCD. rs is the RS signal (register select), 0
00016 * selects instruction register, 1 selects the data register.
00017 */
00018 void hd44780_outbyte(uint8_t b, uint8_t rs);
00019
00020 /*
00021 * Read one byte from the LCD controller. rs is the RS signal, 0
00022 * selects busy flag (bit 7) and address counter, 1 selects the data
00023 * register.
00024 */
00025 uint8_t hd44780_inbyte(uint8_t rs);
00026
00027 /*
00028 * Wait for the busy flag to clear.
00029 */
00030 void hd44780_wait_ready(bool islong);
00031
00032 /*
00033 * Initialize the LCD controller hardware.
00034 */
00035 void hd44780_init(void);
00036
00037 /*
00038 * Prepare the LCD controller pins for powerdown.
00039 */
00040 void hd44780_powerdown(void);
00041
00042
00043 /* Send a command to the LCD controller. */
00044 #define hd44780_outcmd(n) hd44780_outbyte((n), 0)
00045
00046 /* Send a data byte to the LCD controller. */
00047 #define hd44780_outdata(n) hd44780_outbyte((n), 1)
00048
00049 /* Read the address counter and busy flag from the LCD. */
00050 #define hd44780_incmd() hd44780_inbyte(0)
00051
00052 /* Read the current data byte from the LCD. */
00053 #define hd44780_indata() hd44780_inbyte(1)
00054
00055
00056 /* Clear LCD display command. */
00057 #define HD44780_CLR \
00058     0x01
00059
00060 /* Home cursor command. */
00061 #define HD44780_HOME \
00062     0x02
00063
00064 /*
00065 * Select the entry mode. inc determines whether the address counter
00066 * auto-increments, shift selects an automatic display shift.

```

```

00067 */
00068 #define HD44780_ENTMODE(inc, shift) \
00069     (0x04 | ((inc)? 0x02: 0) | ((shift)? 1: 0))
00070
00071 /*
00072 * Selects disp[lay] on/off, cursor on/off, cursor blink[ing]
00073 * on/off.
00074 */
00075 #define HD44780_DISPCTL(disp, cursor, blink) \
00076     (0x08 | ((disp)? 0x04: 0) | ((cursor)? 0x02: 0) | ((blink)? 1: 0))
00077
00078 /*
00079 * With shift = 1, shift display right or left.
00080 * With shift = 0, move cursor right or left.
00081 */
00082 #define HD44780_SHIFT(shift, right) \
00083     (0x10 | ((shift)? 0x08: 0) | ((right)? 0x04: 0))
00084
00085 /*
00086 * Function set.  if8bit selects an 8-bit data path, twoline arranges
00087 * for a two-line display, font5x10 selects the 5x10 dot font (5x8
00088 * dots if clear).
00089 */
00090 #define HD44780_FNSET(if8bit, twoline, font5x10) \
00091     (0x20 | ((if8bit)? 0x10: 0) | ((twoline)? 0x08: 0) | \
00092     ((font5x10)? 0x04: 0))
00093
00094 /*
00095 * Set the next character generator address to addr.
00096 */
00097 #define HD44780_CGADDR(addr) \
00098     (0x40 | ((addr) & 0x3f))
00099
00100 /*
00101 * Set the next display address to addr.
00102 */
00103 #define HD44780_DDADDR(addr) \
00104     (0x80 | ((addr) & 0x7f))
00105

```

## 23.5 lcd.h

```

00001 /*
00002 * -----
00003 * "THE BEER-WARE LICENSE" (Revision 42):
00004 * <joerg@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
00005 * can do whatever you want with this stuff.  If we meet some day, and you think
00006 * this stuff is worth it, you can buy me a beer in return.          Joerg Wunsch
00007 * -----
00008 *
00009 * Stdio demo, upper layer of LCD driver.
00010 *
00011 * $Id$
00012 */
00013
00014 /*
00015 * Initialize LCD controller.  Performs a software reset.
00016 */
00017 void    lcd_init(void);
00018
00019 /*
00020 * Send one character to the LCD.
00021 */
00022 int lcd_putchar(char c, FILE *stream);

```

## 23.6 uart.h

```

00001 /*
00002 * -----
00003 * "THE BEER-WARE LICENSE" (Revision 42):
00004 * <joerg@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
00005 * can do whatever you want with this stuff.  If we meet some day, and you think
00006 * this stuff is worth it, you can buy me a beer in return.          Joerg Wunsch
00007 * -----
00008 *
00009 * Stdio demo, UART declarations
00010 *
00011 * $Id$
00012 */
00013
00014 /*

```

```

00015  * Perform UART startup initialization.
00016  */
00017 void    uart_init(void);
00018
00019 /*
00020  * Send one character to the UART.
00021  */
00022 int    uart_putchar(char c, FILE *stream);
00023
00024 /*
00025  * Size of internal line buffer used by uart_getchar().
00026  */
00027 #define RX_BUFSIZE 80
00028
00029 /*
00030  * Receive one character from the UART. The actual reception is
00031  * line-buffered, and one character is returned from the buffer at
00032  * each invocation.
00033  */
00034 int    uart_getchar(FILE *stream);

```

## 23.7 alloca.h

```

00001 /* Copyright (c) 2007, Dmitry Xmelkov
00002    All rights reserved.
00003
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions are met:
00006
00007    * Redistributions of source code must retain the above copyright
00008    notice, this list of conditions and the following disclaimer.
00009    * Redistributions in binary form must reproduce the above copyright
00010    notice, this list of conditions and the following disclaimer in
00011    the documentation and/or other materials provided with the
00012    distribution.
00013    * Neither the name of the copyright holders nor the names of
00014    contributors may be used to endorse or promote products derived
00015    from this software without specific prior written permission.
00016
00017    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00018    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00019    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00020    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00021    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00022    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00023    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00024    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00026    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00027    POSSIBILITY OF SUCH DAMAGE. */
00028
00029 /* $Id$ */
00030
00031 #ifndef _ALLOCA_H
00032 #define _ALLOCA_H 1
00033
00034 #include <stddef.h>
00035
00036 /** \defgroup alloca <alloca.h>: Allocate space in the stack */
00037
00038 /** \ingroup alloca
00039     \brief Allocate \a __size bytes of space in the stack frame of the caller.
00040
00041     This temporary space is automatically freed when the function that
00042     called alloca() returns to its caller. AVR-LibC defines the alloca() as
00043     a macro, which is translated into the inlined \c __builtin_alloca()
00044     function. The fact that the code is inlined, means that it is impossible
00045     to take the address of this function, or to change its behaviour by
00046     linking with a different library.
00047
00048     \return alloca() returns a pointer to the beginning of the allocated
00049     space. If the allocation causes stack overflow, program behaviour is
00050     undefined.
00051
00052     \warning Avoid use alloca() inside the list of arguments of a function
00053     call.
00054 */
00055 extern void *alloca (size_t __size);
00056
00057 #define alloca(size)    __builtin_alloca (size)
00058
00059 #endif /* alloca.h */

```

## 23.8 assert.h File Reference

### Macros

- #define `assert`(expression)

## 23.9 assert.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2005,2007 Joerg Wunsch
00002    All rights reserved.
00003
00004    Portions of documentation Copyright (c) 1991, 1993
00005    The Regents of the University of California.
00006
00007    All rights reserved.
00008
00009    Redistribution and use in source and binary forms, with or without
00010    modification, are permitted provided that the following conditions are met:
00011
00012    * Redistributions of source code must retain the above copyright
00013      notice, this list of conditions and the following disclaimer.
00014
00015    * Redistributions in binary form must reproduce the above copyright
00016      notice, this list of conditions and the following disclaimer in
00017      the documentation and/or other materials provided with the
00018      distribution.
00019
00020    * Neither the name of the copyright holders nor the names of
00021      contributors may be used to endorse or promote products derived
00022      from this software without specific prior written permission.
00023
00024    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00025    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00026    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00027    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00028    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00029    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00030    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00031    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00032    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00033    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00034    POSSIBILITY OF SUCH DAMAGE.
00035
00036    $Id$
00037 */
00038
00039 /** \file */
00040 /** \defgroup avr_assert <assert.h>: Diagnostics
00041     \code #include <assert.h> \endcode
00042
00043     This header file defines a debugging aid.
00044
00045     As there is no standard error output stream available for many
00046     applications using this library, the generation of a printable
00047     error message is not enabled by default. These messages will
00048     only be generated if the application defines the macro
00049
00050     \code __ASSERT_USE_STDERR \endcode
00051
00052     before including the \c <assert.h> header file. By default,
00053     only abort() will be called to halt the application.
00054 */
00055
00056 /**@{*/
00057
00058 /*
00059  * The ability to include this file (with or without NDEBUG) is a
00060  * feature.
00061  */
00062
00063 #undef assert
00064
00065 #include <stdlib.h>
00066
00067 #if defined(__DOXYGEN__)
00068 /**
00069  * \def assert
00070  * \param expression Expression to test for.

```

```

00071 *
00072 * The assert() macro tests the given expression and if it is false,
00073 * the calling process is terminated. A diagnostic message is written
00074 * to stderr and the function abort() is called, effectively
00075 * terminating the program.
00076 *
00077 * If expression is true, the assert() macro does nothing.
00078 *
00079 * The assert() macro may be removed at compile time by defining
00080 * NDEBUG as a macro (e.g., by using the compiler option -DNDEBUG).
00081 */
00082 # define assert(expression)
00083
00084 #else /* !DOXYGEN */
00085
00086 # if defined(NDEBUG)
00087 #   define assert(e) ((void)0)
00088 # else /* !NDEBUG */
00089 #   if defined(__ASSERT_USE_STDERR)
00090 #     define assert(e) ((e) ? (void)0 : \
00091         __assert(__func__, __FILE__, __LINE__, #e))
00092 #   else /* !__ASSERT_USE_STDERR */
00093 #     define assert(e) ((e) ? (void)0 : abort())
00094 #   endif /* __ASSERT_USE_STDERR */
00095 # endif /* NDEBUG */
00096 #endif /* DOXYGEN */
00097
00098 #if (defined __STDC_VERSION__ && __STDC_VERSION__ >= 201112L) || \
00099     (_GNUCC_ > 4 || (_GNUCC_ == 4 && _GNUCC_MINOR_ >= 6)) && !defined __cplusplus)
00100 # undef static_assert
00101 # define static_assert _Static_assert
00102 #endif
00103
00104 #ifndef __cplusplus
00105 extern "C" {
00106 #endif
00107
00108 #if !defined(__DOXYGEN__)
00109
00110 extern void __assert(const char *__func, const char *__file,
00111     int __lineno, const char *__sexp);
00112
00113 #endif /* not __DOXYGEN__ */
00114
00115 #ifndef __cplusplus
00116 }
00117 #endif
00118
00119 /**@}*/
00120 /* EOF */

```

## 23.10 boot.h File Reference

### Macros

- #define `BOOTLOADER_SECTION` `__attribute__((__section__(".bootloader")))`
- #define `boot_spm_interrupt_enable()` `(__SPM_REG |= (uint8_t)_BV(SPMIE))`
- #define `boot_spm_interrupt_disable()` `(__SPM_REG &= (uint8_t)~_BV(SPMIE))`
- #define `boot_is_spm_interrupt()` `(__SPM_REG & (uint8_t)_BV(SPMIE))`
- #define `boot_rww_busy()` `(__SPM_REG & (uint8_t)_BV(__COMMON_ASB))`
- #define `boot_spm_busy()` `(__SPM_REG & (uint8_t)_BV(__SPM_ENABLE))`
- #define `boot_spm_busy_wait()` `do{}while(boot_spm_busy())`
- #define `GET_LOW_FUSE_BITS` `(0x0000)`
- #define `GET_LOCK_BITS` `(0x0001)`
- #define `GET_EXTENDED_FUSE_BITS` `(0x0002)`
- #define `GET_HIGH_FUSE_BITS` `(0x0003)`
- #define `boot_lock_fuse_bits_get(address)`
- #define `boot_signature_byte_get(addr)`
- #define `boot_page_fill(address, data)` `__boot_page_fill_normal(address, data)`
- #define `boot_page_erase(address)` `__boot_page_erase_normal(address)`
- #define `boot_page_write(address)` `__boot_page_write_normal(address)`
- #define `boot_rww_enable()` `__boot_rww_enable()`

- #define `boot_lock_bits_set(lock_bits) __boot_lock_bits_set(lock_bits)`
- #define `boot_page_fill_safe(address, data)`
- #define `boot_page_erase_safe(address)`
- #define `boot_page_write_safe(address)`
- #define `boot_rww_enable_safe()`
- #define `boot_lock_bits_set_safe(lock_bits)`

## 23.11 boot.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002,2003,2004,2005,2006,2007,2008,2009 Eric B. Weddington
00002 All rights reserved.
00003
00004 Redistribution and use in source and binary forms, with or without
00005 modification, are permitted provided that the following conditions are met:
00006
00007 * Redistributions of source code must retain the above copyright
00008 notice, this list of conditions and the following disclaimer.
00009 * Redistributions in binary form must reproduce the above copyright
00010 notice, this list of conditions and the following disclaimer in
00011 the documentation and/or other materials provided with the
00012 distribution.
00013 * Neither the name of the copyright holders nor the names of
00014 contributors may be used to endorse or promote products derived
00015 from this software without specific prior written permission.
00016
00017 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00018 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00019 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00020 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00021 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00022 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00023 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00024 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00026 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00027 POSSIBILITY OF SUCH DAMAGE. */
00028
00029 /* $Id$ */
00030
00031 #ifndef _AVR_BOOT_H_
00032 #define _AVR_BOOT_H_ 1
00033
00034 /** \file */
00035 /** \defgroup avr_boot <avr/boot.h>: Bootloader Support Utilities
00036     \code
00037     #include <avr/io.h>
00038     #include <avr/boot.h>
00039     \endcode
00040
00041 The macros in this module provide a C language interface to the
00042 bootloader support functionality of certain AVR processors. These
00043 macros are designed to work with all sizes of flash memory.
00044
00045 Global interrupts are not automatically disabled for these macros. It
00046 is left up to the programmer to do this. See the code example below.
00047 Also see the processor datasheet for caveats on having global interrupts
00048 enabled during writing of the Flash.
00049
00050 \note Not all AVR processors provide bootloader support. See your
00051 processor datasheet to see if it provides bootloader support.
00052
00053 \par API Usage Example
00054 The following code shows typical usage of the boot API.
00055
00056 \code
00057 #include <stdint.h>
00058 #include <avr/interrupt.h>
00059 #include <avr/pgmspace.h>
00060
00061 void boot_program_page (uint32_t page, uint8_t *buf)
00062 {
00063     // Disable interrupts.
00064     uint8_t sreg = SREG;
00065     cli();
00066
00067     eeprom_busy_wait ();
00068
00069     boot_page_erase (page);

```



```

00070     boot_spm_busy_wait ();      // Wait until the memory is erased.
00071
00072     for (uint16_t i = 0; i < SPM_PAGESIZE; i += 2)
00073     {
00074         // Set up little-endian word.
00075         uint16_t w = *buf++;
00076         w += (*buf++) << 8;
00077
00078         boot_page_fill (page + i, w);
00079     }
00080
00081     boot_page_write (page);      // Store buffer in flash page.
00082     boot_spm_busy_wait();      // Wait until the memory is written.
00083
00084     // Reenable RWW-section again. We need this if we want to jump back
00085     // to the application after bootloading.
00086     boot_rww_enable ();
00087
00088     // Re-enable interrupts (if they were ever enabled).
00089     SREG = sreg;
00090 } \endcode */
00091
00092 #include <avr/eeprom.h>
00093 #include <avr/io.h>
00094 #include <inttypes.h>
00095 #include <limits.h>
00096
00097 /* Check for SPM Control Register in processor. */
00098 #if defined (SPMCSR)
00099 # define __SPM_REG    SPMCSR
00100 #else
00101 # if defined (SPMCR)
00102 #   define __SPM_REG    SPMCR
00103 # else
00104 #   error AVR processor does not provide bootloader support!
00105 # endif
00106 #endif
00107
00108
00109 /* Check for SPM Enable bit. */
00110 #if defined (SPMEN)
00111 # define __SPM_ENABLE    SPMEN
00112 #elif defined (SELPFRGEN)
00113 # define __SPM_ENABLE    SELPFRGEN
00114 #else
00115 # error Cannot find SPM Enable bit definition!
00116 #endif
00117
00118 /** \ingroup avr_boot
00119     \def BOOTLOADER_SECTION
00120
00121     Used to declare a function or variable to be placed into a
00122     new section called .bootloader. This section and its contents
00123     can then be relocated to any address (such as the bootloader
00124     NRWW area) at link-time. */
00125
00126 #define BOOTLOADER_SECTION    __attribute__ ((__section__(".bootloader")))
00127
00128 #ifndef __DOXYGEN__
00129 /* Create common bit definitions. */
00130 #ifndef ASB
00131 #define __COMMON_ASB    ASB
00132 #else
00133 #define __COMMON_ASB    RWWSB
00134 #endif
00135
00136 #ifndef ASRE
00137 #define __COMMON_ASRE    ASRE
00138 #else
00139 #define __COMMON_ASRE    RWWSRE
00140 #endif
00141
00142 /* Define the bit positions of the Boot Lock Bits. */
00143
00144 #define BLB12    5
00145 #define BLB11    4
00146 #define BLB02    3
00147 #define BLB01    2
00148 #endif /* __DOXYGEN__ */
00149
00150 /** \ingroup avr_boot
00151     \def boot_spm_interrupt_enable()
00152     Enable the SPM interrupt. */
00153
00154 #define boot_spm_interrupt_enable()    (__SPM_REG |= (uint8_t)_BV (SPMIE))
00155
00156 /** \ingroup avr_boot

```

```

00157     \def boot_spm_interrupt_disable()
00158     Disable the SPM interrupt. */
00159
00160 #define boot_spm_interrupt_disable()  (__SPM_REG &= (uint8_t)~_BV(SPMIE))
00161
00162 /** \ingroup avr_boot
00163     \def boot_is_spm_interrupt()
00164     Check if the SPM interrupt is enabled. */
00165
00166 #define boot_is_spm_interrupt()      (__SPM_REG & (uint8_t)_BV(SPMIE))
00167
00168 /** \ingroup avr_boot
00169     \def boot_rww_busy()
00170     Check if the RWW section is busy. */
00171
00172 #define boot_rww_busy()              (__SPM_REG & (uint8_t)_BV(__COMMON_ASB))
00173
00174 /** \ingroup avr_boot
00175     \def boot_spm_busy()
00176     Check if the SPM instruction is busy. */
00177
00178 #define boot_spm_busy()              (__SPM_REG & (uint8_t)_BV(__SPM_ENABLE))
00179
00180 /** \ingroup avr_boot
00181     \def boot_spm_busy_wait()
00182     Wait while the SPM instruction is busy. */
00183
00184 #define boot_spm_busy_wait()        do{}while(boot_spm_busy())
00185
00186 #ifndef __DOXYGEN__
00187 #define __BOOT_PAGE_ERASE           (_BV(__SPM_ENABLE) | _BV(PGERS))
00188 #define __BOOT_PAGE_WRITE          (_BV(__SPM_ENABLE) | _BV(PGWRT))
00189 #define __BOOT_PAGE_FILL           _BV(__SPM_ENABLE)
00190 #define __BOOT_RWW_ENABLE          (_BV(__SPM_ENABLE) | _BV(__COMMON_ASRE))
00191 #if defined(BLBSET)
00192 #define __BOOT_LOCK_BITS_SET       (_BV(__SPM_ENABLE) | _BV(BLBSET))
00193 #elif defined(RFLB) /* Some devices have RFLB defined instead of BLBSET. */
00194 #define __BOOT_LOCK_BITS_SET       (_BV(__SPM_ENABLE) | _BV(RFLB))
00195 #elif defined(RWFLB) /* Some devices have RWFLB defined instead of BLBSET. */
00196 #define __BOOT_LOCK_BITS_SET       (_BV(__SPM_ENABLE) | _BV(RWFLB))
00197 #endif
00198
00199 #define __boot_page_fill_normal(address, data) \
00200  (__extension__({
00201     if (_SFR_IO_REG_P(__SPM_REG))
00202         __asm__ __volatile__ (
00203             "movw r0, %3"        "\n\t"
00204             "out  %0, %1"        "\n\t"
00205             "spm"                 "\n\t"
00206             "clr  __zero_reg__"
00207             :
00208             : "i" (_SFR_IO_ADDR(__SPM_REG)),
00209               "r" ((uint8_t)(__BOOT_PAGE_FILL)),
00210               "z" ((uint16_t)(address)),
00211               "r" ((uint16_t)(data))
00212             : "r0");
00213     else
00214         __asm__ __volatile__ (
00215             "movw r0, %3"        "\n\t"
00216             "sts  %0, %1"        "\n\t"
00217             "spm"                 "\n\t"
00218             "clr  __zero_reg__"
00219             :
00220             : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00221               "r" ((uint8_t)(__BOOT_PAGE_FILL)),
00222               "z" ((uint16_t)(address)),
00223               "r" ((uint16_t)(data))
00224             : "r0");
00225  }))
00226
00227 #define __boot_page_fill_alternate(address, data) \
00228  (__extension__({
00229     __asm__ __volatile__
00230     (
00231         "movw r0, %3"        "\n\t"
00232         "sts  %0, %1"        "\n\t"
00233         "spm"                 "\n\t"
00234         ".word 0xffff"        "\n\t"
00235         "nop"                 "\n\t"
00236         "clr  __zero_reg__"
00237         :
00238         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00239           "r" ((uint8_t)(__BOOT_PAGE_FILL)),
00240           "z" ((uint16_t)(address)),
00241           "r" ((uint16_t)(data))
00242         : "r0"
00243     );

```

```

00244 )))
00245
00246 #define __boot_page_fill_extended(address, data) \
00247 (__extension__({
00248     __asm__ __volatile__
00249     (
00250         "movw r0, %4"           "\n\t"
00251         "movw r30, %A3"        "\n\t"
00252         "out %1, %C3"          "\n\t"
00253         "sts %0, %2"           "\n\t"
00254         "spm"                  "\n\t"
00255         "clr __zero_reg__"
00256         :
00257         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00258           "i" (_SFR_IO_ADDR(RAMPZ)),
00259           "r" ((uint8_t)(__BOOT_PAGE_FILL)),
00260           "r" ((uint32_t)(address)),
00261           "r" ((uint16_t)(data))
00262         : "r0", "r30", "r31"
00263     );
00264 )))
00265
00266 #define __boot_page_erase_normal(address)
00267 (__extension__({
00268     if (_SFR_IO_REG_P(__SPM_REG))
00269         __asm__ __volatile__ (
00270             "out %0, %1"       "\n\t"
00271             "spm"
00272             :
00273             : "i" (_SFR_IO_ADDR(__SPM_REG)),
00274               "r" ((uint8_t)(__BOOT_PAGE_ERASE)),
00275               "z" ((uint16_t)(address)));
00276     else
00277         __asm__ __volatile__ (
00278             "sts %0, %1"       "\n\t"
00279             "spm"
00280             :
00281             : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00282               "r" ((uint8_t)(__BOOT_PAGE_ERASE)),
00283               "z" ((uint16_t)(address)));
00284     )))
00285
00286 #define __boot_page_erase_alternate(address)
00287 (__extension__({
00288     __asm__ __volatile__
00289     (
00290         "sts %0, %1"           "\n\t"
00291         "spm"                  "\n\t"
00292         ".word 0xffff"         "\n\t"
00293         "nop"
00294         :
00295         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00296           "r" ((uint8_t)(__BOOT_PAGE_ERASE)),
00297           "z" ((uint16_t)(address))
00298         );
00299     )))
00300
00301 #define __boot_page_erase_extended(address)
00302 (__extension__({
00303     __asm__ __volatile__
00304     (
00305         "movw r30, %A3"        "\n\t"
00306         "out %1, %C3"          "\n\t"
00307         "sts %0, %2"           "\n\t"
00308         "spm"
00309         :
00310         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00311           "i" (_SFR_IO_ADDR(RAMPZ)),
00312           "r" ((uint8_t)(__BOOT_PAGE_ERASE)),
00313           "r" ((uint32_t)(address))
00314         : "r30", "r31"
00315     );
00316     )))
00317
00318 #define __boot_page_write_normal(address)
00319 (__extension__({
00320     if (_SFR_IO_REG_P(__SPM_REG))
00321         __asm__ __volatile__ (
00322             "out %0, %1"       "\n\t"
00323             "spm"
00324             :
00325             : "i" (_SFR_IO_ADDR(__SPM_REG)),
00326               "r" ((uint8_t)(__BOOT_PAGE_WRITE)),
00327               "z" ((uint16_t)(address)));
00328     else
00329         __asm__ __volatile__ (
00330             "sts %0, %1"       "\n\t"

```

```

00331     "spm"
00332     :
00333     : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00334     "r" ((uint8_t)(__BOOT_PAGE_WRITE)),
00335     "z" ((uint16_t)(address));
00336 ))
00337
00338 #define __boot_page_write_alternate(address)
00339 (__extension__({
00340     __asm__ __volatile__
00341     (
00342         "sts %0, %1"        "\n\t"
00343         "spm"              "\n\t"
00344         ".word 0xffff"     "\n\t"
00345         "nop"
00346         :
00347         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00348         "r" ((uint8_t)(__BOOT_PAGE_WRITE)),
00349         "z" ((uint16_t)(address))
00350     );
00351 ))
00352
00353 #define __boot_page_write_extended(address)
00354 (__extension__({
00355     __asm__ __volatile__
00356     (
00357         "movw r30, %A3"     "\n\t"
00358         "out %1, %C3"      "\n\t"
00359         "sts %0, %2"      "\n\t"
00360         "spm"
00361         :
00362         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00363         "i" (_SFR_IO_ADDR(RAMPZ)),
00364         "r" ((uint8_t)(__BOOT_PAGE_WRITE)),
00365         "r" ((uint32_t)(address))
00366         : "r30", "r31"
00367     );
00368 ))
00369
00370 #define __boot_rww_enable()
00371 (__extension__({
00372     __asm__ __volatile__
00373     (
00374         "sts %0, %1"      "\n\t"
00375         "spm"
00376         :
00377         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00378         "r" ((uint8_t)(__BOOT_RWW_ENABLE))
00379     );
00380 ))
00381
00382 #define __boot_rww_enable_alternate()
00383 (__extension__({
00384     __asm__ __volatile__
00385     (
00386         "sts %0, %1"      "\n\t"
00387         "spm"              "\n\t"
00388         ".word 0xffff"     "\n\t"
00389         "nop"
00390         :
00391         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00392         "r" ((uint8_t)(__BOOT_RWW_ENABLE))
00393     );
00394 ))
00395
00396 /* From the megal6/megal28 data sheets (maybe others):
00397
00398 Bits by SPM To set the Boot Loader Lock bits, write the desired data to
00399 R0, write "X0001001" to SPMCR and execute SPM within four clock cycles
00400 after writing SPMCR. The only accessible Lock bits are the Boot Lock bits
00401 that may prevent the Application and Boot Loader section from any
00402 software update by the MCU.
00403
00404 If bits 5..2 in R0 are cleared (zero), the corresponding Boot Lock bit
00405 will be programmed if an SPM instruction is executed within four cycles
00406 after BLBSET and SPEN (or SELFPGEN) are set in SPMCR. The Z-pointer is
00407 don't care during this operation, but for future compatibility it is
00408 recommended to load the Z-pointer with $0001 (same as used for reading the
00409 Lock bits). For future compatibility It is also recommended to set bits 7,
00410 6, 1, and 0 in R0 to 1 when writing the Lock bits. When programming the
00411 Lock bits the entire Flash can be read during the operation. */
00412
00413 #define __boot_lock_bits_set(lock_bits)
00414 (__extension__({
00415     uint8_t value = (uint8_t)(~(lock_bits));
00416     __asm__ __volatile__
00417     (

```

```

00418         "ldi r30, 1"           "\n\t"
00419         "ldi r31, 0"           "\n\t"
00420         "mov r0, %2"           "\n\t"
00421         "sts %0, %1"           "\n\t"
00422         "spm"
00423         :
00424         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00425           "r" ((uint8_t)(__BOOT_LOCK_BITS_SET)),
00426           "r" (value)
00427         : "r0", "r30", "r31"
00428     );
00429 )))
00430
00431 #define __boot_lock_bits_set_alternate(lock_bits)
00432 (__extension__({
00433     uint8_t value = (uint8_t)(~(lock_bits));
00434     __asm__ __volatile__
00435     (
00436         "ldi r30, 1"           "\n\t"
00437         "ldi r31, 0"           "\n\t"
00438         "mov r0, %2"           "\n\t"
00439         "sts %0, %1"           "\n\t"
00440         "spm"                  "\n\t"
00441         ".word 0xffff"         "\n\t"
00442         "nop"
00443         :
00444         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00445           "r" ((uint8_t)(__BOOT_LOCK_BITS_SET)),
00446           "r" (value)
00447         : "r0", "r30", "r31"
00448     );
00449 )))
00450 #endif /* __DOXYGEN__ */
00451
00452 /*
00453     Reading lock and fuse bits:
00454
00455     Similarly to writing the lock bits above, set BLBSET and SPMMEN (or
00456     SELFPRGEN) bits in __SPMREG, and then (within four clock cycles) issue an
00457     LPM instruction.
00458
00459     Z address:      contents:
00460     0x0000          low fuse bits
00461     0x0001          lock bits
00462     0x0002          extended fuse bits
00463     0x0003          high fuse bits
00464
00465     Sounds confusing, doesn't it?
00466
00467     Unlike the macros in pgmspace.h, no need to care for non-enhanced
00468     cores here as these old cores do not provide SPM support anyway.
00469 */
00470
00471 /** \ingroup avr_boot
00472     \def GET_LOW_FUSE_BITS
00473     address to read the low fuse bits, using boot_lock_fuse_bits_get
00474 */
00475 #define GET_LOW_FUSE_BITS          (0x0000)
00476 /** \ingroup avr_boot
00477     \def GET_LOCK_BITS
00478     address to read the lock bits, using boot_lock_fuse_bits_get
00479 */
00480 #define GET_LOCK_BITS              (0x0001)
00481 /** \ingroup avr_boot
00482     \def GET_EXTENDED_FUSE_BITS
00483     address to read the extended fuse bits, using boot_lock_fuse_bits_get
00484 */
00485 #define GET_EXTENDED_FUSE_BITS     (0x0002)
00486 /** \ingroup avr_boot
00487     \def GET_HIGH_FUSE_BITS
00488     address to read the high fuse bits, using boot_lock_fuse_bits_get
00489 */
00490 #define GET_HIGH_FUSE_BITS         (0x0003)
00491
00492 /** \ingroup avr_boot
00493     \def boot_lock_fuse_bits_get(address)
00494
00495     Read the lock or fuse bits at \c address.
00496
00497     Parameter \c address can be any of GET_LOW_FUSE_BITS,
00498     GET_LOCK_BITS, GET_EXTENDED_FUSE_BITS, or GET_HIGH_FUSE_BITS.
00499
00500     \note The lock and fuse bits returned are the physical values,
00501     i.e. a bit returned as 0 means the corresponding fuse or lock bit
00502     is programmed.
00503 */
00504 #define boot_lock_fuse_bits_get(address)

```

```

00505 (__extension__({
00506     uint8_t __result;
00507     __asm__ __volatile__
00508     (
00509         "sts %1, %2\n\t"
00510         "lpm %0, Z\n\t"
00511         : "=r" (__result)
00512         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00513           "r" ((uint8_t)(__BOOT_LOCK_BITS_SET)),
00514           "z" ((uint16_t)(address))
00515         );
00516     __result;
00517 )))
00518
00519 #ifndef __DOXYGEN__
00520 #   if defined(SIGRD)
00521 #       define __BOOT_SIGROW_READ (_BV(__SPM_ENABLE) | _BV(SIGRD))
00522 #   elif defined(RSIG)
00523 #       define __BOOT_SIGROW_READ (_BV(__SPM_ENABLE) | _BV(RSIG))
00524 #   endif
00525 #endif
00526
00527 /** \ingroup avr_boot
00528     \def boot_signature_byte_get(address)
00529
00530     Read the Signature Row byte at \c address. For some MCU types,
00531     this function can also retrieve the factory-stored oscillator
00532     calibration bytes.
00533
00534     Parameter \c address can be 0-0x1f as documented by the datasheet.
00535     \note The values are MCU type dependent.
00536 */
00537
00538 #define boot_signature_byte_get(addr) \
00539     (__extension__({
00540         uint8_t __result;
00541         __asm__ __volatile__
00542         (
00543             "sts %1, %2" "\n\t"
00544             "lpm %0, Z"
00545             : "=r" (__result)
00546             : "i" (_SFR_MEM_ADDR(__SPM_REG)),
00547               "r" ((uint8_t)(__BOOT_SIGROW_READ)),
00548               "z" ((uint16_t)(addr))
00549             );
00550         __result;
00551     })))
00552
00553 /** \ingroup avr_boot
00554     \def boot_page_fill(address, data)
00555
00556     Fill the bootloader temporary page buffer for flash
00557     address with data word.
00558
00559     \note The address is a byte address. The data is a word. The AVR
00560     writes data to the buffer a word at a time, but addresses the buffer
00561     per byte! So, increment your address by 2 between calls, and send 2
00562     data bytes in a word format! The LSB of the data is written to the lower
00563     address; the MSB of the data is written to the higher address.*/
00564
00565 /** \ingroup avr_boot
00566     \def boot_page_erase(address)
00567
00568     Erase the flash page that contains address.
00569
00570     \note address is a byte address in flash, not a word address. */
00571
00572 /** \ingroup avr_boot
00573     \def boot_page_write(address)
00574
00575     Write the bootloader temporary page buffer
00576     to flash page that contains address.
00577
00578     \note address is a byte address in flash, not a word address. */
00579
00580 /** \ingroup avr_boot
00581     \def boot_rww_enable()
00582
00583     Enable the Read-While-Write memory section. */
00584
00585 /** \ingroup avr_boot
00586     \def boot_lock_bits_set(lock_bits)
00587
00588     Set the bootloader lock bits.
00589
00590     \param lock_bits A mask of which Boot Loader Lock Bits to set.
00591

```

```

00592     \note In this context, a 'set bit' will be written to a zero value.
00593     Note also that only BLBxx bits can be programmed by this command.
00594
00595     For example, to disallow the SPM instruction from writing to the Boot
00596     Loader memory section of flash, you would use this macro as such:
00597
00598     \code
00599     boot_lock_bits_set (_BV (BLB11));
00600     \endcode
00601
00602     \note Like any lock bits, the Boot Loader Lock Bits, once set,
00603     cannot be cleared again except by a chip erase which will in turn
00604     also erase the boot loader itself. */
00605
00606 /* Normal versions of the macros use 16-bit addresses.
00607     Extended versions of the macros use 32-bit addresses.
00608     Alternate versions of the macros use 16-bit addresses and require special
00609     instruction sequences after LPM.
00610
00611     FLASHEND is defined in the ioXXXX.h file.
00612     USHRT_MAX is defined in <limits.h>. */
00613
00614 #if defined(__AVR_ATmega161__) || defined(__AVR_ATmega163__) \
00615     || defined(__AVR_ATmega323__)
00616
00617 /* Alternate: ATmega161/163/323 and 16 bit address */
00618 #define boot_page_fill(address, data) __boot_page_fill_alternate(address, data)
00619 #define boot_page_erase(address)      __boot_page_erase_alternate(address)
00620 #define boot_page_write(address)     __boot_page_write_alternate(address)
00621 #define boot_rww_enable()            __boot_rww_enable_alternate()
00622 #define boot_lock_bits_set(lock_bits) __boot_lock_bits_set_alternate(lock_bits)
00623
00624 #elif (FLASHEND > USHRT_MAX)
00625
00626 /* Extended: >16 bit address */
00627 #define boot_page_fill(address, data) __boot_page_fill_extended(address, data)
00628 #define boot_page_erase(address)     __boot_page_erase_extended(address)
00629 #define boot_page_write(address)    __boot_page_write_extended(address)
00630 #define boot_rww_enable()            __boot_rww_enable()
00631 #define boot_lock_bits_set(lock_bits) __boot_lock_bits_set(lock_bits)
00632
00633 #else
00634
00635 /* Normal: 16 bit address */
00636 #define boot_page_fill(address, data) __boot_page_fill_normal(address, data)
00637 #define boot_page_erase(address)     __boot_page_erase_normal(address)
00638 #define boot_page_write(address)    __boot_page_write_normal(address)
00639 #define boot_rww_enable()            __boot_rww_enable()
00640 #define boot_lock_bits_set(lock_bits) __boot_lock_bits_set(lock_bits)
00641
00642 #endif
00643
00644 /** \ingroup avr_boot
00645
00646     Same as boot_page_fill() except it waits for eeprom and spm operations to
00647     complete before filling the page. */
00648
00649 #define boot_page_fill_safe(address, data) \
00650 do { \
00651     boot_spm_busy_wait();           \
00652     eeprom_busy_wait();             \
00653     boot_page_fill(address, data);  \
00654 } while (0)
00655
00656 /** \ingroup avr_boot
00657
00658     Same as boot_page_erase() except it waits for eeprom and spm operations to
00659     complete before erasing the page. */
00660
00661 #define boot_page_erase_safe(address) \
00662 do { \
00663     boot_spm_busy_wait();           \
00664     eeprom_busy_wait();             \
00665     boot_page_erase (address);      \
00666 } while (0)
00667
00668 /** \ingroup avr_boot
00669
00670     Same as boot_page_write() except it waits for eeprom and spm operations to
00671     complete before writing the page. */
00672
00673 #define boot_page_write_safe(address) \
00674 do { \
00675     boot_spm_busy_wait();           \
00676     eeprom_busy_wait();             \
00677     boot_page_write (address);      \
00678 } while (0)

```

```

00679
00680 /** \ingroup avr_boot
00681
00682     Same as boot_rww_enable() except waits for eeprom and spm operations to
00683     complete before enabling the RWW mameory. */
00684
00685 #define boot_rww_enable_safe() \
00686 do { \
00687     boot_spm_busy_wait();           \
00688     eeprom_busy_wait();           \
00689     boot_rww_enable();           \
00690 } while (0)
00691
00692 /** \ingroup avr_boot
00693
00694     Same as boot_lock_bits_set() except waits for eeprom and spm operations to
00695     complete before setting the lock bits. */
00696
00697 #define boot_lock_bits_set_safe(lock_bits) \
00698 do { \
00699     boot_spm_busy_wait();           \
00700     eeprom_busy_wait();           \
00701     boot_lock_bits_set (lock_bits); \
00702 } while (0)
00703
00704 #endif /* _AVR_BOOT_H_ */

```

## 23.12 builtins.h File Reference

### Macros

- #define `__HAS_DELAY_CYCLES` 1

### Functions

- void `__builtin_avr_sei` (void)
- void `__builtin_avr_cli` (void)
- void `__builtin_avr_sleep` (void)
- void `__builtin_avr_wdr` (void)
- `uint8_t` `__builtin_avr_swap` (`uint8_t` \_\_b)
- `uint16_t` `__builtin_avr_fmul` (`uint8_t` \_\_a, `uint8_t` \_\_b)
- `int16_t` `__builtin_avr_fmuls` (`int8_t` \_\_a, `int8_t` \_\_b)
- `int16_t` `__builtin_avr_fmulsu` (`int8_t` \_\_a, `uint8_t` \_\_b)

## 23.13 builtins.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2008 Anatoly Sokolov
00002     Copyright (c) 2010 Joerg Wunsch
00003     All rights reserved.
00004
00005     Redistribution and use in source and binary forms, with or without
00006     modification, are permitted provided that the following conditions are met:
00007
00008     * Redistributions of source code must retain the above copyright
00009     notice, this list of conditions and the following disclaimer.
00010
00011     * Redistributions in binary form must reproduce the above copyright
00012     notice, this list of conditions and the following disclaimer in
00013     the documentation and/or other materials provided with the
00014     distribution.
00015
00016     * Neither the name of the copyright holders nor the names of
00017     contributors may be used to endorse or promote products derived
00018     from this software without specific prior written permission.
00019
00020     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

```



```

00022 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030 POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033
00034 /*
00035 avr/builtins.h - Intrinsic functions built into the compiler
00036 */
00037
00038 #ifndef _AVR_BUILTINS_H_
00039 #define _AVR_BUILTINS_H_
00040
00041 #ifndef __HAS_DELAY_CYCLES
00042 #define __HAS_DELAY_CYCLES 1
00043 #endif
00044
00045 /* For GCC built-ins, we should not define prototypes,
00046 hence only document that stuff. */
00047 #ifdef __DOXYGEN__
00048
00049 /** \file */
00050 /** \defgroup avr_builtins <avr/builtins.h>: avr-gcc builtins documentation
00051 \code #include <avr/builtins.h> \endcode
00052
00053 \note This file only documents some avr-gcc builtins.
00054 For functions built-in in the compiler, there should be no
00055 prototype declarations.
00056
00057 See also the
00058 <a href="https://gcc.gnu.org/onlinedocs/gcc/AVR-Built-in-Functions.html"
00059 >GCC documentation</a> for a full list of avr-gcc builtins.
00060 */
00061
00062 /**
00063 \ingroup avr_builtins
00064
00065 Enables interrupts by setting the global interrupt mask. */
00066 extern void __builtin_avr_sei(void);
00067
00068 /**
00069 \ingroup avr_builtins
00070
00071 Disables all interrupts by clearing the global interrupt mask. */
00072 extern void __builtin_avr_cli(void);
00073
00074 /**
00075 \ingroup avr_builtins
00076
00077 Emits a \c SLEEP instruction. */
00078
00079 extern void __builtin_avr_sleep(void);
00080
00081 /**
00082 \ingroup avr_builtins
00083
00084 Emits a WDR (watchdog reset) instruction. */
00085 extern void __builtin_avr_wdr(void);
00086
00087 /**
00088 \ingroup avr_builtins
00089
00090 Emits a SWAP (nibble swap) instruction on __b. */
00091 extern uint8_t __builtin_avr_swap(uint8_t __b);
00092
00093 /**
00094 \ingroup avr_builtins
00095
00096 Emits an FMUL (fractional multiply unsigned) instruction. */
00097 extern uint16_t __builtin_avr_fmuls(uint8_t __a, uint8_t __b);
00098
00099 /**
00100 \ingroup avr_builtins
00101
00102 Emits an FMUL (fractional multiply signed) instruction. */
00103 extern int16_t __builtin_avr_fmuls(int8_t __a, int8_t __b);
00104
00105 /**
00106 \ingroup avr_builtins
00107
00108 Emits an FMUL (fractional multiply signed with unsigned) instruction. */

```

```

00109 extern int16_t __builtin_avr_fmulsu(int8_t __a, uint8_t __b);
00110
00111 #if __HAS_DELAY_CYCLES
00112 /**
00113     \ingroup avr_builtins
00114
00115     Emits a sequence of instructions causing the CPU to spend
00116     \c __n cycles on it. */
00117 extern void __builtin_avr_delay_cycles(uint32_t __n);
00118 #endif
00119 #endif /* DOXYGEN */
00120 #endif /* _AVR_BUILTINS_H_ */

```

## 23.14 cpufunc.h File Reference

### Macros

- `#define _NOP()`
- `#define _MemoryBarrier()`

### Functions

- void `ccp_write_io` (volatile `uint8_t *` \_\_ioaddr, `uint8_t` \_\_value)
- void `ccp_write_spm` (volatile `uint8_t *` \_\_ioaddr, `uint8_t` \_\_value)

## 23.15 cpufunc.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2010, Joerg Wunsch
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011     notice, this list of conditions and the following disclaimer in
00012     the documentation and/or other materials provided with the
00013     distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016     contributors may be used to endorse or promote products derived
00017     from this software without specific prior written permission.
00018
00019     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029     POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /* avr/cpufunc.h - Special CPU functions */
00034
00035 #ifndef _AVR_CPUFUNC_H_
00036 #define _AVR_CPUFUNC_H_ 1
00037
00038 #include <stdint.h>
00039
00040 /** \file */
00041 /** \defgroup avr_cpufunc <avr/cpufunc.h>: Special AVR CPU functions
00042     \code #include <avr/cpufunc.h> \endcode
00043

```

```

00044     This header file contains macros that access special functions of
00045     the AVR CPU which do not fit into any of the other header files.
00046
00047 */
00048
00049 #if defined(__DOXYGEN__)
00050 /**
00051     \ingroup avr_cpufunc
00052     \def _NOP
00053
00054     Execute a <i>no operation</i> (NOP) CPU instruction. This
00055     should not be used to implement delays, better use the functions
00056     from <util/delay_basic.h> or <util/delay.h> for this. For
00057     debugging purposes, a NOP can be useful to have an instruction that
00058     is guaranteed to be not optimized away by the compiler, so it can
00059     always become a breakpoint in the debugger.
00060 */
00061 #define _NOP()
00062 #else /* real code */
00063 #define _NOP() __asm__ __volatile__("nop")
00064 #endif /* __DOXYGEN__ */
00065
00066 #if defined(__DOXYGEN__)
00067 /**
00068     \ingroup avr_cpufunc
00069     \def _MemoryBarrier
00070
00071     Implement a read/write <i>memory barrier</i>. A memory
00072     barrier instructs the compiler to not cache any memory data in
00073     registers beyond the barrier. This can sometimes be more effective
00074     than blocking certain optimizations by declaring some object with a
00075     \c volatile qualifier.
00076
00077     See \ref optim_code_reorder for things to be taken into account
00078     with respect to compiler optimizations.
00079 */
00080 #define _MemoryBarrier()
00081 #else /* real code */
00082 #define _MemoryBarrier() __asm__ __volatile__("":::"memory")
00083 #endif /* __DOXYGEN__ */
00084
00085 #ifndef __cplusplus
00086 extern "C" {
00087 #endif
00088
00089 /**
00090     \ingroup avr_cpufunc
00091
00092     Write \a __value to IO Register Protected (CCP) IO register
00093     at \a __ioaddr. . See also \c _PROTECTED_WRITE(). */
00094 void ccp_write_io (volatile uint8_t *__ioaddr, uint8_t __value);
00095
00096 /**
00097     \ingroup avr_cpufunc
00098
00099     Write \a __value to SPM Instruction Protected (CCP) IO register
00100     at \a __ioaddr. See also \c _PROTECTED_WRITE_SPM(). */
00101 void ccp_write_spm (volatile uint8_t *__ioaddr, uint8_t __value);
00102
00103 #ifndef __cplusplus
00104 }
00105 #endif
00106
00107 #endif /* _AVR_CPUFUNC_H_ */

```

## 23.16 eeprom.h

```

00001 /* Copyright (c) 2002, 2003, 2004, 2007 Marek Michalkiewicz
00002     Copyright (c) 2005, 2006 Bjoern Haase
00003     Copyright (c) 2008 Atmel Corporation
00004     Copyright (c) 2008 Wouter van Gulik
00005     Copyright (c) 2009 Dmitry Xmelkov
00006     All rights reserved.
00007
00008     Redistribution and use in source and binary forms, with or without
00009     modification, are permitted provided that the following conditions are met:
00010
00011     * Redistributions of source code must retain the above copyright
00012     notice, this list of conditions and the following disclaimer.
00013     * Redistributions in binary form must reproduce the above copyright
00014     notice, this list of conditions and the following disclaimer in
00015     the documentation and/or other materials provided with the
00016     distribution.
00017     * Neither the name of the copyright holders nor the names of

```

```

00018     contributors may be used to endorse or promote products derived
00019     from this software without specific prior written permission.
00020
00021     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00022     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00023     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00024     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00025     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00026     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00027     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00028     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00029     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00030     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00031     POSSIBILITY OF SUCH DAMAGE. */
00032
00033 /* $Id$ */
00034
00035 #ifndef _AVR_EEPROM_H_
00036 #define _AVR_EEPROM_H_ 1
00037
00038 #include <avr/io.h>
00039
00040 #if !E2END && !defined(__DOXYGEN__) && !defined(__COMPILING_AVR_LIBC__)
00041 # warning "Device does not have EEPROM available."
00042 #else
00043
00044 #if defined (EEAR) && !defined (EEARL) && !defined (EEARH)
00045 #define EEARL EEAR
00046 #endif
00047
00048 #ifndef __ASSEMBLER__
00049
00050 #include <stddef.h> /* size_t */
00051 #include <stdint.h>
00052
00053 /** \defgroup avr_eeprom <avr/eeprom.h>: EEPROM handling
00054     \code #include <avr/eeprom.h> \endcode
00055
00056     This header file declares the interface to some simple library
00057     routines suitable for handling the data EEPROM contained in the
00058     AVR microcontrollers. The implementation uses a simple polled
00059     mode interface. Applications that require interrupt-controlled
00060     EEPROM access to ensure that no time will be wasted in spinloops
00061     will have to deploy their own implementation.
00062
00063     \par Notes:
00064
00065     - In addition to the write functions there is a set of update ones.
00066     This functions read each byte first and skip the burning if the
00067     old value is the same with new. The scanning direction is from
00068     high address to low, to obtain quick return in common cases.
00069
00070     - All of the read/write functions first make sure the EEPROM is
00071     ready to be accessed. Since this may cause long delays if a
00072     write operation is still pending, time-critical applications
00073     should first poll the EEPROM e. g. using eeprom_is_ready() before
00074     attempting any actual I/O. But this functions does not wait until
00075     SELFPGEN in SPMCSR becomes zero. Do this manually, if your
00076     softwate contains the Flash burning.
00077
00078     - As these functions modify IO registers, they are known to be
00079     non-reentrant. If any of these functions are used from both,
00080     standard and interrupt context, the applications must ensure
00081     proper protection (e.g. by disabling interrupts before accessing
00082     them).
00083
00084     - All write functions force erase_and_write programming mode.
00085
00086     - For Xmega the EEPROM start address is 0, like other architectures.
00087     The reading functions add the 0x2000 value to use EEPROM mapping into
00088     data space.
00089 */
00090
00091 #ifdef __cplusplus
00092 extern "C" {
00093 #endif
00094
00095 #ifndef __ATTR_PURE__
00096 # ifdef __DOXYGEN__
00097 # define __ATTR_PURE__
00098 # else
00099 # define __ATTR_PURE__ __attribute__((__pure__))
00100 # endif
00101 #endif
00102
00103 /** \def EEMEM
00104     \ingroup avr_eeprom

```

```

00105     Attribute expression causing a variable to be allocated within the
00106     .eeprom section. */
00107 #define EEMEM __attribute__((__section__(".eeprom")))
00108
00109 /** \def eeprom_is_ready
00110     \ingroup avr_eeprom
00111     \returns 1 if EEPROM is ready for a new read/write operation, 0 if not.
00112 */
00113 #if defined (__DOXYGEN__)
00114 # define eeprom_is_ready()
00115 #elif defined (NVM_STATUS)
00116 # define eeprom_is_ready() bit_is_clear (NVM_STATUS, NVM_NVMBUSY_bp)
00117 #elif defined (NVMCTRL_STATUS)
00118 # define eeprom_is_ready() bit_is_clear (NVMCTRL_STATUS, NVMCTRL_EEBUSY_bp)
00119 #elif defined (DEECCR)
00120 # define eeprom_is_ready() bit_is_clear (DEECCR, BSY)
00121 #elif defined (EEPE)
00122 # define eeprom_is_ready() bit_is_clear (EECR, EEPE)
00123 #else
00124 # define eeprom_is_ready() bit_is_clear (EECR, EEWE)
00125 #endif
00126
00127
00128 /** \def eeprom_busy_wait
00129     \ingroup avr_eeprom
00130     Loops until the eeprom is no longer busy.
00131     \returns Nothing.
00132 */
00133 #define eeprom_busy_wait() do {} while (!eeprom_is_ready())
00134
00135
00136 /** \ingroup avr_eeprom
00137     Read one byte from EEPROM address \a __p.
00138 */
00139 uint8_t eeprom_read_byte (const uint8_t *__p) __ATTR_PURE__;
00140
00141 /** \ingroup avr_eeprom
00142     Read one 16-bit word (little endian) from EEPROM address \a __p.
00143 */
00144 uint16_t eeprom_read_word (const uint16_t *__p) __ATTR_PURE__;
00145
00146 /** \ingroup avr_eeprom
00147     Read one 32-bit double word (little endian) from EEPROM address \a __p.
00148 */
00149 uint32_t eeprom_read_dword (const uint32_t *__p) __ATTR_PURE__;
00150
00151 /** \ingroup avr_eeprom
00152     Read one 64-bit quad word (little endian) from EEPROM address \a __p.
00153 */
00154 #if defined(__DOXYGEN__) || __SIZEOF_LONG_LONG__ == 8
00155 uint64_t eeprom_read_qword (const uint64_t *__p) __ATTR_PURE__;
00156 #endif
00157
00158 /** \ingroup avr_eeprom
00159     Read one float value (little endian) from EEPROM address \a __p.
00160 */
00161 float eeprom_read_float (const float *__p) __ATTR_PURE__;
00162
00163 /** \ingroup avr_eeprom
00164     Read one double value (little endian) from EEPROM address \a __p.
00165 */
00166 #if defined(__DOXYGEN__)
00167 double eeprom_read_double (const double *__p);
00168 #elif __SIZEOF_DOUBLE__ == 4
00169 double eeprom_read_double (const double *__p) __asm("eeprom_read_dword");
00170 #elif __SIZEOF_DOUBLE__ == 8
00171 double eeprom_read_double (const double *__p) __asm("eeprom_read_qword");
00172 #endif
00173
00174 /** \ingroup avr_eeprom
00175     Read one long double value (little endian) from EEPROM address \a __p.
00176 */
00177 #if defined(__DOXYGEN__)
00178 long double eeprom_read_long_double (const long double *__p);
00179 #elif __SIZEOF_LONG_DOUBLE__ == 4
00180 long double eeprom_read_long_double (const long double *__p) __asm("eeprom_read_dword");
00181 #elif __SIZEOF_LONG_DOUBLE__ == 8
00182 long double eeprom_read_long_double (const long double *__p) __asm("eeprom_read_qword");
00183 #endif
00184
00185 /** \ingroup avr_eeprom
00186     Read a block of \a __n bytes from EEPROM address \a __src to SRAM
00187     \a __dst.
00188 */
00189 void eeprom_read_block (void *__dst, const void *__src, size_t __n);
00190
00191

```

```
00192 /** \ingroup avr_eeprom
00193 Write a byte \a __value to EEPROM address \a __p.
00194 */
00195 void eeprom_write_byte (uint8_t *__p, uint8_t __value);
00196
00197 /** \ingroup avr_eeprom
00198 Write a word \a __value to EEPROM address \a __p.
00199 */
00200 void eeprom_write_word (uint16_t *__p, uint16_t __value);
00201
00202 /** \ingroup avr_eeprom
00203 Write a 32-bit double word \a __value to EEPROM address \a __p.
00204 */
00205 void eeprom_write_dword (uint32_t *__p, uint32_t __value);
00206
00207 /** \ingroup avr_eeprom
00208 Write a 64-bit quad word \a __value to EEPROM address \a __p.
00209 */
00210 #if defined(__DOXYGEN__) || __SIZEOF_LONG_LONG__ == 8
00211 void eeprom_write_qword (uint64_t *__p, uint64_t __value);
00212 #endif
00213
00214 /** \ingroup avr_eeprom
00215 Write a float \a __value to EEPROM address \a __p.
00216 */
00217 void eeprom_write_float (float *__p, float __value);
00218
00219 /** \ingroup avr_eeprom
00220 Write a double \a __value to EEPROM address \a __p.
00221 */
00222 #if defined(__DOXYGEN__)
00223 void eeprom_write_double (double *__p, double __value);
00224 #elif __SIZEOF_DOUBLE__ == 4
00225 void eeprom_write_double (double *__p, double __value) __asm("eeprom_write_dword");
00226 #elif __SIZEOF_DOUBLE__ == 8
00227 void eeprom_write_double (double *__p, double __value) __asm("eeprom_write_qword");
00228 #endif
00229
00230 /** \ingroup avr_eeprom
00231 Write a long double \a __value to EEPROM address \a __p.
00232 */
00233 #if defined(__DOXYGEN__)
00234 void eeprom_write_long_double (long double *__p, long double __value);
00235 #elif __SIZEOF_LONG_DOUBLE__ == 4
00236 void eeprom_write_long_double (long double *__p, long double __value) __asm("eeprom_write_dword");
00237 #elif __SIZEOF_LONG_DOUBLE__ == 8
00238 void eeprom_write_long_double (long double *__p, long double __value) __asm("eeprom_write_qword");
00239 #endif
00240
00241 /** \ingroup avr_eeprom
00242 Write a block of \a __n bytes to EEPROM address \a __dst from \a __src.
00243 \note The argument order is mismatch with common functions like strcpy().
00244 */
00245 void eeprom_write_block (const void *__src, void *__dst, size_t __n);
00246
00247
00248 /** \ingroup avr_eeprom
00249 Update a byte \a __value at EEPROM address \a __p.
00250 */
00251 void eeprom_update_byte (uint8_t *__p, uint8_t __value);
00252
00253 /** \ingroup avr_eeprom
00254 Update a word \a __value at EEPROM address \a __p.
00255 */
00256 void eeprom_update_word (uint16_t *__p, uint16_t __value);
00257
00258 /** \ingroup avr_eeprom
00259 Update a 32-bit double word \a __value at EEPROM address \a __p.
00260 */
00261 void eeprom_update_dword (uint32_t *__p, uint32_t __value);
00262
00263 /** \ingroup avr_eeprom
00264 Update a 64-bit quad word \a __value at EEPROM address \a __p.
00265 */
00266 #if defined(__DOXYGEN__) || __SIZEOF_LONG_LONG__ == 8
00267 void eeprom_update_qword (uint64_t *__p, uint64_t __value);
00268 #endif
00269
00270 /** \ingroup avr_eeprom
00271 Update a float \a __value at EEPROM address \a __p.
00272 */
00273 void eeprom_update_float (float *__p, float __value);
00274
00275 /** \ingroup avr_eeprom
00276 Update a double \a __value at EEPROM address \a __p.
00277 */
00278 #if defined(__DOXYGEN__)
```

```

00279 void eeprom_update_double (double *__p, double __value);
00280 #elif __SIZEOF_DOUBLE__ == 4
00281 void eeprom_update_double (double *__p, double __value) __asm("eeprom_update_dword");
00282 #elif __SIZEOF_DOUBLE__ == 8
00283 void eeprom_update_double (double *__p, double __value) __asm("eeprom_update_qword");
00284 #endif
00285
00286 /** \ingroup avr_eeprom
00287     Update a long double \a __value at EEPROM address \a __p.
00288 */
00289 #if defined(__DOXYGEN__)
00290 void eeprom_update_long_double (long double *__p, long double __value);
00291 #elif __SIZEOF_LONG_DOUBLE__ == 4
00292 void eeprom_update_long_double (long double *__p, long double __value) __asm("eeprom_update_dword");
00293 #elif __SIZEOF_LONG_DOUBLE__ == 8
00294 void eeprom_update_long_double (long double *__p, long double __value) __asm("eeprom_update_qword");
00295 #endif
00296
00297 /** \ingroup avr_eeprom
00298     Update a block of \a __n bytes at EEPROM address \a __dst from \a __src.
00299     \note The argument order is mismatch with common functions like strcpy().
00300 */
00301 void eeprom_update_block (const void *__src, void *__dst, size_t __n);
00302
00303
00304 /** \name IAR C compatibility defines */
00305 /**@{*/
00306
00307 /** \def __EEPWRITE
00308     \ingroup avr_eeprom
00309     Write a byte to EEPROM. Compatibility define for IAR C. */
00310 #define __EEPWRITE(addr, val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))
00311
00312 /** \def __EEPWRITE
00313     \ingroup avr_eeprom
00314     Write a byte to EEPROM. Compatibility define for IAR C. */
00315 #define __EEPWRITE(addr, val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))
00316
00317 /** \def __EEGET
00318     \ingroup avr_eeprom
00319     Read a byte from EEPROM. Compatibility define for IAR C. */
00320 #define __EEGET(var, addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))
00321
00322 /** \def __EEGET
00323     \ingroup avr_eeprom
00324     Read a byte from EEPROM. Compatibility define for IAR C. */
00325 #define __EEGET(var, addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))
00326
00327 /**@}*/
00328
00329 #ifdef __cplusplus
00330 }
00331 #endif
00332
00333 #endif /* !__ASSEMBLER__ */
00334 #endif /* E2END || defined(__DOXYGEN__) || defined(__COMPILING_AVR_LIBC__) */
00335 #endif /* !_AVR_EEPROM_H_ */

```

## 23.17 fuse.h File Reference

### 23.18 fuse.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2007, Atmel Corporation
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011       notice, this list of conditions and the following disclaimer in
00012       the documentation and/or other materials provided with the
00013       distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016       contributors may be used to endorse or promote products derived
00017       from this software without specific prior written permission.

```

```

00018
00019 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029 POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /* avr/fuse.h - Fuse API */
00034
00035 #ifndef _AVR_FUSE_H_
00036 #define _AVR_FUSE_H_ 1
00037
00038 /* This file must be explicitly included by <avr/io.h>. */
00039 #if !defined(_AVR_IO_H_)
00040 #error "You must #include <avr/io.h> and not <avr/fuse.h> by itself."
00041 #endif
00042
00043
00044 /** \file */
00045 /** \defgroup avr_fuse <avr/fuse.h>: Fuse Support
00046
00047     \par Introduction
00048
00049     The Fuse API allows a user to specify the fuse settings for the specific
00050     AVR device they are compiling for. These fuse settings will be placed
00051     in a special section in the ELF output file, after linking.
00052
00053     Programming tools can take advantage of the fuse information embedded in
00054     the ELF file, by extracting this information and determining if the fuses
00055     need to be programmed before programming the Flash and EEPROM memories.
00056     This also allows a single ELF file to contain all the
00057     information needed to program an AVR.
00058
00059     To use the Fuse API, include the <avr/io.h> header file, which in turn
00060     automatically includes the individual I/O header file and the <avr/fuse.h>
00061     file. These other two files provides everything necessary to set the AVR
00062     fuses.
00063
00064     \par Fuse API
00065
00066     Each I/O header file must define the FUSE_MEMORY_SIZE macro which is
00067     defined to the number of fuse bytes that exist in the AVR device.
00068
00069     A new type, __fuse_t, is defined as a structure. The number of fields in
00070     this structure are determined by the number of fuse bytes in the
00071     FUSE_MEMORY_SIZE macro.
00072
00073     If FUSE_MEMORY_SIZE == 1, there is only a single field: byte, of type
00074     unsigned char.
00075
00076     If FUSE_MEMORY_SIZE == 2, there are two fields: low, and high, of type
00077     unsigned char.
00078
00079     If FUSE_MEMORY_SIZE == 3, there are three fields: low, high, and extended,
00080     of type unsigned char.
00081
00082     If FUSE_MEMORY_SIZE > 3, there is a single field: byte, which is an array
00083     of unsigned char with the size of the array being FUSE_MEMORY_SIZE.
00084
00085     A convenience macro, FUSEMEM, is defined as a GCC attribute for a
00086     custom-named section of ".fuse".
00087
00088     A convenience macro, FUSES, is defined that declares a variable, __fuse, of
00089     type __fuse_t with the attribute defined by FUSEMEM. This variable
00090     allows the end user to easily set the fuse data.
00091
00092     \note If a device-specific I/O header file has previously defined FUSEMEM,
00093     then FUSEMEM is not redefined. If a device-specific I/O header file has
00094     previously defined FUSES, then FUSES is not redefined.
00095
00096     Each AVR device I/O header file has a set of defined macros which specify the
00097     actual fuse bits available on that device. The AVR fuses have inverted
00098     values, logical 1 for an unprogrammed (disabled) bit and logical 0 for a
00099     programmed (enabled) bit. The defined macros for each individual fuse
00100     bit represent this in their definition by a bit-wise inversion of a mask.
00101     For example, the FUSE_EESAVE fuse in the ATmega128 is defined as:
00102     \code
00103     #define FUSE_EESAVE      ~_BV(3)
00104     \endcode

```



```

00105 \note The _BV macro creates a bit mask from a bit number. It is then
00106 inverted to represent logical values for a fuse memory byte.
00107
00108 To combine the fuse bits macros together to represent a whole fuse byte,
00109 use the bitwise AND operator, like so:
00110 \code
00111 (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN)
00112 \endcode
00113
00114 Each device I/O header file also defines macros that provide default values
00115 for each fuse byte that is available. LFUSE_DEFAULT is defined for a Low
00116 Fuse byte. HFUSE_DEFAULT is defined for a High Fuse byte. EFUSE_DEFAULT
00117 is defined for an Extended Fuse byte.
00118
00119 If FUSE_MEMORY_SIZE > 3, then the I/O header file defines macros that
00120 provide default values for each fuse byte like so:
00121 FUSE0_DEFAULT
00122 FUSE1_DEFAULT
00123 FUSE2_DEFAULT
00124 FUSE3_DEFAULT
00125 FUSE4_DEFAULT
00126 ....
00127
00128 \par API Usage Example
00129
00130 Putting all of this together is easy. Using C99's designated initializers:
00131
00132 \code
00133 #include <avr/io.h>
00134
00135 FUSES =
00136 {
00137     .low = LFUSE_DEFAULT,
00138     .high = (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN),
00139     .extended = EFUSE_DEFAULT,
00140 };
00141
00142 int main(void)
00143 {
00144     return 0;
00145 }
00146 \endcode
00147
00148 Or, using the variable directly instead of the FUSES macro,
00149
00150 \code
00151 #include <avr/io.h>
00152
00153 __fuse_t __fuse __attribute__((section (".fuse"))) =
00154 {
00155     .low = LFUSE_DEFAULT,
00156     .high = (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN),
00157     .extended = EFUSE_DEFAULT,
00158 };
00159
00160 int main(void)
00161 {
00162     return 0;
00163 }
00164 \endcode
00165
00166 If you are compiling in C++, you cannot use the designated initializers so
00167 you must do:
00168
00169 \code
00170 #include <avr/io.h>
00171
00172 FUSES =
00173 {
00174     LFUSE_DEFAULT, // .low
00175     (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN), // .high
00176     EFUSE_DEFAULT, // .extended
00177 };
00178
00179 int main(void)
00180 {
00181     return 0;
00182 }
00183 \endcode
00184
00185
00186 However there are a number of caveats that you need to be aware of to
00187 use this API properly.
00188
00189 Be sure to include <avr/io.h> to get all of the definitions for the API.
00190 The FUSES macro defines a global variable to store the fuse data. This
00191 variable is assigned to its own linker section. Assign the desired fuse

```

```

00192     values immediately in the variable initialization.
00193
00194     The .fuse section in the ELF file will get its values from the initial
00195     variable assignment ONLY. This means that you can NOT assign values to
00196     this variable in functions and the new values will not be put into the
00197     ELF .fuse section.
00198
00199     The global variable is declared in the FUSES macro has two leading
00200     underscores, which means that it is reserved for the "implementation",
00201     meaning the library, so it will not conflict with a user-named variable.
00202
00203     You must initialize ALL fields in the __fuse_t structure. This is because
00204     the fuse bits in all bytes default to a logical 1, meaning unprogrammed.
00205     Normal uninitialized data defaults to all logical zeros. So it is vital that
00206     all fuse bytes are initialized, even with default data. If they are not,
00207     then the fuse bits may not be programmed to the desired settings.
00208
00209     Be sure to have the -mmcu=<em>device</em> flag in your compile command line and
00210     your linker command line to have the correct device selected and to have
00211     the correct I/O header file included when you include <avr/io.h>.
00212
00213     You can print out the contents of the .fuse section in the ELF file by
00214     using this command line:
00215     \code
00216     avr-objdump -s -j .fuse <ELF file>
00217     \endcode
00218     The section contents shows the address on the left, then the data going from
00219     lower address to a higher address, left to right.
00220
00221 */
00222
00223 #if !(defined(__ASSEMBLER__) || defined(__DOXYGEN__))
00224
00225 #ifndef FUSEMEM
00226 #define FUSEMEM __attribute__((used, section(".fuse")))
00227 #endif
00228
00229 #if FUSE_MEMORY_SIZE > 3
00230
00231 typedef struct
00232 {
00233     unsigned char byte[FUSE_MEMORY_SIZE];
00234 } __fuse_t;
00235
00236 #elif FUSE_MEMORY_SIZE == 3
00237
00238 typedef struct
00239 {
00240     unsigned char low;
00241     unsigned char high;
00242     unsigned char extended;
00243 } __fuse_t;
00244
00245 #elif FUSE_MEMORY_SIZE == 2
00246
00247 typedef struct
00248 {
00249     unsigned char low;
00250     unsigned char high;
00251 } __fuse_t;
00252
00253 #elif FUSE_MEMORY_SIZE == 1
00254
00255 typedef struct
00256 {
00257     unsigned char byte;
00258 } __fuse_t;
00259 #endif
00260
00261 #endif
00262
00263 #if !defined(FUSES)
00264 #if defined(__AVR_XMEGA__)
00265 #define FUSES NVM_FUSES_t __fuse FUSEMEM
00266 #else
00267 #define FUSES __fuse_t __fuse FUSEMEM
00268 #endif
00269 #endif
00270
00271
00272 #endif /* !(__ASSEMBLER__ || __DOXYGEN__) */
00273
00274 #endif /* _AVR_FUSE_H_ */

```

## 23.19 interrupt.h File Reference

### Macros

#### Global manipulation of the interrupt flag

The global interrupt flag is maintained in the I bit of the status register (SREG).

Handling interrupts frequently requires attention regarding atomic access to objects that could be altered by code running within an interrupt context, see [<util/atomic.h>](#).

Frequently, interrupts are being disabled for periods of time in order to perform certain operations without being disturbed; see [Problems with reordering code](#) for things to be taken into account with respect to compiler optimizations.

- #define `sei()` `__asm__ __volatile__ ("sei" ::: "memory")`
- #define `cli()` `__asm__ __volatile__ ("cli" ::: "memory")`

#### Macros for writing interrupt handler functions

- #define `ISR(vector, attributes)`
- #define `SIGNAL(vector)`
- #define `EMPTY_INTERRUPT(vector)`
- #define `ISR_ALIAS(vector, target_vector)`
- #define `reti()` `__asm__ __volatile__ ("reti" ::: "memory")`
- #define `BADISR_vect`

#### ISR attributes

- #define `ISR_BLOCK`
- #define `ISR_NOBLOCK`
- #define `ISR_NAKED`
- #define `ISR_FLATTEN`
- #define `ISR_NOICF`
- #define `ISR_NOGCCISR`
- #define `ISR_ALIASOF(target_vector)`

## 23.20 interrupt.h

[Go to the documentation of this file.](#)

```
00001 /* Copyright (c) 2002,2005,2007 Marek Michalkiewicz
00002    Copyright (c) 2007, Dean Camera
00003
00004    All rights reserved.
00005
00006    Redistribution and use in source and binary forms, with or without
00007    modification, are permitted provided that the following conditions are met:
00008
00009    * Redistributions of source code must retain the above copyright
00010      notice, this list of conditions and the following disclaimer.
00011
00012    * Redistributions in binary form must reproduce the above copyright
00013      notice, this list of conditions and the following disclaimer in
00014      the documentation and/or other materials provided with the
00015      distribution.
00016
00017    * Neither the name of the copyright holders nor the names of
00018      contributors may be used to endorse or promote products derived
00019      from this software without specific prior written permission.
00020
00021    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00022    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00023    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00024    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00025    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00026    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00027    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00028    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
```

```

00029 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00030 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00031 POSSIBILITY OF SUCH DAMAGE. */
00032
00033 /* $Id$ */
00034
00035 #ifndef _AVR_INTERRUPT_H_
00036 #define _AVR_INTERRUPT_H_
00037
00038 #include <avr/io.h>
00039
00040 #if !defined(__DOXYGEN__) && !defined(__STRINGIFY)
00041 /* Auxiliary macro for ISR_ALIAS(). */
00042 #define __STRINGIFY(x) #x
00043 #endif /* !defined(__DOXYGEN__) */
00044
00045 /** \file */
00046 /**@{*/
00047
00048
00049 /** \name Global manipulation of the interrupt flag
00050
00051 The global interrupt flag is maintained in the I bit of the status
00052 register (SREG).
00053
00054 Handling interrupts frequently requires attention regarding atomic
00055 access to objects that could be altered by code running within an
00056 interrupt context, see <util/atomic.h>.
00057
00058 Frequently, interrupts are being disabled for periods of time in
00059 order to perform certain operations without being disturbed; see
00060 \ref optim_code_reorder for things to be taken into account with
00061 respect to compiler optimizations.
00062 */
00063
00064 /** \def sei()
00065 \ingroup avr_interrupts
00066
00067 Enables interrupts by setting the global interrupt mask. This function
00068 actually compiles into a single line of assembly, so there is no function
00069 call overhead. However, the macro also implies a <i>memory barrier</i>
00070 which can cause additional loss of optimization.
00071
00072 In order to implement atomic access to multi-byte objects,
00073 consider using the macros from <util/atomic.h>, rather than
00074 implementing them manually with cli() and sei().
00075 */
00076 # define sei() __asm__ __volatile__ ("sei" ::: "memory")
00077
00078 /** \def cli()
00079 \ingroup avr_interrupts
00080
00081 Disables all interrupts by clearing the global interrupt mask. This function
00082 actually compiles into a single line of assembly, so there is no function
00083 call overhead. However, the macro also implies a <i>memory barrier</i>
00084 which can cause additional loss of optimization.
00085
00086 In order to implement atomic access to multi-byte objects,
00087 consider using the macros from <util/atomic.h>, rather than
00088 implementing them manually with cli() and sei().
00089 */
00090 # define cli() __asm__ __volatile__ ("cli" ::: "memory")
00091
00092
00093 /** \name Macros for writing interrupt handler functions */
00094
00095
00096 #if defined(__DOXYGEN__)
00097 /** \def ISR(vector [, attributes])
00098 \ingroup avr_interrupts
00099
00100 Introduces an interrupt handler function (interrupt service
00101 routine) that runs with global interrupts initially disabled
00102 by default with no attributes specified.
00103
00104 The attributes are optional and alter the behaviour and resultant
00105 generated code of the interrupt routine. Multiple attributes may
00106 be used for a single function, with a space separating each
00107 attribute.
00108
00109 Valid attributes are #ISR_BLOCK, #ISR_NOBLOCK, #ISR_NAKED,
00110 #ISR_FLATTEN, #ISR_NOICF, #ISR_NOGCCISR and ISR_ALIASOF(vect).
00111
00112 \c vector must be one of the interrupt vector names that are
00113 valid for the particular MCU type.
00114 */
00115 # define ISR(vector, [attributes])

```

```

00116 #else /* real code */
00117
00118 #if (__GNUC__ == 4 && __GNUC_MINOR__ >= 1) || (__GNUC__ > 4)
00119 # define __INTR_ATTRS __used__, __externally_visible__
00120 #else /* GCC < 4.1 */
00121 # define __INTR_ATTRS __used__
00122 #endif
00123
00124 #ifdef __cplusplus
00125 # define ISR(vector, ...) \
00126     extern "C" void vector (void) __attribute__ ((__signal__, __INTR_ATTRS)) __VA_ARGS__; \
00127     void vector (void)
00128 #else
00129 # define ISR(vector, ...) \
00130     void vector (void) __attribute__ ((__signal__, __INTR_ATTRS)) __VA_ARGS__; \
00131     void vector (void)
00132 #endif
00133
00134 #endif /* DOXYGEN */
00135
00136 #if defined(__DOXYGEN__)
00137 /** \def SIGNAL(vector)
00138     \ingroup avr_interrupts
00139
00140     Introduces an interrupt handler function that runs with global interrupts
00141     initially disabled.
00142
00143     This is the same as the ISR macro without optional attributes.
00144     \deprecated Do not use SIGNAL() in new code. Use ISR() instead.
00145 */
00146 # define SIGNAL(vector)
00147 #else /* real code */
00148
00149 #ifdef __cplusplus
00150 # define SIGNAL(vector) \
00151     extern "C" void vector(void) __attribute__ ((__signal__, __INTR_ATTRS)); \
00152     void vector (void)
00153 #else
00154 # define SIGNAL(vector) \
00155     void vector (void) __attribute__ ((__signal__, __INTR_ATTRS)); \
00156     void vector (void)
00157 #endif
00158
00159 #endif /* DOXYGEN */
00160
00161 #if defined(__DOXYGEN__)
00162 /** \def EMPTY_INTERRUPT(vector)
00163     \ingroup avr_interrupts
00164
00165     Defines an empty interrupt handler function. This will not generate
00166     any prolog or epilog code and will only return from the #ISR. Do not
00167     define a function body as this will define it for you.
00168     Example:
00169     \code EMPTY_INTERRUPT(ADC_vect);\endcode */
00170 # define EMPTY_INTERRUPT(vector)
00171 #else /* real code */
00172
00173 #ifdef __cplusplus
00174 # define EMPTY_INTERRUPT(vector) \
00175     extern "C" void vector(void) __attribute__ ((__signal__, __naked__, __INTR_ATTRS)); \
00176     void vector (void) { __asm__ __volatile__ ("reti" ::: "memory"); }
00177 #else
00178 # define EMPTY_INTERRUPT(vector) \
00179     void vector (void) __attribute__ ((__signal__, __naked__, __INTR_ATTRS)); \
00180     void vector (void) { __asm__ __volatile__ ("reti" ::: "memory"); }
00181 #endif
00182
00183 #endif /* DOXYGEN */
00184
00185 #if defined(__DOXYGEN__)
00186 /** \def ISR_ALIAS(vector, target_vector)
00187     \ingroup avr_interrupts
00188
00189     Aliases a given vector to another one in the same manner as the
00190     ISR_ALIASOF attribute for the ISR() macro. Unlike the ISR_ALIASOF
00191     attribute macro however, this is compatible for all versions of
00192     GCC rather than just GCC version 4.2 onwards.
00193
00194     \note This macro creates a trampoline function for the aliased
00195     macro. This will result in a two cycle penalty for the aliased
00196     vector compared to the ISR the vector is aliased to, due to the
00197     JMP/RJMP opcode used.
00198
00199     \deprecated
00200     For new code, the use of ISR(..., ISR_ALIASOF(...)) is
00201     recommended.
00202

```

```

00203     Example:
00204     \code
00205     ISR(INT0_vect)
00206     {
00207         PORTB = 42;
00208     }
00209
00210     ISR_ALIAS(INT1_vect, INT0_vect);
00211     \endcode
00212
00213 */
00214 # define ISR_ALIAS(vector, target_vector)
00215 #else /* real code */
00216
00217 #ifdef __cplusplus
00218 #   define ISR_ALIAS(vector, tgt) extern "C" void vector (void) \
00219         __attribute__((__signal__, __naked__, __INTR_ATTRS)); \
00220         void vector (void) { __asm__ __volatile__ ("%~jmp " __STRINGIFY(tgt) "::); }
00221 #else /* !__cplusplus */
00222 #   define ISR_ALIAS(vector, tgt) void vector (void) \
00223         __attribute__((__signal__, __naked__, __INTR_ATTRS)); \
00224         void vector (void) { __asm__ __volatile__ ("%~jmp " __STRINGIFY(tgt) "::); }
00225 #endif /* __cplusplus */
00226
00227 #endif /* DOXYGEN */
00228
00229 /** \def reti()
00230     \ingroup avr_interrupts
00231
00232     Returns from an interrupt routine, enabling global interrupts. This should
00233     be the last command executed before leaving an #ISR defined with the
00234     #ISR_NAKED attribute.
00235
00236     This macro actually compiles into a single line of assembly, so there is
00237     no function call overhead.
00238
00239     \note According to the GCC documentation, the only code supported in
00240     naked functions is \ref inline_asm "inline assembly".
00241 */
00242 # define reti() __asm__ __volatile__ ("reti" ::: "memory")
00243
00244 #if defined(__DOXYGEN__)
00245 /** \def BADISR_vect
00246     \ingroup avr_interrupts
00247
00248     \code #include <avr/interrupt.h> \endcode
00249
00250     This is a vector which is aliased to __vector_default, the vector
00251     executed when an IRQ fires with no accompanying ISR handler. This
00252     may be used along with the ISR() macro to create a catch-all for
00253     undefined but used ISRs for debugging purposes.
00254 */
00255 # define BADISR_vect
00256 #else /* !DOXYGEN */
00257 # define BADISR_vect __vector_default
00258 #endif /* DOXYGEN */
00259
00260 /** \name ISR attributes */
00261
00262 #if defined(__DOXYGEN__)
00263 /** \def ISR_BLOCK
00264     \ingroup avr_interrupts
00265
00266     Identical to an ISR with no attributes specified. Global
00267     interrupts are initially disabled by the AVR hardware when
00268     entering the ISR, without the compiler modifying this state.
00269
00270     Use this attribute in the attributes parameter of the #ISR macro.
00271 */
00272 # define ISR_BLOCK
00273
00274 /** \def ISR_NOBLOCK
00275     \ingroup avr_interrupts
00276
00277     ISR runs with global interrupts initially enabled. The interrupt
00278     enable flag is activated by the compiler as early as possible
00279     within the ISR to ensure minimal processing delay for nested
00280     interrupts.
00281
00282     This may be used to create nested ISRs, however care should be
00283     taken to avoid stack overflows, or to avoid infinitely entering
00284     the ISR for those cases where the AVR hardware does not clear the
00285     respective interrupt flag before entering the ISR.
00286
00287     Use this attribute in the attributes parameter of the #ISR macro.
00288 */
00289 # define ISR_NOBLOCK

```

```

00290
00291 /** \def ISR_NAKED
00292     \ingroup avr_interrupts
00293
00294     ISR is created with no prologue or epilogue code. The user code is
00295     responsible for preservation of the machine state including the
00296     SREG register, as well as placing a reti() at the end of the
00297     interrupt routine.
00298
00299     Use this attribute in the attributes parameter of the #ISR macro.
00300
00301     \note According to GCC documentation, the only code supported in
00302     naked functions is \ref inline_asm "inline assembly".
00303 */
00304 # define ISR_NAKED
00305
00306 /** \def ISR_FLATTEN
00307     \ingroup avr_interrupts
00308
00309     The compiler will try to inline all called function into the ISR.
00310     This has an effect with GCC 4.6 and newer only.
00311
00312     Use this attribute in the attributes parameter of the #ISR macro.
00313 */
00314 # define ISR_FLATTEN
00315
00316 /** \def ISR_NOICF
00317     \ingroup avr_interrupts
00318
00319     Avoid identical-code-folding optimization against this ISR.
00320     This has an effect with GCC 5 and newer only.
00321
00322     Use this attribute in the attributes parameter of the #ISR macro.
00323 */
00324 # define ISR_NOICF
00325
00326 /** \def ISR_NOGCCISR
00327     \ingroup avr_interrupts
00328
00329     Do not generate
00330     <a href="https://sourceware.org/binutils/docs/as/AVR-Pseudo-Instructions.html">\c __gcc_isr pseudo
00331     instructions</a>
00332     for this ISR.
00333     This has an effect with
00334     <a href="https://gcc.gnu.org/gcc-8/changes.html#avr">GCC 8</a>
00335     and newer only.
00336
00337     Use this attribute in the attributes parameter of the #ISR macro.
00338 */
00339 # define ISR_NOGCCISR
00340
00341 /** \def ISR_ALIASOF(target_vector)
00342     \ingroup avr_interrupts
00343
00344     The ISR is linked to another ISR, specified by the vect parameter.
00345     This is compatible with GCC 4.2 and greater only.
00346
00347     Use this attribute in the attributes parameter of the #ISR macro.
00348     Example:
00349     \code
00350     ISR (INT0_vect)
00351     {
00352         PORTB = 42;
00353     }
00354     ISR (INT1_vect, ISR_ALIASOF (INT0_vect));
00355     \endcode
00356 */
00357 # define ISR_ALIASOF(target_vector)
00358 #else /* !DOXYGEN */
00359 # define ISR_BLOCK /* empty */
00360 /* FIXME: This won't work with older versions of avr-gcc as ISR_NOBLOCK
00361     will use 'signal' and 'interrupt' at the same time. */
00362 # define ISR_NOBLOCK    __attribute__((__interrupt__))
00363 # define ISR_NAKED      __attribute__((__naked__))
00364
00365 #if (__GNUC__ == 4 && __GNUC_MINOR__ >= 6) || (__GNUC__ >= 5)
00366 # define ISR_FLATTEN    __attribute__((__flatten__))
00367 #else
00368 # define ISR_FLATTEN    /* empty */
00369 #endif /* has flatten (GCC 4.6+) */
00370
00371 #if defined (__has_attribute)
00372 #if __has_attribute (__no_icf__)
00373 # define ISR_NOICF      __attribute__((__no_icf__))
00374 #else
00375 # define ISR_NOICF      /* empty */

```

```

00376 #endif /* has no_icf */
00377
00378 #if __has_attribute (__no_gccisr__)
00379 # define ISR_NOGCCISR __attribute__((__no_gccisr__))
00380 #else
00381 # define ISR_NOGCCISR /* empty */
00382 #endif /* has no_gccisr */
00383 #endif /* has __has_attribute (GCC 5+) */
00384
00385 # define ISR_ALIASOF(v) __attribute__((__alias__(__STRINGIFY(v))))
00386 #endif /* DOXYGEN */
00387
00388 /**@*/
00389
00390 #endif

```

## 23.21 io.h File Reference

### 23.22 io.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002,2003,2005,2006,2007 Marek Michalkiewicz, Joerg Wunsch
00002 Copyright (c) 2007 Eric B. Weddington
00003 All rights reserved.
00004
00005 Redistribution and use in source and binary forms, with or without
00006 modification, are permitted provided that the following conditions are met:
00007
00008 * Redistributions of source code must retain the above copyright
00009 notice, this list of conditions and the following disclaimer.
00010
00011 * Redistributions in binary form must reproduce the above copyright
00012 notice, this list of conditions and the following disclaimer in
00013 the documentation and/or other materials provided with the
00014 distribution.
00015
00016 * Neither the name of the copyright holders nor the names of
00017 contributors may be used to endorse or promote products derived
00018 from this software without specific prior written permission.
00019
00020 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00022 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030 POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033
00034 /** \file */
00035 /** \defgroup avr_io <avr/io.h>: AVR device-specific IO definitions
00036 \code #include <avr/io.h> \endcode
00037
00038 This header file includes the appropriate IO definitions for the
00039 device that has been specified by the <tt>-mmcu=</tt> compiler
00040 command-line switch. This is done by diverting to the appropriate
00041 file <tt>&lt;avr/io</tt><em>XXXX</em><tt>.h</tt> which should
00042 never be included directly. Some register names common to all
00043 AVR devices are defined directly within <tt>&lt;avr/common.h</tt>,
00044 which is included in <tt>&lt;avr/io.h</tt>,
00045 but most of the details come from the respective include file.
00046
00047 Note that this file always includes the following files:
00048 \code
00049 #include <avr/sfr_defs.h>
00050 #include <avr/portpins.h>
00051 #include <avr/common.h>
00052 #include <avr/version.h>
00053 \endcode
00054 See \ref avr_sfr for more details about that header file.
00055
00056 Included are definitions of the IO register set and their
00057 respective bit values as specified in the Atmel documentation.
00058 Note that inconsistencies in naming conventions,
00059 so even identical functions sometimes get different names on

```



```

00060     different devices.
00061
00062     Also included are the specific names useable for interrupt
00063     function definitions as documented
00064     \ref avr_signames "here".
00065
00066     Finally, the following macros are defined:
00067
00068     - \b RAMEND
00069     <br>
00070     The last on-chip RAM address.
00071     <br>
00072     - \b XRAMEND
00073     <br>
00074     The last possible RAM location that is addressable. This is equal to
00075     RAMEND for devices that do not allow for external RAM. For devices
00076     that allow external RAM, this will be larger than RAMEND.
00077     <br>
00078     - \b E2END
00079     <br>
00080     The last EEPROM address.
00081     <br>
00082     - \b FLASHEND
00083     <br>
00084     The last byte address in the Flash program space.
00085     <br>
00086     - \b SPM_PAGESIZE
00087     <br>
00088     For devices with bootloader support, the flash pagesize
00089     (in bytes) to be used for the \c SPM instruction.
00090     - \b E2PAGESIZE
00091     <br>
00092     The size of the EEPROM page.
00093
00094 */
00095
00096 #ifndef _AVR_IO_H_
00097 #define _AVR_IO_H_
00098
00099 #include <avr/sfr_defs.h>
00100
00101 #if defined (__AVR_AT94K__)
00102 # include <avr/ioat94k.h>
00103 #elif defined (__AVR_AT43USB320__)
00104 # include <avr/io43u32x.h>
00105 #elif defined (__AVR_AT43USB355__)
00106 # include <avr/io43u35x.h>
00107 #elif defined (__AVR_AT76C711__)
00108 # include <avr/io76c711.h>
00109 #elif defined (__AVR_AT86RF401__)
00110 # include <avr/io86r401.h>
00111 #elif defined (__AVR_AT90PWM1__)
00112 # include <avr/io90pwm1.h>
00113 #elif defined (__AVR_AT90PWM2__)
00114 # include <avr/io90pwmx.h>
00115 #elif defined (__AVR_AT90PWM2B__)
00116 # include <avr/io90pwm2b.h>
00117 #elif defined (__AVR_AT90PWM3__)
00118 # include <avr/io90pwmx.h>
00119 #elif defined (__AVR_AT90PWM3B__)
00120 # include <avr/io90pwm3b.h>
00121 #elif defined (__AVR_AT90PWM216__)
00122 # include <avr/io90pwm216.h>
00123 #elif defined (__AVR_AT90PWM316__)
00124 # include <avr/io90pwm316.h>
00125 #elif defined (__AVR_AT90PWM161__)
00126 # include <avr/io90pwm161.h>
00127 #elif defined (__AVR_AT90PWM81__)
00128 # include <avr/io90pwm81.h>
00129 #elif defined (__AVR_ATmega8U2__)
00130 # include <avr/iom8u2.h>
00131 #elif defined (__AVR_ATmega16M1__)
00132 # include <avr/iom16m1.h>
00133 #elif defined (__AVR_ATmega16U2__)
00134 # include <avr/iom16u2.h>
00135 #elif defined (__AVR_ATmega16U4__)
00136 # include <avr/iom16u4.h>
00137 #elif defined (__AVR_ATmega32C1__)
00138 # include <avr/iom32c1.h>
00139 #elif defined (__AVR_ATmega32M1__)
00140 # include <avr/iom32m1.h>
00141 #elif defined (__AVR_ATmega32U2__)
00142 # include <avr/iom32u2.h>
00143 #elif defined (__AVR_ATmega32U4__)
00144 # include <avr/iom32u4.h>
00145 #elif defined (__AVR_ATmega32U6__)
00146 # include <avr/iom32u6.h>

```

```
00147 #elif defined (__AVR_ATmega64C1__)
00148 # include <avr/iom64c1.h>
00149 #elif defined (__AVR_ATmega64M1__)
00150 # include <avr/iom64m1.h>
00151 #elif defined (__AVR_ATmega128__)
00152 # include <avr/iom128.h>
00153 #elif defined (__AVR_ATmega128A__)
00154 # include <avr/iom128a.h>
00155 #elif defined (__AVR_ATmega1280__)
00156 # include <avr/iom1280.h>
00157 #elif defined (__AVR_ATmega1281__)
00158 # include <avr/iom1281.h>
00159 #elif defined (__AVR_ATmega1284__)
00160 # include <avr/iom1284.h>
00161 #elif defined (__AVR_ATmega1284P__)
00162 # include <avr/iom1284p.h>
00163 #elif defined (__AVR_ATmega128RFA1__)
00164 # include <avr/iom128rfal.h>
00165 #elif defined (__AVR_ATmega1284RFR2__)
00166 # include <avr/iom1284rfr2.h>
00167 #elif defined (__AVR_ATmega128RFR2__)
00168 # include <avr/iom128rfr2.h>
00169 #elif defined (__AVR_ATmega2564RFR2__)
00170 # include <avr/iom2564rfr2.h>
00171 #elif defined (__AVR_ATmega256RFR2__)
00172 # include <avr/iom256rfr2.h>
00173 #elif defined (__AVR_ATmega2560__)
00174 # include <avr/iom2560.h>
00175 #elif defined (__AVR_ATmega2561__)
00176 # include <avr/iom2561.h>
00177 #elif defined (__AVR_AT90CAN32__)
00178 # include <avr/iocan32.h>
00179 #elif defined (__AVR_AT90CAN64__)
00180 # include <avr/iocan64.h>
00181 #elif defined (__AVR_AT90CAN128__)
00182 # include <avr/iocan128.h>
00183 #elif defined (__AVR_AT90USB82__)
00184 # include <avr/iousb82.h>
00185 #elif defined (__AVR_AT90USB162__)
00186 # include <avr/iousb162.h>
00187 #elif defined (__AVR_AT90USB646__)
00188 # include <avr/iousb646.h>
00189 #elif defined (__AVR_AT90USB647__)
00190 # include <avr/iousb647.h>
00191 #elif defined (__AVR_AT90USB1286__)
00192 # include <avr/iousb1286.h>
00193 #elif defined (__AVR_AT90USB1287__)
00194 # include <avr/iousb1287.h>
00195 #elif defined (__AVR_ATmega644RFR2__)
00196 # include <avr/iom644rfr2.h>
00197 #elif defined (__AVR_ATmega64RFR2__)
00198 # include <avr/iom64rfr2.h>
00199 #elif defined (__AVR_ATmega64__)
00200 # include <avr/iom64.h>
00201 #elif defined (__AVR_ATmega64A__)
00202 # include <avr/iom64a.h>
00203 #elif defined (__AVR_ATmega640__)
00204 # include <avr/iom640.h>
00205 #elif defined (__AVR_ATmega644__)
00206 # include <avr/iom644.h>
00207 #elif defined (__AVR_ATmega644A__)
00208 # include <avr/iom644a.h>
00209 #elif defined (__AVR_ATmega644P__)
00210 # include <avr/iom644p.h>
00211 #elif defined (__AVR_ATmega644PA__)
00212 # include <avr/iom644pa.h>
00213 #elif defined (__AVR_ATmega645__) || defined (__AVR_ATmega645A__) || defined (__AVR_ATmega645P__)
00214 # include <avr/iom645.h>
00215 #elif defined (__AVR_ATmega6450__) || defined (__AVR_ATmega6450A__) || defined (__AVR_ATmega6450P__)
00216 # include <avr/iom6450.h>
00217 #elif defined (__AVR_ATmega649__) || defined (__AVR_ATmega649A__)
00218 # include <avr/iom649.h>
00219 #elif defined (__AVR_ATmega6490__) || defined (__AVR_ATmega6490A__) || defined (__AVR_ATmega6490P__)
00220 # include <avr/iom6490.h>
00221 #elif defined (__AVR_ATmega649P__)
00222 # include <avr/iom649p.h>
00223 #elif defined (__AVR_ATmega64HVE__)
00224 # include <avr/iom64hve.h>
00225 #elif defined (__AVR_ATmega64HVE2__)
00226 # include <avr/iom64hve2.h>
00227 #elif defined (__AVR_ATmega103__)
00228 # include <avr/iom103.h>
00229 #elif defined (__AVR_ATmega32__)
00230 # include <avr/iom32.h>
00231 #elif defined (__AVR_ATmega32A__)
00232 # include <avr/iom32a.h>
00233 #elif defined (__AVR_ATmega323__)
```

```
00234 # include <avr/iom323.h>
00235 #elif defined (__AVR_ATmega324P__) || defined (__AVR_ATmega324A__)
00236 # include <avr/iom324.h>
00237 #elif defined (__AVR_ATmega324PA__)
00238 # include <avr/iom324pa.h>
00239 #elif defined (__AVR_ATmega324PB__)
00240 # include <avr/iom324pb.h>
00241 #elif defined (__AVR_ATmega325__) || defined (__AVR_ATmega325A__)
00242 # include <avr/iom325.h>
00243 #elif defined (__AVR_ATmega325P__)
00244 # include <avr/iom325.h>
00245 #elif defined (__AVR_ATmega325PA__)
00246 # include <avr/iom325pa.h>
00247 #elif defined (__AVR_ATmega3250__) || defined (__AVR_ATmega3250A__)
00248 # include <avr/iom3250.h>
00249 #elif defined (__AVR_ATmega3250P__)
00250 # include <avr/iom3250.h>
00251 #elif defined (__AVR_ATmega3250PA__)
00252 # include <avr/iom3250pa.h>
00253 #elif defined (__AVR_ATmega328P__) || defined (__AVR_ATmega328__)
00254 # include <avr/iom328p.h>
00255 #elif defined (__AVR_ATmega328PB__)
00256 # include <avr/iom328pb.h>
00257 #elif defined (__AVR_ATmega329__) || defined (__AVR_ATmega329A__)
00258 # include <avr/iom329.h>
00259 #elif defined (__AVR_ATmega329P__) || defined (__AVR_ATmega329PA__)
00260 # include <avr/iom329.h>
00261 #elif defined (__AVR_ATmega3290__) || defined (__AVR_ATmega3290A__)
00262 # include <avr/iom3290.h>
00263 #elif defined (__AVR_ATmega3290P__)
00264 # include <avr/iom3290.h>
00265 #elif defined (__AVR_ATmega3290PA__)
00266 # include <avr/iom3290pa.h>
00267 #elif defined (__AVR_ATmega32HVB__)
00268 # include <avr/iom32hvb.h>
00269 #elif defined (__AVR_ATmega32HVBREVB__)
00270 # include <avr/iom32hvbrevb.h>
00271 #elif defined (__AVR_ATmega406__)
00272 # include <avr/iom406.h>
00273 #elif defined (__AVR_ATmega16__)
00274 # include <avr/iom16.h>
00275 #elif defined (__AVR_ATmega16A__)
00276 # include <avr/iom16a.h>
00277 #elif defined (__AVR_ATmega161__)
00278 # include <avr/iom161.h>
00279 #elif defined (__AVR_ATmega162__)
00280 # include <avr/iom162.h>
00281 #elif defined (__AVR_ATmega163__)
00282 # include <avr/iom163.h>
00283 #elif defined (__AVR_ATmega164P__) || defined (__AVR_ATmega164A__)
00284 # include <avr/iom164.h>
00285 #elif defined (__AVR_ATmega164PA__)
00286 # include <avr/iom164pa.h>
00287 #elif defined (__AVR_ATmega165__)
00288 # include <avr/iom165.h>
00289 #elif defined (__AVR_ATmega165A__)
00290 # include <avr/iom165a.h>
00291 #elif defined (__AVR_ATmega165P__)
00292 # include <avr/iom165p.h>
00293 #elif defined (__AVR_ATmega165PA__)
00294 # include <avr/iom165pa.h>
00295 #elif defined (__AVR_ATmega168__)
00296 # include <avr/iom168.h>
00297 #elif defined (__AVR_ATmega168A__)
00298 # include <avr/iom168a.h>
00299 #elif defined (__AVR_ATmega168P__)
00300 # include <avr/iom168p.h>
00301 #elif defined (__AVR_ATmega168PA__)
00302 # include <avr/iom168pa.h>
00303 #elif defined (__AVR_ATmega168PB__)
00304 # include <avr/iom168pb.h>
00305 #elif defined (__AVR_ATmega169__) || defined (__AVR_ATmega169A__)
00306 # include <avr/iom169.h>
00307 #elif defined (__AVR_ATmega169P__)
00308 # include <avr/iom169p.h>
00309 #elif defined (__AVR_ATmega169PA__)
00310 # include <avr/iom169pa.h>
00311 #elif defined (__AVR_ATmega8HVA__)
00312 # include <avr/iom8hva.h>
00313 #elif defined (__AVR_ATmega16HVA__)
00314 # include <avr/iom16hva.h>
00315 #elif defined (__AVR_ATmega16HVA2__)
00316 # include <avr/iom16hva2.h>
00317 #elif defined (__AVR_ATmega16HVB__)
00318 # include <avr/iom16hvb.h>
00319 #elif defined (__AVR_ATmega16HVBREVB__)
00320 # include <avr/iom16hvbrevb.h>
```

```
00321 #elif defined (__AVR_ATmega8__)
00322 # include <avr/iom8.h>
00323 #elif defined (__AVR_ATmega8A__)
00324 # include <avr/iom8a.h>
00325 #elif defined (__AVR_ATmega48__)
00326 # include <avr/iom48.h>
00327 #elif defined (__AVR_ATmega48A__)
00328 # include <avr/iom48a.h>
00329 #elif defined (__AVR_ATmega48PA__)
00330 # include <avr/iom48pa.h>
00331 #elif defined (__AVR_ATmega48PB__)
00332 # include <avr/iom48pb.h>
00333 #elif defined (__AVR_ATmega48P__)
00334 # include <avr/iom48p.h>
00335 #elif defined (__AVR_ATmega88__)
00336 # include <avr/iom88.h>
00337 #elif defined (__AVR_ATmega88A__)
00338 # include <avr/iom88a.h>
00339 #elif defined (__AVR_ATmega88P__)
00340 # include <avr/iom88p.h>
00341 #elif defined (__AVR_ATmega88PA__)
00342 # include <avr/iom88pa.h>
00343 #elif defined (__AVR_ATmega88PB__)
00344 # include <avr/iom88pb.h>
00345 #elif defined (__AVR_ATmega8515__)
00346 # include <avr/iom8515.h>
00347 #elif defined (__AVR_ATmega8535__)
00348 # include <avr/iom8535.h>
00349 #elif defined (__AVR_AT90S8535__)
00350 # include <avr/io8535.h>
00351 #elif defined (__AVR_AT90C8534__)
00352 # include <avr/io8534.h>
00353 #elif defined (__AVR_AT90S8515__)
00354 # include <avr/io8515.h>
00355 #elif defined (__AVR_AT90S4434__)
00356 # include <avr/io4434.h>
00357 #elif defined (__AVR_AT90S4433__)
00358 # include <avr/io4433.h>
00359 #elif defined (__AVR_AT90S4414__)
00360 # include <avr/io4414.h>
00361 #elif defined (__AVR_ATtiny22__)
00362 # include <avr/iotn22.h>
00363 #elif defined (__AVR_ATtiny26__)
00364 # include <avr/iotn26.h>
00365 #elif defined (__AVR_AT90S2343__)
00366 # include <avr/io2343.h>
00367 #elif defined (__AVR_AT90S2333__)
00368 # include <avr/io2333.h>
00369 #elif defined (__AVR_AT90S2323__)
00370 # include <avr/io2323.h>
00371 #elif defined (__AVR_AT90S2313__)
00372 # include <avr/io2313.h>
00373 #elif defined (__AVR_ATtiny4__)
00374 # include <avr/iotn4.h>
00375 #elif defined (__AVR_ATtiny5__)
00376 # include <avr/iotn5.h>
00377 #elif defined (__AVR_ATtiny9__)
00378 # include <avr/iotn9.h>
00379 #elif defined (__AVR_ATtiny10__)
00380 # include <avr/iotn10.h>
00381 #elif defined (__AVR_ATtiny102__)
00382 # include <avr/iotn102.h>
00383 #elif defined (__AVR_ATtiny104__)
00384 # include <avr/iotn104.h>
00385 #elif defined (__AVR_ATtiny20__)
00386 # include <avr/iotn20.h>
00387 #elif defined (__AVR_ATtiny40__)
00388 # include <avr/iotn40.h>
00389 #elif defined (__AVR_ATtiny2313__)
00390 # include <avr/iotn2313.h>
00391 #elif defined (__AVR_ATtiny2313A__)
00392 # include <avr/iotn2313a.h>
00393 #elif defined (__AVR_ATtiny13__)
00394 # include <avr/iotn13.h>
00395 #elif defined (__AVR_ATtiny13A__)
00396 # include <avr/iotn13a.h>
00397 #elif defined (__AVR_ATtiny25__)
00398 # include <avr/iotn25.h>
00399 #elif defined (__AVR_ATtiny4313__)
00400 # include <avr/iotn4313.h>
00401 #elif defined (__AVR_ATtiny45__)
00402 # include <avr/iotn45.h>
00403 #elif defined (__AVR_ATtiny85__)
00404 # include <avr/iotn85.h>
00405 #elif defined (__AVR_ATtiny24__)
00406 # include <avr/iotn24.h>
00407 #elif defined (__AVR_ATtiny24A__)
```

```
00408 # include <avr/iotn24a.h>
00409 #elif defined (__AVR_ATTtiny44__)
00410 # include <avr/iotn44.h>
00411 #elif defined (__AVR_ATTtiny44A__)
00412 # include <avr/iotn44a.h>
00413 #elif defined (__AVR_ATTtiny441__)
00414 # include <avr/iotn441.h>
00415 #elif defined (__AVR_ATTtiny84__)
00416 # include <avr/iotn84.h>
00417 #elif defined (__AVR_ATTtiny84A__)
00418 # include <avr/iotn84a.h>
00419 #elif defined (__AVR_ATTtiny841__)
00420 # include <avr/iotn841.h>
00421 #elif defined (__AVR_ATTtiny261__)
00422 # include <avr/iotn261.h>
00423 #elif defined (__AVR_ATTtiny261A__)
00424 # include <avr/iotn261a.h>
00425 #elif defined (__AVR_ATTtiny461__)
00426 # include <avr/iotn461.h>
00427 #elif defined (__AVR_ATTtiny461A__)
00428 # include <avr/iotn461a.h>
00429 #elif defined (__AVR_ATTtiny861__)
00430 # include <avr/iotn861.h>
00431 #elif defined (__AVR_ATTtiny861A__)
00432 # include <avr/iotn861a.h>
00433 #elif defined (__AVR_ATTtiny43U__)
00434 # include <avr/iotn43u.h>
00435 #elif defined (__AVR_ATTtiny48__)
00436 # include <avr/iotn48.h>
00437 #elif defined (__AVR_ATTtiny88__)
00438 # include <avr/iotn88.h>
00439 #elif defined (__AVR_ATTtiny828__)
00440 # include <avr/iotn828.h>
00441 #elif defined (__AVR_ATTtiny87__)
00442 # include <avr/iotn87.h>
00443 #elif defined (__AVR_ATTtiny167__)
00444 # include <avr/iotn167.h>
00445 #elif defined (__AVR_ATTtiny1634__)
00446 # include <avr/iotn1634.h>
00447 #elif defined (__AVR_ATTtiny202__)
00448 # include <avr/iotn202.h>
00449 #elif defined (__AVR_ATTtiny204__)
00450 # include <avr/iotn204.h>
00451 #elif defined (__AVR_ATTtiny212__)
00452 # include <avr/iotn212.h>
00453 #elif defined (__AVR_ATTtiny214__)
00454 # include <avr/iotn214.h>
00455 #elif defined (__AVR_ATTtiny402__)
00456 # include <avr/iotn402.h>
00457 #elif defined (__AVR_ATTtiny404__)
00458 # include <avr/iotn404.h>
00459 #elif defined (__AVR_ATTtiny406__)
00460 # include <avr/iotn406.h>
00461 #elif defined (__AVR_ATTtiny412__)
00462 # include <avr/iotn412.h>
00463 #elif defined (__AVR_ATTtiny414__)
00464 # include <avr/iotn414.h>
00465 #elif defined (__AVR_ATTtiny416__)
00466 # include <avr/iotn416.h>
00467 #elif defined (__AVR_ATTtiny417__)
00468 # include <avr/iotn417.h>
00469 #elif defined (__AVR_ATTtiny424__)
00470 # include <avr/iotn424.h>
00471 #elif defined (__AVR_ATTtiny426__)
00472 # include <avr/iotn426.h>
00473 #elif defined (__AVR_ATTtiny427__)
00474 # include <avr/iotn427.h>
00475 #elif defined (__AVR_ATTtiny804__)
00476 # include <avr/iotn804.h>
00477 #elif defined (__AVR_ATTtiny806__)
00478 # include <avr/iotn806.h>
00479 #elif defined (__AVR_ATTtiny807__)
00480 # include <avr/iotn807.h>
00481 #elif defined (__AVR_ATTtiny814__)
00482 # include <avr/iotn814.h>
00483 #elif defined (__AVR_ATTtiny816__)
00484 # include <avr/iotn816.h>
00485 #elif defined (__AVR_ATTtiny817__)
00486 # include <avr/iotn817.h>
00487 #elif defined (__AVR_ATTtiny824__)
00488 # include <avr/iotn824.h>
00489 #elif defined (__AVR_ATTtiny826__)
00490 # include <avr/iotn826.h>
00491 #elif defined (__AVR_ATTtiny827__)
00492 # include <avr/iotn827.h>
00493 #elif defined (__AVR_ATTtiny1604__)
00494 # include <avr/iotn1604.h>
```

```
00495 #elif defined (__AVR_ATTtiny1606__)
00496 # include <avr/iotn1606.h>
00497 #elif defined (__AVR_ATTtiny1607__)
00498 # include <avr/iotn1607.h>
00499 #elif defined (__AVR_ATTtiny1614__)
00500 # include <avr/iotn1614.h>
00501 #elif defined (__AVR_ATTtiny1616__)
00502 # include <avr/iotn1616.h>
00503 #elif defined (__AVR_ATTtiny1617__)
00504 # include <avr/iotn1617.h>
00505 #elif defined (__AVR_ATTtiny1624__)
00506 # include <avr/iotn1624.h>
00507 #elif defined (__AVR_ATTtiny1626__)
00508 # include <avr/iotn1626.h>
00509 #elif defined (__AVR_ATTtiny1627__)
00510 # include <avr/iotn1627.h>
00511 #elif defined (__AVR_ATTtiny3214__)
00512 # include <avr/iotn3214.h>
00513 #elif defined (__AVR_ATTtiny3216__)
00514 # include <avr/iotn3216.h>
00515 #elif defined (__AVR_ATTtiny3217__)
00516 # include <avr/iotn3217.h>
00517 #elif defined (__AVR_ATTtiny3224__)
00518 # include <avr/iotn3224.h>
00519 #elif defined (__AVR_ATTtiny3226__)
00520 # include <avr/iotn3226.h>
00521 #elif defined (__AVR_ATTtiny3227__)
00522 # include <avr/iotn3227.h>
00523 #elif defined (__AVR_ATmega808__)
00524 # include <avr/iom808.h>
00525 #elif defined (__AVR_ATmega809__)
00526 # include <avr/iom809.h>
00527 #elif defined (__AVR_ATmega1608__)
00528 # include <avr/iom1608.h>
00529 #elif defined (__AVR_ATmega1609__)
00530 # include <avr/iom1609.h>
00531 #elif defined (__AVR_ATmega3208__)
00532 # include <avr/iom3208.h>
00533 #elif defined (__AVR_ATmega3209__)
00534 # include <avr/iom3209.h>
00535 #elif defined (__AVR_ATmega4808__)
00536 # include <avr/iom4808.h>
00537 #elif defined (__AVR_ATmega4809__)
00538 # include <avr/iom4809.h>
00539 #elif defined (__AVR_AT90SCR100__)
00540 # include <avr/io90scr100.h>
00541 #elif defined (__AVR_ATxmega8E5__)
00542 # include <avr/iox8e5.h>
00543 #elif defined (__AVR_ATxmega16A4__)
00544 # include <avr/iox16a4.h>
00545 #elif defined (__AVR_ATxmega16A4U__)
00546 # include <avr/iox16a4u.h>
00547 #elif defined (__AVR_ATxmega16C4__)
00548 # include <avr/iox16c4.h>
00549 #elif defined (__AVR_ATxmega16D4__)
00550 # include <avr/iox16d4.h>
00551 #elif defined (__AVR_ATxmega32A4__)
00552 # include <avr/iox32a4.h>
00553 #elif defined (__AVR_ATxmega32A4U__)
00554 # include <avr/iox32a4u.h>
00555 #elif defined (__AVR_ATxmega32C3__)
00556 # include <avr/iox32c3.h>
00557 #elif defined (__AVR_ATxmega32C4__)
00558 # include <avr/iox32c4.h>
00559 #elif defined (__AVR_ATxmega32D3__)
00560 # include <avr/iox32d3.h>
00561 #elif defined (__AVR_ATxmega32D4__)
00562 # include <avr/iox32d4.h>
00563 #elif defined (__AVR_ATxmega32E5__)
00564 # include <avr/iox32e5.h>
00565 #elif defined (__AVR_ATxmega64A1__)
00566 # include <avr/iox64a1.h>
00567 #elif defined (__AVR_ATxmega64A1U__)
00568 # include <avr/iox64a1u.h>
00569 #elif defined (__AVR_ATxmega64A3__)
00570 # include <avr/iox64a3.h>
00571 #elif defined (__AVR_ATxmega64A3U__)
00572 # include <avr/iox64a3u.h>
00573 #elif defined (__AVR_ATxmega64A4U__)
00574 # include <avr/iox64a4u.h>
00575 #elif defined (__AVR_ATxmega64B1__)
00576 # include <avr/iox64b1.h>
00577 #elif defined (__AVR_ATxmega64B3__)
00578 # include <avr/iox64b3.h>
00579 #elif defined (__AVR_ATxmega64C3__)
00580 # include <avr/iox64c3.h>
00581 #elif defined (__AVR_ATxmega64D3__)
```

```
00582 # include <avr/iox64d3.h>
00583 #elif defined (__AVR_ATxmega64D4__)
00584 # include <avr/iox64d4.h>
00585 #elif defined (__AVR_ATxmega128A1__)
00586 # include <avr/iox128a1.h>
00587 #elif defined (__AVR_ATxmega128A1U__)
00588 # include <avr/iox128a1u.h>
00589 #elif defined (__AVR_ATxmega128A4U__)
00590 # include <avr/iox128a4u.h>
00591 #elif defined (__AVR_ATxmega128A3__)
00592 # include <avr/iox128a3.h>
00593 #elif defined (__AVR_ATxmega128A3U__)
00594 # include <avr/iox128a3u.h>
00595 #elif defined (__AVR_ATxmega128B1__)
00596 # include <avr/iox128b1.h>
00597 #elif defined (__AVR_ATxmega128B3__)
00598 # include <avr/iox128b3.h>
00599 #elif defined (__AVR_ATxmega128C3__)
00600 # include <avr/iox128c3.h>
00601 #elif defined (__AVR_ATxmega128D3__)
00602 # include <avr/iox128d3.h>
00603 #elif defined (__AVR_ATxmega128D4__)
00604 # include <avr/iox128d4.h>
00605 #elif defined (__AVR_ATxmega192A3__)
00606 # include <avr/iox192a3.h>
00607 #elif defined (__AVR_ATxmega192A3U__)
00608 # include <avr/iox192a3u.h>
00609 #elif defined (__AVR_ATxmega192C3__)
00610 # include <avr/iox192c3.h>
00611 #elif defined (__AVR_ATxmega192D3__)
00612 # include <avr/iox192d3.h>
00613 #elif defined (__AVR_ATxmega256A3__)
00614 # include <avr/iox256a3.h>
00615 #elif defined (__AVR_ATxmega256A3U__)
00616 # include <avr/iox256a3u.h>
00617 #elif defined (__AVR_ATxmega256A3B__)
00618 # include <avr/iox256a3b.h>
00619 #elif defined (__AVR_ATxmega256A3BU__)
00620 # include <avr/iox256a3bu.h>
00621 #elif defined (__AVR_ATxmega256C3__)
00622 # include <avr/iox256c3.h>
00623 #elif defined (__AVR_ATxmega256D3__)
00624 # include <avr/iox256d3.h>
00625 #elif defined (__AVR_ATxmega384C3__)
00626 # include <avr/iox384c3.h>
00627 #elif defined (__AVR_ATxmega384D3__)
00628 # include <avr/iox384d3.h>
00629 #elif defined (__AVR_ATA5702M322__)
00630 # include <avr/ia5702m322.h>
00631 #elif defined (__AVR_ATA5782__)
00632 # include <avr/ia5782.h>
00633 #elif defined (__AVR_ATA5790__)
00634 # include <avr/ia5790.h>
00635 #elif defined (__AVR_ATA5790N__)
00636 # include <avr/ia5790n.h>
00637 #elif defined (__AVR_ATA5831__)
00638 # include <avr/ia5831.h>
00639 #elif defined (__AVR_ATA5272__)
00640 # include <avr/ia5272.h>
00641 #elif defined (__AVR_ATA5505__)
00642 # include <avr/ia5505.h>
00643 #elif defined (__AVR_ATA5795__)
00644 # include <avr/ia5795.h>
00645 #elif defined (__AVR_ATA6285__)
00646 # include <avr/ia6285.h>
00647 #elif defined (__AVR_ATA6286__)
00648 # include <avr/ia6286.h>
00649 #elif defined (__AVR_ATA6289__)
00650 # include <avr/ia6289.h>
00651 #elif defined (__AVR_ATA6612C__)
00652 # include <avr/ia6612c.h>
00653 #elif defined (__AVR_ATA6613C__)
00654 # include <avr/ia6613c.h>
00655 #elif defined (__AVR_ATA6614Q__)
00656 # include <avr/ia6614q.h>
00657 #elif defined (__AVR_ATA6616C__)
00658 # include <avr/ia6616c.h>
00659 #elif defined (__AVR_ATA6617C__)
00660 # include <avr/ia6617c.h>
00661 #elif defined (__AVR_ATA664251__)
00662 # include <avr/ia664251.h>
00663 /* avr1: the following only supported for assembler programs */
00664 #elif defined (__AVR_ATTiny28__)
00665 # include <avr/iotn28.h>
00666 #elif defined (__AVR_AT90S1200__)
00667 # include <avr/iol200.h>
00668 #elif defined (__AVR_ATTiny15__)
```

```
00669 # include <avr/iotn15.h>
00670 #elif defined (__AVR_ATTtiny12__)
00671 # include <avr/iotn12.h>
00672 #elif defined (__AVR_ATTtiny11__)
00673 # include <avr/iotn11.h>
00674 #elif defined (__AVR_M3000__)
00675 # include <avr/iom3000.h>
00676 #elif defined (__AVR_AVR32DA28__)
00677 # include <avr/ioavr32da28.h>
00678 #elif defined (__AVR_AVR32DA32__)
00679 # include <avr/ioavr32da32.h>
00680 #elif defined (__AVR_AVR32DA48__)
00681 # include <avr/ioavr32da48.h>
00682 #elif defined (__AVR_AVR64DA28__)
00683 # include <avr/ioavr64da28.h>
00684 #elif defined (__AVR_AVR64DA32__)
00685 # include <avr/ioavr64da32.h>
00686 #elif defined (__AVR_AVR64DA48__)
00687 # include <avr/ioavr64da48.h>
00688 #elif defined (__AVR_AVR64DA64__)
00689 # include <avr/ioavr64da64.h>
00690 #elif defined (__AVR_AVR128DA28__)
00691 # include <avr/ioavr128da28.h>
00692 #elif defined (__AVR_AVR128DA32__)
00693 # include <avr/ioavr128da32.h>
00694 #elif defined (__AVR_AVR128DA48__)
00695 # include <avr/ioavr128da48.h>
00696 #elif defined (__AVR_AVR128DA64__)
00697 # include <avr/ioavr128da64.h>
00698 #elif defined (__AVR_AVR32DB28__)
00699 # include <avr/ioavr32db28.h>
00700 #elif defined (__AVR_AVR32DB32__)
00701 # include <avr/ioavr32db32.h>
00702 #elif defined (__AVR_AVR32DB48__)
00703 # include <avr/ioavr32db48.h>
00704 #elif defined (__AVR_AVR64DB28__)
00705 # include <avr/ioavr64db28.h>
00706 #elif defined (__AVR_AVR64DB32__)
00707 # include <avr/ioavr64db32.h>
00708 #elif defined (__AVR_AVR64DB48__)
00709 # include <avr/ioavr64db48.h>
00710 #elif defined (__AVR_AVR64DB64__)
00711 # include <avr/ioavr64db64.h>
00712 #elif defined (__AVR_AVR128DB28__)
00713 # include <avr/ioavr128db28.h>
00714 #elif defined (__AVR_AVR128DB32__)
00715 # include <avr/ioavr128db32.h>
00716 #elif defined (__AVR_AVR128DB48__)
00717 # include <avr/ioavr128db48.h>
00718 #elif defined (__AVR_AVR128DB64__)
00719 # include <avr/ioavr128db64.h>
00720 #elif defined (__AVR_AVR16DD14__)
00721 # include <avr/ioavr16dd14.h>
00722 #elif defined (__AVR_AVR16DD20__)
00723 # include <avr/ioavr16dd20.h>
00724 #elif defined (__AVR_AVR16DD28__)
00725 # include <avr/ioavr16dd28.h>
00726 #elif defined (__AVR_AVR16DD32__)
00727 # include <avr/ioavr16dd32.h>
00728 #elif defined (__AVR_AVR32DD14__)
00729 # include <avr/ioavr32dd14.h>
00730 #elif defined (__AVR_AVR32DD20__)
00731 # include <avr/ioavr32dd20.h>
00732 #elif defined (__AVR_AVR32DD28__)
00733 # include <avr/ioavr32dd28.h>
00734 #elif defined (__AVR_AVR32DD32__)
00735 # include <avr/ioavr32dd32.h>
00736 #elif defined (__AVR_AVR64DD14__)
00737 # include <avr/ioavr64dd14.h>
00738 #elif defined (__AVR_AVR64DD20__)
00739 # include <avr/ioavr64dd20.h>
00740 #elif defined (__AVR_AVR64DD28__)
00741 # include <avr/ioavr64dd28.h>
00742 #elif defined (__AVR_AVR64DD32__)
00743 # include <avr/ioavr64dd32.h>
00744 #elif defined (__AVR_AVR64DU28__)
00745 # include <avr/ioavr64du28.h>
00746 #elif defined (__AVR_AVR64DU32__)
00747 # include <avr/ioavr64du32.h>
00748 #elif defined (__AVR_AVR16EA28__)
00749 # include <avr/ioavr16ea28.h>
00750 #elif defined (__AVR_AVR16EA32__)
00751 # include <avr/ioavr16ea32.h>
00752 #elif defined (__AVR_AVR16EA48__)
00753 # include <avr/ioavr16ea48.h>
00754 #elif defined (__AVR_AVR16EB14__)
00755 # include <avr/ioavr16eb14.h>
```



```

00756 #elif defined (__AVR_AVR16EB20__)
00757 # include <avr/ioavr16eb20.h>
00758 #elif defined (__AVR_AVR16EB28__)
00759 # include <avr/ioavr16eb28.h>
00760 #elif defined (__AVR_AVR16EB32__)
00761 # include <avr/ioavr16eb32.h>
00762 #elif defined (__AVR_AVR32EA28__)
00763 # include <avr/ioavr32ea28.h>
00764 #elif defined (__AVR_AVR32EA32__)
00765 # include <avr/ioavr32ea32.h>
00766 #elif defined (__AVR_AVR32EA48__)
00767 # include <avr/ioavr32ea48.h>
00768 #elif defined (__AVR_AVR64EA28__)
00769 # include <avr/ioavr64ea28.h>
00770 #elif defined (__AVR_AVR64EA32__)
00771 # include <avr/ioavr64ea32.h>
00772 #elif defined (__AVR_AVR64EA48__)
00773 # include <avr/ioavr64ea48.h>
00774 #elif defined (__AVR_DEV_LIB_NAME__)
00775 # define __concat__(a,b) a##b
00776 # define __header1__(a,b) __concat__(a,b)
00777 # define __AVR_DEVICE_HEADER__ <avr/__header1__(io,__AVR_DEV_LIB_NAME__)>.h>
00778 # include __AVR_DEVICE_HEADER__
00779 #else
00780 # if !defined(__COMPILING_AVR_LIBC__)
00781 # warning "device type not defined"
00782 # endif
00783 #endif
00784
00785 #include <avr/portpins.h>
00786
00787 #include <avr/common.h>
00788
00789 #include <avr/version.h>
00790
00791 #if __AVR_ARCH__ >= 100
00792 # include <avr/xmega.h>
00793 #endif
00794
00795 /* Include fuse.h after individual IO header files. */
00796 #include <avr/fuse.h>
00797
00798 /* Include lock.h after individual IO header files. */
00799 #include <avr/lock.h>
00800
00801 #endif /* _AVR_IO_H_ */

```

## 23.23 lock.h File Reference

### 23.24 lock.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2007, Atmel Corporation
00002 All rights reserved.
00003
00004 Redistribution and use in source and binary forms, with or without
00005 modification, are permitted provided that the following conditions are met:
00006
00007 * Redistributions of source code must retain the above copyright
00008 notice, this list of conditions and the following disclaimer.
00009
00010 * Redistributions in binary form must reproduce the above copyright
00011 notice, this list of conditions and the following disclaimer in
00012 the documentation and/or other materials provided with the
00013 distribution.
00014
00015 * Neither the name of the copyright holders nor the names of
00016 contributors may be used to endorse or promote products derived
00017 from this software without specific prior written permission.
00018
00019 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

```

```

00029 POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /* avr/lock.h - Lock Bits API */
00034
00035 #ifndef _AVR_LOCK_H_
00036 #define _AVR_LOCK_H_ 1
00037
00038
00039 /** \file */
00040 /** \defgroup avr_lock <avr/lock.h>: Lockbit Support
00041
00042 \par Introduction
00043
00044 The Lockbit API allows a user to specify the lockbit settings for the
00045 specific AVR device they are compiling for. These lockbit settings will be
00046 placed in a special section in the ELF output file, after linking.
00047
00048 Programming tools can take advantage of the lockbit information embedded in
00049 the ELF file, by extracting this information and determining if the lockbits
00050 need to be programmed after programming the Flash and EEPROM memories.
00051 This also allows a single ELF file to contain all the
00052 information needed to program an AVR.
00053
00054 To use the Lockbit API, include the <avr/io.h> header file, which in turn
00055 automatically includes the individual I/O header file and the <avr/lock.h>
00056 file. These other two files provides everything necessary to set the AVR
00057 lockbits.
00058
00059 \par Lockbit API
00060
00061 Each I/O header file may define up to 3 macros that controls what kinds
00062 of lockbits are available to the user.
00063
00064 If __LOCK_BITS_EXIST is defined, then two lock bits are available to the
00065 user and 3 mode settings are defined for these two bits.
00066
00067 If __BOOT_LOCK_BITS_0_EXIST is defined, then the two BLB0 lock bits are
00068 available to the user and 4 mode settings are defined for these two bits.
00069
00070 If __BOOT_LOCK_BITS_1_EXIST is defined, then the two BLB1 lock bits are
00071 available to the user and 4 mode settings are defined for these two bits.
00072
00073 If __BOOT_LOCK_APPLICATION_TABLE_BITS_EXIST is defined then two lock bits
00074 are available to set the locking mode for the Application Table Section
00075 (which is used in the XMEGA family).
00076
00077 If __BOOT_LOCK_APPLICATION_BITS_EXIST is defined then two lock bits are
00078 available to set the locking mode for the Application Section (which is used
00079 in the XMEGA family).
00080
00081 If __BOOT_LOCK_BOOT_BITS_EXIST is defined then two lock bits are available
00082 to set the locking mode for the Boot Loader Section (which is used in the
00083 XMEGA family).
00084
00085 The AVR lockbit modes have inverted values, logical 1 for an unprogrammed
00086 (disabled) bit and logical 0 for a programmed (enabled) bit. The defined
00087 macros for each individual lock bit represent this in their definition by a
00088 bit-wise inversion of a mask. For example, the LB_MODE_3 macro is defined
00089 as:
00090 \code
00091 #define LB_MODE_3 (0xFC)
00092 \endcode
00093
00094 To combine the lockbit mode macros together to represent a whole byte,
00095 use the bitwise AND operator, like so:
00096 \code
00097 (LB_MODE_3 & BLB0_MODE_2)
00098 \endcode
00099
00100 <avr/lock.h> also defines a macro that provides a default lockbit value:
00101 LOCKBITS_DEFAULT which is defined to be 0xFF.
00102
00103 See the AVR device specific datasheet for more details about these
00104 lock bits and the available mode settings.
00105
00106 A convenience macro, LOCKMEM, is defined as a GCC attribute for a
00107 custom-named section of ".lock".
00108
00109 A convenience macro, LOCKBITS, is defined that declares a variable, __lock,
00110 of type unsigned char with the attribute defined by LOCKMEM. This variable
00111 allows the end user to easily set the lockbit data.
00112
00113 \note If a device-specific I/O header file has previously defined LOCKMEM,
00114 then LOCKMEM is not redefined. If a device-specific I/O header file has
00115 previously defined LOCKBITS, then LOCKBITS is not redefined. LOCKBITS is

```

```

00116     currently known to be defined in the I/O header files for the XMEGA devices.
00117
00118     \par API Usage Example
00119
00120     Putting all of this together is easy:
00121
00122     \code
00123     #include <avr/io.h>
00124
00125     LOCKBITS = (LB_MODE_1 & BLB0_MODE_3 & BLB1_MODE_4);
00126
00127     int main(void)
00128     {
00129         return 0;
00130     }
00131     \endcode
00132
00133     Or:
00134
00135     \code
00136     #include <avr/io.h>
00137
00138     unsigned char __lock __attribute__((section (".lock"))) =
00139         (LB_MODE_1 & BLB0_MODE_3 & BLB1_MODE_4);
00140
00141     int main(void)
00142     {
00143         return 0;
00144     }
00145     \endcode
00146
00147
00148
00149     However there are a number of caveats that you need to be aware of to
00150     use this API properly.
00151
00152     Be sure to include <avr/io.h> to get all of the definitions for the API.
00153     The LOCKBITS macro defines a global variable to store the lockbit data. This
00154     variable is assigned to its own linker section. Assign the desired lockbit
00155     values immediately in the variable initialization.
00156
00157     The .lock section in the ELF file will get its values from the initial
00158     variable assignment ONLY. This means that you can NOT assign values to
00159     this variable in functions and the new values will not be put into the
00160     ELF .lock section.
00161
00162     The global variable is declared in the LOCKBITS macro has two leading
00163     underscores, which means that it is reserved for the "implementation",
00164     meaning the library, so it will not conflict with a user-named variable.
00165
00166     You must initialize the lockbit variable to some meaningful value, even
00167     if it is the default value. This is because the lockbits default to a
00168     logical 1, meaning unprogrammed. Normal uninitialized data defaults to all
00169     logical zeros. So it is vital that all lockbits are initialized, even with
00170     default data. If they are not, then the lockbits may not be programmed to the
00171     desired settings and can possibly put your device into an unrecoverable
00172     state.
00173
00174     Be sure to have the -mmcu=<em>device</em> flag in your compile command line and
00175     your linker command line to have the correct device selected and to have
00176     the correct I/O header file included when you include <avr/io.h>.
00177
00178     You can print out the contents of the .lock section in the ELF file by
00179     using this command line:
00180     \code
00181     avr-objdump -s -j .lock <ELF file>
00182     \endcode
00183
00184     */
00185
00186
00187     #if !(defined(__ASSEMBLER__) || defined(__DOXYGEN__))
00188
00189     #ifndef LOCKMEM
00190     #define LOCKMEM __attribute__((used, section (".lock")))
00191     #endif
00192
00193     #ifndef LOCKBITS
00194     #define LOCKBITS unsigned char __lock LOCKMEM
00195     #endif
00196
00197     /* Lock Bit Modes */
00198     #if defined(__LOCK_BITS_EXIST)
00199     #define LB_MODE_1 (0xFF)
00200     #define LB_MODE_2 (0xFE)
00201     #define LB_MODE_3 (0xFC)
00202     #endif

```

```

00203
00204 #if defined(__BOOT_LOCK_BITS_0_EXIST)
00205 #define BLB0_MODE_1 (0xFF)
00206 #define BLB0_MODE_2 (0xFB)
00207 #define BLB0_MODE_3 (0xF3)
00208 #define BLB0_MODE_4 (0xF7)
00209 #endif
00210
00211 #if defined(__BOOT_LOCK_BITS_1_EXIST)
00212 #define BLB1_MODE_1 (0xFF)
00213 #define BLB1_MODE_2 (0xEF)
00214 #define BLB1_MODE_3 (0xCF)
00215 #define BLB1_MODE_4 (0xDF)
00216 #endif
00217
00218 #if defined(__BOOT_LOCK_APPLICATION_TABLE_BITS_EXIST)
00219 #define BLBAT0 ~_BV(2)
00220 #define BLBAT1 ~_BV(3)
00221 #endif
00222
00223 #if defined(__BOOT_LOCK_APPLICATION_BITS_EXIST)
00224 #define BLBA0 ~_BV(4)
00225 #define BLBA1 ~_BV(5)
00226 #endif
00227
00228 #if defined(__BOOT_LOCK_BOOT_BITS_EXIST)
00229 #define BLBB0 ~_BV(6)
00230 #define BLBB1 ~_BV(7)
00231 #endif
00232
00233 #ifndef LOCKBITS_DEFAULT
00234 #define LOCKBITS_DEFAULT (0xFF)
00235 #endif
00236
00237 #endif /* !(__ASSEMBLER || __DOXYGEN__) */
00238
00239
00240 #endif /* _AVR_LOCK_H_ */

```

## 23.25 pgmspace.h File Reference

### Macros

- #define [PROGMEM\\_FAR](#) \_\_attribute\_\_((\_\_section\_\_(".progmemx.data")))
- #define [PROGMEM](#) \_\_attribute\_\_((\_\_progmem\_\_))
- #define [PSTR](#)(str) ({ static const [PROGMEM](#) char c[] = (str); &c[0]; })
- #define [PSTR\\_FAR](#)(str) ({ static const [PROGMEM\\_FAR](#) char c[] = (str); [pgm\\_get\\_far\\_address](#)(c[0]); })
- #define [pgm\\_read\\_byte\\_near](#)(\_\_addr) \_\_LPM((uint16\_t)(\_\_addr))
- #define [pgm\\_read\\_word\\_near](#)(\_\_addr) \_\_LPM\_word((uint16\_t)(\_\_addr))
- #define [pgm\\_read\\_dword\\_near](#)(\_\_addr) \_\_LPM\_dword((uint16\_t)(\_\_addr))
- #define [pgm\\_read\\_qword\\_near](#)(\_\_addr) \_\_LPM\_qword((uint16\_t)(\_\_addr))
- #define [pgm\\_read\\_float\\_near](#)(addr) [pgm\\_read\\_float](#)(addr)
- #define [pgm\\_read\\_ptr\\_near](#)(\_\_addr) ((void\*) \_\_LPM\_word((uint16\_t)(\_\_addr)))
- #define [pgm\\_read\\_byte\\_far](#)(\_\_addr) \_\_ELPM(\_\_addr)
- #define [pgm\\_read\\_word\\_far](#)(\_\_addr) \_\_ELPM\_word(\_\_addr)
- #define [pgm\\_read\\_dword\\_far](#)(\_\_addr) \_\_ELPM\_dword(\_\_addr)
- #define [pgm\\_read\\_qword\\_far](#)(\_\_addr) \_\_ELPM\_qword(\_\_addr)
- #define [pgm\\_read\\_ptr\\_far](#)(\_\_addr) ((void\*) \_\_ELPM\_word(\_\_addr))
- #define [pgm\\_read\\_byte](#)(\_\_addr) [pgm\\_read\\_byte\\_near](#)(\_\_addr)
- #define [pgm\\_read\\_word](#)(\_\_addr) [pgm\\_read\\_word\\_near](#)(\_\_addr)
- #define [pgm\\_read\\_dword](#)(\_\_addr) [pgm\\_read\\_dword\\_near](#)(\_\_addr)
- #define [pgm\\_read\\_qword](#)(\_\_addr) [pgm\\_read\\_qword\\_near](#)(\_\_addr)
- #define [pgm\\_read\\_ptr](#)(\_\_addr) [pgm\\_read\\_ptr\\_near](#)(\_\_addr)
- #define [pgm\\_get\\_far\\_address](#)(var)

## Functions

- static char [pgm\\_read\\_char](#) (const char \*\_\_addr)
- static unsigned char [pgm\\_read\\_unsigned\\_char](#) (const unsigned char \*\_\_addr)
- static signed char [pgm\\_read\\_signed\\_char](#) (const signed char \*\_\_addr)
- static [uint8\\_t](#) [pgm\\_read\\_u8](#) (const [uint8\\_t](#) \*\_\_addr)
- static [int8\\_t](#) [pgm\\_read\\_i8](#) (const [int8\\_t](#) \*\_\_addr)
- static short [pgm\\_read\\_short](#) (const short \*\_\_addr)
- static unsigned short [pgm\\_read\\_unsigned\\_short](#) (const unsigned short \*\_\_addr)
- static [uint16\\_t](#) [pgm\\_read\\_u16](#) (const [uint16\\_t](#) \*\_\_addr)
- static [int16\\_t](#) [pgm\\_read\\_i16](#) (const [int16\\_t](#) \*\_\_addr)
- static int [pgm\\_read\\_int](#) (const int \*\_\_addr)
- static signed [pgm\\_read\\_signed](#) (const signed \*\_\_addr)
- static unsigned [pgm\\_read\\_unsigned](#) (const unsigned \*\_\_addr)
- static signed int [pgm\\_read\\_signed\\_int](#) (const signed int \*\_\_addr)
- static unsigned int [pgm\\_read\\_unsigned\\_int](#) (const unsigned int \*\_\_addr)
- static [\\_\\_int24](#) [pgm\\_read\\_i24](#) (const [\\_\\_int24](#) \*\_\_addr)
- static [\\_\\_uint24](#) [pgm\\_read\\_u24](#) (const [\\_\\_uint24](#) \*\_\_addr)
- static [uint32\\_t](#) [pgm\\_read\\_u32](#) (const [uint32\\_t](#) \*\_\_addr)
- static [int32\\_t](#) [pgm\\_read\\_i32](#) (const [int32\\_t](#) \*\_\_addr)
- static long [pgm\\_read\\_long](#) (const long \*\_\_addr)
- static unsigned long [pgm\\_read\\_unsigned\\_long](#) (const unsigned long \*\_\_addr)
- static long long [pgm\\_read\\_long\\_long](#) (const long long \*\_\_addr)
- static unsigned long long [pgm\\_read\\_unsigned\\_long\\_long](#) (const unsigned long long \*\_\_addr)
- static [uint64\\_t](#) [pgm\\_read\\_u64](#) (const [uint64\\_t](#) \*\_\_addr)
- static [int64\\_t](#) [pgm\\_read\\_i64](#) (const [int64\\_t](#) \*\_\_addr)
- static float [pgm\\_read\\_float](#) (const float \*\_\_addr)
- static double [pgm\\_read\\_double](#) (const double \*\_\_addr)
- static long double [pgm\\_read\\_long\\_double](#) (const long double \*\_\_addr)
- static char [pgm\\_read\\_char\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static unsigned char [pgm\\_read\\_unsigned\\_char\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static signed char [pgm\\_read\\_signed\\_char\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [uint8\\_t](#) [pgm\\_read\\_u8\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [int8\\_t](#) [pgm\\_read\\_i8\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static short [pgm\\_read\\_short\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static unsigned short [pgm\\_read\\_unsigned\\_short\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [uint16\\_t](#) [pgm\\_read\\_u16\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [int16\\_t](#) [pgm\\_read\\_i16\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static int [pgm\\_read\\_int\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static unsigned [pgm\\_read\\_unsigned\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static unsigned int [pgm\\_read\\_unsigned\\_int\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static signed [pgm\\_read\\_signed\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static signed int [pgm\\_read\\_signed\\_int\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static long [pgm\\_read\\_long\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static unsigned long [pgm\\_read\\_unsigned\\_long\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [\\_\\_int24](#) [pgm\\_read\\_i24\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [\\_\\_uint24](#) [pgm\\_read\\_u24\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [uint32\\_t](#) [pgm\\_read\\_u32\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [int32\\_t](#) [pgm\\_read\\_i32\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static long long [pgm\\_read\\_long\\_long\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static unsigned long long [pgm\\_read\\_unsigned\\_long\\_long\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [uint64\\_t](#) [pgm\\_read\\_u64\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static [int64\\_t](#) [pgm\\_read\\_i64\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static float [pgm\\_read\\_float\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)
- static double [pgm\\_read\\_double\\_far](#) ([uint\\_farptr\\_t](#) \_\_addr)

- static long double [pgm\\_read\\_long\\_double\\_far](#) (uint\_farptr\_t \_\_addr)
- const void \* [memchr\\_P](#) (const void \*, int \_\_val, size\_t \_\_len)
- int [memcmp\\_P](#) (const void \*, const void \*, size\_t)
- void \* [memcpy\\_P](#) (void \*, const void \*, int \_\_val, size\_t)
- void \* [memcpy\\_P](#) (void \*, const void \*, size\_t)
- void \* [memmem\\_P](#) (const void \*, size\_t, const void \*, size\_t)
- const void \* [memrchr\\_P](#) (const void \*, int \_\_val, size\_t \_\_len)
- char \* [strcat\\_P](#) (char \*, const char \*)
- const char \* [strchr\\_P](#) (const char \*, int \_\_val)
- const char \* [strchrnul\\_P](#) (const char \*, int \_\_val)
- int [strcmp\\_P](#) (const char \*, const char \*)
- char \* [strcpy\\_P](#) (char \*, const char \*)
- int [strcasecmp\\_P](#) (const char \*, const char \*)
- char \* [strcasestr\\_P](#) (const char \*, const char \*)
- size\_t [strcspn\\_P](#) (const char \*\_\_s, const char \*\_\_reject)
- size\_t [strlcat\\_P](#) (char \*, const char \*, size\_t)
- size\_t [strlcpy\\_P](#) (char \*, const char \*, size\_t)
- size\_t [strlen\\_P](#) (const char \*, size\_t)
- int [strncmp\\_P](#) (const char \*, const char \*, size\_t)
- int [strncasecmp\\_P](#) (const char \*, const char \*, size\_t)
- char \* [strncat\\_P](#) (char \*, const char \*, size\_t)
- char \* [strncpy\\_P](#) (char \*, const char \*, size\_t)
- char \* [strpbrk\\_P](#) (const char \*\_\_s, const char \*\_\_accept)
- const char \* [strrchr\\_P](#) (const char \*, int \_\_val)
- char \* [strsep\\_P](#) (char \*\*\_\_sp, const char \*\_\_delim)
- size\_t [strspn\\_P](#) (const char \*\_\_s, const char \*\_\_accept)
- char \* [strstr\\_P](#) (const char \*, const char \*)
- char \* [strtok\\_P](#) (char \*\_\_s, const char \*\_\_delim)
- char \* [strtok\\_rP](#) (char \*\_\_s, const char \*\_\_delim, char \*\*\_\_last)
- size\_t [strlen\\_PF](#) (uint\_farptr\_t src)
- size\_t [strlen\\_PF](#) (uint\_farptr\_t src, size\_t len)
- void \* [memcpy\\_PF](#) (void \*dest, uint\_farptr\_t src, size\_t len)
- char \* [strcpy\\_PF](#) (char \*dest, uint\_farptr\_t src)
- char \* [strncpy\\_PF](#) (char \*dest, uint\_farptr\_t src, size\_t len)
- char \* [strcat\\_PF](#) (char \*dest, uint\_farptr\_t src)
- size\_t [strlcat\\_PF](#) (char \*dst, uint\_farptr\_t src, size\_t siz)
- char \* [strncat\\_PF](#) (char \*dest, uint\_farptr\_t src, size\_t len)
- int [strcmp\\_PF](#) (const char \*s1, uint\_farptr\_t s2)
- int [strncmp\\_PF](#) (const char \*s1, uint\_farptr\_t s2, size\_t n)
- int [strcasecmp\\_PF](#) (const char \*s1, uint\_farptr\_t s2)
- int [strncasecmp\\_PF](#) (const char \*s1, uint\_farptr\_t s2, size\_t n)
- uint\_farptr\_t [strchr\\_PF](#) (uint\_farptr\_t, int \_\_val)
- char \* [strstr\\_PF](#) (const char \*s1, uint\_farptr\_t s2)
- size\_t [strlcpy\\_PF](#) (char \*dst, uint\_farptr\_t src, size\_t siz)
- int [memcmp\\_PF](#) (const void \*, uint\_farptr\_t, size\_t)
- static size\_t [strlen\\_P](#) (const char \*s)
- template<typename T >  
T [pgm\\_read< T >](#) (const T \*addr)
- template<typename T >  
T [pgm\\_read\\_far< T >](#) (uint\_farptr\_t addr)

## 23.26 pgmspace.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002-2007 Marek Michalkiewicz
00002    Copyright (c) 2006, Carlos Lamas
00003    Copyright (c) 2009-2010, Jan Waclawek
00004    All rights reserved.
00005
00006    Redistribution and use in source and binary forms, with or without
00007    modification, are permitted provided that the following conditions are met:
00008
00009    * Redistributions of source code must retain the above copyright
00010    notice, this list of conditions and the following disclaimer.
00011    * Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in
00013    the documentation and/or other materials provided with the
00014    distribution.
00015    * Neither the name of the copyright holders nor the names of
00016    contributors may be used to endorse or promote products derived
00017    from this software without specific prior written permission.
00018
00019    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029    POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /*
00034    pgmspace.h
00035
00036    Contributors:
00037    Created by Marek Michalkiewicz <marekm@linux.org.pl>
00038    Eric B. Weddington <eric@ecentral.com>
00039    Wolfgang Haidinger <wh@vmars.tuwien.ac.at> (pgm_read_dword())
00040    Ivanov Anton <anton@arc.com.ru> (pgm_read_float())
00041 */
00042
00043 /** \file */
00044 /** \defgroup avr_pgmspace <avr/pgmspace.h>: Program Space Utilities
00045     \code
00046     #include <avr/io.h>
00047     #include <avr/pgmspace.h>
00048     \endcode
00049
00050     The functions in this module provide interfaces for a program to access
00051     data stored in program space (flash memory) of the device.
00052
00053     \note These functions are an attempt to provide some compatibility with
00054     header files that come with IAR C, to make porting applications between
00055     different compilers easier. This is not 100% compatibility though (GCC
00056     does not have full support for multiple address spaces yet).
00057
00058     \note If you are working with strings which are completely based in RAM,
00059     use the standard string functions described in \ref avr_string.
00060
00061     \note If possible, put your constant tables in the lower 64 KB and use
00062     pgm_read_byte_near() or pgm_read_word_near() instead of
00063     pgm_read_byte_far() or pgm_read_word_far() since it is more efficient that
00064     way, and you can still use the upper 64K for executable code.
00065     All functions that are suffixed with a \c _P \e require their
00066     arguments to be in the lower 64 KB of the flash ROM, as they do
00067     not use ELPM instructions. This is normally not a big concern as
00068     the linker setup arranges any program space constants declared
00069     using the macros from this header file so they are placed right after
00070     the interrupt vectors, and in front of any executable code. However,
00071     it can become a problem if there are too many of these constants, or
00072     for bootloaders on devices with more than 64 KB of ROM.
00073     <em>All these functions will not work in that situation.</em>
00074
00075     \note For <b>Xmega</b> devices, make sure the NVM controller
00076     command register (\c NVM.CMD or \c NVM_CMD) is set to 0x00 (NOP)
00077     before using any of these functions.
00078 */
00079
00080 #ifndef __PGMSPACE_H_
00081 #define __PGMSPACE_H_ 1
00082
00083 #ifndef __DOXYGEN__

```

```

00084 #define __need_size_t
00085 #endif
00086 #include <inttypes.h>
00087 #include <stddef.h>
00088 #include <avr/io.h>
00089
00090 #ifndef __DOXYGEN__
00091 #ifndef __ATTR_CONST__
00092 #define __ATTR_CONST__ __attribute__((__const__))
00093 #endif
00094
00095 #ifndef __ATTR_PROGMEM__
00096 #define __ATTR_PROGMEM__ __attribute__((__progmem__))
00097 #endif
00098
00099 #ifndef __ATTR_PURE__
00100 #define __ATTR_PURE__ __attribute__((__pure__))
00101 #endif
00102
00103 #ifndef __ATTR_ALWAYS_INLINE__
00104 #define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00105 #endif
00106
00107 #define PROGMEM __ATTR_PROGMEM__
00108
00109 #endif /* !__DOXYGEN__ */
00110
00111 /**
00112  \ingroup avr_pgmspace
00113  \def PROGMEM_FAR
00114
00115  Attribute to use in order to declare an object being located in
00116  far flash ROM. This is similar to #PROGMEM, except that it puts
00117  the static storage object in section
00118  <tt>\ref sec_dot_progmemx ".progmemx.data"</tt>.
00119  In order to access the object,
00120  the <tt>pgm_read_*_far</tt> and \c _PF functions declare in this header
00121  can be used. In order to get its address, see pgm_get_far_address().
00122
00123  It only makes sense to put read-only objects in this section,
00124  though the compiler does not diagnose when this is not the case. */
00125 #define PROGMEM_FAR __attribute__((__section__(".progmemx.data")))
00126
00127 #ifdef __DOXYGEN__
00128
00129 /**
00130  \ingroup avr_pgmspace
00131  \def PROGMEM
00132
00133  Attribute to use in order to declare an object being located in
00134  flash ROM. */
00135 #define PROGMEM __attribute__((__progmem__))
00136
00137 /** \ingroup avr_pgmspace
00138  \fn char pgm_read_char (const char *__addr)
00139  Read a <tt>char</tt> from 16-bit (near) byte-address \p __addr.
00140  The address is in the lower 64 KiB of program memory. */
00141 static inline char pgm_read_char (const char *__addr);
00142
00143 /** \ingroup avr_pgmspace
00144  \fn unsigned char pgm_read_unsigned_char (const unsigned char *__addr)
00145  Read an <tt>unsigned char</tt> from 16-bit (near) byte-address \p __addr.
00146  The address is in the lower 64 KiB of program memory. */
00147 static inline unsigned char pgm_read_unsigned_char (const unsigned char *__addr);
00148
00149 /** \ingroup avr_pgmspace
00150  \fn signed char pgm_read_signed_char (const signed char *__addr)
00151  Read a <tt>signed char</tt> from 16-bit (near) byte-address \p __addr.
00152  The address is in the lower 64 KiB of program memory. */
00153 static inline signed char pgm_read_signed_char (const signed char *__addr);
00154
00155 /** \ingroup avr_pgmspace
00156  \fn uint8_t pgm_read_u8 (const uint8_t *__addr)
00157  Read an <tt>uint8_t</tt> from 16-bit (near) byte-address \p __addr.
00158  The address is in the lower 64 KiB of program memory. */
00159 static inline uint8_t pgm_read_u8 (const uint8_t *__addr);
00160
00161 /** \ingroup avr_pgmspace
00162  \fn int8_t pgm_read_i8 (const int8_t *__addr)
00163  Read an <tt>int8_t</tt> from 16-bit (near) byte-address \p __addr.
00164  The address is in the lower 64 KiB of program memory. */
00165 static inline int8_t pgm_read_i8 (const int8_t *__addr);
00166
00167 /** \ingroup avr_pgmspace
00168  \fn short pgm_read_short (const short *__addr)
00169  Read a <tt>short</tt> from 16-bit (near) byte-address \p __addr.
00170  The address is in the lower 64 KiB of program memory. */

```



```
00171 static inline short pgm_read_short (const short *__addr);
00172
00173 /** \ingroup avr_pgmSPACE
00174     \fn unsigned short pgm_read_unsigned_short (const unsigned short *__addr)
00175     Read an <tt>unsigned short</tt> from 16-bit (near) byte-address \p __addr.
00176     The address is in the lower 64 KiB of program memory. */
00177 static inline unsigned short pgm_read_unsigned_short (const unsigned short *__addr);
00178
00179 /** \ingroup avr_pgmSPACE
00180     \fn uint16_t pgm_read_u16 (const uint16_t *__addr)
00181     Read an <tt>uint16_t</tt> from 16-bit (near) byte-address \p __addr.
00182     The address is in the lower 64 KiB of program memory. */
00183 static inline uint16_t pgm_read_u16 (const uint16_t *__addr);
00184
00185 /** \ingroup avr_pgmSPACE
00186     \fn int16_t pgm_read_i16 (const int16_t *__addr)
00187     Read an <tt>int16_t</tt> from 16-bit (near) byte-address \p __addr.
00188     The address is in the lower 64 KiB of program memory. */
00189 static inline int16_t pgm_read_i16 (const int16_t *__addr);
00190
00191 /** \ingroup avr_pgmSPACE
00192     \fn int pgm_read_int (const int *__addr)
00193     Read an <tt>int</tt> from 16-bit (near) byte-address \p __addr.
00194     The address is in the lower 64 KiB of program memory. */
00195 static inline int pgm_read_int (const int *__addr);
00196
00197 /** \ingroup avr_pgmSPACE
00198     \fn signed pgm_read_signed (const signed *__addr)
00199     Read a <tt>signed</tt> from 16-bit (near) byte-address \p __addr.
00200     The address is in the lower 64 KiB of program memory. */
00201 static inline signed pgm_read_signed (const signed *__addr);
00202
00203 /** \ingroup avr_pgmSPACE
00204     \fn unsigned pgm_read_unsigned (const unsigned *__addr)
00205     Read an <tt>unsigned</tt> from 16-bit (near) byte-address \p __addr.
00206     The address is in the lower 64 KiB of program memory. */
00207 static inline unsigned pgm_read_unsigned (const unsigned *__addr);
00208
00209 /** \ingroup avr_pgmSPACE
00210     \fn signed int pgm_read_signed_int (const signed int *__addr)
00211     Read a <tt>signed int</tt> from 16-bit (near) byte-address \p __addr.
00212     The address is in the lower 64 KiB of program memory. */
00213 static inline signed int pgm_read_signed_int (const signed int *__addr);
00214
00215 /** \ingroup avr_pgmSPACE
00216     \fn unsigned int pgm_read_unsigned_int (const unsigned int *__addr)
00217     Read an <tt>unsigned int</tt> from 16-bit (near) byte-address \p __addr.
00218     The address is in the lower 64 KiB of program memory. */
00219 static inline unsigned int pgm_read_unsigned_int (const unsigned int *__addr);
00220
00221 /** \ingroup avr_pgmSPACE
00222     \fn __int24 pgm_read_i24 (const __int24 *__addr)
00223     Read an <tt>__int24</tt> from 16-bit (near) byte-address \p __addr.
00224     The address is in the lower 64 KiB of program memory. */
00225 static inline __int24 pgm_read_i24 (const __int24 *__addr);
00226
00227 /** \ingroup avr_pgmSPACE
00228     \fn __uint24 pgm_read_u24 (const __uint24 *__addr)
00229     Read an <tt>__uint24</tt> from 16-bit (near) byte-address \p __addr.
00230     The address is in the lower 64 KiB of program memory. */
00231 static inline __uint24 pgm_read_u24 (const __uint24 *__addr);
00232
00233 /** \ingroup avr_pgmSPACE
00234     \fn uint32_t pgm_read_u32 (const uint32_t *__addr)
00235     Read an <tt>uint32_t</tt> from 16-bit (near) byte-address \p __addr.
00236     The address is in the lower 64 KiB of program memory. */
00237 static inline uint32_t pgm_read_u32 (const uint32_t *__addr);
00238
00239 /** \ingroup avr_pgmSPACE
00240     \fn int32_t pgm_read_i32 (const int32_t *__addr)
00241     Read an <tt>int32_t</tt> from 16-bit (near) byte-address \p __addr.
00242     The address is in the lower 64 KiB of program memory. */
00243 static inline int32_t pgm_read_i32 (const int32_t *__addr);
00244
00245 /** \ingroup avr_pgmSPACE
00246     \fn long pgm_read_long (const long *__addr)
00247     Read a <tt>long</tt> from 16-bit (near) byte-address \p __addr.
00248     The address is in the lower 64 KiB of program memory. */
00249 static inline long pgm_read_long (const long *__addr);
00250
00251 /** \ingroup avr_pgmSPACE
00252     \fn unsigned long pgm_read_unsigned_long (const unsigned long *__addr)
00253     Read an <tt>unsigned long</tt> from 16-bit (near) byte-address \p __addr.
00254     The address is in the lower 64 KiB of program memory. */
00255 static inline unsigned long pgm_read_unsigned_long (const unsigned long *__addr);
00256
00257 /** \ingroup avr_pgmSPACE
```

```

00258     \fn long long pgm_read_long_long (const long long *__addr)
00259     Read a <tt>long long</tt> from 16-bit (near) byte-address \p __addr.
00260     The address is in the lower 64 KiB of program memory. */
00261 static inline long long pgm_read_long_long (const long long *__addr);
00262
00263 /** \ingroup avr_pgmspace
00264     \fn unsigned long long pgm_read_unsigned_long_long (const unsigned long long *__addr)
00265     Read an <tt>unsigned long long</tt> from 16-bit (near) byte-address
00266     \p __addr.
00267     The address is in the lower 64 KiB of program memory. */
00268 static inline unsigned long long pgm_read_unsigned_long_long (const unsigned long long *__addr);
00269
00270 /** \ingroup avr_pgmspace
00271     \fn uint64_t pgm_read_u64 (const uint64_t *__addr)
00272     Read an <tt>uint64_t</tt> from 16-bit (near) byte-address \p __addr.
00273     The address is in the lower 64 KiB of program memory. */
00274 static inline uint64_t pgm_read_u64 (const uint64_t *__addr);
00275
00276 /** \ingroup avr_pgmspace
00277     \fn int64_t pgm_read_i64 (const int64_t *__addr)
00278     Read a <tt>int64_t</tt> from 16-bit (near) byte-address \p __addr.
00279     The address is in the lower 64 KiB of program memory. */
00280 static inline int64_t pgm_read_i64 (const int64_t *__addr);
00281
00282 /** \ingroup avr_pgmspace
00283     \fn float pgm_read_float (const float *__addr)
00284     Read a <tt>float</tt> from 16-bit (near) byte-address \p __addr.
00285     The address is in the lower 64 KiB of program memory. */
00286 static inline float pgm_read_float (const float *__addr);
00287
00288 /** \ingroup avr_pgmspace
00289     \fn double pgm_read_double (const double *__addr)
00290     Read a <tt>double</tt> from 16-bit (near) byte-address \p __addr.
00291     The address is in the lower 64 KiB of program memory. */
00292 static inline double pgm_read_double (const double *__addr);
00293
00294 /** \ingroup avr_pgmspace
00295     \fn long double pgm_read_long_double (const long double *__addr)
00296     Read a <tt>long double</tt> from 16-bit (near) byte-address \p __addr.
00297     The address is in the lower 64 KiB of program memory. */
00298 static inline long double pgm_read_long_double (const long double *__addr);
00299
00300 #else /* !DOXYGEN */
00301 #if defined(__AVR_TINY__)
00302 /* For Reduced Tiny devices, avr-gcc adds 0x4000 when it takes the address
00303 of a PROGMEM object. This means we can use open coded C/C++ to read
00304 from progmem. This assumes we have
00305 - GCC PR71948 - Make progmem work on Reduced Tiny (GCC v7 / 2016-08) */
00306 #define __LPM__1(res, addr) res = *addr
00307 #define __LPM__2(res, addr) res = *addr
00308 #define __LPM__3(res, addr) res = *addr
00309 #define __LPM__4(res, addr) res = *addr
00310 #define __LPM__8(res, addr) res = *addr
00311
00312 #elif defined(__AVR_HAVE_LPMX__)
00313 #define __LPM__1(res, addr) \
00314     __asm __volatile__ ("lpm %0,%a1" \
00315         : "=r" (res) : "z" (addr)) \
00316
00317 #define __LPM__2(res, addr) \
00318     __asm __volatile__ ("lpm %A0,%a1+" \
00319         : "=r" (res), "+z" (addr)) \
00320     "\n\t" \
00321
00322 #define __LPM__3(res, addr) \
00323     __asm __volatile__ ("lpm %A0,%a1+" \
00324         : "=r" (res), "+z" (addr)) \
00325     "\n\t" \
00326     "lpm %B0,%a1+" \
00327     "\n\t" \
00328     "lpm %C0,%a1+" \
00329     "\n\t" \
00330     "lpm %D0,%a1+" \
00331     : "=r" (res), "+z" (addr)) \
00332
00333 #define __LPM__8(res, addr) \
00334     __asm __volatile__ ("lpm %r0+0,%a1+" \
00335         : "=r" (res), "+z" (addr)) \
00336     "\n\t" \
00337     "lpm %r0+1,%a1+" \
00338     "\n\t" \
00339     "lpm %r0+2,%a1+" \
00340     "\n\t" \
00341     "lpm %r0+3,%a1+" \
00342     "\n\t" \
00343     "lpm %r0+4,%a1+" \
00344     "\n\t" \
00345     "lpm %r0+5,%a1+" \
00346     "\n\t" \
00347     "lpm %r0+6,%a1+" \
00348     "\n\t" \
00349     "lpm %r0+7,%a1+" \
00350     : "=r" (res), "+z" (addr)) \

```

```

00345 #else /* Has no LPMx and no Reduced Tiny => Has LPM. */
00346 #define __LPM_1(res, addr) \
00347     __asm __volatile__ ("lpm $ mov %A0,r0" \
00348         : "=r" (res) : "z" (addr) : "r0")
00349
00350 #define __LPM_2(res, addr) \
00351     __asm __volatile__ ("lpm $ mov %A0,r0 $ adiw %1,1" "\n\t" \
00352         "lpm $ mov %B0,r0" \
00353         : "=r" (res), "+z" (addr) :: "r0")
00354
00355 #define __LPM_3(res, addr) \
00356     __asm __volatile__ ("lpm $ mov %A0,r0 $ adiw %1,1" "\n\t" \
00357         "lpm $ mov %B0,r0 $ adiw %1,1" "\n\t" \
00358         "lpm $ mov %C0,r0" \
00359         : "=r" (res), "+z" (addr) :: "r0")
00360
00361 #define __LPM_4(res, addr) \
00362     __asm __volatile__ ("lpm $ mov %A0,r0 $ adiw %1,1" "\n\t" \
00363         "lpm $ mov %B0,r0 $ adiw %1,1" "\n\t" \
00364         "lpm $ mov %C0,r0 $ adiw %1,1" "\n\t" \
00365         "lpm $ mov %D0,r0" \
00366         : "=r" (res), "+z" (addr) :: "r0")
00367
00368 #define __LPM_8(res, addr) \
00369     __asm __volatile__ ("lpm $ mov %r0+0,r0 $ adiw %1,1" "\n\t" \
00370         "lpm $ mov %r0+1,r0 $ adiw %1,1" "\n\t" \
00371         "lpm $ mov %r0+2,r0 $ adiw %1,1" "\n\t" \
00372         "lpm $ mov %r0+3,r0 $ adiw %1,1" "\n\t" \
00373         "lpm $ mov %r0+4,r0 $ adiw %1,1" "\n\t" \
00374         "lpm $ mov %r0+5,r0 $ adiw %1,1" "\n\t" \
00375         "lpm $ mov %r0+6,r0 $ adiw %1,1" "\n\t" \
00376         "lpm $ mov %r0+7,r0" \
00377         : "=r" (res), "+z" (addr) :: "r0")
00378 #endif /* LPM cases */
00379
00380 #define _Avrlibc_Def_Pgm_1(Name, Typ) \
00381     static __ATTR_ALWAYS_INLINE__ \
00382     Typ pgm_read_##Name (const Typ *__addr) \
00383     { \
00384         Typ __res; \
00385         __LPM_1 (__res, __addr); \
00386         return __res; \
00387     }
00388
00389 #define _Avrlibc_Def_Pgm_2(Name, Typ) \
00390     static __ATTR_ALWAYS_INLINE__ \
00391     Typ pgm_read_##Name (const Typ *__addr) \
00392     { \
00393         Typ __res; \
00394         __LPM_2 (__res, __addr); \
00395         return __res; \
00396     }
00397
00398 #define _Avrlibc_Def_Pgm_3(Name, Typ) \
00399     static __ATTR_ALWAYS_INLINE__ \
00400     Typ pgm_read_##Name (const Typ *__addr) \
00401     { \
00402         Typ __res; \
00403         __LPM_3 (__res, __addr); \
00404         return __res; \
00405     }
00406
00407 #define _Avrlibc_Def_Pgm_4(Name, Typ) \
00408     static __ATTR_ALWAYS_INLINE__ \
00409     Typ pgm_read_##Name (const Typ *__addr) \
00410     { \
00411         Typ __res; \
00412         __LPM_4 (__res, __addr); \
00413         return __res; \
00414     }
00415
00416 #define _Avrlibc_Def_Pgm_8(Name, Typ) \
00417     static __ATTR_ALWAYS_INLINE__ \
00418     Typ pgm_read_##Name (const Typ *__addr) \
00419     { \
00420         Typ __res; \
00421         __LPM_8 (__res, __addr); \
00422         return __res; \
00423     }
00424
00425 _Avrlibc_Def_Pgm_1 (char, char)
00426 _Avrlibc_Def_Pgm_1 (unsigned_char, unsigned char)
00427 _Avrlibc_Def_Pgm_1 (signed_char, signed char)
00428 _Avrlibc_Def_Pgm_1 (u8, uint8_t)
00429 _Avrlibc_Def_Pgm_1 (i8, int8_t)
00430 #if __SIZEOF_INT__ == 1
00431 _Avrlibc_Def_Pgm_1 (int, int)

```

```
00432 _Avrlibc_Def_Pgm_1 (signed, signed)
00433 _Avrlibc_Def_Pgm_1 (unsigned, unsigned)
00434 _Avrlibc_Def_Pgm_1 (signed_int, signed int)
00435 _Avrlibc_Def_Pgm_1 (unsigned_int, unsigned int)
00436 #endif
00437 #if __SIZEOF_SHORT__ == 1
00438 _Avrlibc_Def_Pgm_1 (short, short)
00439 _Avrlibc_Def_Pgm_1 (unsigned_short, unsigned short)
00440 #endif
00441
00442 _Avrlibc_Def_Pgm_2 (u16, uint16_t)
00443 _Avrlibc_Def_Pgm_2 (i16, int16_t)
00444 #if __SIZEOF_INT__ == 2
00445 _Avrlibc_Def_Pgm_2 (int, int)
00446 _Avrlibc_Def_Pgm_2 (signed, signed)
00447 _Avrlibc_Def_Pgm_2 (unsigned, unsigned)
00448 _Avrlibc_Def_Pgm_2 (signed_int, signed int)
00449 _Avrlibc_Def_Pgm_2 (unsigned_int, unsigned int)
00450 #endif
00451 #if __SIZEOF_SHORT__ == 2
00452 _Avrlibc_Def_Pgm_2 (short, short)
00453 _Avrlibc_Def_Pgm_2 (unsigned_short, unsigned short)
00454 #endif
00455 #if __SIZEOF_LONG__ == 2
00456 _Avrlibc_Def_Pgm_2 (long, long)
00457 _Avrlibc_Def_Pgm_2 (unsigned_long, unsigned long)
00458 #endif
00459
00460 #if defined(__INT24_MAX__)
00461 _Avrlibc_Def_Pgm_3 (i24, __int24)
00462 _Avrlibc_Def_Pgm_3 (u24, __uint24)
00463 #endif /* Have __int24 */
00464
00465 _Avrlibc_Def_Pgm_4 (u32, uint32_t)
00466 _Avrlibc_Def_Pgm_4 (i32, int32_t)
00467 _Avrlibc_Def_Pgm_4 (float, float)
00468 #if __SIZEOF_LONG__ == 4
00469 _Avrlibc_Def_Pgm_4 (long, long)
00470 _Avrlibc_Def_Pgm_4 (unsigned_long, unsigned long)
00471 #endif
00472 #if __SIZEOF_LONG_LONG__ == 4
00473 _Avrlibc_Def_Pgm_4 (long_long, long long)
00474 _Avrlibc_Def_Pgm_4 (unsigned_long_long, unsigned long long)
00475 #endif
00476 #if __SIZEOF_DOUBLE__ == 4
00477 _Avrlibc_Def_Pgm_4 (double, double)
00478 #endif
00479 #if __SIZEOF_LONG_DOUBLE__ == 4
00480 _Avrlibc_Def_Pgm_4 (long_double, long double)
00481 #endif
00482
00483 #if __SIZEOF_LONG_LONG__ == 8
00484 _Avrlibc_Def_Pgm_8 (u64, uint64_t)
00485 _Avrlibc_Def_Pgm_8 (i64, int64_t)
00486 _Avrlibc_Def_Pgm_8 (long_long, long long)
00487 _Avrlibc_Def_Pgm_8 (unsigned_long_long, unsigned long long)
00488 #endif
00489 #if __SIZEOF_DOUBLE__ == 8
00490 _Avrlibc_Def_Pgm_8 (double, double)
00491 #endif
00492 #if __SIZEOF_LONG_DOUBLE__ == 8
00493 _Avrlibc_Def_Pgm_8 (long_double, long double)
00494 #endif
00495
00496 #endif /* DOXYGEN */
00497
00498 #ifdef __DOXYGEN__
00499
00500 /** \ingroup avr_pgmspace
00501     \fn char pgm_read_char_far (uint_farptr_t __addr)
00502     Read a <tt>char</tt> from far byte-address \p __addr.
00503     The address is in the program memory. */
00504 static inline char pgm_read_char_far (uint_farptr_t __addr);
00505
00506 /** \ingroup avr_pgmspace
00507     \fn unsigned char pgm_read_unsigned_char_far (uint_farptr_t __addr)
00508     Read an <tt>unsigned char</tt> from far byte-address \p __addr.
00509     The address is in the program memory. */
00510 static inline unsigned char pgm_read_unsigned_char_far (uint_farptr_t __addr);
00511
00512 /** \ingroup avr_pgmspace
00513     \fn signed char pgm_read_signed_char_far (uint_farptr_t __addr)
00514     Read a <tt>signed char</tt> from far byte-address \p __addr.
00515     The address is in the program memory. */
00516 static inline signed char pgm_read_signed_char_far (uint_farptr_t __addr);
00517
00518 /** \ingroup avr_pgmspace
```

```
00519     \fn uint8_t pgm_read_u8_far (uint_farptr_t __addr)
00520     Read an <tt>uint8_t</tt> from far byte-address \p __addr.
00521     The address is in the program memory. */
00522 static inline uint8_t pgm_read_u8_far (uint_farptr_t __addr);
00523
00524 /** \ingroup avr_pgmspace
00525     \fn int8_t pgm_read_i8_far (uint_farptr_t __addr)
00526     Read an <tt>int8_t</tt> from far byte-address \p __addr.
00527     The address is in the program memory. */
00528 static inline int8_t pgm_read_i8_far (uint_farptr_t __addr);
00529
00530 /** \ingroup avr_pgmspace
00531     \fn short pgm_read_short_far (uint_farptr_t __addr)
00532     Read a <tt>short</tt> from far byte-address \p __addr.
00533     The address is in the program memory. */
00534 static inline short pgm_read_short_far (uint_farptr_t __addr);
00535
00536 /** \ingroup avr_pgmspace
00537     \fn unsigned short pgm_read_unsigned_short_far (uint_farptr_t __addr)
00538     Read an <tt>unsigned short</tt> from far byte-address \p __addr.
00539     The address is in the program memory. */
00540 static inline unsigned short pgm_read_unsigned_short_far (uint_farptr_t __addr);
00541
00542 /** \ingroup avr_pgmspace
00543     \fn uint16_t pgm_read_u16_far (uint_farptr_t __addr)
00544     Read an <tt>uint16_t</tt> from far byte-address \p __addr.
00545     The address is in the program memory. */
00546 static inline uint16_t pgm_read_u16_far (uint_farptr_t __addr);
00547
00548 /** \ingroup avr_pgmspace
00549     \fn int16_t pgm_read_i16_far (uint_farptr_t __addr)
00550     Read an <tt>int16_t</tt> from far byte-address \p __addr.
00551     The address is in the program memory. */
00552 static inline int16_t pgm_read_i16_far (uint_farptr_t __addr);
00553
00554 /** \ingroup avr_pgmspace
00555     \fn int pgm_read_int_far (uint_farptr_t __addr)
00556     Read an <tt>int</tt> from far byte-address \p __addr.
00557     The address is in the program memory. */
00558 static inline int pgm_read_int_far (uint_farptr_t __addr);
00559
00560 /** \ingroup avr_pgmspace
00561     \fn unsigned pgm_read_unsigned_far (uint_farptr_t __addr)
00562     Read an <tt>unsigned</tt> from far byte-address \p __addr.
00563     The address is in the program memory. */
00564 static inline unsigned pgm_read_unsigned_far (uint_farptr_t __addr);
00565
00566 /** \ingroup avr_pgmspace
00567     \fn unsigned int pgm_read_unsigned_int_far (uint_farptr_t __addr)
00568     Read an <tt>unsigned int</tt> from far byte-address \p __addr.
00569     The address is in the program memory. */
00570 static inline unsigned int pgm_read_unsigned_int_far (uint_farptr_t __addr);
00571
00572 /** \ingroup avr_pgmspace
00573     \fn signed pgm_read_signed_far (uint_farptr_t __addr)
00574     Read a <tt>signed</tt> from far byte-address \p __addr.
00575     The address is in the program memory. */
00576 static inline signed pgm_read_signed_far (uint_farptr_t __addr);
00577
00578 /** \ingroup avr_pgmspace
00579     \fn signed int pgm_read_signed_int_far (uint_farptr_t __addr)
00580     Read a <tt>signed int</tt> from far byte-address \p __addr.
00581     The address is in the program memory. */
00582 static inline signed int pgm_read_signed_int_far (uint_farptr_t __addr);
00583
00584 /** \ingroup avr_pgmspace
00585     \fn long pgm_read_long_far (uint_farptr_t __addr)
00586     Read a <tt>long</tt> from far byte-address \p __addr.
00587     The address is in the program memory. */
00588 static inline long pgm_read_long_far (uint_farptr_t __addr);
00589
00590 /** \ingroup avr_pgmspace
00591     \fn unsigned long pgm_read_unsigned_long_far (uint_farptr_t __addr)
00592     Read an <tt>unsigned long</tt> from far byte-address \p __addr.
00593     The address is in the program memory. */
00594 static inline unsigned long pgm_read_unsigned_long_far (uint_farptr_t __addr);
00595
00596 /** \ingroup avr_pgmspace
00597     \fn __int24 pgm_read_i24_far (uint_farptr_t __addr)
00598     Read an <tt>__int24</tt> from far byte-address \p __addr.
00599     The address is in the program memory. */
00600 static inline __int24 pgm_read_i24_far (uint_farptr_t __addr);
00601
00602 /** \ingroup avr_pgmspace
00603     \fn __uint24 pgm_read_u24_far (uint_farptr_t __addr)
00604     Read an <tt>__uint24</tt> from far byte-address \p __addr.
00605     The address is in the program memory. */
```

```

00606 static inline __uint24 pgm_read_u24_far (uint_farptr_t __addr);
00607
00608 /** \ingroup avr_pgmspace
00609     \fn uint32_t pgm_read_u32_far (uint_farptr_t __addr)
00610     Read an <tt>uint32_t</tt> from far byte-address \p __addr.
00611     The address is in the program memory. */
00612 static inline uint32_t pgm_read_u32_far (uint_farptr_t __addr);
00613
00614 /** \ingroup avr_pgmspace
00615     \fn int32_t pgm_read_i32_far (uint_farptr_t __addr)
00616     Read an <tt>int32_t</tt> from far byte-address \p __addr.
00617     The address is in the program memory. */
00618 static inline int32_t pgm_read_i32_far (uint_farptr_t __addr);
00619
00620 /** \ingroup avr_pgmspace
00621     \fn long long pgm_read_long_long_far (uint_farptr_t __addr)
00622     Read a <tt>long long</tt> from far byte-address \p __addr.
00623     The address is in the program memory. */
00624 static inline long long pgm_read_long_long_far (uint_farptr_t __addr);
00625
00626 /** \ingroup avr_pgmspace
00627     \fn unsigned long long pgm_read_unsigned_long_long_far (uint_farptr_t __addr)
00628     Read an <tt>unsigned long long</tt> from far byte-address \p __addr.
00629     The address is in the program memory. */
00630 static inline unsigned long long pgm_read_unsigned_long_long_far (uint_farptr_t __addr);
00631
00632 /** \ingroup avr_pgmspace
00633     \fn uint64_t pgm_read_u64_far (uint_farptr_t __addr)
00634     Read an <tt>uint64_t</tt> from far byte-address \p __addr.
00635     The address is in the program memory. */
00636 static inline uint64_t pgm_read_u64_far (uint_farptr_t __addr);
00637
00638 /** \ingroup avr_pgmspace
00639     \fn int64_t pgm_read_i64_far (uint_farptr_t __addr)
00640     Read an <tt>int64_t</tt> from far byte-address \p __addr.
00641     The address is in the program memory. */
00642 static inline int64_t pgm_read_i64_far (uint_farptr_t __addr);
00643
00644 /** \ingroup avr_pgmspace
00645     \fn float pgm_read_float_far (uint_farptr_t __addr)
00646     Read a <tt>float</tt> from far byte-address \p __addr.
00647     The address is in the program memory. */
00648 static inline float pgm_read_float_far (uint_farptr_t __addr);
00649
00650 /** \ingroup avr_pgmspace
00651     \fn double pgm_read_double_far (uint_farptr_t __addr)
00652     Read a <tt>double</tt> from far byte-address \p __addr.
00653     The address is in the program memory. */
00654 static inline double pgm_read_double_far (uint_farptr_t __addr);
00655
00656 /** \ingroup avr_pgmspace
00657     \fn long double pgm_read_long_double_far (uint_farptr_t __addr)
00658     Read a <tt>long double</tt> from far byte-address \p __addr.
00659     The address is in the program memory. */
00660 static inline long double pgm_read_long_double_far (uint_farptr_t __addr);
00661
00662 #else /* !DOXYGEN */
00663
00664 #if defined(__AVR_HAVE_ELPMX__)
00665
00666 #ifdef __AVR_HAVE_RAMPD__
00667 /* For devices with EBI, reset RAMPZ to zero after. */
00668 #define __pgm_clr_RAMPZ_ "\n\t" "out %i2, __zero_reg_"
00669 #else
00670 /* Devices without EBI: no need to reset RAMPZ. */
00671 #define __pgm_clr_RAMPZ_ /* empty */
00672 #endif
00673
00674 #define __ELPM__1(res, addr, T)
00675     __asm __volatile__ ("movw r30,%1"           "\n\t" //
00676                        "out %i2,%C1"          "\n\t" //
00677                        "elpm %A0,Z"           //
00678                        __pgm_clr_RAMPZ_       //
00679                        : "=r" (res)           //
00680                        : "r" (addr), "n" (& RAMPZ) //
00681                        : "r30", "r31")
00682
00683 #define __ELPM__2(res, addr, T)
00684     __asm __volatile__ ("movw r30,%1"           "\n\t" //
00685                        "out %i2,%C1"          "\n\t" //
00686                        "elpm %A0,Z+"         "\n\t" //
00687                        "elpm %B0,Z+"         //
00688                        __pgm_clr_RAMPZ_       //
00689                        : "=r" (res)           //
00690                        : "r" (addr), "n" (& RAMPZ) //
00691                        : "r30", "r31")
00692

```

```

00693 #define __ELPM__3(res, addr, T)
00694 __asm __volatile__ ("movw r30,%1" "\n\t" //
00695 "out %i2,%C1" "\n\t" //
00696 "elpm %A0,Z+" "\n\t" //
00697 "elpm %B0,Z+" "\n\t" //
00698 "elpm %C0,Z+" //
00699 __pgm_clr_RAMPZ_ //
00700 : "=r" (res) //
00701 : "r" (addr), "n" (& RAMPZ) //
00702 : "r30", "r31") //
00703
00704 #define __ELPM__4(res, addr, T)
00705 __asm __volatile__ ("movw r30,%1" "\n\t" //
00706 "out %i2,%C1" "\n\t" //
00707 "elpm %A0,Z+" "\n\t" //
00708 "elpm %B0,Z+" "\n\t" //
00709 "elpm %C0,Z+" "\n\t" //
00710 "elpm %D0,Z+" //
00711 __pgm_clr_RAMPZ_ //
00712 : "=r" (res) //
00713 : "r" (addr), "n" (& RAMPZ) //
00714 : "r30", "r31") //
00715
00716 #define __ELPM__8(res, addr, T)
00717 __asm __volatile__ ("movw r30,%1" "\n\t" //
00718 "out %i2,%C1" "\n\t" //
00719 "elpm %r0+0,Z+" "\n\t" //
00720 "elpm %r0+1,Z+" "\n\t" //
00721 "elpm %r0+2,Z+" "\n\t" //
00722 "elpm %r0+3,Z+" "\n\t" //
00723 "elpm %r0+4,Z+" "\n\t" //
00724 "elpm %r0+5,Z+" "\n\t" //
00725 "elpm %r0+6,Z+" "\n\t" //
00726 "elpm %r0+7,Z+" //
00727 __pgm_clr_RAMPZ_ //
00728 : "=r" (res) //
00729 : "r" (addr), "n" (& RAMPZ) //
00730 : "r30", "r31") //
00731
00732 /* FIXME: AT43USB320 does not have RAMPZ but supports (external) program
00733 memory of 64 KiW, at least that's what the comments in io43usb32x.h are
00734 indicating. A solution would be to put the device in a different
00735 multilib-set (see GCC PR78275), as io.h has "#define FLASHEND 0x0FFFF".
00736 For now, just exclude AT43USB320 from code that uses RAMPZ. Also note
00737 that the manual asserts that the entire program memory can be accessed
00738 by LPM, implying only 64 KiB of program memory. */
00739 #elif defined(__AVR_HAVE_ELPM__) \
00740 && !defined(__AVR_AT43USB320__)
00741 /* The poor devices without ELPMx: Do 24-bit addresses by hand... */
00742 #define __ELPM__1(res, addr, T)
00743 __asm __volatile__ ("mov r30,%A1" "\n\t" //
00744 "mov r31,%B1" "\n\t" //
00745 "out %i2,%C1 $ elpm $ mov %A0,r0" //
00746 : "=r" (res) //
00747 : "r" (addr), "n" (& RAMPZ) //
00748 : "r30", "r31", "r0") //
00749
00750 #define __ELPM__2(res, addr, T)
00751 __asm __volatile__ //
00752 ("mov r30,%A1" "\n\t" //
00753 "mov r31,%B1" "\n\t" //
00754 "mov %B0,%C1" "\n\t" //
00755 "out %i2,%B0 $ elpm $ mov %A0,r0 $ adiw r30,1 $ adc %B0,r1" "\n\t" //
00756 "out %i2,%B0 $ elpm $ mov %B0,r0" //
00757 : "=r" (res) //
00758 : "r" (addr), "n" (& RAMPZ) //
00759 : "r30", "r31", "r0") //
00760
00761 #define __ELPM__3(res, addr, T)
00762 __asm __volatile__ //
00763 ("mov r30,%A1" "\n\t" //
00764 "mov r31,%B1" "\n\t" //
00765 "mov %C0,%C1" "\n\t" //
00766 "out %i2,%C0 $ elpm $ mov %A0,r0 $ adiw r30,1 $ adc %C0,r1" "\n\t" //
00767 "out %i2,%C0 $ elpm $ mov %B0,r0 $ adiw r30,1 $ adc %C0,r1" "\n\t" //
00768 "out %i2,%C0 $ elpm $ mov %C0,r0" //
00769 : "=r" (res) //
00770 : "r" (addr), "n" (& RAMPZ) //
00771 : "r30", "r31", "r0") //
00772
00773 #define __ELPM__4(res, addr, T)
00774 __asm __volatile__ //
00775 ("mov r30,%A1" "\n\t" //
00776 "mov r31,%B1" "\n\t" //
00777 "mov %D0,%C1" "\n\t" //
00778 "out %i2,%D0 $ elpm $ mov %A0,r0 $ adiw r30,1 $ adc %D0,r1" "\n\t" //
00779 "out %i2,%D0 $ elpm $ mov %B0,r0 $ adiw r30,1 $ adc %D0,r1" "\n\t" //

```

```

00780     "out %i2,%D0 $ elpm $ mov %C0,r0 $ adiw r30,1 $ adc %D0,r1" "\n\t"  \
00781     "out %i2,%D0 $ elpm $ mov %D0,r0"                                \
00782     : "r" (res)                                                       \
00783     : "r" (addr), "n" (& RAMPZ)                                       \
00784     : "r30", "r31", "r0"                                             \
00785
00786 #define __ELPM__8(res, addr, T)                                       \
00787     __asm __volatile__                                                \
00788     ("mov r30,%A1" "\n\t"                                             \
00789     "mov r31,%B1" "\n\t"                                             \
00790     "mov %r0+7,%C1" "\n\t"                                           \
00791     "out %i2,%r0+7 $ elpm $ mov %r0+0,r0 $ adiw r30,1 $ adc %r0+7,r1" "\n\t" \
00792     "out %i2,%r0+7 $ elpm $ mov %r0+1,r0 $ adiw r30,1 $ adc %r0+7,r1" "\n\t" \
00793     "out %i2,%r0+7 $ elpm $ mov %r0+2,r0 $ adiw r30,1 $ adc %r0+7,r1" "\n\t" \
00794     "out %i2,%r0+7 $ elpm $ mov %r0+3,r0 $ adiw r30,1 $ adc %r0+7,r1" "\n\t" \
00795     "out %i2,%r0+7 $ elpm $ mov %r0+4,r0 $ adiw r30,1 $ adc %r0+7,r1" "\n\t" \
00796     "out %i2,%r0+7 $ elpm $ mov %r0+5,r0 $ adiw r30,1 $ adc %r0+7,r1" "\n\t" \
00797     "out %i2,%r0+7 $ elpm $ mov %r0+6,r0 $ adiw r30,1 $ adc %r0+7,r1" "\n\t" \
00798     "out %i2,%r0+7 $ elpm $ mov %r0+7,r0"                             \
00799     : "r" (res)                                                       \
00800     : "r" (addr), "n" (& RAMPZ)                                       \
00801     : "r30", "r31", "r0"                                             \
00802 #else
00803 /* No ELPM: Fall back to __LPM__<N>. */
00804 #define __ELPM__1(r,a,T) const T *__a = (const T*)(uint16_t) a; __LPM__1(r,__a)
00805 #define __ELPM__2(r,a,T) const T *__a = (const T*)(uint16_t) a; __LPM__2(r,__a)
00806 #define __ELPM__3(r,a,T) const T *__a = (const T*)(uint16_t) a; __LPM__3(r,__a)
00807 #define __ELPM__4(r,a,T) const T *__a = (const T*)(uint16_t) a; __LPM__4(r,__a)
00808 #define __ELPM__8(r,a,T) const T *__a = (const T*)(uint16_t) a; __LPM__8(r,__a)
00809 #endif /* ELPM cases */
00810
00811 #define _Avrlibc_Def_Pgm_Far_1(Name, Typ)                               \
00812     static __ATTR_ALWAYS_INLINE__                                     \
00813     Typ pgm_read_##Name##_far (uint_farptr_t __addr)               \
00814     {                                                                 \
00815         Typ __res;                                                  \
00816         __ELPM__1 (__res, __addr, Typ);                             \
00817         return __res;                                               \
00818     }
00819
00820 #define _Avrlibc_Def_Pgm_Far_2(Name, Typ)                               \
00821     static __ATTR_ALWAYS_INLINE__                                     \
00822     Typ pgm_read_##Name##_far (uint_farptr_t __addr)               \
00823     {                                                                 \
00824         Typ __res;                                                  \
00825         __ELPM__2 (__res, __addr, Typ);                             \
00826         return __res;                                               \
00827     }
00828
00829 #define _Avrlibc_Def_Pgm_Far_3(Name, Typ)                               \
00830     static __ATTR_ALWAYS_INLINE__                                     \
00831     Typ pgm_read_##Name##_far (uint_farptr_t __addr)               \
00832     {                                                                 \
00833         Typ __res;                                                  \
00834         __ELPM__3 (__res, __addr, Typ);                             \
00835         return __res;                                               \
00836     }
00837
00838 #define _Avrlibc_Def_Pgm_Far_4(Name, Typ)                               \
00839     static __ATTR_ALWAYS_INLINE__                                     \
00840     Typ pgm_read_##Name##_far (uint_farptr_t __addr)               \
00841     {                                                                 \
00842         Typ __res;                                                  \
00843         __ELPM__4 (__res, __addr, Typ);                             \
00844         return __res;                                               \
00845     }
00846
00847 #define _Avrlibc_Def_Pgm_Far_8(Name, Typ)                               \
00848     static __ATTR_ALWAYS_INLINE__                                     \
00849     Typ pgm_read_##Name##_far (uint_farptr_t __addr)               \
00850     {                                                                 \
00851         Typ __res;                                                  \
00852         __ELPM__8 (__res, __addr, Typ);                             \
00853         return __res;                                               \
00854     }
00855
00856 _Avrlibc_Def_Pgm_Far_1 (char, char)
00857 _Avrlibc_Def_Pgm_Far_1 (unsigned_char, unsigned char)
00858 _Avrlibc_Def_Pgm_Far_1 (signed_char, signed char)
00859 _Avrlibc_Def_Pgm_Far_1 (u8, uint8_t)
00860 _Avrlibc_Def_Pgm_Far_1 (i8, int8_t)
00861 #if __SIZEOF_INT__ == 1
00862 _Avrlibc_Def_Pgm_Far_1 (int, int)
00863 _Avrlibc_Def_Pgm_Far_1 (unsigned, unsigned)
00864 _Avrlibc_Def_Pgm_Far_1 (unsigned_int, unsigned int)
00865 _Avrlibc_Def_Pgm_Far_1 (signed, signed)
00866 _Avrlibc_Def_Pgm_Far_1 (signed_int, signed int)

```



```

00867 #endif
00868 #if __SIZEOF_SHORT__ == 1
00869 _Avrlibc_Def_Pgm_Far_1 (short, short)
00870 _Avrlibc_Def_Pgm_Far_1 (unsigned_short, unsigned short)
00871 #endif
00872
00873 _Avrlibc_Def_Pgm_Far_2 (u16, uint16_t)
00874 _Avrlibc_Def_Pgm_Far_2 (i16, int16_t)
00875 #if __SIZEOF_INT__ == 2
00876 _Avrlibc_Def_Pgm_Far_2 (int, int)
00877 _Avrlibc_Def_Pgm_Far_2 (unsigned, unsigned)
00878 _Avrlibc_Def_Pgm_Far_2 (unsigned_int, unsigned int)
00879 _Avrlibc_Def_Pgm_Far_2 (signed, signed)
00880 _Avrlibc_Def_Pgm_Far_2 (signed_int, signed int)
00881 #endif
00882 #if __SIZEOF_SHORT__ == 2
00883 _Avrlibc_Def_Pgm_Far_2 (short, short)
00884 _Avrlibc_Def_Pgm_Far_2 (unsigned_short, unsigned short)
00885 #endif
00886 #if __SIZEOF_LONG__ == 2
00887 _Avrlibc_Def_Pgm_Far_2 (long, long)
00888 _Avrlibc_Def_Pgm_Far_2 (unsigned_long, unsigned long)
00889 #endif
00890
00891 #if defined(__INT24_MAX__)
00892 _Avrlibc_Def_Pgm_Far_3 (i24, __int24)
00893 _Avrlibc_Def_Pgm_Far_3 (u24, __uint24)
00894 #endif /* Have __int24 */
00895
00896 _Avrlibc_Def_Pgm_Far_4 (u32, uint32_t)
00897 _Avrlibc_Def_Pgm_Far_4 (i32, int32_t)
00898 _Avrlibc_Def_Pgm_Far_4 (float, float)
00899 #if __SIZEOF_LONG__ == 4
00900 _Avrlibc_Def_Pgm_Far_4 (long, long)
00901 _Avrlibc_Def_Pgm_Far_4 (unsigned_long, unsigned long)
00902 #endif
00903 #if __SIZEOF_LONG_LONG__ == 4
00904 _Avrlibc_Def_Pgm_Far_4 (long_long, long long)
00905 _Avrlibc_Def_Pgm_Far_4 (unsigned_long_long, unsigned long long)
00906 #endif
00907 #if __SIZEOF_DOUBLE__ == 4
00908 _Avrlibc_Def_Pgm_Far_4 (double, double)
00909 #endif
00910 #if __SIZEOF_LONG_DOUBLE__ == 4
00911 _Avrlibc_Def_Pgm_Far_4 (long_double, long double)
00912 #endif
00913
00914 #if __SIZEOF_LONG_LONG__ == 8
00915 _Avrlibc_Def_Pgm_Far_8 (u64, uint64_t)
00916 _Avrlibc_Def_Pgm_Far_8 (i64, int64_t)
00917 _Avrlibc_Def_Pgm_Far_8 (long_long, long long)
00918 _Avrlibc_Def_Pgm_Far_8 (unsigned_long_long, unsigned long long)
00919 #endif
00920 #if __SIZEOF_DOUBLE__ == 8
00921 _Avrlibc_Def_Pgm_Far_8 (double, double)
00922 #endif
00923 #if __SIZEOF_LONG_DOUBLE__ == 8
00924 _Avrlibc_Def_Pgm_Far_8 (long_double, long double)
00925 #endif
00926
00927 #endif /* DOXYGEN */
00928
00929 #ifdef __cplusplus
00930 extern "C" {
00931 #endif
00932
00933 #if defined(__DOXYGEN__)
00934 /* No documentation for the deprecated stuff. */
00935 #elif defined(__PROG_TYPES_COMPAT__) /* !DOXYGEN */
00936
00937 typedef void prog_void __attribute__((__progmem__, __deprecated__("prog_void type is deprecated.")));
00938 typedef char prog_char __attribute__((__progmem__, __deprecated__("prog_char type is deprecated.")));
00939 typedef unsigned char prog_uchar __attribute__((__progmem__, __deprecated__("prog_uchar type is
deprecated.")));
00940 typedef int8_t prog_int8_t __attribute__((__progmem__, __deprecated__("prog_int8_t type is
deprecated.")));
00941 typedef uint8_t prog_uint8_t __attribute__((__progmem__, __deprecated__("prog_uint8_t type is
deprecated.")));
00942 typedef int16_t prog_int16_t __attribute__((__progmem__, __deprecated__("prog_int16_t type is
deprecated.")));
00943 typedef uint16_t prog_uint16_t __attribute__((__progmem__, __deprecated__("prog_uint16_t type is
deprecated.")));
00944 typedef int32_t prog_int32_t __attribute__((__progmem__, __deprecated__("prog_int32_t type is
deprecated.")));
00945 typedef uint32_t prog_uint32_t __attribute__((__progmem__, __deprecated__("prog_uint32_t type is
deprecated.")));
00946 #if !__USING_MINT8

```

```

00947 typedef int64_t    prog_int64_t  __attribute__((__progmem__, __deprecated__("prog_int64_t type is
depreciated.")));
00948 typedef uint64_t   prog_uint64_t  __attribute__((__progmem__, __deprecated__("prog_uint64_t type is
depreciated.")));
00949 #endif
00950
00951 #ifndef PGM_P
00952 #define PGM_P const prog_char *
00953 #endif
00954
00955 #ifndef PGM_VOID_P
00956 #define PGM_VOID_P const prog_void *
00957 #endif
00958
00959 #else /* !defined(__DOXYGEN__), !defined(__PROG_TYPES_COMPAT__) */
00960
00961 #ifndef PGM_P
00962 #define PGM_P const char *
00963 #endif
00964
00965 #ifndef PGM_VOID_P
00966 #define PGM_VOID_P const void *
00967 #endif
00968 #endif /* defined(__DOXYGEN__), defined(__PROG_TYPES_COMPAT__) */
00969
00970 /* Although in C, we can get away with just using __c, it does not work in
00971 C++. We need to use &__c[0] to avoid the compiler puking. Dave Hylands
00972 explained it thusly,
00973
00974     Let's suppose that we use PSTR("Test"). In this case, the type returned
00975     by __c is a prog_char[5] and not a prog_char *. While these are
00976     compatible, they aren't the same thing (especially in C++). The type
00977     returned by &__c[0] is a prog_char *, which explains why it works
00978     fine. */
00979
00980 #if defined(__DOXYGEN__)
00981 /*
00982  * The #define below is just a dummy that serves documentation
00983  * purposes only.
00984  */
00985 /** \ingroup avr_pgmspace
00986     \def PSTR(str)
00987
00988     Used to declare a static pointer to a string in program space. */
00989 # define PSTR(str) ({ static const PROGMEM char c[] = (str); &c[0]; })
00990 #else /* !DOXYGEN */
00991 /* The real thing. */
00992 # define PSTR(s) (__extension__({static const char __c[] PROGMEM = (s); &__c[0];}))
00993 #endif /* DOXYGEN */
00994
00995 #if defined(__DOXYGEN__)
00996 /** \ingroup avr_pgmspace
00997     \def PSTR_FAR(str)
00998
00999     Used to define a string literal in far program space, and to return its
01000     address of type #uint_farptr_t. */
01001 # define PSTR_FAR(str) ({ static const PROGMEM_FAR char c[] = (str); pgm_get_far_address(c[0]); })
01002 #else /* !DOXYGEN */
01003 /* The real thing. */
01004 # define PSTR_FAR(s) (__extension__({static const char __c[] PROGMEM_FAR = (s);
pgm_get_far_address(__c[0]);}))
01005 #endif /* DOXYGEN */
01006
01007 #ifndef __DOXYGEN__
01008
01009 /* These are used down the line for pgm_read_byte[_near] etc. */
01010
01011 #if defined (__AVR_TINY__)
01012 /* Attribute __progmem__ on Reduced Tiny works different than for
01013 all the other devices: When taking the address of a symbol that's
01014 attributed as progmem, then the compiler adds an offset of 0x4000
01015 to the value of the symbol. This means that accessing data in
01016 progmem can be performed by vanilla C/C++ code. This requires
01017 - GCC PR71948 - Make progmem work on Reduced Tiny (GCC v7 / 2016-08) */
01018 #define __LPM(addr)      (* (const uint8_t*) (addr))
01019 #define __LPM_word(addr) (* (const uint16_t*) (addr))
01020 #define __LPM_dword(addr) (* (const uint32_t*) (addr))
01021 # if __SIZEOF_LONG_LONG__ == 8
01022 # define __LPM_qword(addr) (* (const uint64_t*) (addr))
01023 # endif
01024 #else
01025 #define __LPM(addr)      \
01026     (__extension__({    \
01027     uint16_t __addr16 = (uint16_t) (addr); \
01028     uint8_t __result; \
01029     __LPM_1 (__result, __addr16); \
01030     __result; \

```

```

01031 )))
01032
01033 #define __LPM_word(addr)           \
01034     (__extension__({               \
01035         uint16_t __addr16 = (uint16_t) (addr); \
01036         uint16_t __result;          \
01037         __LPM__2 (__result, __addr16); \
01038         __result;                   \
01039     }))
01040
01041 #define __LPM_dword(addr)           \
01042     (__extension__({               \
01043         uint16_t __addr16 = (uint16_t) (addr); \
01044         uint32_t __result;          \
01045         __LPM__4 (__result, __addr16); \
01046         __result;                   \
01047     }))
01048
01049 #if __SIZEOF_LONG_LONG__ == 8
01050 #define __LPM_qword(addr)           \
01051     (__extension__({               \
01052         uint16_t __addr16 = (uint16_t) (addr); \
01053         uint64_t __result;          \
01054         __LPM__8 (__result, __addr16); \
01055         __result;                   \
01056     }))
01057 #endif
01058 #endif /* AVR_TINY */
01059
01060
01061 #define __ELPM(addr)                 \
01062     (__extension__({               \
01063         uint_farptr_t __addr32 = (addr); \
01064         uint8_t __result;            \
01065         __ELPM__1 (__result, __addr32, uint8_t); \
01066         __result;                   \
01067     }))
01068
01069 #define __ELPM_word(addr)            \
01070     (__extension__({               \
01071         uint_farptr_t __addr32 = (addr); \
01072         uint16_t __result;           \
01073         __ELPM__2 (__result, __addr32, uint16_t); \
01074         __result;                   \
01075     }))
01076
01077 #define __ELPM_dword(addr)           \
01078     (__extension__({               \
01079         uint_farptr_t __addr32 = (addr); \
01080         uint32_t __result;           \
01081         __ELPM__4 (__result, __addr32, uint32_t); \
01082         __result;                   \
01083     }))
01084
01085 #if __SIZEOF_LONG_LONG__ == 8
01086 #define __ELPM_qword(addr)          \
01087     (__extension__({               \
01088         uint_farptr_t __addr32 = (addr); \
01089         uint64_t __result;           \
01090         __ELPM__8 (__result, __addr32, uint64_t); \
01091         __result;                   \
01092     }))
01093 #endif
01094
01095 #endif /* !__DOXYGEN__ */
01096
01097 /** \ingroup avr_pgmspace
01098     \def pgm_read_byte_near(__addr)
01099     Read a byte from the program space with a 16-bit (near) byte-address. */
01100
01101 #define pgm_read_byte_near(__addr) __LPM ((uint16_t)(__addr))
01102
01103 /** \ingroup avr_pgmspace
01104     \def pgm_read_word_near(__addr)
01105     Read a word from the program space with a 16-bit (near) byte-address. */
01106
01107 #define pgm_read_word_near(__addr) __LPM_word ((uint16_t)(__addr))
01108
01109 /** \ingroup avr_pgmspace
01110     \def pgm_read_dword_near(__addr)
01111     Read a double word from the program space with a 16-bit (near)
01112     byte-address. */
01113
01114 #define pgm_read_dword_near(__addr) \
01115     __LPM_dword ((uint16_t)(__addr))
01116
01117 /** \ingroup avr_pgmspace

```

```

01118     \def pgm_read_qword_near(__addr)
01119     Read a quad-word from the program space with a 16-bit (near)
01120     byte-address. */
01121
01122 #define pgm_read_qword_near(__addr) __LPM_qword ((uint16_t) (__addr))
01123
01124 /** \ingroup avr_pgmspace
01125     \def pgm_read_float_near (const float *address)
01126     Read a \c float from the program space with a 16-bit (near) byte-address.*/
01127
01128 #define pgm_read_float_near(addr) pgm_read_float (addr)
01129
01130 /** \ingroup avr_pgmspace
01131     \def pgm_read_ptr_near(__addr)
01132     Read a pointer from the program space with a 16-bit (near) byte-address. */
01133
01134 #define pgm_read_ptr_near(__addr) \
01135     ((void*) __LPM_word ((uint16_t) (__addr)))
01136
01137 /** \ingroup avr_pgmspace
01138     \def pgm_read_byte_far(__addr)
01139     Read a byte from the program space with a 32-bit (far) byte-address. */
01140
01141 #define pgm_read_byte_far(__addr) __ELPM (__addr)
01142
01143 /** \ingroup avr_pgmspace
01144     \def pgm_read_word_far(__addr)
01145     Read a word from the program space with a 32-bit (far) byte-address. */
01146
01147 #define pgm_read_word_far(__addr) __ELPM_word (__addr)
01148
01149 /** \ingroup avr_pgmspace
01150     \def pgm_read_dword_far(__addr)
01151     Read a double word from the program space with a 32-bit (far)
01152     byte-address. */
01153
01154 #define pgm_read_dword_far(__addr) __ELPM_dword (__addr)
01155
01156 /** \ingroup avr_pgmspace
01157     \def pgm_read_qword_far(__addr)
01158     Read a quad-word from the program space with a 32-bit (far)
01159     byte-address. */
01160
01161 #define pgm_read_qword_far(__addr) __ELPM_qword (__addr)
01162
01163 /** \ingroup avr_pgmspace
01164     \def pgm_read_ptr_far(__addr)
01165     Read a pointer from the program space with a 32-bit (far) byte-address. */
01166
01167 #define pgm_read_ptr_far(__addr) ((void*) __ELPM_word (__addr))
01168
01169 /** \ingroup avr_pgmspace
01170     \def pgm_read_byte(__addr)
01171     Read a byte from the program space with a 16-bit (near) nyte-address. */
01172
01173 #define pgm_read_byte(__addr) pgm_read_byte_near(__addr)
01174
01175 /** \ingroup avr_pgmspace
01176     \def pgm_read_word(__addr)
01177     Read a word from the program space with a 16-bit (near) byte-address. */
01178
01179 #define pgm_read_word(__addr) pgm_read_word_near(__addr)
01180
01181 /** \ingroup avr_pgmspace
01182     \def pgm_read_dword(__addr)
01183     Read a double word from the program space with a 16-bit (near)
01184     byte-address. */
01185
01186 #define pgm_read_dword(__addr) pgm_read_dword_near(__addr)
01187
01188 /** \ingroup avr_pgmspace
01189     \def pgm_read_qword(__addr)
01190     Read a quad-word from the program space with a 16-bit (near)
01191     byte-address. */
01192
01193 #define pgm_read_qword(__addr) pgm_read_qword_near(__addr)
01194
01195 /** \ingroup avr_pgmspace
01196     \def pgm_read_ptr(__addr)
01197     Read a pointer from the program space with a 16-bit (near) byte-address. */
01198
01199 #define pgm_read_ptr(__addr) pgm_read_ptr_near(__addr)
01200
01201 /** \ingroup avr_pgmspace
01202     \def pgm_get_far_address(var)
01203
01204     This macro evaluates to a ::uint_farptr_t 32-bit "far" pointer (only

```

```

01205 24 bits used) to data even beyond the 64 KiB limit for the 16-bit ordinary
01206 pointer. It is similar to the '&' operator, with some limitations.
01207 Example:
01208 \code
01209 #include <avr/pgmspace.h>
01210
01211 // Section .progmemx.data is located after all the code sections.
01212 __attribute__((section(".progmemx.data")))
01213 const int data[] = { 2, 3, 5, 7, 9, 11 };
01214
01215 int get_data (uint8_t idx)
01216 {
01217     uint_farptr_t pdata = pgm_get_far_address (data[0]);
01218     return pgm_read_int_far (pdata + idx * sizeof(int));
01219 }
01220 \endcode
01221
01222 Comments:
01223
01224 - The overhead is minimal and it's mainly due to the 32-bit size operation.
01225
01226 - 24 bit sizes guarantees the code compatibility for use in future devices.
01227
01228 - \p var has to be resolved at link-time as an existing symbol,
01229   i.e. a simple variable name, an array name, or an array or structure
01230   element provided the offset is known at compile-time, and \p var is
01231   located in static storage, etc.
01232
01233 - The returned value is the symbol's \ref sec_vma "VMA"
01234   (virtual memory address)
01235   determined by the linker and falls in the corresponding memory region.
01236   The AVR Harvard architecture requires non-overlapping VMA areas for
01237   the multiple \ref sec_memory_regions "memory regions" in the processor:
01238   Flash ROM, RAM, and EEPROM. Typical offset for these are
01239   \c 0x0, \c 0x800xx0, and \c 0x810000 respectively, derived from the
01240   linker script used and linker options.
01241 */
01242
01243 #define pgm_get_far_address(var)          \|
01244 (__extension__({                          \|
01245     uint_farptr_t __tmp;                  \|
01246                                           \|
01247     __asm__ __volatile__ (                \|
01248         "ldi    %A0, lo8(%1)"             "\n\t" \|
01249         "ldi    %B0, hi8(%1)"             "\n\t" \|
01250         "ldi    %C0, hh8(%1)"             "\n\t" \|
01251         "clr    %D0"                       \|
01252         : "=d" (__tmp)                    \|
01253         : "i" (&(var))                   \|
01254     );                                     \|
01255     __tmp;                                 \|
01256 ))) \|
01257
01258
01259
01260 /** \ingroup avr_pgmspace
01261     \fn const void * memchr_P(const void *s, int val, size_t len)
01262     \brief Scan flash memory for a character.
01263
01264     The memchr_P() function scans the first \p len bytes of the flash
01265     memory area pointed to by \p s for the character \p val. The first
01266     byte to match \p val (interpreted as an unsigned character) stops
01267     the operation.
01268
01269     \return The memchr_P() function returns a pointer to the matching
01270     byte or \c NULL if the character does not occur in the given memory
01271     area. */
01272 extern const void * memchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;
01273
01274 /** \ingroup avr_pgmspace
01275     \fn int memcmp_P(const void *s1, const void *s2, size_t len)
01276     \brief Compare memory areas
01277
01278     The memcmp_P() function compares the first \p len bytes of the memory
01279     areas \p s1 and flash \p s2. The comparison is performed using unsigned
01280     char operations.
01281
01282     \returns The memcmp_P() function returns an integer less than, equal
01283     to, or greater than zero if the first \p len bytes of \p s1 is found,
01284     respectively, to be less than, to match, or be greater than the first
01285     \p len bytes of \p s2. */
01286 extern int memcmp_P(const void *, const void *, size_t) __ATTR_PURE__;
01287
01288 /** \ingroup avr_pgmspace
01289     \fn void *memccpy_P (void *dest, const void *src, int val, size_t len)
01290
01291     This function is similar to memccpy() except that \p src is pointer

```

```

01292     to a string in program space.  */
01293 extern void *memccpy_P(void *, const void *, int __val, size_t);
01294
01295 /** \ingroup avr_pgmspace
01296     \fn void *memcpy_P(void *dest, const void *src, size_t n)
01297
01298     The memcpy_P() function is similar to memcpy(), except the src string
01299     resides in program space.
01300
01301     \returns The memcpy_P() function returns a pointer to dest.  */
01302 extern void *memcpy_P(void *, const void *, size_t);
01303
01304 /** \ingroup avr_pgmspace
01305     \fn void *memmem_P(const void *s1, size_t len1, const void *s2, size_t len2)
01306
01307     The memmem_P() function is similar to memmem() except that \p s2 is
01308     pointer to a string in program space.  */
01309 extern void *memmem_P(const void *, size_t, const void *, size_t) __ATTR_PURE__;
01310
01311 /** \ingroup avr_pgmspace
01312     \fn const void *memrchr_P(const void *src, int val, size_t len)
01313
01314     The memrchr_P() function is like the memchr_P() function, except
01315     that it searches backwards from the end of the \p len bytes pointed
01316     to by \p src instead of forwards from the front. (Glibc, GNU extension.)
01317
01318     \return The memrchr_P() function returns a pointer to the matching
01319     byte or \c NULL if the character does not occur in the given memory
01320     area.  */
01321 extern const void * memrchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;
01322
01323 /** \ingroup avr_pgmspace
01324     \fn char *strcat_P(char *dest, const char *src)
01325
01326     The strcat_P() function is similar to strcat() except that the \e src
01327     string must be located in program space (flash).
01328
01329     \returns The strcat() function returns a pointer to the resulting string
01330     \e dest.  */
01331 extern char *strcat_P(char *, const char *);
01332
01333 /** \ingroup avr_pgmspace
01334     \fn const char *strchr_P(const char *s, int val)
01335     \brief Locate character in program space string.
01336
01337     The strchr_P() function locates the first occurrence of \p val
01338     (converted to a char) in the string pointed to by \p s in program
01339     space. The terminating null character is considered to be part of
01340     the string.
01341
01342     The strchr_P() function is similar to strchr() except that \p s is
01343     pointer to a string in program space.
01344
01345     \returns The strchr_P() function returns a pointer to the matched
01346     character or \c NULL if the character is not found.  */
01347 extern const char * strchr_P(const char *, int __val) __ATTR_CONST__;
01348
01349 /** \ingroup avr_pgmspace
01350     \fn const char *strchrnul_P(const char *s, int c)
01351
01352     The strchrnul_P() function is like strchr_P() except that if \p c is
01353     not found in \p s, then it returns a pointer to the null byte at the
01354     end of \p s, rather than \c NULL. (Glibc, GNU extension.)
01355
01356     \return The strchrnul_P() function returns a pointer to the matched
01357     character, or a pointer to the null byte at the end of \p s (i.e.,
01358     \c s+strlen(s)) if the character is not found.  */
01359 extern const char * strchrnul_P(const char *, int __val) __ATTR_CONST__;
01360
01361 /** \ingroup avr_pgmspace
01362     \fn int strcmp_P(const char *s1, const char *s2)
01363
01364     The strcmp_P() function is similar to strcmp() except that \p s2 is
01365     pointer to a string in program space.
01366
01367     \returns The strcmp_P() function returns an integer less than, equal
01368     to, or greater than zero if \p s1 is found, respectively, to be less
01369     than, to match, or be greater than \p s2. A consequence of the
01370     ordering used by strcmp_P() is that if \p s1 is an initial substring
01371     of \p s2, then \p s1 is considered to be "less than" \p s2.  */
01372 extern int strcmp_P(const char *, const char *) __ATTR_PURE__;
01373
01374 /** \ingroup avr_pgmspace
01375     \fn char *strcpy_P(char *dest, const char *src)
01376
01377     The strcpy_P() function is similar to strcpy() except that src is a
01378     pointer to a string in program space.

```

```

01379
01380     \returns The strcpy_P() function returns a pointer to the destination
01381     string dest. */
01382 extern char *strcpy_P(char *, const char *);
01383
01384 /** \ingroup avr_pgmspace
01385     \fn int strcmp_P(const char *s1, const char *s2)
01386     \brief Compare two strings ignoring case.
01387
01388     The strcmp_P() function compares the two strings \p s1 and \p s2,
01389     ignoring the case of the characters.
01390
01391     \param s1 A pointer to a string in the devices SRAM.
01392     \param s2 A pointer to a string in the devices Flash.
01393
01394     \returns The strcmp_P() function returns an integer less than,
01395     equal to, or greater than zero if \p s1 is found, respectively, to
01396     be less than, to match, or be greater than \p s2. A consequence of
01397     the ordering used by strcmp_P() is that if \p s1 is an initial
01398     substring of \p s2, then \p s1 is considered to be "less than" \p s2. */
01399 extern int strcmp_P(const char *, const char *) __ATTR_PURE__;
01400
01401 /** \ingroup avr_pgmspace
01402     \fn char *strcasestr_P(const char *s1, const char *s2)
01403
01404     This function is similar to strcasestr() except that \p s2 is pointer
01405     to a string in program space. */
01406 extern char *strcasestr_P(const char *, const char *) __ATTR_PURE__;
01407
01408 /** \ingroup avr_pgmspace
01409     \fn size_t strcspn_P(const char *s, const char *reject)
01410
01411     The strcspn_P() function calculates the length of the initial segment
01412     of \p s which consists entirely of characters not in \p reject. This
01413     function is similar to strcspn() except that \p reject is a pointer
01414     to a string in program space.
01415
01416     \return The strcspn_P() function returns the number of characters in
01417     the initial segment of \p s which are not in the string \p reject.
01418     The terminating zero is not considered as a part of string. */
01419 extern size_t strcspn_P(const char *__s, const char * __reject) __ATTR_PURE__;
01420
01421 /** \ingroup avr_pgmspace
01422     \fn size_t strlcat_P(char *dst, const char *src, size_t siz)
01423     \brief Concatenate two strings.
01424
01425     The strlcat_P() function is similar to strlcat(), except that the \p src
01426     string must be located in program space (flash).
01427
01428     Appends \p src to string \p dst of size \p siz (unlike strncat(),
01429     \p siz is the full size of \p dst, not space left). At most \p siz-1
01430     characters will be copied. Always NULL terminates (unless \p siz <=
01431     \p strlen(dst)).
01432
01433     \returns The strlcat_P() function returns strlen(src) + MIN(siz,
01434     strlen(initial dst)). If retval >= siz, truncation occurred. */
01435 extern size_t strlcat_P(char *, const char *, size_t );
01436
01437 /** \ingroup avr_pgmspace
01438     \fn size_t strlcpy_P(char *dst, const char *src, size_t siz)
01439     \brief Copy a string from progmem to RAM.
01440
01441     Copy \p src to string \p dst of size \p siz. At most \p siz-1
01442     characters will be copied. Always NULL terminates (unless \p siz == 0).
01443     The strlcpy_P() function is similar to strlcpy() except that the
01444     \p src is pointer to a string in memory space.
01445
01446     \returns The strlcpy_P() function returns strlen(src). If
01447     retval >= siz, truncation occurred. */
01448 extern size_t strlcpy_P(char *, const char *, size_t );
01449
01450 /** \ingroup avr_pgmspace
01451     \fn size_t strnlen_P(const char *src, size_t len)
01452     \brief Determine the length of a fixed-size string.
01453
01454     The strnlen_P() function is similar to strnlen(), except that \c src is a
01455     pointer to a string in program space.
01456
01457     \returns The strnlen_P function returns strnlen_P(src), if that is less than
01458     \c len, or \c len if there is no '\\0' character among the first \c len
01459     characters pointed to by \c src. */
01460 extern size_t strnlen_P(const char *, size_t) __ATTR_CONST__; /* program memory can't change */
01461
01462 /** \ingroup avr_pgmspace
01463     \fn int strncmp_P(const char *s1, const char *s2, size_t n)
01464
01465     The strncmp_P() function is similar to strcmp_P() except it only compares

```

```

01466     the first (at most) n characters of s1 and s2.
01467
01468     \returns The strcmp_P() function returns an integer less than, equal to,
01469     or greater than zero if s1 (or the first n bytes thereof) is found,
01470     respectively, to be less than, to match, or be greater than s2. */
01471 extern int strcmp_P(const char *, const char *, size_t) __ATTR_PURE__;
01472
01473 /** \ingroup avr_pgmspace
01474     \fn int strcasecmp_P(const char *s1, const char *s2, size_t n)
01475     \brief Compare two strings ignoring case.
01476
01477     The strcasecmp_P() function is similar to strcmp_P(), except it
01478     only compares the first \p n characters of \p s1.
01479
01480     \param s1 A pointer to a string in the devices SRAM.
01481     \param s2 A pointer to a string in the devices Flash.
01482     \param n The maximum number of bytes to compare.
01483
01484     \returns The strcasecmp_P() function returns an integer less than,
01485     equal to, or greater than zero if \p s1 (or the first \p n bytes
01486     thereof) is found, respectively, to be less than, to match, or be
01487     greater than \p s2. A consequence of the ordering used by
01488     strcmp_P() is that if \p s1 is an initial substring of \p s2,
01489     then \p s1 is considered to be "less than" \p s2. */
01490 extern int strcasecmp_P(const char *, const char *, size_t) __ATTR_PURE__;
01491
01492 /** \ingroup avr_pgmspace
01493     \fn char *strncat_P(char *dest, const char *src, size_t len)
01494     \brief Concatenate two strings.
01495
01496     The strncat_P() function is similar to strncat(), except that the \e src
01497     string must be located in program space (flash).
01498
01499     \returns The strncat_P() function returns a pointer to the resulting string
01500     dest. */
01501 extern char *strncat_P(char *, const char *, size_t);
01502
01503 /** \ingroup avr_pgmspace
01504     \fn char *strncpy_P(char *dest, const char *src, size_t n)
01505
01506     The strncpy_P() function is similar to strncpy_P() except that not more
01507     than n bytes of src are copied. Thus, if there is no null byte among the
01508     first n bytes of src, the result will not be null-terminated.
01509
01510     In the case where the length of src is less than that of n, the remainder
01511     of dest will be padded with nulls.
01512
01513     \returns The strncpy_P() function returns a pointer to the destination
01514     string dest. */
01515 extern char *strncpy_P(char *, const char *, size_t);
01516
01517 /** \ingroup avr_pgmspace
01518     \fn char *strpbrk_P(const char *s, const char *accept)
01519
01520     The strpbrk_P() function locates the first occurrence in the string
01521     \p s of any of the characters in the flash string \p accept. This
01522     function is similar to strpbrk() except that \p accept is a pointer
01523     to a string in program space.
01524
01525     \return The strpbrk_P() function returns a pointer to the character
01526     in \p s that matches one of the characters in \p accept, or \c NULL
01527     if no such character is found. The terminating zero is not considered
01528     as a part of string: if one or both args are empty, the result will
01529     \c NULL. */
01530 extern char *strpbrk_P(const char *s, const char *accept) __ATTR_PURE__;
01531
01532 /** \ingroup avr_pgmspace
01533     \fn const char *strrchr_P(const char *s, int val)
01534     \brief Locate character in string.
01535
01536     The strrchr_P() function returns a pointer to the last occurrence of
01537     the character \p val in the flash string \p s.
01538
01539     \return The strrchr_P() function returns a pointer to the matched
01540     character or \c NULL if the character is not found. */
01541 extern const char *strrchr_P(const char *, int val) __ATTR_CONST__;
01542
01543 /** \ingroup avr_pgmspace
01544     \fn char *strsep_P(char **sp, const char *delim)
01545     \brief Parse a string into tokens.
01546
01547     The strsep_P() function locates, in the string referenced by \p *sp,
01548     the first occurrence of any character in the string \p delim (or the
01549     terminating '\\0' character) and replaces it with a '\\0'. The
01550     location of the next character after the delimiter character (or \c
01551     NULL, if the end of the string was reached) is stored in \p *sp. An
01552     "empty" field, i.e. one caused by two adjacent delimiter

```



```

01553     characters, can be detected by comparing the location referenced by
01554     the pointer returned in \p *sp to '\\0'. This function is similar to
01555     strsep() except that \p delim is a pointer to a string in program
01556     space.
01557
01558     \return The strsep_P() function returns a pointer to the original
01559     value of \p *sp. If \p *sp is initially \c NULL, strsep_P() returns
01560     \c NULL. */
01561 extern char *strsep_P(char **__sp, const char * __delim);
01562
01563 /** \ingroup avr_pgmspace
01564     \fn size_t strspn_P(const char *s, const char *accept)
01565
01566     The strspn_P() function calculates the length of the initial segment
01567     of \p s which consists entirely of characters in \p accept. This
01568     function is similar to strspn() except that \p accept is a pointer
01569     to a string in program space.
01570
01571     \return The strspn_P() function returns the number of characters in
01572     the initial segment of \p s which consist only of characters from \p
01573     accept. The terminating zero is not considered as a part of string. */
01574 extern size_t strspn_P(const char *__s, const char * __accept) __ATTR_PURE__;
01575
01576 /** \ingroup avr_pgmspace
01577     \fn char *strstr_P(const char *s1, const char *s2)
01578     \brief Locate a substring.
01579
01580     The strstr_P() function finds the first occurrence of the substring
01581     \p s2 in the string \p s1. The terminating '\\0' characters are not
01582     compared. The strstr_P() function is similar to strstr() except that
01583     \p s2 is pointer to a string in program space.
01584
01585     \returns The strstr_P() function returns a pointer to the beginning
01586     of the substring, or NULL if the substring is not found. If \p s2
01587     points to a string of zero length, the function returns \p s1. */
01588 extern char *strstr_P(const char *, const char *) __ATTR_PURE__;
01589
01590 /** \ingroup avr_pgmspace
01591     \fn char *strtok_P(char *s, const char * delim)
01592     \brief Parses the string into tokens.
01593
01594     strtok_P() parses the string \p s into tokens. The first call to
01595     strtok_P() should have \p s as its first argument. Subsequent calls
01596     should have the first argument set to NULL. If a token ends with a
01597     delimiter, this delimiting character is overwritten with a '\\0' and a
01598     pointer to the next character is saved for the next call to strtok_P().
01599     The delimiter string \p delim may be different for each call.
01600
01601     The strtok_P() function is similar to strtok() except that \p delim
01602     is pointer to a string in program space.
01603
01604     \returns The strtok_P() function returns a pointer to the next token or
01605     NULL when no more tokens are found.
01606
01607     \note strtok_P() is NOT reentrant. For a reentrant version of this
01608     function see strtok_rP().
01609     */
01610 extern char *strtok_P(char *__s, const char * __delim);
01611
01612 /** \ingroup avr_pgmspace
01613     \fn char *strtok_rP(char *string, const char *delim, char **last)
01614     \brief Parses string into tokens.
01615
01616     The strtok_rP() function parses \p string into tokens. The first call to
01617     strtok_rP() should have string as its first argument. Subsequent calls
01618     should have the first argument set to NULL. If a token ends with a
01619     delimiter, this delimiting character is overwritten with a '\\0' and a
01620     pointer to the next character is saved for the next call to strtok_rP().
01621     The delimiter string \p delim may be different for each call. \p last is
01622     a user allocated char* pointer. It must be the same while parsing the
01623     same string. strtok_rP() is a reentrant version of strtok_P().
01624
01625     The strtok_rP() function is similar to strtok_r() except that \p delim
01626     is pointer to a string in program space.
01627
01628     \returns The strtok_rP() function returns a pointer to the next token or
01629     NULL when no more tokens are found. */
01630 extern char *strtok_rP(char *__s, const char * __delim, char **__last);
01631
01632 /** \ingroup avr_pgmspace
01633     \fn size_t strlen_PF(uint_farptr_t s)
01634     \brief Obtain the length of a string
01635
01636     The strlen_PF() function is similar to strlen(), except that \e s is a
01637     far pointer to a string in program space.
01638
01639     \param s A far pointer to the string in flash

```

```

01640
01641 \returns The strlen_PF() function returns the number of characters in
01642 \e s. The contents of RAMPZ SFR are undefined when the function returns. */
01643 extern size_t strlen_PF(uint_farptr_t src) __ATTR_CONST__; /* program memory can't change */
01644
01645 /** \ingroup avr_pgmspace
01646 \fn size_t strlen_PF(uint_farptr_t s, size_t len)
01647 \brief Determine the length of a fixed-size string
01648
01649 The strlen_PF() function is similar to strlen(), except that \e s is a
01650 far pointer to a string in program space.
01651
01652 \param s A far pointer to the string in Flash
01653 \param len The maximum number of length to return
01654
01655 \returns The strlen_PF function returns strlen_P(\e s), if that is less
01656 than \e len, or \e len if there is no '\\0' character among the first \e
01657 len characters pointed to by \e s. The contents of RAMPZ SFR are
01658 undefined when the function returns. */
01659 extern size_t strlen_PF(uint_farptr_t src, size_t len) __ATTR_CONST__; /* program memory can't change
*/
01660
01661 /** \ingroup avr_pgmspace
01662 \fn void *memcpy_PF(void *dest, uint_farptr_t src, size_t n)
01663 \brief Copy a memory block from flash to SRAM
01664
01665 The memcpy_PF() function is similar to memcpy(), except the data
01666 is copied from the program space and is addressed using a far pointer.
01667
01668 \param dest A pointer to the destination buffer
01669 \param src A far pointer to the origin of data in flash memory
01670 \param n The number of bytes to be copied
01671
01672 \returns The memcpy_PF() function returns a pointer to \e dst. The contents
01673 of RAMPZ SFR are undefined when the function returns. */
01674 extern void *memcpy_PF(void *dest, uint_farptr_t src, size_t len);
01675
01676 /** \ingroup avr_pgmspace
01677 \fn char *strcpy_PF(char *dst, uint_farptr_t src)
01678 \brief Duplicate a string
01679
01680 The strcpy_PF() function is similar to strcpy() except that \e src is a far
01681 pointer to a string in program space.
01682
01683 \param dst A pointer to the destination string in SRAM
01684 \param src A far pointer to the source string in Flash
01685
01686 \returns The strcpy_PF() function returns a pointer to the destination
01687 string \e dst. The contents of RAMPZ SFR are undefined when the function
01688 returns. */
01689 extern char *strcpy_PF(char *dst, uint_farptr_t src);
01690
01691 /** \ingroup avr_pgmspace
01692 \fn char *strncpy_PF(char *dst, uint_farptr_t src, size_t n)
01693 \brief Duplicate a string until a limited length
01694
01695 The strncpy_PF() function is similar to strncpy() except that not more
01696 than \e n bytes of \e src are copied. Thus, if there is no null byte among
01697 the first \e n bytes of \e src, the result will not be null-terminated.
01698
01699 In the case where the length of \e src is less than that of \e n, the
01700 remainder of \e dst will be padded with nulls.
01701
01702 \param dst A pointer to the destination string in SRAM
01703 \param src A far pointer to the source string in Flash
01704 \param n The maximum number of bytes to copy
01705
01706 \returns The strncpy_PF() function returns a pointer to the destination
01707 string \e dst. The contents of RAMPZ SFR are undefined when the function
01708 returns. */
01709 extern char *strncpy_PF(char *dst, uint_farptr_t src, size_t len);
01710
01711 /** \ingroup avr_pgmspace
01712 \fn char *strcat_PF(char *dst, uint_farptr_t src)
01713 \brief Concatenates two strings
01714
01715 The strcat_PF() function is similar to strcat() except that the \e src
01716 string must be located in program space (flash) and is addressed using
01717 a far pointer
01718
01719 \param dst A pointer to the destination string in SRAM
01720 \param src A far pointer to the string to be appended in Flash
01721
01722 \returns The strcat_PF() function returns a pointer to the resulting
01723 string \e dst. The contents of RAMPZ SFR are undefined when the function
01724 returns */
01725 extern char *strcat_PF(char *dst, uint_farptr_t src);

```

```

01726
01727 /** \ingroup avr_pgmspace
01728     \fn size_t strlcat_PF(char *dst, uint_farptr_t src, size_t n)
01729     \brief Concatenate two strings
01730
01731     The strlcat_PF() function is similar to strlcat(), except that the \e src
01732     string must be located in program space (flash) and is addressed using
01733     a far pointer.
01734
01735     Appends src to string dst of size \e n (unlike strncat(), \e n is the
01736     full size of \e dst, not space left). At most \e n-1 characters
01737     will be copied. Always NULL terminates (unless \e n <= strlen(\e dst)).
01738
01739     \param dst A pointer to the destination string in SRAM
01740     \param src A far pointer to the source string in Flash
01741     \param n The total number of bytes allocated to the destination string
01742
01743     \returns The strlcat_PF() function returns strlen(\e src) + MIN(\e n,
01744     strlen(initial \e dst)). If retval >= \e n, truncation occurred. The
01745     contents of RAMPZ SFR are undefined when the function returns. */
01746 extern size_t strlcat_PF(char *dst, uint_farptr_t src, size_t siz);
01747
01748 /** \ingroup avr_pgmspace
01749     \fn char *strncat_PF(char *dst, uint_farptr_t src, size_t n)
01750     \brief Concatenate two strings
01751
01752     The strncat_PF() function is similar to strncat(), except that the \e src
01753     string must be located in program space (flash) and is addressed using a
01754     far pointer.
01755
01756     \param dst A pointer to the destination string in SRAM
01757     \param src A far pointer to the source string in Flash
01758     \param n The maximum number of bytes to append
01759
01760     \returns The strncat_PF() function returns a pointer to the resulting
01761     string \e dst. The contents of RAMPZ SFR are undefined when the function
01762     returns. */
01763 extern char *strncat_PF(char *dest, uint_farptr_t src, size_t len);
01764
01765 /** \ingroup avr_pgmspace
01766     \fn int strcmp_PF(const char *s1, uint_farptr_t s2)
01767     \brief Compares two strings
01768
01769     The strcmp_PF() function is similar to strcmp() except that \e s2 is a far
01770     pointer to a string in program space.
01771
01772     \param s1 A pointer to the first string in SRAM
01773     \param s2 A far pointer to the second string in Flash
01774
01775     \returns The strcmp_PF() function returns an integer less than, equal to,
01776     or greater than zero if \e s1 is found, respectively, to be less than, to
01777     match, or be greater than \e s2. The contents of RAMPZ SFR are undefined
01778     when the function returns. */
01779 extern int strcmp_PF(const char *s1, uint_farptr_t s2) __ATTR_PURE__;
01780
01781 /** \ingroup avr_pgmspace
01782     \fn int strncmp_PF(const char *s1, uint_farptr_t s2, size_t n)
01783     \brief Compare two strings with limited length
01784
01785     The strncmp_PF() function is similar to strcmp_PF() except it only
01786     compares the first (at most) \e n characters of \e s1 and \e s2.
01787
01788     \param s1 A pointer to the first string in SRAM
01789     \param s2 A far pointer to the second string in Flash
01790     \param n The maximum number of bytes to compare
01791
01792     \returns The strncmp_PF() function returns an integer less than, equal
01793     to, or greater than zero if \e s1 (or the first \e n bytes thereof) is found,
01794     respectively, to be less than, to match, or be greater than \e s2. The
01795     contents of RAMPZ SFR are undefined when the function returns. */
01796 extern int strncmp_PF(const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;
01797
01798 /** \ingroup avr_pgmspace
01799     \fn int strcasecmp_PF(const char *s1, uint_farptr_t s2)
01800     \brief Compare two strings ignoring case
01801
01802     The strcasecmp_PF() function compares the two strings \e s1 and \e s2, ignoring
01803     the case of the characters.
01804
01805     \param s1 A pointer to the first string in SRAM
01806     \param s2 A far pointer to the second string in Flash
01807
01808     \returns The strcasecmp_PF() function returns an integer less than, equal
01809     to, or greater than zero if \e s1 is found, respectively, to be less than, to
01810     match, or be greater than \e s2. The contents of RAMPZ SFR are undefined
01811     when the function returns. */
01812 extern int strcasecmp_PF(const char *s1, uint_farptr_t s2) __ATTR_PURE__;

```

```

01813
01814 /** \ingroup avr_pgmspace
01815     \fn int strcasecmp_PF(const char *s1, uint_farptr_t s2, size_t n)
01816     \brief Compare two strings ignoring case
01817
01818     The strcasecmp_PF() function is similar to strcmp_PF(), except it
01819     only compares the first \e n characters of \e s1 and the string in flash is
01820     addressed using a far pointer.
01821
01822     \param s1 A pointer to a string in SRAM
01823     \param s2 A far pointer to a string in Flash
01824     \param n The maximum number of bytes to compare
01825
01826     \returns The strcmp_PF() function returns an integer less than, equal
01827     to, or greater than zero if \e s1 (or the first \e n bytes thereof) is found,
01828     respectively, to be less than, to match, or be greater than \e s2. The
01829     contents of RAMPZ SFR are undefined when the function returns. */
01830 extern int strcasecmp_PF(const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;
01831
01832 /** \ingroup avr_pgmspace
01833     \fn uint_farptr_t strchr_PF(uint_farptr_t s, int val)
01834     \brief Locate character in far program space string.
01835
01836     The strchr_PF() function locates the first occurrence of \p val
01837     (converted to a char) in the string pointed to by \p s in far program
01838     space. The terminating null character is considered to be part of
01839     the string.
01840
01841     The strchr_PF() function is similar to strchr() except that \p s is
01842     a far pointer to a string in program space that's \e not \e required to be
01843     located in the lower 64 KiB block like it is the case for strchr_P().
01844
01845     \returns The strchr_PF() function returns a far pointer to the matched
01846     character or \c 0 if the character is not found. */
01847 extern uint_farptr_t strchr_PF(uint_farptr_t, int __val) __ATTR_CONST__;
01848
01849 /** \ingroup avr_pgmspace
01850     \fn char *strstr_PF(const char *s1, uint_farptr_t s2)
01851     \brief Locate a substring.
01852
01853     The strstr_PF() function finds the first occurrence of the substring \c s2
01854     in the string \c s1. The terminating '\\0' characters are not
01855     compared.
01856     The strstr_PF() function is similar to strstr() except that \c s2 is a
01857     far pointer to a string in program space.
01858
01859     \returns The strstr_PF() function returns a pointer to the beginning of the
01860     substring, or NULL if the substring is not found.
01861     If \c s2 points to a string of zero length, the function returns \c s1. The
01862     contents of RAMPZ SFR are undefined when the function returns. */
01863 extern char *strstr_PF(const char *s1, uint_farptr_t s2);
01864
01865 /** \ingroup avr_pgmspace
01866     \fn size_t strlcpy_PF(char *dst, uint_farptr_t src, size_t siz)
01867     \brief Copy a string from progmem to RAM.
01868
01869     Copy src to string dst of size siz. At most siz-1 characters will be
01870     copied. Always NULL terminates (unless siz == 0).
01871
01872     \returns The strlcpy_PF() function returns strlen(src). If retval >= siz,
01873     truncation occurred. The contents of RAMPZ SFR are undefined when the
01874     function returns. */
01875 extern size_t strlcpy_PF(char *dst, uint_farptr_t src, size_t siz);
01876
01877 /** \ingroup avr_pgmspace
01878     \fn int memcmp_PF(const void *s1, uint_farptr_t s2, size_t len)
01879     \brief Compare memory areas
01880
01881     The memcmp_PF() function compares the first \p len bytes of the memory
01882     areas \p s1 and flash \p s2. The comparison is performed using unsigned
01883     char operations. It is an equivalent of memcmp_P() function, except
01884     that it is capable working on all FLASH including the extended area
01885     above 64KB.
01886
01887     \returns The memcmp_PF() function returns an integer less than, equal
01888     to, or greater than zero if the first \p len bytes of \p s1 is found,
01889     respectively, to be less than, to match, or be greater than the first
01890     \p len bytes of \p s2. */
01891 extern int memcmp_PF(const void *, uint_farptr_t, size_t) __ATTR_PURE__;
01892
01893 #ifdef __DOXYGEN__
01894 /** \ingroup avr_pgmspace
01895     \fn size_t strlen_P(const char *src)
01896
01897     The strlen_P() function is similar to strlen(), except that src is a
01898     pointer to a string in program space.
01899
01899

```

```

01900     \returns The strlen_P() function returns the number of characters in src.
01901
01902     \note strlen_P() is implemented as an inline function in the avr/pgmspace.h
01903     header file, which will check if the length of the string is a constant
01904     and known at compile time. If it is not known at compile time, the macro
01905     will issue a call to __strlen_P() which will then calculate the length
01906     of the string as normal.
01907 */
01908 static inline size_t strlen_P(const char * s);
01909 #else /* !DOXYGEN */
01910
01911 #ifdef __AVR_TINY__
01912 #define __strlen_P strlen
01913 extern size_t strlen (const char*);
01914 #else
01915 extern size_t __strlen_P(const char *) __ATTR_CONST__; /* internal helper function */
01916 #endif
01917
01918 static __ATTR_ALWAYS_INLINE__ size_t strlen_P(const char * s);
01919 size_t strlen_P(const char *s)
01920 {
01921     return __builtin_constant_p(__builtin_strlen(s))
01922         ? __builtin_strlen(s) : __strlen_P(s);
01923 }
01924 #endif /* DOXYGEN */
01925
01926 #ifdef __cplusplus
01927 } // extern "C"
01928 #endif
01929
01930 #if defined(__cplusplus) && defined(__pgm_read_template__)
01931
01932 /* Caveat: When this file is found via -isystem <path>, then some older
01933    avr-g++ versions come up with
01934
01935        error: template with C linkage
01936
01937    because the target description did not define NO_IMPLICIT_EXTERN_C. */
01938
01939 template<typename __T, size_t>
01940 struct __pgm_read_impl
01941 {
01942     // A default implementaton for T's with a size not in { 1, 2, 3, 4, 8 }.
01943     // While this works, the performance is absolute scrap because GCC does
01944     // not handle objects well that don't fit in a register (i.e. avr-gcc
01945     // has no respective machine_mode).
01946     __T operator() (const __T *__addr) const
01947     {
01948         __T __res;
01949         memcpy_P (&__res, __addr, sizeof(__T));
01950         return __res;
01951     }
01952 };
01953
01954 template<typename __T>
01955 struct __pgm_read_impl<__T, 1>
01956 {
01957     __T operator() (const __T *__addr) const
01958     {
01959         __T __res; __LPM_1 (__res, __addr); return __res;
01960     }
01961 };
01962
01963 template<typename __T>
01964 struct __pgm_read_impl<__T, 2>
01965 {
01966     __T operator() (const __T *__addr) const
01967     {
01968         __T __res; __LPM_2 (__res, __addr); return __res;
01969     }
01970 };
01971
01972 template<typename __T>
01973 struct __pgm_read_impl<__T, 3>
01974 {
01975     __T operator() (const __T *__addr) const
01976     {
01977         __T __res; __LPM_3 (__res, __addr); return __res;
01978     }
01979 };
01980
01981 template<typename __T>
01982 struct __pgm_read_impl<__T, 4>
01983 {
01984     __T operator() (const __T *__addr) const
01985     {
01986         __T __res; __LPM_4 (__res, __addr); return __res;

```

```

01987 }
01988 };
01989
01990 template<typename __T>
01991 struct __pgm_read_impl<__T, 8>
01992 {
01993     __T operator() (const __T *__addr) const
01994     {
01995         __T __res; __LPM_8 (__res, __addr); return __res;
01996     }
01997 };
01998
01999 template<typename __T>
02000 __T pgm_read (const __T *__addr)
02001 {
02002     return __pgm_read_impl<__T, sizeof(__T)>() (__addr);
02003 }
02004
02005 ///////////////////////////////////////////////////////////////////
02006
02007 template<typename __T, size_t>
02008 struct __pgm_read_far_impl
02009 {
02010     // A default implementaton for T's with a size not in { 1, 2, 3, 4, 8 }.
02011     // While this works, the performance is absolute scrap because GCC does
02012     // not handle objects well that don't fit in a register (i.e. avr-gcc
02013     // has no respective machine_mode).
02014     __T operator() (const __T *__addr) const
02015     {
02016         __T __res;
02017         memcpy_PF (&__res, __addr, sizeof(__T));
02018         return __res;
02019     }
02020 };
02021
02022 template<typename __T>
02023 struct __pgm_read_far_impl<__T, 1>
02024 {
02025     __T operator() (uint_farptr_t __addr) const
02026     {
02027         __T __res; __ELPM_1 (__res, __addr, __T); return __res;
02028     }
02029 };
02030
02031 template<typename __T>
02032 struct __pgm_read_far_impl<__T, 2>
02033 {
02034     __T operator() (uint_farptr_t __addr) const
02035     {
02036         __T __res; __ELPM_2 (__res, __addr, __T); return __res;
02037     }
02038 };
02039
02040 template<typename __T>
02041 struct __pgm_read_far_impl<__T, 3>
02042 {
02043     __T operator() (uint_farptr_t __addr) const
02044     {
02045         __T __res; __ELPM_3 (__res, __addr, __T); return __res;
02046     }
02047 };
02048
02049 template<typename __T>
02050 struct __pgm_read_far_impl<__T, 4>
02051 {
02052     __T operator() (uint_farptr_t __addr) const
02053     {
02054         __T __res; __ELPM_4 (__res, __addr, __T); return __res;
02055     }
02056 };
02057
02058 template<typename __T>
02059 struct __pgm_read_far_impl<__T, 8>
02060 {
02061     __T operator() (uint_farptr_t __addr) const
02062     {
02063         __T __res; __ELPM_8 (__res, __addr, __T); return __res;
02064     }
02065 };
02066
02067 template<typename __T>
02068 __T pgm_read_far (uint_farptr_t __addr)
02069 {
02070     return __pgm_read_far_impl<__T, sizeof(__T)>() (__addr);
02071 }
02072
02073 #endif /* C++ */

```

```

02074
02075 #ifndef __DOXYGEN__
02076 /** \ingroup avr_pgmspace
02077     \fn T pgm_read<T> (const T *addr)
02078
02079     Read an object of type \c T from program memory address \p addr and
02080     return it.
02081     This template is only available when macro \c __pgm_read_template__
02082     is defined. */
02083 template<typename T>
02084 T pgm_read<T> (const T *addr);
02085
02086 /** \ingroup avr_pgmspace
02087     \fn T pgm_read_far<T> (uint_farptr_t addr)
02088
02089     Read an object of type \c T from program memory address \p addr and
02090     return it.
02091     This template is only available when macro \c __pgm_read_template__
02092     is defined. */
02093 template<typename T>
02094 T pgm_read_far<T> (uint_farptr_t addr);
02095 #endif /* DOXYGEN */
02096
02097 #endif /* __PGMSPACE_H_ */

```

## 23.27 portpins.h

```

00001 /* Copyright (c) 2003 Theodore A. Roth
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011     notice, this list of conditions and the following disclaimer in
00012     the documentation and/or other materials provided with the
00013     distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016     contributors may be used to endorse or promote products derived
00017     from this software without specific prior written permission.
00018
00019     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029     POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 #ifndef _AVR_PORTPINS_H_
00034 #define _AVR_PORTPINS_H_ 1
00035
00036 /* This file should only be included from <avr/io.h>, never directly. */
00037
00038 #ifndef _AVR_IO_H_
00039 # error "Include <avr/io.h> instead of this file."
00040 #endif
00041
00042 /* Define Generic PORTn, DDn, and PINn values. */
00043
00044 /* Port Data Register (generic) */
00045 #define PORT7 7
00046 #define PORT6 6
00047 #define PORT5 5
00048 #define PORT4 4
00049 #define PORT3 3
00050 #define PORT2 2
00051 #define PORT1 1
00052 #define PORT0 0
00053
00054 /* Port Data Direction Register (generic) */
00055 #define DD7 7
00056 #define DD6 6
00057 #define DD5 5

```

```
00058 #define DD4 4
00059 #define DD3 3
00060 #define DD2 2
00061 #define DD1 1
00062 #define DD0 0
00063
00064 /* Port Input Pins (generic) */
00065 #define PIN7 7
00066 #define PIN6 6
00067 #define PIN5 5
00068 #define PIN4 4
00069 #define PIN3 3
00070 #define PIN2 2
00071 #define PIN1 1
00072 #define PIN0 0
00073
00074 /* Define PORTxn an Pxn values for all possible port pins if not defined already by io.h. */
00075
00076 /* PORT A */
00077
00078 #if defined(PA0) && !defined(PORTA0)
00079 # define PORTA0 PA0
00080 #elif defined(PORTA0) && !defined(PA0)
00081 # define PA0 PORTA0
00082 #endif
00083 #if defined(PA1) && !defined(PORTA1)
00084 # define PORTA1 PA1
00085 #elif defined(PORTA1) && !defined(PA1)
00086 # define PA1 PORTA1
00087 #endif
00088 #if defined(PA2) && !defined(PORTA2)
00089 # define PORTA2 PA2
00090 #elif defined(PORTA2) && !defined(PA2)
00091 # define PA2 PORTA2
00092 #endif
00093 #if defined(PA3) && !defined(PORTA3)
00094 # define PORTA3 PA3
00095 #elif defined(PORTA3) && !defined(PA3)
00096 # define PA3 PORTA3
00097 #endif
00098 #if defined(PA4) && !defined(PORTA4)
00099 # define PORTA4 PA4
00100 #elif defined(PORTA4) && !defined(PA4)
00101 # define PA4 PORTA4
00102 #endif
00103 #if defined(PA5) && !defined(PORTA5)
00104 # define PORTA5 PA5
00105 #elif defined(PORTA5) && !defined(PA5)
00106 # define PA5 PORTA5
00107 #endif
00108 #if defined(PA6) && !defined(PORTA6)
00109 # define PORTA6 PA6
00110 #elif defined(PORTA6) && !defined(PA6)
00111 # define PA6 PORTA6
00112 #endif
00113 #if defined(PA7) && !defined(PORTA7)
00114 # define PORTA7 PA7
00115 #elif defined(PORTA7) && !defined(PA7)
00116 # define PA7 PORTA7
00117 #endif
00118
00119 /* PORT B */
00120
00121 #if defined(PB0) && !defined(PORTB0)
00122 # define PORTB0 PB0
00123 #elif defined(PORTB0) && !defined(PB0)
00124 # define PB0 PORTB0
00125 #endif
00126 #if defined(PB1) && !defined(PORTB1)
00127 # define PORTB1 PB1
00128 #elif defined(PORTB1) && !defined(PB1)
00129 # define PB1 PORTB1
00130 #endif
00131 #if defined(PB2) && !defined(PORTB2)
00132 # define PORTB2 PB2
00133 #elif defined(PORTB2) && !defined(PB2)
00134 # define PB2 PORTB2
00135 #endif
00136 #if defined(PB3) && !defined(PORTB3)
00137 # define PORTB3 PB3
00138 #elif defined(PORTB3) && !defined(PB3)
00139 # define PB3 PORTB3
00140 #endif
00141 #if defined(PB4) && !defined(PORTB4)
00142 # define PORTB4 PB4
00143 #elif defined(PORTB4) && !defined(PB4)
00144 # define PB4 PORTB4
```



```
00145 #endif
00146 #if defined(PB5) && !defined(PORTB5)
00147 # define PORTB5 PB5
00148 #elif defined(PORTB5) && !defined(PB5)
00149 # define PB5 PORTB5
00150 #endif
00151 #if defined(PB6) && !defined(PORTB6)
00152 # define PORTB6 PB6
00153 #elif defined(PORTB6) && !defined(PB6)
00154 # define PB6 PORTB6
00155 #endif
00156 #if defined(PB7) && !defined(PORTB7)
00157 # define PORTB7 PB7
00158 #elif defined(PORTB7) && !defined(PB7)
00159 # define PB7 PORTB7
00160 #endif
00161
00162 /* PORT C */
00163
00164 #if defined(PC0) && !defined(PORTC0)
00165 # define PORTC0 PC0
00166 #elif defined(PORTC0) && !defined(PC0)
00167 # define PC0 PORTC0
00168 #endif
00169 #if defined(PC1) && !defined(PORTC1)
00170 # define PORTC1 PC1
00171 #elif defined(PORTC1) && !defined(PC1)
00172 # define PC1 PORTC1
00173 #endif
00174 #if defined(PC2) && !defined(PORTC2)
00175 # define PORTC2 PC2
00176 #elif defined(PORTC2) && !defined(PC2)
00177 # define PC2 PORTC2
00178 #endif
00179 #if defined(PC3) && !defined(PORTC3)
00180 # define PORTC3 PC3
00181 #elif defined(PORTC3) && !defined(PC3)
00182 # define PC3 PORTC3
00183 #endif
00184 #if defined(PC4) && !defined(PORTC4)
00185 # define PORTC4 PC4
00186 #elif defined(PORTC4) && !defined(PC4)
00187 # define PC4 PORTC4
00188 #endif
00189 #if defined(PC5) && !defined(PORTC5)
00190 # define PORTC5 PC5
00191 #elif defined(PORTC5) && !defined(PC5)
00192 # define PC5 PORTC5
00193 #endif
00194 #if defined(PC6) && !defined(PORTC6)
00195 # define PORTC6 PC6
00196 #elif defined(PORTC6) && !defined(PC6)
00197 # define PC6 PORTC6
00198 #endif
00199 #if defined(PC7) && !defined(PORTC7)
00200 # define PORTC7 PC7
00201 #elif defined(PORTC7) && !defined(PC7)
00202 # define PC7 PORTC7
00203 #endif
00204
00205 /* PORT D */
00206
00207 #if defined(PD0) && !defined(PORTD0)
00208 # define PORTD0 PD0
00209 #elif defined(PORTD0) && !defined(PD0)
00210 # define PD0 PORTD0
00211 #endif
00212 #if defined(PD1) && !defined(PORTD1)
00213 # define PORTD1 PD1
00214 #elif defined(PORTD1) && !defined(PD1)
00215 # define PD1 PORTD1
00216 #endif
00217 #if defined(PD2) && !defined(PORTD2)
00218 # define PORTD2 PD2
00219 #elif defined(PORTD2) && !defined(PD2)
00220 # define PD2 PORTD2
00221 #endif
00222 #if defined(PD3) && !defined(PORTD3)
00223 # define PORTD3 PD3
00224 #elif defined(PORTD3) && !defined(PD3)
00225 # define PD3 PORTD3
00226 #endif
00227 #if defined(PD4) && !defined(PORTD4)
00228 # define PORTD4 PD4
00229 #elif defined(PORTD4) && !defined(PD4)
00230 # define PD4 PORTD4
00231 #endif
```

```
00232 #if defined(PD5) && !defined(PORTD5)
00233 # define PORTD5 PD5
00234 #elif defined(PORTD5) && !defined(PD5)
00235 # define PD5 PORTD5
00236 #endif
00237 #if defined(PD6) && !defined(PORTD6)
00238 # define PORTD6 PD6
00239 #elif defined(PORTD6) && !defined(PD6)
00240 # define PD6 PORTD6
00241 #endif
00242 #if defined(PD7) && !defined(PORTD7)
00243 # define PORTD7 PD7
00244 #elif defined(PORTD7) && !defined(PD7)
00245 # define PD7 PORTD7
00246 #endif
00247
00248 /* PORT E */
00249
00250 #if defined(PE0) && !defined(PORTE0)
00251 # define PORTE0 PE0
00252 #elif defined(PORTE0) && !defined(PE0)
00253 # define PE0 PORTE0
00254 #endif
00255 #if defined(PE1) && !defined(PORTE1)
00256 # define PORTE1 PE1
00257 #elif defined(PORTE1) && !defined(PE1)
00258 # define PE1 PORTE1
00259 #endif
00260 #if defined(PE2) && !defined(PORTE2)
00261 # define PORTE2 PE2
00262 #elif defined(PORTE2) && !defined(PE2)
00263 # define PE2 PORTE2
00264 #endif
00265 #if defined(PE3) && !defined(PORTE3)
00266 # define PORTE3 PE3
00267 #elif defined(PORTE3) && !defined(PE3)
00268 # define PE3 PORTE3
00269 #endif
00270 #if defined(PE4) && !defined(PORTE4)
00271 # define PORTE4 PE4
00272 #elif defined(PORTE4) && !defined(PE4)
00273 # define PE4 PORTE4
00274 #endif
00275 #if defined(PE5) && !defined(PORTE5)
00276 # define PORTE5 PE5
00277 #elif defined(PORTE5) && !defined(PE5)
00278 # define PE5 PORTE5
00279 #endif
00280 #if defined(PE6) && !defined(PORTE6)
00281 # define PORTE6 PE6
00282 #elif defined(PORTE6) && !defined(PE6)
00283 # define PE6 PORTE6
00284 #endif
00285 #if defined(PE7) && !defined(PORTE7)
00286 # define PORTE7 PE7
00287 #elif defined(PORTE7) && !defined(PE7)
00288 # define PE7 PORTE7
00289 #endif
00290
00291 /* PORT F */
00292
00293 #if defined(PF0) && !defined(PORTF0)
00294 # define PORTF0 PF0
00295 #elif defined(PORTF0) && !defined(PF0)
00296 # define PF0 PORTF0
00297 #endif
00298 #if defined(PF1) && !defined(PORTF1)
00299 # define PORTF1 PF1
00300 #elif defined(PORTF1) && !defined(PF1)
00301 # define PF1 PORTF1
00302 #endif
00303 #if defined(PF2) && !defined(PORTF2)
00304 # define PORTF2 PF2
00305 #elif defined(PORTF2) && !defined(PF2)
00306 # define PF2 PORTF2
00307 #endif
00308 #if defined(PF3) && !defined(PORTF3)
00309 # define PORTF3 PF3
00310 #elif defined(PORTF3) && !defined(PF3)
00311 # define PF3 PORTF3
00312 #endif
00313 #if defined(PF4) && !defined(PORTF4)
00314 # define PORTF4 PF4
00315 #elif defined(PORTF4) && !defined(PF4)
00316 # define PF4 PORTF4
00317 #endif
00318 #if defined(PF5) && !defined(PORTF5)
```

```
00319 # define PORTF5 PF5
00320 #elif defined(PORTF5) && !defined(PF5)
00321 # define PF5 PORTF5
00322 #endif
00323 #if defined(PF6) && !defined(PORTF6)
00324 # define PORTF6 PF6
00325 #elif defined(PORTF6) && !defined(PF6)
00326 # define PF6 PORTF6
00327 #endif
00328 #if defined(PF7) && !defined(PORTF7)
00329 # define PORTF7 PF7
00330 #elif defined(PORTF7) && !defined(PF7)
00331 # define PF7 PORTF7
00332 #endif
00333
00334 /* PORT G */
00335
00336 #if defined(PG0) && !defined(PORTG0)
00337 # define PORTG0 PG0
00338 #elif defined(PORTG0) && !defined(PG0)
00339 # define PG0 PORTG0
00340 #endif
00341 #if defined(PG1) && !defined(PORTG1)
00342 # define PORTG1 PG1
00343 #elif defined(PORTG1) && !defined(PG1)
00344 # define PG1 PORTG1
00345 #endif
00346 #if defined(PG2) && !defined(PORTG2)
00347 # define PORTG2 PG2
00348 #elif defined(PORTG2) && !defined(PG2)
00349 # define PG2 PORTG2
00350 #endif
00351 #if defined(PG3) && !defined(PORTG3)
00352 # define PORTG3 PG3
00353 #elif defined(PORTG3) && !defined(PG3)
00354 # define PG3 PORTG3
00355 #endif
00356 #if defined(PG4) && !defined(PORTG4)
00357 # define PORTG4 PG4
00358 #elif defined(PORTG4) && !defined(PG4)
00359 # define PG4 PORTG4
00360 #endif
00361 #if defined(PG5) && !defined(PORTG5)
00362 # define PORTG5 PG5
00363 #elif defined(PORTG5) && !defined(PG5)
00364 # define PG5 PORTG5
00365 #endif
00366 #if defined(PG6) && !defined(PORTG6)
00367 # define PORTG6 PG6
00368 #elif defined(PORTG6) && !defined(PG6)
00369 # define PG6 PORTG6
00370 #endif
00371 #if defined(PG7) && !defined(PORTG7)
00372 # define PORTG7 PG7
00373 #elif defined(PORTG7) && !defined(PG7)
00374 # define PG7 PORTG7
00375 #endif
00376
00377 /* PORT H */
00378
00379 #if defined(PH0) && !defined(PORTH0)
00380 # define PORTH0 PH0
00381 #elif defined(PORTH0) && !defined(PH0)
00382 # define PH0 PORTH0
00383 #endif
00384 #if defined(PH1) && !defined(PORTH1)
00385 # define PORTH1 PH1
00386 #elif defined(PORTH1) && !defined(PH1)
00387 # define PH1 PORTH1
00388 #endif
00389 #if defined(PH2) && !defined(PORTH2)
00390 # define PORTH2 PH2
00391 #elif defined(PORTH2) && !defined(PH2)
00392 # define PH2 PORTH2
00393 #endif
00394 #if defined(PH3) && !defined(PORTH3)
00395 # define PORTH3 PH3
00396 #elif defined(PORTH3) && !defined(PH3)
00397 # define PH3 PORTH3
00398 #endif
00399 #if defined(PH4) && !defined(PORTH4)
00400 # define PORTH4 PH4
00401 #elif defined(PORTH4) && !defined(PH4)
00402 # define PH4 PORTH4
00403 #endif
00404 #if defined(PH5) && !defined(PORTH5)
00405 # define PORTH5 PH5
```

```
00406 #elif defined(PORTH5) && !defined(PH5)
00407 # define PH5 PORTH5
00408 #endif
00409 #if defined(PH6) && !defined(PORTH6)
00410 # define PORTH6 PH6
00411 #elif defined(PORTH6) && !defined(PH6)
00412 # define PH6 PORTH6
00413 #endif
00414 #if defined(PH7) && !defined(PORTH7)
00415 # define PORTH7 PH7
00416 #elif defined(PORTH7) && !defined(PH7)
00417 # define PH7 PORTH7
00418 #endif
00419
00420 /* PORT J */
00421
00422 #if defined(PJ0) && !defined(PORTJ0)
00423 # define PORTJ0 PJ0
00424 #elif defined(PORTJ0) && !defined(PJ0)
00425 # define PJ0 PORTJ0
00426 #endif
00427 #if defined(PJ1) && !defined(PORTJ1)
00428 # define PORTJ1 PJ1
00429 #elif defined(PORTJ1) && !defined(PJ1)
00430 # define PJ1 PORTJ1
00431 #endif
00432 #if defined(PJ2) && !defined(PORTJ2)
00433 # define PORTJ2 PJ2
00434 #elif defined(PORTJ2) && !defined(PJ2)
00435 # define PJ2 PORTJ2
00436 #endif
00437 #if defined(PJ3) && !defined(PORTJ3)
00438 # define PORTJ3 PJ3
00439 #elif defined(PORTJ3) && !defined(PJ3)
00440 # define PJ3 PORTJ3
00441 #endif
00442 #if defined(PJ4) && !defined(PORTJ4)
00443 # define PORTJ4 PJ4
00444 #elif defined(PORTJ4) && !defined(PJ4)
00445 # define PJ4 PORTJ4
00446 #endif
00447 #if defined(PJ5) && !defined(PORTJ5)
00448 # define PORTJ5 PJ5
00449 #elif defined(PORTJ5) && !defined(PJ5)
00450 # define PJ5 PORTJ5
00451 #endif
00452 #if defined(PJ6) && !defined(PORTJ6)
00453 # define PORTJ6 PJ6
00454 #elif defined(PORTJ6) && !defined(PJ6)
00455 # define PJ6 PORTJ6
00456 #endif
00457 #if defined(PJ7) && !defined(PORTJ7)
00458 # define PORTJ7 PJ7
00459 #elif defined(PORTJ7) && !defined(PJ7)
00460 # define PJ7 PORTJ7
00461 #endif
00462
00463 /* PORT K */
00464
00465 #if defined(PK0) && !defined(PORTK0)
00466 # define PORTK0 PK0
00467 #elif defined(PORTK0) && !defined(PK0)
00468 # define PK0 PORTK0
00469 #endif
00470 #if defined(PK1) && !defined(PORTK1)
00471 # define PORTK1 PK1
00472 #elif defined(PORTK1) && !defined(PK1)
00473 # define PK1 PORTK1
00474 #endif
00475 #if defined(PK2) && !defined(PORTK2)
00476 # define PORTK2 PK2
00477 #elif defined(PORTK2) && !defined(PK2)
00478 # define PK2 PORTK2
00479 #endif
00480 #if defined(PK3) && !defined(PORTK3)
00481 # define PORTK3 PK3
00482 #elif defined(PORTK3) && !defined(PK3)
00483 # define PK3 PORTK3
00484 #endif
00485 #if defined(PK4) && !defined(PORTK4)
00486 # define PORTK4 PK4
00487 #elif defined(PORTK4) && !defined(PK4)
00488 # define PK4 PORTK4
00489 #endif
00490 #if defined(PK5) && !defined(PORTK5)
00491 # define PORTK5 PK5
00492 #elif defined(PORTK5) && !defined(PK5)
```

```

00493 # define PK5 PORTK5
00494 #endif
00495 #if defined(PK6) && !defined(PORTK6)
00496 # define PORTK6 PK6
00497 #elif defined(PORTK6) && !defined(PK6)
00498 # define PK6 PORTK6
00499 #endif
00500 #if defined(PK7) && !defined(PORTK7)
00501 # define PORTK7 PK7
00502 #elif defined(PORTK7) && !defined(PK7)
00503 # define PK7 PORTK7
00504 #endif
00505
00506 /* PORT L */
00507
00508 #if defined(PL0) && !defined(PORTL0)
00509 # define PORTL0 PL0
00510 #elif defined(PORTL0) && !defined(PL0)
00511 # define PL0 PORTL0
00512 #endif
00513 #if defined(PL1) && !defined(PORTL1)
00514 # define PORTL1 PL1
00515 #elif defined(PORTL1) && !defined(PL1)
00516 # define PL1 PORTL1
00517 #endif
00518 #if defined(PL2) && !defined(PORTL2)
00519 # define PORTL2 PL2
00520 #elif defined(PORTL2) && !defined(PL2)
00521 # define PL2 PORTL2
00522 #endif
00523 #if defined(PL3) && !defined(PORTL3)
00524 # define PORTL3 PL3
00525 #elif defined(PORTL3) && !defined(PL3)
00526 # define PL3 PORTL3
00527 #endif
00528 #if defined(PL4) && !defined(PORTL4)
00529 # define PORTL4 PL4
00530 #elif defined(PORTL4) && !defined(PL4)
00531 # define PL4 PORTL4
00532 #endif
00533 #if defined(PL5) && !defined(PORTL5)
00534 # define PORTL5 PL5
00535 #elif defined(PORTL5) && !defined(PL5)
00536 # define PL5 PORTL5
00537 #endif
00538 #if defined(PL6) && !defined(PORTL6)
00539 # define PORTL6 PL6
00540 #elif defined(PORTL6) && !defined(PL6)
00541 # define PL6 PORTL6
00542 #endif
00543 #if defined(PL7) && !defined(PORTL7)
00544 # define PORTL7 PL7
00545 #elif defined(PORTL7) && !defined(PL7)
00546 # define PL7 PORTL7
00547 #endif
00548
00549 #endif /* _AVR_PORTPINS_H_ */

```

## 23.28 power.h File Reference

### Macros

- #define [clock\\_prescale\\_get\(\)](#) (clock\_div\_t)(CLKPR & (uint8\_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))

### Functions

- static void [power\\_all\\_enable](#) ()
- static void [power\\_all\\_disable](#) ()
- void [clock\\_prescale\\_set](#) (clock\_div\_t \_\_x)

## 23.29 power.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2006, 2007, 2008 Eric B. Weddington
00002    Copyright (c) 2011 Frédéric Nadeau
00003    All rights reserved.
00004
00005    Redistribution and use in source and binary forms, with or without
00006    modification, are permitted provided that the following conditions are met:
00007
00008    * Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010    * Redistributions in binary form must reproduce the above copyright
00011    notice, this list of conditions and the following disclaimer in
00012    the documentation and/or other materials provided with the
00013    distribution.
00014    * Neither the name of the copyright holders nor the names of
00015    contributors may be used to endorse or promote products derived
00016    from this software without specific prior written permission.
00017
00018    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00019    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00020    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00021    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00022    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00023    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00024    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00025    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00026    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00027    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00028    POSSIBILITY OF SUCH DAMAGE. */
00029
00030 /* $Id$ */
00031
00032 #ifndef _AVR_POWER_H_
00033 #define _AVR_POWER_H_ 1
00034
00035 #include <avr/io.h>
00036 #include <stdint.h>
00037
00038 #ifndef __DOXYGEN__
00039 #ifndef __ATTR_ALWAYS_INLINE__
00040 #define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00041 #endif
00042 #endif /* ! DOXYGEN */
00043
00044 /** \file */
00045 /** \defgroup avr_power <avr/power.h>: Power Reduction Management
00046
00047 \code #include <avr/power.h>\endcode
00048
00049 Many AVRs contain a Power Reduction Register (PRR) or Registers (PRRx) that
00050 allow you to reduce power consumption by disabling or enabling various on-board
00051 peripherals as needed. Some devices have the XTAL Divide Control Register
00052 (XDIV) which offer similar functionality as System Clock Prescale
00053 Register (CLKPR).
00054
00055 There are many macros in this header file that provide an easy interface
00056 to enable or disable on-board peripherals to reduce power. See the table below.
00057
00058 \note Not all AVR devices have a Power Reduction Register (for example
00059 the ATmega8). On those devices without a Power Reduction Register, the
00060 power reduction macros are not available..
00061
00062 \note Not all AVR devices contain the same peripherals (for example, the LCD
00063 interface), or they will be named differently (for example, USART and
00064 USART0). Please consult your device's datasheet, or the header file, to
00065 find out which macros are applicable to your device.
00066
00067 \note For device using the XTAL Divide Control Register (XDIV), when prescaler
00068 is used, Timer/Counter0 can only be used in asynchronous mode. Keep in mind
00069 that Timer/Counter0 source shall be less than ¼th of peripheral clock.
00070 Therefore, when using a typical 32.768 kHz crystal, one shall not scale
00071 the clock below 131.072 kHz.
00072
00073 \anchor avr_powermacros
00074 <small>
00075 <table>
00076 <caption>Power Macros</caption>
00077 <tr>
00078 <th>Power Macro
00079 <th>Description
00080 </tr>
00081 <tr>
00082 <td>\c power_aca_disable()</td>
00083 <td>Disable the Analog Comparator on PortA</td>

```

```
00084 </tr>
00085 <tr>
00086 <td>\c power_aca_enable()</td>
00087 <td>Enable the Analog Comparator on PortA</td>
00088 </tr>
00089 <tr>
00090 <td>\c power_adc_enable()</td>
00091 <td>Enable the Analog to Digital Converter module</td>
00092 </tr>
00093 <tr>
00094 <td>\c power_adc_disable()</td>
00095 <td>Disable the Analog to Digital Converter module</td>
00096 </tr>
00097 <tr>
00098 <td>\c power_adca_disable()</td>
00099 <td>Disable the Analog to Digital Converter module on PortA</td>
00100 </tr>
00101 <tr>
00102 <td>\c power_adca_enable()</td>
00103 <td>Enable the Analog to Digital Converter module on PortA</td>
00104 </tr>
00105 <tr>
00106 <td>\c power_evsys_disable()</td>
00107 <td>Disable the EVSYS module</td>
00108 </tr>
00109 <tr>
00110 <td>\c power_evsys_enable()</td>
00111 <td>Enable the EVSYS module</td>
00112 </tr>
00113 <tr>
00114 <td>\c power_hiresc_disable()</td>
00115 <td>Disable the HIRES module on PortC</td>
00116 </tr>
00117 <tr>
00118 <td>\c power_hiresc_enable()</td>
00119 <td>Enable the HIRES module on PortC</td>
00120 </tr>
00121 <tr>
00122 <td>\c power_lcd_enable()</td>
00123 <td>Enable the LCD module</td>
00124 </tr>
00125 <tr>
00126 <td>\c power_lcd_disable()</td>
00127 <td>Disable the LCD module</td>
00128 </tr>
00129 <tr>
00130 <td>\c power_pga_enable()</td>
00131 <td>Enable the Programmable Gain Amplifier module</td>
00132 </tr>
00133 <tr>
00134 <td>\c power_pga_disable()</td>
00135 <td>Disable the Programmable Gain Amplifier module</td>
00136 </tr>
00137 <tr>
00138 <td>\c power_pscr_enable()</td>
00139 <td>Enable the Reduced Power Stage Controller module</td>
00140 </tr>
00141 <tr>
00142 <td>\c power_pscr_disable()</td>
00143 <td>Disable the Reduced Power Stage Controller module</td>
00144 </tr>
00145 <tr>
00146 <td>\c power_psc0_enable()</td>
00147 <td>Enable the Power Stage Controller 0 module</td>
00148 </tr>
00149 <tr>
00150 <td>\c power_psc0_disable()</td>
00151 <td>Disable the Power Stage Controller 0 module</td>
00152 </tr>
00153 <tr>
00154 <td>\c power_psc1_enable()</td>
00155 <td>Enable the Power Stage Controller 1 module</td>
00156 </tr>
00157 <tr>
00158 <td>\c power_psc1_disable()</td>
00159 <td>Disable the Power Stage Controller 1 module</td>
00160 </tr>
00161 <tr>
00162 <td>\c power_psc2_enable()</td>
00163 <td>Enable the Power Stage Controller 2 module</td>
00164 </tr>
00165 <tr>
00166 <td>\c power_psc2_disable()</td>
00167 <td>Disable the Power Stage Controller 2 module</td>
00168 </tr>
00169 <tr>
00170 <td>\c power_ram0_enable()</td>
```

```
00171     <td>Enable the SRAM block 0</td>
00172 </tr>
00173 <tr>
00174     <td>\c power_ram0_disable()</td>
00175     <td>Disable the SRAM block 0</td>
00176 </tr>
00177 <tr>
00178     <td>\c power_ram1_enable()</td>
00179     <td>Enable the SRAM block 1</td>
00180 </tr>
00181 <tr>
00182     <td>\c power_ram1_disable()</td>
00183     <td>Disable the SRAM block 1</td>
00184 </tr>
00185 <tr>
00186     <td>\c power_ram2_enable()</td>
00187     <td>Enable the SRAM block 2</td>
00188 </tr>
00189 <tr>
00190     <td>\c power_ram2_disable()</td>
00191     <td>Disable the SRAM block 2</td>
00192 </tr>
00193 <tr>
00194     <td>\c power_ram3_enable()</td>
00195     <td>Enable the SRAM block 3</td>
00196 </tr>
00197 <tr>
00198     <td>\c power_ram3_disable()</td>
00199     <td>Disable the SRAM block 3</td>
00200 </tr>
00201 <tr>
00202     <td>\c power_rtc_disable()</td>
00203     <td>Disable the RTC module</td>
00204 </tr>
00205 <tr>
00206     <td>\c power_rtc_enable()</td>
00207     <td>Enable the RTC module</td>
00208 </tr>
00209 <tr>
00210     <td>\c power_spi_enable()</td>
00211     <td>Enable the Serial Peripheral Interface module</td>
00212 </tr>
00213 <tr>
00214     <td>\c power_spi_disable()</td>
00215     <td>Disable the Serial Peripheral Interface module</td>
00216 </tr>
00217 <tr>
00218     <td>\c power_spic_disable()</td>
00219     <td>Disable the SPI module on PortC</td>
00220 </tr>
00221 <tr>
00222     <td>\c power_spic_enable()</td>
00223     <td>Enable the SPI module on PortC</td>
00224 </tr>
00225 <tr>
00226     <td>\c power_spid_disable()</td>
00227     <td>Disable the SPI module on PortD</td>
00228 </tr>
00229 <tr>
00230     <td>\c power_spid_enable()</td>
00231     <td>Enable the SPI module on PortD</td>
00232 </tr>
00233 <tr>
00234     <td>\c power_tc0c_disable()</td>
00235     <td>Disable the TC0 module on PortC</td>
00236 </tr>
00237 <tr>
00238     <td>\c power_tc0c_enable()</td>
00239     <td>Enable the TC0 module on PortC</td>
00240 </tr>
00241 <tr>
00242     <td>\c power_tc0d_disable()</td>
00243     <td>Disable the TC0 module on PortD</td>
00244 </tr>
00245 <tr>
00246     <td>\c power_tc0d_enable()</td>
00247     <td>Enable the TC0 module on PortD</td>
00248 </tr>
00249 <tr>
00250     <td>\c power_tc0e_disable()</td>
00251     <td>Disable the TC0 module on PortE</td>
00252 </tr>
00253 <tr>
00254     <td>\c power_tc0e_enable()</td>
00255     <td>Enable the TC0 module on PortE</td>
00256 </tr>
00257 <tr>
```



```
00258     <td>\c power_tc0f_disable()</td>
00259     <td>Disable the TC0 module on PortF</td>
00260 </tr>
00261 <tr>
00262     <td>\c power_tc0f_enable()</td>
00263     <td>Enable the TC0 module on PortF</td>
00264 </tr>
00265 <tr>
00266     <td>\c power_tclc_disable()</td>
00267     <td>Disable the TC1 module on PortC</td>
00268 </tr>
00269 <tr>
00270     <td>\c power_tclc_enable()</td>
00271     <td>Enable the TC1 module on PortC</td>
00272 </tr>
00273 <tr>
00274     <td>\c power_twic_disable()</td>
00275     <td>Disable the Two Wire Interface module on PortC</td>
00276 </tr>
00277 <tr>
00278     <td>\c power_twic_enable()</td>
00279     <td>Enable the Two Wire Interface module on PortC</td>
00280 </tr>
00281 <tr>
00282     <td>\c power_twie_disable()</td>
00283     <td>Disable the Two Wire Interface module on PortE</td>
00284 </tr>
00285 <tr>
00286     <td>\c power_twie_enable()</td>
00287     <td>Enable the Two Wire Interface module on PortE</td>
00288 </tr>
00289 <tr>
00290     <td>\c power_timer0_enable()</td>
00291     <td>Enable the Timer 0 module</td>
00292 </tr>
00293 <tr>
00294     <td>\c power_timer0_disable()</td>
00295     <td>Disable the Timer 0 module</td>
00296 </tr>
00297 <tr>
00298     <td>\c power_timer1_enable()</td>
00299     <td>Enable the Timer 1 module</td>
00300 </tr>
00301 <tr>
00302     <td>\c power_timer1_disable()</td>
00303     <td>Disable the Timer 1 module</td>
00304 </tr>
00305 <tr>
00306     <td>\c power_timer2_enable()</td>
00307     <td>Enable the Timer 2 module</td>
00308 </tr>
00309 <tr>
00310     <td>\c power_timer2_disable()</td>
00311     <td>Disable the Timer 2 module</td>
00312 </tr>
00313 <tr>
00314     <td>\c power_timer3_enable()</td>
00315     <td>Enable the Timer 3 module</td>
00316 </tr>
00317 <tr>
00318     <td>\c power_timer3_disable()</td>
00319     <td>Disable the Timer 3 module</td>
00320 </tr>
00321 <tr>
00322     <td>\c power_timer4_enable()</td>
00323     <td>Enable the Timer 4 module</td>
00324 </tr>
00325 <tr>
00326     <td>\c power_timer4_disable()</td>
00327     <td>Disable the Timer 4 module</td>
00328 </tr>
00329 <tr>
00330     <td>\c power_timer5_enable()</td>
00331     <td>Enable the Timer 5 module</td>
00332 </tr>
00333 <tr>
00334     <td>\c power_timer5_disable()</td>
00335     <td>Disable the Timer 5 module</td>
00336 </tr>
00337 <tr>
00338     <td>\c power_twi_enable()</td>
00339     <td>Enable the Two Wire Interface module</td>
00340 </tr>
00341 <tr>
00342     <td>\c power_twi_disable()</td>
00343     <td>Disable the Two Wire Interface module</td>
00344 </tr>
```

```
00345 <tr>
00346 <td>\c power_usart_enable()</td>
00347 <td>Enable the USART module</td>
00348 </tr>
00349 <tr>
00350 <td>\c power_usart_disable()</td>
00351 <td>Disable the USART module</td>
00352 </tr>
00353 <tr>
00354 <td>\c power_usart0_enable()</td>
00355 <td>Enable the USART 0 module</td>
00356 </tr>
00357 <tr>
00358 <td>\c power_usart0_disable()</td>
00359 <td>Disable the USART 0 module</td>
00360 </tr>
00361 <tr>
00362 <td>\c power_usart1_enable()</td>
00363 <td>Enable the USART 1 module</td>
00364 </tr>
00365 <tr>
00366 <td>\c power_usart1_disable()</td>
00367 <td>Disable the USART 1 module</td>
00368 </tr>
00369 <tr>
00370 <td>\c power_usart2_enable()</td>
00371 <td>Enable the USART 2 module</td>
00372 </tr>
00373 <tr>
00374 <td>\c power_usart2_disable()</td>
00375 <td>Disable the USART 2 module</td>
00376 </tr>
00377 <tr>
00378 <td>\c power_usart3_enable()</td>
00379 <td>Enable the USART 3 module</td>
00380 </tr>
00381 <tr>
00382 <td>\c power_usart3_disable()</td>
00383 <td>Disable the USART 3 module</td>
00384 </tr>
00385 <tr>
00386 <td>\c power_usartc0_disable()</td>
00387 <td> Disable the USART0 module on PortC</td>
00388 </tr>
00389 <tr>
00390 <td>\c power_usartc0_enable()</td>
00391 <td> Enable the USART0 module on PortC</td>
00392 </tr>
00393 <tr>
00394 <td>\c power_usartd0_disable()</td>
00395 <td> Disable the USART0 module on PortD</td>
00396 </tr>
00397 <tr>
00398 <td>\c power_usartd0_enable()</td>
00399 <td> Enable the USART0 module on PortD</td>
00400 </tr>
00401 <tr>
00402 <td>\c power_usarte0_disable()</td>
00403 <td> Disable the USART0 module on PortE</td>
00404 </tr>
00405 <tr>
00406 <td>\c power_usarte0_enable()</td>
00407 <td> Enable the USART0 module on PortE</td>
00408 </tr>
00409 <tr>
00410 <td>\c power_usartf0_disable()</td>
00411 <td> Disable the USART0 module on PortF</td>
00412 </tr>
00413 <tr>
00414 <td>\c power_usartf0_enable()</td>
00415 <td> Enable the USART0 module on PortF</td>
00416 </tr>
00417 <tr>
00418 <td>\c power_usb_enable()</td>
00419 <td>Enable the USB module</td>
00420 </tr>
00421 <tr>
00422 <td>\c power_usb_disable()</td>
00423 <td>Disable the USB module</td>
00424 </tr>
00425 <tr>
00426 <td>\c power_usi_enable()</td>
00427 <td>Enable the Universal Serial Interface module</td>
00428 </tr>
00429 <tr>
00430 <td>\c power_usi_disable()</td>
00431 <td>Disable the Universal Serial Interface module</td>
```

```
00432 </tr>
00433 <tr>
00434 <td>\c power_vadc_enable()</td>
00435 <td>Enable the Voltage ADC module</td>
00436 </tr>
00437 <tr>
00438 <td>\c power_vadc_disable()</td>
00439 <td>Disable the Voltage ADC module</td>
00440 </tr>
00441 <tr>
00442 <td>\c power_all_enable()</td>
00443 <td>Enable all modules</td>
00444 </tr>
00445 <tr>
00446 <td>\c power_all_disable()</td>
00447 <td>Disable all modules</td>
00448 </tr>
00449 </table>
00450 </small>
00451 */
00452
00453 #if defined(__AVR_HAVE_PRR_PRADC)
00454 #define power_adc_enable() (PRR &= (uint8_t)~(1 << PRADC))
00455 #define power_adc_disable() (PRR |= (uint8_t)(1 << PRADC))
00456 #endif
00457
00458 #if defined(__AVR_HAVE_PRR_PRCAN)
00459 #define power_can_enable() (PRR &= (uint8_t)~(1 << PRCAN))
00460 #define power_can_disable() (PRR |= (uint8_t)(1 << PRCAN))
00461 #endif
00462
00463 #if defined(__AVR_HAVE_PRR_PRLCD)
00464 #define power_lcd_enable() (PRR &= (uint8_t)~(1 << PRLCD))
00465 #define power_lcd_disable() (PRR |= (uint8_t)(1 << PRLCD))
00466 #endif
00467
00468 #if defined(__AVR_HAVE_PRR_PRLIN)
00469 #define power_lin_enable() (PRR &= (uint8_t)~(1 << PRLIN))
00470 #define power_lin_disable() (PRR |= (uint8_t)(1 << PRLIN))
00471 #endif
00472
00473 #if defined(__AVR_HAVE_PRR_PRPSC)
00474 #define power_psc_enable() (PRR &= (uint8_t)~(1 << PRPSC))
00475 #define power_psc_disable() (PRR |= (uint8_t)(1 << PRPSC))
00476 #endif
00477
00478 #if defined(__AVR_HAVE_PRR_PRPSC0)
00479 #define power_psc0_enable() (PRR &= (uint8_t)~(1 << PRPSC0))
00480 #define power_psc0_disable() (PRR |= (uint8_t)(1 << PRPSC0))
00481 #endif
00482
00483 #if defined(__AVR_HAVE_PRR_PRPSC1)
00484 #define power_psc1_enable() (PRR &= (uint8_t)~(1 << PRPSC1))
00485 #define power_psc1_disable() (PRR |= (uint8_t)(1 << PRPSC1))
00486 #endif
00487
00488 #if defined(__AVR_HAVE_PRR_PRPSC2)
00489 #define power_psc2_enable() (PRR &= (uint8_t)~(1 << PRPSC2))
00490 #define power_psc2_disable() (PRR |= (uint8_t)(1 << PRPSC2))
00491 #endif
00492
00493 #if defined(__AVR_HAVE_PRR_PRPSCR)
00494 #define power_pscr_enable() (PRR &= (uint8_t)~(1 << PRPSCR))
00495 #define power_pscr_disable() (PRR |= (uint8_t)(1 << PRPSCR))
00496 #endif
00497
00498 #if defined(__AVR_HAVE_PRR_PRSPI)
00499 #define power_spi_enable() (PRR &= (uint8_t)~(1 << PRSPI))
00500 #define power_spi_disable() (PRR |= (uint8_t)(1 << PRSPI))
00501 #endif
00502
00503 #if defined(__AVR_HAVE_PRR_PRTIM0)
00504 #define power_timer0_enable() (PRR &= (uint8_t)~(1 << PRTIM0))
00505 #define power_timer0_disable() (PRR |= (uint8_t)(1 << PRTIM0))
00506 #endif
00507
00508 #if defined(__AVR_HAVE_PRR_PRTIM1)
00509 #define power_timer1_enable() (PRR &= (uint8_t)~(1 << PRTIM1))
00510 #define power_timer1_disable() (PRR |= (uint8_t)(1 << PRTIM1))
00511 #endif
00512
00513 #if defined(__AVR_HAVE_PRR_PRTIM2)
00514 #define power_timer2_enable() (PRR &= (uint8_t)~(1 << PRTIM2))
00515 #define power_timer2_disable() (PRR |= (uint8_t)(1 << PRTIM2))
00516 #endif
00517
00518 #if defined(__AVR_HAVE_PRR_PRTWI)
```

```
00519 #define power_twi_enable()      (PRR &= (uint8_t)~(1 << PRTWI))
00520 #define power_twi_disable()     (PRR |= (uint8_t)(1 << PRTWI))
00521 #endif
00522
00523 #if defined(__AVR_HAVE_PRR_PRUSART)
00524 #define power_usart_enable()     (PRR &= (uint8_t)~(1 << PRUSART))
00525 #define power_usart_disable()   (PRR |= (uint8_t)(1 << PRUSART))
00526 #endif
00527
00528 #if defined(__AVR_HAVE_PRR_PRUSART0)
00529 #define power_usart0_enable()    (PRR &= (uint8_t)~(1 << PRUSART0))
00530 #define power_usart0_disable()  (PRR |= (uint8_t)(1 << PRUSART0))
00531 #endif
00532
00533 #if defined(__AVR_HAVE_PRR_PRUSART1)
00534 #define power_usart1_enable()    (PRR &= (uint8_t)~(1 << PRUSART1))
00535 #define power_usart1_disable()  (PRR |= (uint8_t)(1 << PRUSART1))
00536 #endif
00537
00538 #if defined(__AVR_HAVE_PRR_PRUSI)
00539 #define power_usi_enable()       (PRR &= (uint8_t)~(1 << PRUSI))
00540 #define power_usi_disable()     (PRR |= (uint8_t)(1 << PRUSI))
00541 #endif
00542
00543 #if defined(__AVR_HAVE_PRR0_PRADC)
00544 #define power_adc_enable()       (PRR0 &= (uint8_t)~(1 << PRADC))
00545 #define power_adc_disable()     (PRR0 |= (uint8_t)(1 << PRADC))
00546 #endif
00547
00548 #if defined(__AVR_HAVE_PRR0_PRCO)
00549 #define power_clock_output_enable() (PRR0 &= (uint8_t)~(1 << PRCO))
00550 #define power_clock_output_disable() (PRR0 |= (uint8_t)(1 << PRCO))
00551 #endif
00552
00553 #if defined(__AVR_HAVE_PRR0_PRCRC)
00554 #define power_crc_enable()       (PRR0 &= (uint8_t)~(1 << PRCRC))
00555 #define power_crc_disable()     (PRR0 |= (uint8_t)(1 << PRCRC))
00556 #endif
00557
00558 #if defined(__AVR_HAVE_PRR0_PRCU)
00559 #define power_crypto_enable()    (PRR0 &= (uint8_t)~(1 << PRCU))
00560 #define power_crypto_disable()  (PRR0 |= (uint8_t)(1 << PRCU))
00561 #endif
00562
00563 #if defined(__AVR_HAVE_PRR0_PRDS)
00564 #define power_irdriver_enable()  (PRR0 &= (uint8_t)~(1 << PRDS))
00565 #define power_irdriver_disable() (PRR0 |= (uint8_t)(1 << PRDS))
00566 #endif
00567
00568 #if defined(__AVR_HAVE_PRR0_PRLFR)
00569 #define power_lfreceiver_enable() (PRR0 &= (uint8_t)~(1 << PRLFR))
00570 #define power_lfreceiver_disable() (PRR0 |= (uint8_t)(1 << PRLFR))
00571 #endif
00572
00573 #if defined(__AVR_HAVE_PRR0_PRLFRS)
00574 #define power_lfrs_enable()      (PRR0 &= (uint8_t)~(1 << PRLFRS))
00575 #define power_lfrs_disable()    (PRR0 |= (uint8_t)(1 << PRLFRS))
00576 #endif
00577
00578 #if defined(__AVR_HAVE_PRR0_PRLIN)
00579 #define power_lin_enable()       (PRR0 &= (uint8_t)~(1 << PRLIN))
00580 #define power_lin_disable()     (PRR0 |= (uint8_t)(1 << PRLIN))
00581 #endif
00582
00583 #if defined(__AVR_HAVE_PRR0_PRPGA)
00584 #define power_pga_enable()       (PRR0 &= (uint8_t)~(1 << PRPGA))
00585 #define power_pga_disable()     (PRR0 |= (uint8_t)(1 << PRPGA))
00586 #endif
00587
00588 #if defined(__AVR_HAVE_PRR0_PRRXDC)
00589 #define power_receive_dsp_control_enable() (PRR0 &= (uint8_t)~(1 << PRRXDC))
00590 #define power_receive_dsp_control_disable() (PRR0 |= (uint8_t)(1 << PRRXDC))
00591 #endif
00592
00593 #if defined(__AVR_HAVE_PRR0_PRSPI)
00594 #define power_spi_enable()       (PRR0 &= (uint8_t)~(1 << PRSPI))
00595 #define power_spi_disable()     (PRR0 |= (uint8_t)(1 << PRSPI))
00596 #endif
00597
00598 #if defined(__AVR_HAVE_PRR0_PRT0)
00599 #define power_timer0_enable()    (PRR0 &= (uint8_t)~(1 << PRT0))
00600 #define power_timer0_disable()  (PRR0 |= (uint8_t)(1 << PRT0))
00601 #endif
00602
00603 #if defined(__AVR_HAVE_PRR0_PRTIM0)
00604 #define power_timer0_enable()    (PRR0 &= (uint8_t)~(1 << PRTIM0))
00605 #define power_timer0_disable()  (PRR0 |= (uint8_t)(1 << PRTIM0))
```

```
00606 #endif
00607
00608 #if defined(__AVR_HAVE_PRR0_PRT1)
00609 #define power_timer1_enable() (PRR0 &= (uint8_t)~(1 << PRT1))
00610 #define power_timer1_disable() (PRR0 |= (uint8_t)(1 << PRT1))
00611 #endif
00612
00613 #if defined(__AVR_HAVE_PRR0_PRTIM1)
00614 #define power_timer1_enable() (PRR0 &= (uint8_t)~(1 << PRTIM1))
00615 #define power_timer1_disable() (PRR0 |= (uint8_t)(1 << PRTIM1))
00616 #endif
00617
00618 #if defined(__AVR_HAVE_PRR0_PRT2)
00619 #define power_timer2_enable() (PRR0 &= (uint8_t)~(1 << PRT2))
00620 #define power_timer2_disable() (PRR0 |= (uint8_t)(1 << PRT2))
00621 #endif
00622
00623 #if defined(__AVR_HAVE_PRR0_PRTIM2)
00624 #define power_timer2_enable() (PRR0 &= (uint8_t)~(1 << PRTIM2))
00625 #define power_timer2_disable() (PRR0 |= (uint8_t)(1 << PRTIM2))
00626 #endif
00627
00628 #if defined(__AVR_HAVE_PRR0_PRT3)
00629 #define power_timer3_enable() (PRR0 &= (uint8_t)~(1 << PRT3))
00630 #define power_timer3_disable() (PRR0 |= (uint8_t)(1 << PRT3))
00631 #endif
00632
00633 #if defined(__AVR_HAVE_PRR0_PRTM)
00634 #define power_timermodulator_enable() (PRR0 &= (uint8_t)~(1 << PRTM))
00635 #define power_timermodulator_disable() (PRR0 |= (uint8_t)(1 << PRTM))
00636 #endif
00637
00638 #if defined(__AVR_HAVE_PRR0_PRTWI)
00639 #define power_twi_enable() (PRR0 &= (uint8_t)~(1 << PRTWI))
00640 #define power_twi_disable() (PRR0 |= (uint8_t)(1 << PRTWI))
00641 #endif
00642
00643 #if defined(__AVR_HAVE_PRR0_PRTWI1)
00644 #define power_twi1_enable() (PRR0 &= (uint8_t)~(1 << PRTWI1))
00645 #define power_twi1_disable() (PRR0 |= (uint8_t)(1 << PRTWI1))
00646 #endif
00647
00648 #if defined(__AVR_HAVE_PRR0_PRTXDC)
00649 #define power_transmit_dsp_control_enable() (PRR0 &= (uint8_t)~(1 << PRTXDC))
00650 #define power_transmit_dsp_control_disable() (PRR0 |= (uint8_t)(1 << PRTXDC))
00651 #endif
00652
00653 #if defined(__AVR_HAVE_PRR0_PRUSART0)
00654 #define power_usart0_enable() (PRR0 &= (uint8_t)~(1 << PRUSART0))
00655 #define power_usart0_disable() (PRR0 |= (uint8_t)(1 << PRUSART0))
00656 #endif
00657
00658 #if defined(__AVR_HAVE_PRR0_PRUSART1)
00659 #define power_usart1_enable() (PRR0 &= (uint8_t)~(1 << PRUSART1))
00660 #define power_usart1_disable() (PRR0 |= (uint8_t)(1 << PRUSART1))
00661 #endif
00662
00663 #if defined(__AVR_HAVE_PRR0_PRVADC)
00664 #define power_vadc_enable() (PRR0 &= (uint8_t)~(1 << PRVADC))
00665 #define power_vadc_disable() (PRR0 |= (uint8_t)(1 << PRVADC))
00666 #endif
00667
00668 #if defined(__AVR_HAVE_PRR0_PRVM)
00669 #define power_voltage_monitor_enable() (PRR0 &= (uint8_t)~(1 << PRVM))
00670 #define power_voltage_monitor_disable() (PRR0 |= (uint8_t)(1 << PRVM))
00671 #endif
00672
00673 #if defined(__AVR_HAVE_PRR0_PRVRM)
00674 #define power_vrm_enable() (PRR0 &= (uint8_t)~(1 << PRVRM))
00675 #define power_vrm_disable() (PRR0 |= (uint8_t)(1 << PRVRM))
00676 #endif
00677
00678 #if defined(__AVR_HAVE_PRR1_PRAES)
00679 #define power_aes_enable() (PRR1 &= (uint8_t)~(1 << PRAES))
00680 #define power_aes_disable() (PRR1 |= (uint8_t)(1 << PRAES))
00681 #endif
00682
00683 #if defined(__AVR_HAVE_PRR1_PRCI)
00684 #define power_cinterface_enable() (PRR1 &= (uint8_t)~(1 << PRCI))
00685 #define power_cinterface_disable() (PRR1 |= (uint8_t)(1 << PRCI))
00686 #endif
00687
00688 #if defined(__AVR_HAVE_PRR1_PRHSSPI)
00689 #define power_hsspi_enable() (PRR1 &= (uint8_t)~(1 << PRHSSPI))
00690 #define power_hsspi_disable() (PRR1 |= (uint8_t)(1 << PRHSSPI))
00691 #endif
00692
```

```
00693 #if defined(__AVR_HAVE_PRR1_PRKB)
00694 #define power_kb_enable() (PRR1 &= (uint8_t)~(1 << PRKB))
00695 #define power_kb_disable() (PRR1 |= (uint8_t)(1 << PRKB))
00696 #endif
00697
00698 #if defined(__AVR_HAVE_PRR1_PRLFPH)
00699 #define power_lfph_enable() (PRR1 &= (uint8_t)~(1 << PRLFPH))
00700 #define power_lfph_disable() (PRR1 |= (uint8_t)(1 << PRLFPH))
00701 #endif
00702
00703 #if defined(__AVR_HAVE_PRR1_PRLFR)
00704 #define power_lfreceiver_enable() (PRR1 &= (uint8_t)~(1 << PRLFR))
00705 #define power_lfreceiver_disable() (PRR1 |= (uint8_t)(1 << PRLFR))
00706 #endif
00707
00708 #if defined(__AVR_HAVE_PRR1_PRLFTP)
00709 #define power_lftp_enable() (PRR1 &= (uint8_t)~(1 << PRLFTP))
00710 #define power_lftp_disable() (PRR1 |= (uint8_t)(1 << PRLFTP))
00711 #endif
00712
00713 #if defined(__AVR_HAVE_PRR1_PRSCI)
00714 #define power_sci_enable() (PRR1 &= (uint8_t)~(1 << PRSCI))
00715 #define power_sci_disable() (PRR1 |= (uint8_t)(1 << PRSCI))
00716 #endif
00717
00718 #if defined(__AVR_HAVE_PRR1_PRSPI)
00719 #define power_spi_enable() (PRR1 &= (uint8_t)~(1 << PRSPI))
00720 #define power_spi_disable() (PRR1 |= (uint8_t)(1 << PRSPI))
00721 #endif
00722
00723 #if defined(__AVR_HAVE_PRR1_PRT1)
00724 #define power_timer1_enable() (PRR1 &= (uint8_t)~(1 << PRT1))
00725 #define power_timer1_disable() (PRR1 |= (uint8_t)(1 << PRT1))
00726 #endif
00727
00728 #if defined(__AVR_HAVE_PRR1_PRT2)
00729 #define power_timer2_enable() (PRR1 &= (uint8_t)~(1 << PRT2))
00730 #define power_timer2_disable() (PRR1 |= (uint8_t)(1 << PRT2))
00731 #endif
00732
00733 #if defined(__AVR_HAVE_PRR1_PRT3)
00734 #define power_timer3_enable() (PRR1 &= (uint8_t)~(1 << PRT3))
00735 #define power_timer3_disable() (PRR1 |= (uint8_t)(1 << PRT3))
00736 #endif
00737
00738 #if defined(__AVR_HAVE_PRR1_PRT4)
00739 #define power_timer4_enable() (PRR1 &= (uint8_t)~(1 << PRT4))
00740 #define power_timer4_disable() (PRR1 |= (uint8_t)(1 << PRT4))
00741 #endif
00742
00743 #if defined(__AVR_HAVE_PRR1_PRT5)
00744 #define power_timer5_enable() (PRR1 &= (uint8_t)~(1 << PRT5))
00745 #define power_timer5_disable() (PRR1 |= (uint8_t)(1 << PRT5))
00746 #endif
00747
00748 #if defined(__AVR_HAVE_PRR1_PRTIM3)
00749 #define power_timer3_enable() (PRR1 &= (uint8_t)~(1 << PRTIM3))
00750 #define power_timer3_disable() (PRR1 |= (uint8_t)(1 << PRTIM3))
00751 #endif
00752
00753 #if defined(__AVR_HAVE_PRR1_PRTIM4)
00754 #define power_timer4_enable() (PRR1 &= (uint8_t)~(1 << PRTIM4))
00755 #define power_timer4_disable() (PRR1 |= (uint8_t)(1 << PRTIM4))
00756 #endif
00757
00758 #if defined(__AVR_HAVE_PRR1_PRTIM5)
00759 #define power_timer5_enable() (PRR1 &= (uint8_t)~(1 << PRTIM5))
00760 #define power_timer5_disable() (PRR1 |= (uint8_t)(1 << PRTIM5))
00761 #endif
00762
00763 #if defined(__AVR_HAVE_PRR1_PRTRX24)
00764 #define power_transceiver_enable() (PRR1 &= (uint8_t)~(1 << PRTRX24))
00765 #define power_transceiver_disable() (PRR1 |= (uint8_t)(1 << PRTRX24))
00766 #endif
00767
00768 #if defined(__AVR_HAVE_PRR1_PRUSART1)
00769 #define power_usart1_enable() (PRR1 &= (uint8_t)~(1 << PRUSART1))
00770 #define power_usart1_disable() (PRR1 |= (uint8_t)(1 << PRUSART1))
00771 #endif
00772
00773 #if defined(__AVR_HAVE_PRR1_PRUSART2)
00774 #define power_usart2_enable() (PRR1 &= (uint8_t)~(1 << PRUSART2))
00775 #define power_usart2_disable() (PRR1 |= (uint8_t)(1 << PRUSART2))
00776 #endif
00777
00778 #if defined(__AVR_HAVE_PRR1_PRUSART3)
00779 #define power_usart3_enable() (PRR1 &= (uint8_t)~(1 << PRUSART3))
```

```
00780 #define power_usart3_disable() (PRR1 |= (uint8_t)(1 << PRUSART3))
00781 #endif
00782
00783 #if defined(__AVR_HAVE_PRR1_PRUSB)
00784 #define power_usb_enable() (PRR1 &= (uint8_t)~(1 << PRUSB))
00785 #define power_usb_disable() (PRR1 |= (uint8_t)(1 << PRUSB))
00786 #endif
00787
00788 #if defined(__AVR_HAVE_PRR1_PRUSBH)
00789 #define power_usbh_enable() (PRR1 &= (uint8_t)~(1 << PRUSBH))
00790 #define power_usbh_disable() (PRR1 |= (uint8_t)(1 << PRUSBH))
00791 #endif
00792
00793 #if defined(__AVR_HAVE_PRR2_PRDF)
00794 #define power_data_fifo_enable() (PRR2 &= (uint8_t)~(1 << PRDF))
00795 #define power_data_fifo_disable() (PRR2 |= (uint8_t)(1 << PRDF))
00796 #endif
00797
00798 #if defined(__AVR_HAVE_PRR2_PRIDS)
00799 #define power_id_scan_enable() (PRR2 &= (uint8_t)~(1 << PRIDS))
00800 #define power_id_scan_disable() (PRR2 |= (uint8_t)(1 << PRIDS))
00801 #endif
00802
00803 #if defined(__AVR_HAVE_PRR2_PRRAM0)
00804 #define power_ram0_enable() (PRR2 &= (uint8_t)~(1 << PRRAM0))
00805 #define power_ram0_disable() (PRR2 |= (uint8_t)(1 << PRRAM0))
00806 #endif
00807
00808 #if defined(__AVR_HAVE_PRR2_PRRAM1)
00809 #define power_ram1_enable() (PRR2 &= (uint8_t)~(1 << PRRAM1))
00810 #define power_ram1_disable() (PRR2 |= (uint8_t)(1 << PRRAM1))
00811 #endif
00812
00813 #if defined(__AVR_HAVE_PRR2_PRRAM2)
00814 #define power_ram2_enable() (PRR2 &= (uint8_t)~(1 << PRRAM2))
00815 #define power_ram2_disable() (PRR2 |= (uint8_t)(1 << PRRAM2))
00816 #endif
00817
00818 #if defined(__AVR_HAVE_PRR2_PRRAM3)
00819 #define power_ram3_enable() (PRR2 &= (uint8_t)~(1 << PRRAM3))
00820 #define power_ram3_disable() (PRR2 |= (uint8_t)(1 << PRRAM3))
00821 #endif
00822
00823 #if defined(__AVR_HAVE_PRR2_PRRS)
00824 #define power_rssi_buffer_enable() (PRR2 &= (uint8_t)~(1 << PRRS))
00825 #define power_rssi_buffer_disable() (PRR2 |= (uint8_t)(1 << PRRS))
00826 #endif
00827
00828 #if defined(__AVR_HAVE_PRR2_PRSF)
00829 #define power_preamble_rssi_fifo_enable() (PRR2 &= (uint8_t)~(1 << PRSF))
00830 #define power_preamble_rssi_fifo_disable() (PRR2 |= (uint8_t)(1 << PRSF))
00831 #endif
00832
00833 #if defined(__AVR_HAVE_PRR2_PRSPI2)
00834 #define power_spi2_enable() (PRR2 &= (uint8_t)~(1 << PRSPI2))
00835 #define power_spi2_disable() (PRR2 |= (uint8_t)(1 << PRSPI2))
00836 #endif
00837
00838 #if defined(__AVR_HAVE_PRR2_PRSSM)
00839 #define power_sequencer_state_machine_enable() (PRR2 &= (uint8_t)~(1 << PRSSM))
00840 #define power_sequencer_state_machine_disable() (PRR2 |= (uint8_t)(1 << PRSSM))
00841 #endif
00842
00843 #if defined(__AVR_HAVE_PRR2_PRTM)
00844 #define power_tx_modulator_enable() (PRR2 &= (uint8_t)~(1 << PRTM))
00845 #define power_tx_modulator_disable() (PRR2 |= (uint8_t)(1 << PRTM))
00846 #endif
00847
00848 #if defined(__AVR_HAVE_PRR2_PRTWI2)
00849 #define power_twi2_enable() (PRR2 &= (uint8_t)~(1 << PRTWI2))
00850 #define power_twi2_disable() (PRR2 |= (uint8_t)(1 << PRTWI2))
00851 #endif
00852
00853 #if defined(__AVR_HAVE_PRR2_PRXA)
00854 #define power_rx_buffer_A_enable() (PRR2 &= (uint8_t)~(1 << PRXA))
00855 #define power_rx_buffer_A_disable() (PRR2 |= (uint8_t)(1 << PRXA))
00856 #endif
00857
00858 #if defined(__AVR_HAVE_PRR2_PRXB)
00859 #define power_rx_buffer_B_enable() (PRR2 &= (uint8_t)~(1 << PRXB))
00860 #define power_rx_buffer_B_disable() (PRR2 |= (uint8_t)(1 << PRXB))
00861 #endif
00862
00863 #if defined(__AVR_HAVE_PRGEN_AES)
00864 #define power_aes_enable() (PR_PRGEN &= (uint8_t)~(PR_AES_bm))
00865 #define power_aes_disable() (PR_PRGEN |= (uint8_t)PR_AES_bm)
00866 #endif
```

```
00867
00868 #if defined(__AVR_HAVE_PRGEN_DMA)
00869 #define power_dma_enable() (PR_PRGEN &= (uint8_t)~(PR_DMA_bm))
00870 #define power_dma_disable() (PR_PRGEN |= (uint8_t)PR_DMA_bm)
00871 #endif
00872
00873 #if defined(__AVR_HAVE_PRGEN_EBI)
00874 #define power_ebi_enable() (PR_PRGEN &= (uint8_t)~(PR_EBI_bm))
00875 #define power_ebi_disable() (PR_PRGEN |= (uint8_t)PR_EBI_bm)
00876 #endif
00877
00878 #if defined(__AVR_HAVE_PRGEN_EDMA)
00879 #define power_edma_enable() (PR_PRGEN &= (uint8_t)~(PR_EDMA_bm))
00880 #define power_edma_disable() (PR_PRGEN |= (uint8_t)PR_EDMA_bm)
00881 #endif
00882
00883 #if defined(__AVR_HAVE_PRGEN_EVSYS)
00884 #define power_evsys_enable() (PR_PRGEN &= (uint8_t)~(PR_EVSYS_bm))
00885 #define power_evsys_disable() (PR_PRGEN |= (uint8_t)PR_EVSYS_bm)
00886 #endif
00887
00888 #if defined(__AVR_HAVE_PRGEN_LCD)
00889 #define power_lcd_enable() (PR_PRGEN &= (uint8_t)~(PR_LCD_bm))
00890 #define power_lcd_disable() (PR_PRGEN |= (uint8_t)PR_LCD_bm)
00891 #endif
00892
00893 #if defined(__AVR_HAVE_PRGEN_RTC)
00894 #define power_rtc_enable() (PR_PRGEN &= (uint8_t)~(PR_RTC_bm))
00895 #define power_rtc_disable() (PR_PRGEN |= (uint8_t)PR_RTC_bm)
00896 #endif
00897
00898 #if defined(__AVR_HAVE_PRGEN_USB)
00899 #define power_usb_enable() (PR_PRGEN &= (uint8_t)~(PR_USB_bm))
00900 #define power_usb_disable() (PR_PRGEN &= (uint8_t)PR_USB_bm)
00901 #endif
00902
00903 #if defined(__AVR_HAVE_PRGEN_XCL)
00904 #define power_xcl_enable() (PR_PRGEN &= (uint8_t)~(PR_XCL_bm))
00905 #define power_xcl_disable() (PR_PRGEN |= (uint8_t)PR_XCL_bm)
00906 #endif
00907
00908 #if defined(__AVR_HAVE_PRPA_AC)
00909 #define power_aca_enable() (PR_PRPA &= (uint8_t)~(PR_AC_bm))
00910 #define power_aca_disable() (PR_PRPA |= (uint8_t)PR_AC_bm)
00911 #endif
00912
00913 #if defined(__AVR_HAVE_PRPA_ADC)
00914 #define power_adca_enable() (PR_PRPA &= (uint8_t)~(PR_ADC_bm))
00915 #define power_adca_disable() (PR_PRPA |= (uint8_t)PR_ADC_bm)
00916 #endif
00917
00918 #if defined(__AVR_HAVE_PRPA_DAC)
00919 #define power_daca_enable() (PR_PRPA &= (uint8_t)~(PR_DAC_bm))
00920 #define power_daca_disable() (PR_PRPA |= (uint8_t)PR_DAC_bm)
00921 #endif
00922
00923 #if defined(__AVR_HAVE_PRPB_AC)
00924 #define power_acb_enable() (PR_PRPB &= (uint8_t)~(PR_AC_bm))
00925 #define power_acb_disable() (PR_PRPB |= (uint8_t)PR_AC_bm)
00926 #endif
00927
00928 #if defined(__AVR_HAVE_PRPB_ADC)
00929 #define power_adcb_enable() (PR_PRPB &= (uint8_t)~(PR_ADC_bm))
00930 #define power_adcb_disable() (PR_PRPB |= (uint8_t)PR_ADC_bm)
00931 #endif
00932
00933 #if defined(__AVR_HAVE_PRPB_DAC)
00934 #define power_dacb_enable() (PR_PRPB &= (uint8_t)~(PR_DAC_bm))
00935 #define power_dacb_disable() (PR_PRPB |= (uint8_t)PR_DAC_bm)
00936 #endif
00937
00938 #if defined(__AVR_HAVE_PRPC_HIRES)
00939 #define power_hiresc_enable() (PR_PRPC &= (uint8_t)~(PR_HIRES_bm))
00940 #define power_hiresc_disable() (PR_PRPC |= (uint8_t)PR_HIRES_bm)
00941 #endif
00942
00943 #if defined(__AVR_HAVE_PRPC_SPI)
00944 #define power_spic_enable() (PR_PRPC &= (uint8_t)~(PR_SPI_bm))
00945 #define power_spic_disable() (PR_PRPC |= (uint8_t)PR_SPI_bm)
00946 #endif
00947
00948 #if defined(__AVR_HAVE_PRPC_TC0)
00949 #define power_tc0c_enable() (PR_PRPC &= (uint8_t)~(PR_TC0_bm))
00950 #define power_tc0c_disable() (PR_PRPC |= (uint8_t)PR_TC0_bm)
00951 #endif
00952
00953 #if defined(__AVR_HAVE_PRPC_TC1)
```



```
00954 #define power_tclc_enable() (PR_PRPC &= (uint8_t)~(PR_TC1_bm))
00955 #define power_tclc_disable() (PR_PRPC |= (uint8_t)PR_TC1_bm)
00956 #endif
00957
00958 #if defined(__AVR_HAVE_PRPC_TC4)
00959 #define power_tc4c_enable() (PR_PRPC &= (uint8_t)~(PR_TC4_bm))
00960 #define power_tc4c_disable() (PR_PRPC |= (uint8_t)PR_TC4_bm)
00961 #endif
00962
00963 #if defined(__AVR_HAVE_PRPC_TC5)
00964 #define power_tc5c_enable() (PR_PRPC &= (uint8_t)~(PR_TC5_bm))
00965 #define power_tc5c_disable() (PR_PRPC |= (uint8_t)PR_TC5_bm)
00966 #endif
00967
00968 #if defined(__AVR_HAVE_PRPC_TWI)
00969 #define power_twic_enable() (PR_PRPC &= (uint8_t)~(PR_TWI_bm))
00970 #define power_twic_disable() (PR_PRPC |= (uint8_t)PR_TWI_bm)
00971 #endif
00972
00973 #if defined(__AVR_HAVE_PRPC_USART0)
00974 #define power_usartc0_enable() (PR_PRPC &= (uint8_t)~(PR_USART0_bm))
00975 #define power_usartc0_disable() (PR_PRPC |= (uint8_t)PR_USART0_bm)
00976 #endif
00977
00978 #if defined(__AVR_HAVE_PRPC_USART1)
00979 #define power_usartc1_enable() (PR_PRPC &= (uint8_t)~(PR_USART1_bm))
00980 #define power_usartc1_disable() (PR_PRPC |= (uint8_t)PR_USART1_bm)
00981 #endif
00982
00983 #if defined(__AVR_HAVE_PRPD_HIRES)
00984 #define power_hiresd_enable() (PR_PRPD &= (uint8_t)~(PR_HIRES_bm))
00985 #define power_hiresd_disable() (PR_PRPD |= (uint8_t)PR_HIRES_bm)
00986 #endif
00987
00988 #if defined(__AVR_HAVE_PRPD_SPI)
00989 #define power_spid_enable() (PR_PRPD &= (uint8_t)~(PR_SPI_bm))
00990 #define power_spid_disable() (PR_PRPD |= (uint8_t)PR_SPI_bm)
00991 #endif
00992
00993 #if defined(__AVR_HAVE_PRPD_TC0)
00994 #define power_tc0d_enable() (PR_PRPD &= (uint8_t)~(PR_TC0_bm))
00995 #define power_tc0d_disable() (PR_PRPD |= (uint8_t)PR_TC0_bm)
00996 #endif
00997
00998 #if defined(__AVR_HAVE_PRPD_TC1)
00999 #define power_tc1d_enable() (PR_PRPD &= (uint8_t)~(PR_TC1_bm))
01000 #define power_tc1d_disable() (PR_PRPD |= (uint8_t)PR_TC1_bm)
01001 #endif
01002
01003 #if defined(__AVR_HAVE_PRPD_TC5)
01004 #define power_tc5d_enable() (PR_PRPD &= (uint8_t)~(PR_TC5_bm))
01005 #define power_tc5d_disable() (PR_PRPD |= (uint8_t)PR_TC5_bm)
01006 #endif
01007
01008 #if defined(__AVR_HAVE_PRPD_TWI)
01009 #define power_twid_enable() (PR_PRPD &= (uint8_t)~(PR_TWI_bm))
01010 #define power_twid_disable() (PR_PRPD |= (uint8_t)PR_TWI_bm)
01011 #endif
01012
01013 #if defined(__AVR_HAVE_PRPD_USART0)
01014 #define power_usartd0_enable() (PR_PRPD &= (uint8_t)~(PR_USART0_bm))
01015 #define power_usartd0_disable() (PR_PRPD |= (uint8_t)PR_USART0_bm)
01016 #endif
01017
01018 #if defined(__AVR_HAVE_PRPD_USART1)
01019 #define power_usartd1_enable() (PR_PRPD &= (uint8_t)~(PR_USART1_bm))
01020 #define power_usartd1_disable() (PR_PRPD |= (uint8_t)PR_USART1_bm)
01021 #endif
01022
01023 #if defined(__AVR_HAVE_PRPE_HIRES)
01024 #define power_hirese_enable() (PR_PRPE &= (uint8_t)~(PR_HIRES_bm))
01025 #define power_hirese_disable() (PR_PRPE |= (uint8_t)PR_HIRES_bm)
01026 #endif
01027
01028 #if defined(__AVR_HAVE_PRPE_SPI)
01029 #define power_spie_enable() (PR_PRPE &= (uint8_t)~(PR_SPI_bm))
01030 #define power_spie_disable() (PR_PRPE |= (uint8_t)PR_SPI_bm)
01031 #endif
01032
01033 #if defined(__AVR_HAVE_PRPE_TC0)
01034 #define power_tc0e_enable() (PR_PRPE &= (uint8_t)~(PR_TC0_bm))
01035 #define power_tc0e_disable() (PR_PRPE |= (uint8_t)PR_TC0_bm)
01036 #endif
01037
01038 #if defined(__AVR_HAVE_PRPE_TC1)
01039 #define power_tc1e_enable() (PR_PRPE &= (uint8_t)~(PR_TC1_bm))
01040 #define power_tc1e_disable() (PR_PRPE |= (uint8_t)PR_TC1_bm)
```

```

01041 #endif
01042
01043 #if defined(__AVR_HAVE_PRPE_TWI)
01044 #define power_twie_enable() (PR_PRPE &= (uint8_t)~(PR_TWI_bm))
01045 #define power_twie_disable() (PR_PRPE |= (uint8_t)PR_TWI_bm)
01046 #endif
01047
01048 #if defined(__AVR_HAVE_PRPE_USART0)
01049 #define power_usarte0_enable() (PR_PRPE &= (uint8_t)~(PR_USART0_bm))
01050 #define power_usarte0_disable() (PR_PRPE |= (uint8_t)PR_USART0_bm)
01051 #endif
01052
01053 #if defined(__AVR_HAVE_PRPE_USART1)
01054 #define power_usarte1_enable() (PR_PRPE &= (uint8_t)~(PR_USART1_bm))
01055 #define power_usarte1_disable() (PR_PRPE |= (uint8_t)PR_USART1_bm)
01056 #endif
01057
01058 #if defined(__AVR_HAVE_PRPF_HIRES)
01059 #define power_hiresf_enable() (PR_PRPF &= (uint8_t)~(PR_HIRES_bm))
01060 #define power_hiresf_disable() (PR_PRPF |= (uint8_t)PR_HIRES_bm)
01061 #endif
01062
01063 #if defined(__AVR_HAVE_PRPF_SPI)
01064 #define power_spif_enable() (PR_PRPF &= (uint8_t)~(PR_SPI_bm))
01065 #define power_spif_disable() (PR_PRPF |= (uint8_t)PR_SPI_bm)
01066 #endif
01067
01068 #if defined(__AVR_HAVE_PRPF_TC0)
01069 #define power_tc0f_enable() (PR_PRPF &= (uint8_t)~(PR_TC0_bm))
01070 #define power_tc0f_disable() (PR_PRPF |= (uint8_t)PR_TC0_bm)
01071 #endif
01072
01073 #if defined(__AVR_HAVE_PRPF_TC1)
01074 #define power_tclf_enable() (PR_PRPF &= (uint8_t)~(PR_TC1_bm))
01075 #define power_tclf_disable() (PR_PRPF |= (uint8_t)PR_TC1_bm)
01076 #endif
01077
01078 #if defined(__AVR_HAVE_PRPF_TWI)
01079 #define power_twif_enable() (PR_PRPF &= (uint8_t)~(PR_TWI_bm))
01080 #define power_twif_disable() (PR_PRPF |= (uint8_t)PR_TWI_bm)
01081 #endif
01082
01083 #if defined(__AVR_HAVE_PRPF_USART0)
01084 #define power_usartf0_enable() (PR_PRPF &= (uint8_t)~(PR_USART0_bm))
01085 #define power_usartf0_disable() (PR_PRPF |= (uint8_t)PR_USART0_bm)
01086 #endif
01087
01088 #if defined(__AVR_HAVE_PRPF_USART1)
01089 #define power_usartf1_enable() (PR_PRPF &= (uint8_t)~(PR_USART1_bm))
01090 #define power_usartf1_disable() (PR_PRPF |= (uint8_t)PR_USART1_bm)
01091 #endif
01092
01093 #ifdef __DOXYGEN__
01094 /**
01095  \ingroup avr_power
01096  \fn void power_all_enable()
01097  Enable all modules.
01098  */
01099 static __ATTR_ALWAYS_INLINE__ void power_all_enable();
01100 #else
01101 static __ATTR_ALWAYS_INLINE__ void __power_all_enable()
01102 {
01103 #ifdef __AVR_HAVE_PRR
01104     PRR &= (uint8_t)~(__AVR_HAVE_PRR);
01105 #endif
01106
01107 #ifdef __AVR_HAVE_PRR0
01108     PRR0 &= (uint8_t)~(__AVR_HAVE_PRR0);
01109 #endif
01110
01111 #ifdef __AVR_HAVE_PRR1
01112     PRR1 &= (uint8_t)~(__AVR_HAVE_PRR1);
01113 #endif
01114
01115 #ifdef __AVR_HAVE_PRR2
01116     PRR2 &= (uint8_t)~(__AVR_HAVE_PRR2);
01117 #endif
01118
01119 #ifdef __AVR_HAVE_PRGEN
01120     PR_PRGEN &= (uint8_t)~(__AVR_HAVE_PRGEN);
01121 #endif
01122
01123 #ifdef __AVR_HAVE_PRPA
01124     PR_PRPA &= (uint8_t)~(__AVR_HAVE_PRPA);
01125 #endif
01126
01127 #ifdef __AVR_HAVE_PRPB

```

```
01128     PR_PRPB &= (uint8_t)~(__AVR_HAVE_PRPB);
01129 #endif
01130
01131 #ifdef __AVR_HAVE_PRPC
01132     PR_PRPC &= (uint8_t)~(__AVR_HAVE_PRPC);
01133 #endif
01134
01135 #ifdef __AVR_HAVE_PRPD
01136     PR_PRPD &= (uint8_t)~(__AVR_HAVE_PRPD);
01137 #endif
01138
01139 #ifdef __AVR_HAVE_PRPE
01140     PR_PRPE &= (uint8_t)~(__AVR_HAVE_PRPE);
01141 #endif
01142
01143 #ifdef __AVR_HAVE_PRPF
01144     PR_PRPF &= (uint8_t)~(__AVR_HAVE_PRPF);
01145 #endif
01146 }
01147 #endif /* __DOXYGEN__ */
01148
01149 #ifdef __DOXYGEN__
01150 /**
01151  \ingroup avr_power
01152  \fn void power_all_disable()
01153  Disable all modules.
01154  */
01155 static __ATTR_ALWAYS_INLINE__ void power_all_disable();
01156 #else
01157 static __ATTR_ALWAYS_INLINE__ void __power_all_disable()
01158 {
01159 #ifdef __AVR_HAVE_PRR
01160     PRR |= (uint8_t)(__AVR_HAVE_PRR);
01161 #endif
01162
01163 #ifdef __AVR_HAVE_PRR0
01164     PRR0 |= (uint8_t)(__AVR_HAVE_PRR0);
01165 #endif
01166
01167 #ifdef __AVR_HAVE_PRR1
01168     PRR1 |= (uint8_t)(__AVR_HAVE_PRR1);
01169 #endif
01170
01171 #ifdef __AVR_HAVE_PRR2
01172     PRR2 |= (uint8_t)(__AVR_HAVE_PRR2);
01173 #endif
01174
01175 #ifdef __AVR_HAVE_PRGEN
01176     PR_PRGEN |= (uint8_t)(__AVR_HAVE_PRGEN);
01177 #endif
01178
01179 #ifdef __AVR_HAVE_PRPA
01180     PR_PRPA |= (uint8_t)(__AVR_HAVE_PRPA);
01181 #endif
01182
01183 #ifdef __AVR_HAVE_PRPB
01184     PR_PRPB |= (uint8_t)(__AVR_HAVE_PRPB);
01185 #endif
01186
01187 #ifdef __AVR_HAVE_PRPC
01188     PR_PRPC |= (uint8_t)(__AVR_HAVE_PRPC);
01189 #endif
01190
01191 #ifdef __AVR_HAVE_PRPD
01192     PR_PRPD |= (uint8_t)(__AVR_HAVE_PRPD);
01193 #endif
01194
01195 #ifdef __AVR_HAVE_PRPE
01196     PR_PRPE |= (uint8_t)(__AVR_HAVE_PRPE);
01197 #endif
01198
01199 #ifdef __AVR_HAVE_PRPF
01200     PR_PRPF |= (uint8_t)(__AVR_HAVE_PRPF);
01201 #endif
01202 }
01203 #endif /* __DOXYGEN__ */
01204
01205 #ifndef __DOXYGEN__
01206 #ifndef power_all_enable
01207 #define power_all_enable() __power_all_enable()
01208 #endif
01209
01210 #ifndef power_all_disable
01211 #define power_all_disable() __power_all_disable()
01212 #endif
01213 #endif /* !__DOXYGEN__ */
01214
```

```
01215
01216 #if defined(__DOXYGEN__) \
01217 || defined(__AVR_AT90CAN32__) \
01218 || defined(__AVR_AT90CAN64__) \
01219 || defined(__AVR_AT90CAN128__) \
01220 || defined(__AVR_AT90PWM1__) \
01221 || defined(__AVR_AT90PWM2__) \
01222 || defined(__AVR_AT90PWM2B__) \
01223 || defined(__AVR_AT90PWM3__) \
01224 || defined(__AVR_AT90PWM3B__) \
01225 || defined(__AVR_AT90PWM81__) \
01226 || defined(__AVR_AT90PWM161__) \
01227 || defined(__AVR_AT90PWM216__) \
01228 || defined(__AVR_AT90PWM316__) \
01229 || defined(__AVR_AT90SCR100__) \
01230 || defined(__AVR_AT90USB646__) \
01231 || defined(__AVR_AT90USB647__) \
01232 || defined(__AVR_AT90USB82__) \
01233 || defined(__AVR_AT90USB1286__) \
01234 || defined(__AVR_AT90USB1287__) \
01235 || defined(__AVR_AT90USB162__) \
01236 || defined(__AVR_ATA5505__) \
01237 || defined(__AVR_ATA5272__) \
01238 || defined(__AVR_ATmega1280__) \
01239 || defined(__AVR_ATmega1281__) \
01240 || defined(__AVR_ATmega1284__) \
01241 || defined(__AVR_ATmega128RFA1__) \
01242 || defined(__AVR_ATmega1284RFR2__) \
01243 || defined(__AVR_ATmega128RFR2__) \
01244 || defined(__AVR_ATmega1284P__) \
01245 || defined(__AVR_ATmega162__) \
01246 || defined(__AVR_ATmega164A__) \
01247 || defined(__AVR_ATmega164P__) \
01248 || defined(__AVR_ATmega164PA__) \
01249 || defined(__AVR_ATmega165__) \
01250 || defined(__AVR_ATmega165A__) \
01251 || defined(__AVR_ATmega165P__) \
01252 || defined(__AVR_ATmega165PA__) \
01253 || defined(__AVR_ATmega168__) \
01254 || defined(__AVR_ATmega168P__) \
01255 || defined(__AVR_ATmega168A__) \
01256 || defined(__AVR_ATmega168PA__) \
01257 || defined(__AVR_ATmega168PB__) \
01258 || defined(__AVR_ATmega169__) \
01259 || defined(__AVR_ATmega169A__) \
01260 || defined(__AVR_ATmega169P__) \
01261 || defined(__AVR_ATmega169PA__) \
01262 || defined(__AVR_ATmega16M1__) \
01263 || defined(__AVR_ATmega16U2__) \
01264 || defined(__AVR_ATmega16U4__) \
01265 || defined(__AVR_ATmega2560__) \
01266 || defined(__AVR_ATmega2561__) \
01267 || defined(__AVR_ATmega2564RFR2__) \
01268 || defined(__AVR_ATmega256RFR2__) \
01269 || defined(__AVR_ATmega324A__) \
01270 || defined(__AVR_ATmega324P__) \
01271 || defined(__AVR_ATmega324PA__) \
01272 || defined(__AVR_ATmega324PB__) \
01273 || defined(__AVR_ATmega325__) \
01274 || defined(__AVR_ATmega325A__) \
01275 || defined(__AVR_ATmega325PA__) \
01276 || defined(__AVR_ATmega3250__) \
01277 || defined(__AVR_ATmega3250A__) \
01278 || defined(__AVR_ATmega3250PA__) \
01279 || defined(__AVR_ATmega328__) \
01280 || defined(__AVR_ATmega328P__) \
01281 || defined(__AVR_ATmega328PB__) \
01282 || defined(__AVR_ATmega329__) \
01283 || defined(__AVR_ATmega329A__) \
01284 || defined(__AVR_ATmega329P__) \
01285 || defined(__AVR_ATmega329PA__) \
01286 || defined(__AVR_ATmega3290__) \
01287 || defined(__AVR_ATmega3290A__) \
01288 || defined(__AVR_ATmega3290P__) \
01289 || defined(__AVR_ATmega3290PA__) \
01290 || defined(__AVR_ATmega32C1__) \
01291 || defined(__AVR_ATmega32M1__) \
01292 || defined(__AVR_ATmega32U2__) \
01293 || defined(__AVR_ATmega32U4__) \
01294 || defined(__AVR_ATmega32U6__) \
01295 || defined(__AVR_ATmega48__) \
01296 || defined(__AVR_ATmega48A__) \
01297 || defined(__AVR_ATmega48PA__) \
01298 || defined(__AVR_ATmega48P__) \
01299 || defined(__AVR_ATmega640__) \
01300 || defined(__AVR_ATmega649P__) \
01301 || defined(__AVR_ATmega644__) \
```

```

01302 || defined(__AVR_ATmega644A__) \
01303 || defined(__AVR_ATmega644P__) \
01304 || defined(__AVR_ATmega644PA__) \
01305 || defined(__AVR_ATmega645__) \
01306 || defined(__AVR_ATmega645A__) \
01307 || defined(__AVR_ATmega645P__) \
01308 || defined(__AVR_ATmega6450__) \
01309 || defined(__AVR_ATmega6450A__) \
01310 || defined(__AVR_ATmega6450P__) \
01311 || defined(__AVR_ATmega649__) \
01312 || defined(__AVR_ATmega649A__) \
01313 || defined(__AVR_ATmega64M1__) \
01314 || defined(__AVR_ATmega64C1__) \
01315 || defined(__AVR_ATmega6490__) \
01316 || defined(__AVR_ATmega6490A__) \
01317 || defined(__AVR_ATmega6490P__) \
01318 || defined(__AVR_ATmega644RFR2__) \
01319 || defined(__AVR_ATmega64RFR2__) \
01320 || defined(__AVR_ATmega88__) \
01321 || defined(__AVR_ATmega88A__) \
01322 || defined(__AVR_ATmega88P__) \
01323 || defined(__AVR_ATmega88PA__) \
01324 || defined(__AVR_ATmega8U2__) \
01325 || defined(__AVR_ATmega16U2__) \
01326 || defined(__AVR_ATmega32U2__) \
01327 || defined(__AVR_ATtiny48__) \
01328 || defined(__AVR_ATtiny88__) \
01329 || defined(__AVR_ATtiny87__) \
01330 || defined(__AVR_ATtiny167__)
01331
01332
01333 /** \addtogroup avr_power
01334
01335 Some of the newer AVR's contain a System Clock Prescale Register (CLKPR) that
01336 allows you to decrease the system clock frequency and the power consumption
01337 when the need for processing power is low.
01338 On some earlier AVR's (ATmega103, ATmega64, ATmega128), similar
01339 functionality can be achieved through the XTAL Divide Control Register.
01340 Below are two macros and an enumerated type that can be used to
01341 interface to the Clock Prescale Register or
01342 XTAL Divide Control Register.
01343
01344 \note Not all AVR devices have a clock prescaler. On those devices
01345 without a Clock Prescale Register or XTAL Divide Control Register, these
01346 macros are not available.
01347
01348 \code
01349 typedef enum
01350 {
01351     clock_div_1 = 0,
01352     clock_div_2 = 1,
01353     clock_div_4 = 2,
01354     clock_div_8 = 3,
01355     clock_div_16 = 4,
01356     clock_div_32 = 5,
01357     clock_div_64 = 6,
01358     clock_div_128 = 7,
01359     clock_div_256 = 8,
01360     clock_div_1_rc = 15, // ATmega128RFAl only
01361 } clock_div_t;
01362 \endcode
01363 Clock prescaler setting enumerations for device using
01364 System Clock Prescale Register.
01365
01366 \code
01367 typedef enum
01368 {
01369     clock_div_1 = 1,
01370     clock_div_2 = 2,
01371     clock_div_4 = 4,
01372     clock_div_8 = 8,
01373     clock_div_16 = 16,
01374     clock_div_32 = 32,
01375     clock_div_64 = 64,
01376     clock_div_128 = 128
01377 } clock_div_t;
01378 \endcode
01379 Clock prescaler setting enumerations for device using
01380 XTAL Divide Control Register.
01381
01382 */
01383 #ifndef __DOXYGEN__
01384 typedef enum
01385 {
01386     clock_div_1 = 0,
01387     clock_div_2 = 1,
01388     clock_div_4 = 2,

```

```

01389     clock_div_8 = 3,
01390     clock_div_16 = 4,
01391     clock_div_32 = 5,
01392     clock_div_64 = 6,
01393     clock_div_128 = 7,
01394     clock_div_256 = 8
01395 #if defined(__AVR_ATmega128RFA1__) \
01396 || defined(__AVR_ATmega2564RFR2__) \
01397 || defined(__AVR_ATmega1284RFR2__) \
01398 || defined(__AVR_ATmega644RFR2__) \
01399 || defined(__AVR_ATmega256RFR2__) \
01400 || defined(__AVR_ATmega128RFR2__) \
01401 || defined(__AVR_ATmega64RFR2__) \
01402     , clock_div_1_rc = 15
01403 #endif
01404 } clock_div_t;
01405
01406 static __ATTR_ALWAYS_INLINE__ void clock_prescale_set(clock_div_t);
01407 #endif /* !__DOXYGEN__ */
01408
01409 /**
01410  \ingroup avr_power
01411  \fn clock_prescale_set(clock_div_t x)
01412
01413  Set the clock prescaler register select bits, selecting a system clock
01414  division setting. This function is inlined, even if compiler
01415  optimizations are disabled.
01416
01417  The type of \c x is \c clock_div_t.
01418
01419  \note For device with XTAL Divide Control Register (XDIV), \c x can actually range
01420  from 1 to 129. Thus, one does not need to use \c clock_div_t type as argument.
01421  */
01422 void clock_prescale_set(clock_div_t __x)
01423 {
01424     uint8_t __tmp = _BV(CLKPCE);
01425     __asm__ __volatile__ (
01426         "in __tmp_reg__, __SREG__" "\n\t"
01427         "cli" "\n\t"
01428         "sts %1, %0" "\n\t"
01429         "sts %1, %2" "\n\t"
01430         "out __SREG__, __tmp_reg__"
01431         : /* no outputs */
01432         : "d" (__tmp),
01433         "M" (_SFR_MEM_ADDR(CLKPR)),
01434         "d" (__x)
01435         : "r0");
01436 }
01437
01438 /** \ingroup avr_power
01439  \def clock_prescale_get()
01440  Gets and returns the clock prescaler register setting. The return type is \c clock_div_t.
01441
01442  \note For device with XTAL Divide Control Register (XDIV), return can actually
01443  range from 1 to 129. Care should be taken has the return value could differ from the
01444  typedef enum clock_div_t. This should only happen if clock_prescale_set was previously
01445  called with a value other than those defined by \c clock_div_t.
01446  */
01447 #define clock_prescale_get() (clock_div_t)(CLKPR &
01448 (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))
01449
01449 #elif defined(__AVR_ATmega16HVB__) \
01450 || defined(__AVR_ATmega16HVBREVB__) \
01451 || defined(__AVR_ATmega32HVB__) \
01452 || defined(__AVR_ATmega32HVBREVB__)
01453
01454 typedef enum
01455 {
01456     clock_div_1 = 0,
01457     clock_div_2 = 1,
01458     clock_div_4 = 2,
01459     clock_div_8 = 3
01460 } clock_div_t;
01461
01462 static __ATTR_ALWAYS_INLINE__ void clock_prescale_set(clock_div_t);
01463
01464 void clock_prescale_set(clock_div_t __x)
01465 {
01466     uint8_t __tmp = _BV(CLKPCE);
01467     __asm__ __volatile__ (
01468         "in __tmp_reg__, __SREG__" "\n\t"
01469         "cli" "\n\t"
01470         "sts %1, %0" "\n\t"
01471         "sts %1, %2" "\n\t"
01472         "out __SREG__, __tmp_reg__"
01473         : /* no outputs */
01474         : "d" (__tmp),

```

```

01475         "M" (_SFR_MEM_ADDR(CLKPR)),
01476         "d" (__x)
01477     : "r0");
01478 }
01479
01480 #define clock_prescale_get() (clock_div_t)(CLKPR & (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)))
01481
01482 #elif defined(__AVR_ATmega5790__) \
01483 || defined (__AVR_ATmega5795__)
01484
01485 typedef enum
01486 {
01487     clock_div_1 = 0,
01488     clock_div_2 = 1,
01489     clock_div_4 = 2,
01490     clock_div_8 = 3,
01491     clock_div_16 = 4,
01492     clock_div_32 = 5,
01493     clock_div_64 = 6,
01494     clock_div_128 = 7,
01495 } clock_div_t;
01496
01497 static __ATTR_ALWAYS_INLINE__ void system_clock_prescale_set(clock_div_t);
01498
01499 void system_clock_prescale_set(clock_div_t __x)
01500 {
01501     uint8_t __tmp = _BV(CLKPCE);
01502     __asm__ __volatile__ (
01503         "in __tmp_reg__, __SREG__" "\n\t"
01504         "cli" "\n\t"
01505         "out %1, %0" "\n\t"
01506         "out %1, %2" "\n\t"
01507         "out __SREG__, __tmp_reg__"
01508         : /* no outputs */
01509         : "d" (__tmp),
01510           "I" (_SFR_IO_ADDR(CLKPR)),
01511           "d" (__x)
01512         : "r0");
01513 }
01514
01515 #define system_clock_prescale_get() (clock_div_t)(CLKPR &
01516 (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)))
01517
01518 typedef enum
01519 {
01520     timer_clock_div_reset = 0,
01521     timer_clock_div_1 = 1,
01522     timer_clock_div_2 = 2,
01523     timer_clock_div_4 = 3,
01524     timer_clock_div_8 = 4,
01525     timer_clock_div_16 = 5,
01526     timer_clock_div_32 = 6,
01527     timer_clock_div_64 = 7
01528 } timer_clock_div_t;
01529
01530 static __ATTR_ALWAYS_INLINE__ void timer_clock_prescale_set(timer_clock_div_t);
01531
01532 void timer_clock_prescale_set(timer_clock_div_t __x)
01533 {
01534     uint8_t __t;
01535     __asm__ __volatile__ (
01536         "in __tmp_reg__, __SREG__" "\n\t"
01537         "cli" "\n\t"
01538         "in %[temp],[clkpr]" "\n\t"
01539         "out %[clkpr],[enable]" "\n\t"
01540         "andi %[temp],[not_CLTPS]" "\n\t"
01541         "or %[temp],[set_value]" "\n\t"
01542         "out %[clkpr],[temp]" "\n\t"
01543         "out __SREG__, __tmp_reg__"
01544         : [temp] "=d" (__t)
01545         : [clkpr] "I" (_SFR_IO_ADDR(CLKPR)),
01546           [enable] "r" (_BV(CLKPCE)),
01547           [not_CLTPS] "M" (0xFF & ~( (1 << CLTPS2) | (1 << CLTPS1) | (1 << CLTPS0))),
01548           [set_value] "r" ((__x & 7) << 3)
01549         : "r0");
01550 }
01551
01552 #define timer_clock_prescale_get() (timer_clock_div_t)(CLKPR &
01553 (uint8_t)((1<<CLTPS0)|(1<<CLTPS1)|(1<<CLTPS2)))
01554
01555 #elif defined(__AVR_ATmega6285__) \
01556 || defined (__AVR_ATmega6286__)
01557
01558 typedef enum
01559 {
01560     clock_div_1 = 0,
01561     clock_div_2 = 1,

```

```

01560     clock_div_4 = 2,
01561     clock_div_8 = 3,
01562     clock_div_16 = 4,
01563     clock_div_32 = 5,
01564     clock_div_64 = 6,
01565     clock_div_128 = 7
01566 } clock_div_t;
01567
01568 static __ATTR_ALWAYS_INLINE__ void system_clock_prescale_set(clock_div_t);
01569
01570 void system_clock_prescale_set(clock_div_t __x)
01571 {
01572     uint8_t __t;
01573     __asm__ __volatile__ (
01574         "in __tmp_reg__, __SREG__"      "\n\t"
01575         "cli"                            "\n\t"
01576         "in %[temp], %[clpr]"           "\n\t"
01577         "out %[clpr], %[enable]"        "\n\t"
01578         "andi %[temp], %[not_CLKPS]"    "\n\t"
01579         "or %[temp], %[set_value]"      "\n\t"
01580         "out %[clpr], %[temp]"          "\n\t"
01581         "out __SREG__, __tmp_reg__"
01582         : [temp] "=d" (__t)
01583         : [clpr] "I" (_SFR_IO_ADDR(CLKPR)),
01584           [enable] "r" (_BV(CLPCPE)),
01585           [not_CLKPS] "M" (0xFF & (~ ((1 < CLKPS2) | (1 < CLKPS1) | (1 < CLKPS0)))),
01586           [set_value] "r" (__x & 7)
01587         : "r0");
01588 }
01589
01590 #define system_clock_prescale_get() (clock_div_t)(CLKPR &
01591 (uint8_t)((1<CLKPS0)|(1<CLKPS1)|(1<CLKPS2)))
01592
01592 typedef enum
01593 {
01594     timer_clock_div_reset = 0,
01595     timer_clock_div_1 = 1,
01596     timer_clock_div_2 = 2,
01597     timer_clock_div_4 = 3,
01598     timer_clock_div_8 = 4,
01599     timer_clock_div_16 = 5,
01600     timer_clock_div_32 = 6,
01601     timer_clock_div_64 = 7
01602 } timer_clock_div_t;
01603
01604 static __ATTR_ALWAYS_INLINE__ void timer_clock_prescale_set(timer_clock_div_t);
01605
01606 void timer_clock_prescale_set(timer_clock_div_t __x)
01607 {
01608     uint8_t __t;
01609     __asm__ __volatile__ (
01610         "in __tmp_reg__, __SREG__"      "\n\t"
01611         "cli"                            "\n\t"
01612         "in %[temp], %[clpr]"           "\n\t"
01613         "out %[clpr], %[enable]"        "\n\t"
01614         "andi %[temp], %[not_CLTPS]"    "\n\t"
01615         "or %[temp], %[set_value]"      "\n\t"
01616         "out %[clpr], %[temp]"          "\n\t"
01617         "out __SREG__, __tmp_reg__"
01618         : [temp] "=d" (__t)
01619         : [clpr] "I" (_SFR_IO_ADDR(CLKPR)),
01620           [enable] "r" (_BV(CLPCPE)),
01621           [not_CLTPS] "M" (0xFF & (~ ((1 < CLTPS2) | (1 < CLTPS1) | (1 < CLTPS0)))),
01622           [set_value] "r" ((__x & 7) < 3)
01623         : "r0");
01624 }
01625
01626 #define timer_clock_prescale_get() (timer_clock_div_t)(CLKPR &
01627 (uint8_t)((1<CLTPS0)|(1<CLTPS1)|(1<CLTPS2)))
01628
01628 #elif defined(__AVR_ATtiny24__) \
01629 || defined(__AVR_ATtiny24A__) \
01630 || defined(__AVR_ATtiny44__) \
01631 || defined(__AVR_ATtiny44A__) \
01632 || defined(__AVR_ATtiny84__) \
01633 || defined(__AVR_ATtiny84A__) \
01634 || defined(__AVR_ATtiny25__) \
01635 || defined(__AVR_ATtiny45__) \
01636 || defined(__AVR_ATtiny85__) \
01637 || defined(__AVR_ATtiny261A__) \
01638 || defined(__AVR_ATtiny261__) \
01639 || defined(__AVR_ATtiny461__) \
01640 || defined(__AVR_ATtiny461A__) \
01641 || defined(__AVR_ATtiny861__) \
01642 || defined(__AVR_ATtiny861A__) \
01643 || defined(__AVR_ATtiny2313__) \
01644 || defined(__AVR_ATtiny2313A__) \

```



```

01645 || defined(__AVR_ATtiny4313__) \
01646 || defined(__AVR_ATtiny13__) \
01647 || defined(__AVR_ATtiny13A__) \
01648 || defined(__AVR_ATtiny43U__) \
01649
01650 typedef enum
01651 {
01652     clock_div_1 = 0,
01653     clock_div_2 = 1,
01654     clock_div_4 = 2,
01655     clock_div_8 = 3,
01656     clock_div_16 = 4,
01657     clock_div_32 = 5,
01658     clock_div_64 = 6,
01659     clock_div_128 = 7,
01660     clock_div_256 = 8
01661 } clock_div_t;
01662
01663 static __ATTR_ALWAYS_INLINE__ void clock_prescale_set(clock_div_t);
01664
01665 void clock_prescale_set(clock_div_t __x)
01666 {
01667     uint8_t __tmp = _BV(CLKPCF);
01668     __asm__ __volatile__ (
01669         "in __tmp_reg__, __SREG__" "\n\t"
01670         "cli" "\n\t"
01671         "out %1, %0" "\n\t"
01672         "out %1, %2" "\n\t"
01673         "out __SREG__, __tmp_reg__"
01674         : /* no outputs */
01675         : "d" (__tmp),
01676         "I" (_SFR_IO_ADDR(CLKPR)),
01677         "d" (__x)
01678         : "r0");
01679 }
01680
01681
01682 #define clock_prescale_get() (clock_div_t) (CLKPR &
    (uint8_t) ((1<CLKPS0) | (1<CLKPS1) | (1<CLKPS2) | (1<CLKPS3)))
01683
01684 #elif defined(__AVR_ATtiny441__) \
01685 || defined(__AVR_ATtiny841__)
01686
01687 typedef enum
01688 {
01689     clock_div_1 = 0,
01690     clock_div_2 = 1,
01691     clock_div_4 = 2,
01692     clock_div_8 = 3,
01693     clock_div_16 = 4,
01694     clock_div_32 = 5,
01695     clock_div_64 = 6,
01696     clock_div_128 = 7,
01697     clock_div_256 = 8
01698 } clock_div_t;
01699
01700 static __ATTR_ALWAYS_INLINE__ void clock_prescale_set (clock_div_t);
01701
01702 void clock_prescale_set (clock_div_t __x)
01703 {
01704     __asm__ __volatile__ (
01705         "in __tmp_reg__, __SREG__" "\n\t"
01706         "cli" "\n\t"
01707         "sts %2, %3" "\n\t"
01708         "sts %1, %0" "\n\t"
01709         "out __SREG__, __tmp_reg__"
01710         : /* no outputs */
01711         : "r" (__x),
01712         "n" (_SFR_MEM_ADDR(CLKPR)),
01713         "n" (_SFR_MEM_ADDR(CCP)),
01714         "r" ((uint8_t) 0xD8)
01715         : "r0");
01716 }
01717
01718 #define clock_prescale_get() (clock_div_t) (CLKPR &
    (uint8_t) ((1<CLKPS0) | (1<CLKPS1) | (1<CLKPS2) | (1<CLKPS3)))
01719
01720 #elif defined(__AVR_ATmega64__) \
01721 || defined(__AVR_ATmega103__) \
01722 || defined(__AVR_ATmega128__)
01723
01724 //Enum is declared for code compatibility
01725 typedef enum
01726 {
01727     clock_div_1 = 1,
01728     clock_div_2 = 2,
01729     clock_div_4 = 4,

```

```

01730     clock_div_8 = 8,
01731     clock_div_16 = 16,
01732     clock_div_32 = 32,
01733     clock_div_64 = 64,
01734     clock_div_128 = 128
01735 } clock_div_t;
01736
01737 static __ATTR_ALWAYS_INLINE__ void clock_prescale_set(clock_div_t);
01738
01739 void clock_prescale_set(clock_div_t __x)
01740 {
01741     if((__x <= 0) || (__x > 129))
01742     {
01743         return; //Invalid value.
01744     }
01745     else
01746     {
01747         uint8_t __tmp = 0;
01748         //Algo explained:
01749         //1 - Clear XDIV in order for it to accept a new value (actually only
01750         //   XDIVEN need to be cleared, but clearing XDIV is faster than
01751         //   read-modify-write since we will rewrite XDIV later anyway)
01752         //2 - wait 8 clock cycle for stability, see datasheet errata
01753         //3 - Exit if requested prescaler is 1
01754         //4 - Calculate XDIV6..0 value = 129 - __x
01755         //5 - Set XDIVEN bit in calculated value
01756         //6 - write XDIV with calculated value
01757         //7 - wait 8 clock cycle for stability, see datasheet errata
01758         __asm__ volatile (
01759             "in __tmp_reg__, __SREG__" "\n\t"
01760             "cli" "\n\t"
01761             "out %2, __zero_reg__" "\n\t"
01762             "nop" "\n\t"
01763             "nop" "\n\t"
01764             "nop" "\n\t"
01765             "nop" "\n\t"
01766             "nop" "\n\t"
01767             "nop" "\n\t"
01768             "nop" "\n\t"
01769             "nop" "\n\t"
01770             "cpi %1, 0x01" "\n\t"
01771             "breq L_%= " "\n\t"
01772             "ldi %0, 0x81" "\n\t" //129
01773             "sub %0, %1" "\n\t"
01774             "ori %0, 0x80" "\n\t" //128
01775             "out %2, %0" "\n\t"
01776             "nop" "\n\t"
01777             "nop" "\n\t"
01778             "nop" "\n\t"
01779             "nop" "\n\t"
01780             "nop" "\n\t"
01781             "nop" "\n\t"
01782             "nop" "\n\t"
01783             "nop" "\n\t"
01784             "L_%=: " "out __SREG__, __tmp_reg__"
01785             : "=d" (__tmp)
01786             : "d" (__x),
01787               "I" (_SFR_IO_ADDR(XDIV))
01788             : "r0");
01789     }
01790 }
01791
01792 static __ATTR_ALWAYS_INLINE__ clock_div_t clock_prescale_get(void);
01793
01794 clock_div_t clock_prescale_get(void)
01795 {
01796     if (bit_is_clear(XDIV, XDIVEN))
01797     {
01798         return 1;
01799     }
01800     else
01801     {
01802         return (clock_div_t) (129 - (XDIV & 0x7F));
01803     }
01804 }
01805
01806 #elif defined(__AVR_ATtiny4__) \
01807 || defined(__AVR_ATtiny5__) \
01808 || defined(__AVR_ATtiny9__) \
01809 || defined(__AVR_ATtiny10__) \
01810 || defined(__AVR_ATtiny102__) \
01811 || defined(__AVR_ATtiny104__) \
01812 || defined(__AVR_ATtiny20__) \
01813 || defined(__AVR_ATtiny40__) \
01814
01815 typedef enum
01816 {

```

```

01817     clock_div_1 = 0,
01818     clock_div_2 = 1,
01819     clock_div_4 = 2,
01820     clock_div_8 = 3,
01821     clock_div_16 = 4,
01822     clock_div_32 = 5,
01823     clock_div_64 = 6,
01824     clock_div_128 = 7,
01825     clock_div_256 = 8
01826 } clock_div_t;
01827
01828 static __ATTR_ALWAYS_INLINE__ void clock_prescale_set(clock_div_t);
01829
01830 void clock_prescale_set(clock_div_t __x)
01831 {
01832     uint8_t __tmp = 0xD8;
01833     __asm__ __volatile__ (
01834         "in __tmp_reg__, __SREG__" "\n\t"
01835         "cli" "\n\t"
01836         "out %1, %0" "\n\t"
01837         "out %2, %3" "\n\t"
01838         "out __SREG__, __tmp_reg__"
01839         : /* no outputs */
01840         : "d" (__tmp),
01841           "I" (_SFR_IO_ADDR(CCP)),
01842           "I" (_SFR_IO_ADDR(CLKPSR)),
01843           "d" (__x)
01844         : "r16");
01845 }
01846
01847 #define clock_prescale_get() (clock_div_t)(CLKPSR &
01848 (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))
01849 #endif
01850
01851 #endif /* _AVR_POWER_H_ */

```

## 23.30 sfr\_defs.h

```

00001 /* Copyright (c) 2002, Marek Michalkiewicz <marekm@amelek.gda.pl>
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011     notice, this list of conditions and the following disclaimer in
00012     the documentation and/or other materials provided with the
00013     distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016     contributors may be used to endorse or promote products derived
00017     from this software without specific prior written permission.
00018
00019     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029     POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* avr/sfr_defs.h - macros for accessing AVR special function registers */
00032
00033 /* $Id$ */
00034
00035 #ifndef _AVR_SFR_DEFS_H_
00036 #define _AVR_SFR_DEFS_H_ 1
00037
00038 /** \defgroup avr_sfr_notes Additional notes from <avr/sfr_defs.h>
00039     \ingroup avr_sfr
00040
00041     The <c <avr/sfr_defs.h> file is included by all of the <c <avr/ioXXXX.h>
00042     files, which use macros defined here to make the special function register
00043     definitions look like C variables or simple constants, depending on the
00044     <tt><SFR_ASM_COMPAT</tt> define. Some examples from <c <avr/iocanxx.h> to
00045     show how to define such macros:

```

```

00046
00047 \code
00048 #define PORTA    _SFR_IO8(0x02)
00049 #define EEAR    _SFR_IO16(0x21)
00050 #define UDR0    _SFR_MEM8(0xC6)
00051 #define TCNT3    _SFR_MEM16(0x94)
00052 #define CANIDT    _SFR_MEM32(0xF0)
00053 \endcode
00054
00055     If \c _SFR_ASM_COMPAT is not defined, C programs can use names like
00056 <tt>PORTA</tt> directly in C expressions (also on the left side of
00057 assignment operators) and GCC will do the right thing (use short I/O
00058 instructions if possible). The \c __SFR_OFFSET definition is not used in
00059 any way in this case.
00060
00061     Define \c _SFR_ASM_COMPAT as 1 to make these names work as simple constants
00062 (addresses of the I/O registers). This is necessary when included in
00063 preprocessed assembler (*.S) source files, so it is done automatically if
00064 \c __ASSEMBLER__ is defined. By default, all addresses are defined as if
00065 they were memory addresses (used in \c lds/sts instructions). To use these
00066 addresses in \c in/out instructions, you must subtract 0x20 from them.
00067
00068     For more backwards compatibility, insert the following at the start of your
00069 old assembler source file:
00070
00071 \code
00072 #define __SFR_OFFSET 0
00073 \endcode
00074
00075     This automatically subtracts 0x20 from I/O space addresses, but it's a
00076 hack, so it is recommended to change your source: wrap such addresses in
00077 macros defined here, as shown below. After this is done, the
00078 <tt>__SFR_OFFSET</tt> definition is no longer necessary and can be removed.
00079
00080     Real example - this code could be used in a boot loader that is portable
00081 between devices with \c SPMCR at different addresses.
00082
00083 \verbatim
00084 <avr/iom163.h>: #define SPMCR _SFR_IO8(0x37)
00085 <avr/iom128.h>: #define SPMCR _SFR_MEM8(0x68)
00086 \endverbatim
00087
00088 \code
00089 #if _SFR_IO_REG_P(SPMCR)
00090     out _SFR_IO_ADDR(SPMCR), r24
00091 #else
00092     sts _SFR_MEM_ADDR(SPMCR), r24
00093 #endif
00094 \endcode
00095
00096     You can use the \c in/out/cbi/sbi/sbic/sbis instructions, without the
00097 <tt>_SFR_IO_REG_P</tt> test, if you know that the register is in the I/O
00098 space (as with \c SREG, for example). If it isn't, the assembler will
00099 complain (I/O address out of range 0...0x3f), so this should be fairly
00100 safe.
00101
00102     If you do not define \c __SFR_OFFSET (so it will be 0x20 by default), all
00103 special register addresses are defined as memory addresses (so \c SREG is
00104 0x5f), and (if code size and speed are not important, and you don't like
00105 the ugly \#if above) you can always use lds/sts to access them. But, this
00106 will not work if <tt>__SFR_OFFSET</tt> != 0x20, so use a different macro
00107 (defined only if <tt>__SFR_OFFSET</tt> == 0x20) for safety:
00108
00109 \code
00110     sts _SFR_ADDR(SPMCR), r24
00111 \endcode
00112
00113     In C programs, all 3 combinations of \c _SFR_ASM_COMPAT and
00114 <tt>__SFR_OFFSET</tt> are supported - the \c _SFR_ADDR(SPMCR) macro can be
00115 used to get the address of the \c SPMCR register (0x57 or 0x68 depending on
00116 device). */
00117
00118 #ifndef __ASSEMBLER__
00119 #define _SFR_ASM_COMPAT 1
00120 #elif !defined(_SFR_ASM_COMPAT)
00121 #define _SFR_ASM_COMPAT 0
00122 #endif
00123
00124 #ifndef __ASSEMBLER__
00125 /* These only work in C programs. */
00126 #include <inttypes.h>
00127
00128 #define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *) (mem_addr))
00129 #define _MMIO_WORD(mem_addr) (*(volatile uint16_t *) (mem_addr))
00130 #define _MMIO_DWORD(mem_addr) (*(volatile uint32_t *) (mem_addr))
00131 #endif
00132

```

```

00133 #if _SFR_ASM_COMPAT
00134
00135 #ifndef __SFR_OFFSET
00136 /* Define as 0 before including this file for compatibility with old asm
00137 sources that don't subtract __SFR_OFFSET from symbolic I/O addresses. */
00138 # if __AVR_ARCH__ >= 100
00139 #   define __SFR_OFFSET 0x00
00140 #   else
00141 #   define __SFR_OFFSET 0x20
00142 #   endif
00143 #endif
00144
00145 #if (__SFR_OFFSET != 0) && (__SFR_OFFSET != 0x20)
00146 #error "__SFR_OFFSET must be 0 or 0x20"
00147 #endif
00148
00149 #define _SFR_MEM8(mem_addr) (mem_addr)
00150 #define _SFR_MEM16(mem_addr) (mem_addr)
00151 #define _SFR_MEM32(mem_addr) (mem_addr)
00152 #define _SFR_IO8(io_addr) ((io_addr) + __SFR_OFFSET)
00153 #define _SFR_IO16(io_addr) ((io_addr) + __SFR_OFFSET)
00154
00155 #define _SFR_IO_ADDR(sfr) ((sfr) - __SFR_OFFSET)
00156 #define _SFR_MEM_ADDR(sfr) (sfr)
00157 #define _SFR_IO_REG_P(sfr) ((sfr) < 0x40 + __SFR_OFFSET)
00158
00159 #if (__SFR_OFFSET == 0x20)
00160 /* No need to use ?: operator, so works in assembler too. */
00161 #define _SFR_ADDR(sfr) _SFR_MEM_ADDR(sfr)
00162 #elif !defined(__ASSEMBLER__)
00163 #define _SFR_ADDR(sfr) (_SFR_IO_REG_P(sfr) ? (_SFR_IO_ADDR(sfr) + 0x20) : _SFR_MEM_ADDR(sfr))
00164 #endif
00165
00166 #else /* !_SFR_ASM_COMPAT */
00167
00168 #ifndef __SFR_OFFSET
00169 # if __AVR_ARCH__ >= 100
00170 #   define __SFR_OFFSET 0x00
00171 #   else
00172 #   define __SFR_OFFSET 0x20
00173 #   endif
00174 #endif
00175
00176 #define _SFR_MEM8(mem_addr) _MMIO_BYTE(mem_addr)
00177 #define _SFR_MEM16(mem_addr) _MMIO_WORD(mem_addr)
00178 #define _SFR_MEM32(mem_addr) _MMIO_DWORD(mem_addr)
00179 #define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + __SFR_OFFSET)
00180 #define _SFR_IO16(io_addr) _MMIO_WORD((io_addr) + __SFR_OFFSET)
00181
00182 #define _SFR_MEM_ADDR(sfr) ((uint16_t) &(sfr))
00183 #define _SFR_IO_ADDR(sfr) (_SFR_MEM_ADDR(sfr) - __SFR_OFFSET)
00184 #define _SFR_IO_REG_P(sfr) (_SFR_MEM_ADDR(sfr) < 0x40 + __SFR_OFFSET)
00185
00186 #define _SFR_ADDR(sfr) _SFR_MEM_ADDR(sfr)
00187
00188 #endif /* !_SFR_ASM_COMPAT */
00189
00190 #define _SFR_BYTE(sfr) _MMIO_BYTE(_SFR_ADDR(sfr))
00191 #define _SFR_WORD(sfr) _MMIO_WORD(_SFR_ADDR(sfr))
00192 #define _SFR_DWORD(sfr) _MMIO_DWORD(_SFR_ADDR(sfr))
00193
00194 /** \name Bit manipulation */
00195
00196 /**@{*/
00197 /** \def _BV
00198     \ingroup avr_sfr
00199
00200     \code #include <avr/io.h>\endcode
00201
00202     Converts a bit number into a byte value.
00203
00204     \note The bit shift is performed by the compiler which then inserts the
00205     result into the code. Thus, there is no run-time overhead when using
00206     _BV(). */
00207
00208 #define _BV(bit) (1 << (bit))
00209
00210 /**@}*/
00211
00212 #ifndef _VECTOR
00213 #define _VECTOR(N) __vector_ ## N
00214 #endif
00215
00216 #ifndef __ASSEMBLER__
00217
00218
00219 /** \name IO register bit manipulation */

```

```

00220
00221 /**{*/
00222
00223
00224
00225 /** \def bit_is_set
00226     \ingroup avr_sfr
00227
00228     \code #include <avr/io.h>\endcode
00229
00230     Test whether bit \c bit in IO register \c sfr is set.
00231     This will return a 0 if the bit is clear, and non-zero
00232     if the bit is set. */
00233
00234 #define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
00235
00236 /** \def bit_is_clear
00237     \ingroup avr_sfr
00238
00239     \code #include <avr/io.h>\endcode
00240
00241     Test whether bit \c bit in IO register \c sfr is clear.
00242     This will return non-zero if the bit is clear, and a 0
00243     if the bit is set. */
00244
00245 #define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))
00246
00247 /** \def loop_until_bit_is_set
00248     \ingroup avr_sfr
00249
00250     \code #include <avr/io.h>\endcode
00251
00252     Wait until bit \c bit in IO register \c sfr is set. */
00253
00254 #define loop_until_bit_is_set(sfr, bit) do { } while (bit_is_clear(sfr, bit))
00255
00256 /** \def loop_until_bit_is_clear
00257     \ingroup avr_sfr
00258
00259     \code #include <avr/io.h>\endcode
00260
00261     Wait until bit \c bit in IO register \c sfr is clear. */
00262
00263 #define loop_until_bit_is_clear(sfr, bit) do { } while (bit_is_set(sfr, bit))
00264
00265 /**}*/
00266
00267 #endif /* !__ASSEMBLER__ */
00268
00269 #endif /* _SFR_DEFS_H_ */

```

## 23.31 signal.h

```

00001 /* Copyright (c) 2002,2005,2006 Marek Michalkiewicz
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011       notice, this list of conditions and the following disclaimer in
00012       the documentation and/or other materials provided with the
00013       distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016       contributors may be used to endorse or promote products derived
00017       from this software without specific prior written permission.
00018
00019     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029     POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */

```

```

00032
00033 #ifndef _AVR_SIGNAL_H_
00034 #define _AVR_SIGNAL_H_
00035
00036 #warning "This header file is obsolete. Use <avr/interrupt.h>."
00037 #include <avr/interrupt.h>
00038
00039 #endif /* _AVR_SIGNAL_H_ */

```

## 23.32 signature.h File Reference

### 23.33 signature.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2009, Atmel Corporation
00002    All rights reserved.
00003
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions are met:
00006
00007    * Redistributions of source code must retain the above copyright
00008      notice, this list of conditions and the following disclaimer.
00009
00010    * Redistributions in binary form must reproduce the above copyright
00011      notice, this list of conditions and the following disclaimer in
00012      the documentation and/or other materials provided with the
00013      distribution.
00014
00015    * Neither the name of the copyright holders nor the names of
00016      contributors may be used to endorse or promote products derived
00017      from this software without specific prior written permission.
00018
00019    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029    POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /* avr/signature.h - Signature API */
00034
00035 #ifndef _AVR_SIGNATURE_H_
00036 #define _AVR_SIGNATURE_H_ 1
00037
00038 /** \file */
00039 /** \defgroup avr_signature <avr/signature.h>: Signature Support
00040
00041     \par Introduction
00042
00043     The <avr/signature.h> header file allows the user to automatically
00044     and easily include the device's signature data in a special section of
00045     the final linked ELF file.
00046
00047     This value can then be used by programming software to compare the on-device
00048     signature with the signature recorded in the ELF file to look for a match
00049     before programming the device.
00050
00051     \par API Usage Example
00052
00053     Usage is very simple; just include the header file:
00054
00055     \code
00056     #include <avr/signature.h>
00057     \endcode
00058
00059     This will declare a constant unsigned char array and it is initialized with
00060     the three signature bytes, MSB first, that are defined in the device I/O
00061     header file. This array is then placed in the .signature section in the
00062     resulting linked ELF file.
00063
00064     The three signature bytes that are used to initialize the array are
00065     these defined macros in the device I/O header file, from MSB to LSB:
00066     SIGNATURE_2, SIGNATURE_1, SIGNATURE_0.

```

```

00067
00068     This header file should only be included once in an application.
00069 */
00070
00071 #ifndef __ASSEMBLER__
00072
00073 #include <avr/io.h>
00074
00075 #if defined(SIGNATURE_0) && defined(SIGNATURE_1) && defined(SIGNATURE_2)
00076
00077 const unsigned char __signature[3]
00078 __attribute__((__used__, __section__(".signature"))) =
00079     { SIGNATURE_2, SIGNATURE_1, SIGNATURE_0 };
00080
00081 #endif /* defined(SIGNATURE_0) && defined(SIGNATURE_1) && defined(SIGNATURE_2) */
00082
00083 #endif /* __ASSEMBLER__ */
00084
00085 #endif /* _AVR_SIGNATURE_H_ */

```

## 23.34 sleep.h File Reference

### Functions

- void [sleep\\_enable](#) (void)
- void [sleep\\_disable](#) (void)
- void [sleep\\_cpu](#) (void)
- void [sleep\\_mode](#) (void)
- void [sleep\\_bod\\_disable](#) (void)

## 23.35 sleep.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, 2004 Theodore A. Roth
00002     Copyright (c) 2004, 2007, 2008 Eric B. Weddington
00003     Copyright (c) 2005, 2006, 2007 Joerg Wunsch
00004     All rights reserved.
00005
00006     Redistribution and use in source and binary forms, with or without
00007     modification, are permitted provided that the following conditions are met:
00008
00009     * Redistributions of source code must retain the above copyright
00010     notice, this list of conditions and the following disclaimer.
00011
00012     * Redistributions in binary form must reproduce the above copyright
00013     notice, this list of conditions and the following disclaimer in
00014     the documentation and/or other materials provided with the
00015     distribution.
00016
00017     * Neither the name of the copyright holders nor the names of
00018     contributors may be used to endorse or promote products derived
00019     from this software without specific prior written permission.
00020
00021     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00022     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00023     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00024     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00025     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00026     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00027     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00028     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00029     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00030     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00031     POSSIBILITY OF SUCH DAMAGE. */
00032
00033 /* $Id$ */
00034
00035 #ifndef _AVR_SLEEP_H_
00036 #define _AVR_SLEEP_H_ 1
00037
00038 #include <avr/io.h>
00039 #include <stdint.h>
00040
00041

```



```

00042 /** \file */
00043
00044 /** \defgroup avr_sleep <avr/sleep.h>: Power Management and Sleep Modes */
00045 /**@{*/
00046 /**
00047     \code #include <avr/sleep.h>\endcode
00048
00049     Use of the \c SLEEP instruction can allow an application to reduce its
00050     power consumption considerably. AVR devices can be put into different
00051     sleep modes. Refer to the datasheet for the details relating to the device
00052     you are using.
00053
00054     There are several macros provided in this header file to actually
00055     put the device into sleep mode. The simplest way is to optionally
00056     set the desired sleep mode using \c set_sleep_mode() (it usually
00057     defaults to idle mode where the CPU is put on sleep but all
00058     peripheral clocks are still running), and then call
00059     \c sleep_mode(). This macro automatically sets the sleep enable bit, goes
00060     to sleep, and clears the sleep enable bit.
00061
00062     Example:
00063     \code
00064     #include <avr/sleep.h>
00065
00066     ...
00067     set_sleep_mode(<mode>);
00068     sleep_mode();
00069     \endcode
00070
00071     Note that unless your purpose is to completely lock the CPU (until a
00072     hardware reset), interrupts need to be enabled before going to sleep.
00073
00074     As the \c sleep_mode() macro might cause race conditions in some
00075     situations, the individual steps of manipulating the sleep enable
00076     (SE) bit, and actually issuing the \c SLEEP instruction, are provided
00077     in the macros \c sleep_enable(), \c sleep_disable(), and
00078     \c sleep_cpu(). This also allows for test-and-sleep scenarios that
00079     take care of not missing the interrupt that will awake the device
00080     from sleep.
00081
00082     Example:
00083     \code
00084     #include <avr/interrupt.h>
00085     #include <avr/sleep.h>
00086
00087     ...
00088     set_sleep_mode(<mode>);
00089     cli();
00090     if (some_condition)
00091     {
00092         sleep_enable();
00093         sei();
00094         sleep_cpu();
00095         sleep_disable();
00096     }
00097     sei();
00098     \endcode
00099
00100     This sequence ensures an atomic test of \c some_condition with
00101     interrupts being disabled. If the condition is met, sleep mode
00102     will be prepared, and the \c SLEEP instruction will be scheduled
00103     immediately after an \c SEI instruction. As the instruction right
00104     after the \c SEI is guaranteed to be executed before an interrupt
00105     could trigger, it is sure the device will really be put to sleep.
00106
00107     Some devices have the ability to disable the Brown Out Detector (BOD) before
00108     going to sleep. This will also reduce power while sleeping. If the
00109     specific AVR device has this ability then an additional macro is defined:
00110     \c sleep_bod_disable(). This macro generates inlined assembly code
00111     that will correctly implement the timed sequence for disabling the BOD
00112     before sleeping. However, there is a limited number of cycles after the
00113     BOD has been disabled that the device can be put into sleep mode, otherwise
00114     the BOD will not truly be disabled. Recommended practice is to disable
00115     the BOD (\c sleep_bod_disable()), set the interrupts (\c sei()), and then
00116     put the device to sleep (\c sleep_cpu()), like so:
00117
00118     \code
00119     #include <avr/interrupt.h>
00120     #include <avr/sleep.h>
00121
00122     ...
00123     set_sleep_mode(<mode>);
00124     cli();
00125     if (some_condition)
00126     {
00127         sleep_enable();
00128         sleep_bod_disable();

```

```

00129     sei();
00130     sleep_cpu();
00131     sleep_disable();
00132 }
00133 sei();
00134 \endcode
00135 */
00136
00137
00138 /* Define an internal sleep control register and an internal sleep enable bit mask. */
00139 #if defined(SLEEP_CTRL)
00140
00141     /* XMEGA devices */
00142     #define _SLEEP_CONTROL_REG SLEEP_CTRL
00143     #define _SLEEP_ENABLE_MASK SLEEP_SEN_bm
00144     #define _SLEEP_SMODE_GROUP_MASK SLEEP_SMODE_gm
00145
00146 #elif defined(SLPCTRL)
00147
00148     /* New xmega devices */
00149     #define _SLEEP_CONTROL_REG SLPCTRL_CTRLA
00150     #define _SLEEP_ENABLE_MASK SLPCTRL_SEN_bm
00151     #define _SLEEP_SMODE_GROUP_MASK SLPCTRL_SMODE_gm
00152
00153 #elif defined(SMCR)
00154
00155     #define _SLEEP_CONTROL_REG SMCR
00156     #define _SLEEP_ENABLE_MASK _BV(SE)
00157
00158 #elif defined(__AVR_AT94K__)
00159
00160     #define _SLEEP_CONTROL_REG MCUR
00161     #define _SLEEP_ENABLE_MASK _BV(SE)
00162
00163 #elif !defined(__DOXYGEN__)
00164
00165     #define _SLEEP_CONTROL_REG MCUCR
00166     #define _SLEEP_ENABLE_MASK _BV(SE)
00167
00168 #endif
00169
00170
00171 /* Special casing these three devices - they are the
00172 only ones that need to write to more than one register. */
00173 #if defined(__AVR_ATmega161__)
00174
00175     #define set_sleep_mode(mode) \
00176     do { \
00177         MCUCR = ((MCUCR & ~_BV(SM1)) | ((mode) == SLEEP_MODE_PWR_DOWN || (mode) == SLEEP_MODE_PWR_SAVE
00178 ? _BV(SM1) : 0)); \
00179         EMCUCR = ((EMCUCR & ~_BV(SM0)) | ((mode) == SLEEP_MODE_PWR_SAVE ? _BV(SM0) : 0)); \
00180     } while(0)
00181
00182 #elif defined(__AVR_ATmega162__) \
00183 || defined(__AVR_ATmega8515__)
00184
00185     #define set_sleep_mode(mode) \
00186     do { \
00187         MCUCR = ((MCUCR & ~_BV(SM1)) | ((mode) == SLEEP_MODE_IDLE ? 0 : _BV(SM1))); \
00188         MCUCSR = ((MCUCSR & ~_BV(SM2)) | ((mode) == SLEEP_MODE_STANDBY || (mode) ==
00189 SLEEP_MODE_EXT_STANDBY ? _BV(SM2) : 0)); \
00190         EMCUCR = ((EMCUCR & ~_BV(SM0)) | ((mode) == SLEEP_MODE_PWR_SAVE || (mode) ==
00191 SLEEP_MODE_EXT_STANDBY ? _BV(SM0) : 0)); \
00192     } while(0)
00193
00194 /* For xmegs, check presence of SLEEP_SMODE<n>_bm and define set_sleep_mode accordingly. */
00195 #elif defined(__AVR_XMEGA__)
00196
00197 #define set_sleep_mode(mode) \
00198 do { \
00199     _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~(_SLEEP_SMODE_GROUP_MASK)) | (mode)); \
00200 } while(0)
00201
00202 /* For everything else, check for presence of SM<n> and define set_sleep_mode accordingly. */
00203 #else
00204 #if defined(SM2)
00205
00206     #define set_sleep_mode(mode) \
00207     do { \
00208         _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~(_BV(SM0) | _BV(SM1) | _BV(SM2))) | (mode)); \
00209     } while(0)
00210
00211 #elif defined(SM1)
00212
00213     #define set_sleep_mode(mode) \
00214     do { \

```

```

00213     _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~(_BV(SM0) | _BV(SM1))) | (mode)); \
00214     } while(0)
00215
00216 #elif defined(SM)
00217
00218     #define set_sleep_mode(mode) \
00219     do { \
00220         _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~_BV(SM)) | (mode)); \
00221     } while(0)
00222
00223 #else
00224
00225     #error "No SLEEP mode defined for this device."
00226
00227 #endif /* if defined(SM2) */
00228 #endif /* #if defined(__AVR_ATmega161__) */
00229
00230
00231
00232 /** \ingroup avr_sleep
00233
00234     Put the device in sleep mode. How the device is brought out of sleep mode
00235     depends on the specific mode selected with the set_sleep_mode() function.
00236     See the data sheet for your device for more details. */
00237
00238
00239 #if defined(__DOXYGEN__)
00240
00241 /** \ingroup avr_sleep
00242
00243     Set the SE (sleep enable) bit.
00244 */
00245 extern void sleep_enable (void);
00246
00247 #else
00248
00249 #define sleep_enable() \
00250 do { \
00251     _SLEEP_CONTROL_REG |= (uint8_t)_SLEEP_ENABLE_MASK; \
00252 } while(0)
00253
00254 #endif
00255
00256
00257 #if defined(__DOXYGEN__)
00258
00259 /** \ingroup avr_sleep
00260
00261     Clear the SE (sleep enable) bit.
00262 */
00263 extern void sleep_disable (void);
00264
00265 #else
00266
00267 #define sleep_disable() \
00268 do { \
00269     _SLEEP_CONTROL_REG &= (uint8_t)(~_SLEEP_ENABLE_MASK); \
00270 } while(0)
00271
00272 #endif
00273
00274
00275 /** \ingroup avr_sleep
00276
00277     Put the device into sleep mode. The SE bit must be set
00278     beforehand, and it is recommended to clear it afterwards.
00279 */
00280 #if defined(__DOXYGEN__)
00281
00282 extern void sleep_cpu (void);
00283
00284 #else
00285
00286 #define sleep_cpu() \
00287 do { \
00288     __asm__ __volatile__ ( "sleep" "\n\t" ::: ); \
00289 } while(0)
00290
00291 #endif
00292
00293
00294 #if defined(__DOXYGEN__)
00295
00296 /** \ingroup avr_sleep
00297
00298     Put the device into sleep mode, taking care of setting
00299     the SE bit before, and clearing it afterwards. */

```

```

00300 extern void sleep_mode (void);
00301
00302 #else
00303
00304 #define sleep_mode() \
00305 do { \
00306     sleep_enable(); \
00307     sleep_cpu(); \
00308     sleep_disable(); \
00309 } while (0)
00310
00311 #endif
00312
00313
00314 #if defined(__DOXYGEN__)
00315
00316 /** \ingroup avr_sleep
00317
00318     Disable BOD before going to sleep.
00319     Not available on all devices.
00320 */
00321 extern void sleep_bod_disable (void);
00322
00323 #else
00324
00325 #if defined(BODS) && defined(BODSE)
00326
00327 #ifndef BODCR
00328
00329 #define BOD_CONTROL_REG BODCR
00330
00331 #else
00332
00333 #define BOD_CONTROL_REG MCUCR
00334
00335 #endif
00336
00337 #define sleep_bod_disable() \
00338 do { \
00339     uint8_t tempreg; \
00340     __asm__ __volatile__ ("in %[tempreg], %[mcucr]" "\n\t" \
00341                          "ori %[tempreg], %[bods_bodse]" "\n\t" \
00342                          "out %[mcucr], %[tempreg]" "\n\t" \
00343                          "andi %[tempreg], %[not_bodse]" "\n\t" \
00344                          "out %[mcucr], %[tempreg]" \
00345                          : [tempreg] "=&d" (tempreg) \
00346                          : [mcucr] "I" _SFR_IO_ADDR(BOD_CONTROL_REG), \
00347                          [bods_bodse] "i" (_BV(BODS) | _BV(BODSE)), \
00348                          [not_bodse] "i" (~_BV(BODSE)); \
00349 } while (0)
00350
00351 #endif
00352
00353 #endif
00354
00355
00356 /**@}*/
00357
00358 #endif /* _AVR_SLEEP_H_ */

```

## 23.36 version.h

```

00001 /* Copyright (c) 2005, Joerg Wunsch                               -*- c -*-
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011     notice, this list of conditions and the following disclaimer in
00012     the documentation and/or other materials provided with the
00013     distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016     contributors may be used to endorse or promote products derived
00017     from this software without specific prior written permission.
00018
00019     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

```

```

00023    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029    POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /** \defgroup avr_version <avr/version.h>: avr-libc version macros
00034     \code #include <avr/version.h> \endcode
00035
00036     This header file defines macros that contain version numbers and
00037     strings describing the current version of avr-libc.
00038
00039     The version number itself basically consists of three pieces that
00040     are separated by a dot: the major number, the minor number, and
00041     the revision number. For development versions (which use an odd
00042     minor number), the string representation additionally gets the
00043     date code (YYYYMMDD) appended.
00044
00045     This file will also be included by \c <avr/io.h>. That way,
00046     portable tests can be implemented using \c <avr/io.h> that can be
00047     used in code that wants to remain backwards-compatible to library
00048     versions prior to the date when the library version API had been
00049     added, as referenced but undefined C preprocessor macros
00050     automatically evaluate to 0.
00051 */
00052
00053 #ifndef _AVR_VERSION_H_
00054 #define _AVR_VERSION_H_
00055
00056 /** \ingroup avr_version
00057     String literal representation of the current library version. */
00058 #define __AVR_LIBC_VERSION_STRING__ "2.2.0"
00059
00060 /** \ingroup avr_version
00061     Numerical representation of the current library version.
00062
00063     In the numerical representation, the major number is multiplied by
00064     10000, the minor number by 100, and all three parts are then
00065     added. It is intended to provide a monotonically increasing
00066     numerical value that can easily be used in numerical checks.
00067 */
00068 #define __AVR_LIBC_VERSION__      20200UL
00069
00070 /** \ingroup avr_version
00071     String literal representation of the release date. */
00072 #define __AVR_LIBC_DATE_STRING__  "20240608"
00073
00074 /** \ingroup avr_version
00075     Numerical representation of the release date. */
00076 #define __AVR_LIBC_DATE__        20240608UL
00077
00078 /** \ingroup avr_version
00079     Library major version number. */
00080 #define __AVR_LIBC_MAJOR__      2
00081
00082 /** \ingroup avr_version
00083     Library minor version number. */
00084 #define __AVR_LIBC_MINOR__      2
00085
00086 /** \ingroup avr_version
00087     Library revision number. */
00088 #define __AVR_LIBC_REVISION__    0
00089
00090 #endif /* _AVR_VERSION_H_ */

```

## 23.37 wdt.h File Reference

### Macros

- #define `wdt_reset()` `__asm__ __volatile__ ("wdr")`
- #define `wdt_enable(timeout)`
- #define `WDTO_15MS` 0
- #define `WDTO_30MS` 1
- #define `WDTO_60MS` 2

- `#define WDTO_120MS 3`
- `#define WDTO_250MS 4`
- `#define WDTO_500MS 5`
- `#define WDTO_1S 6`
- `#define WDTO_2S 7`
- `#define WDTO_4S 8`
- `#define WDTO_8S 9`

## Functions

- static void `wdt_enable` (const `uint8_t` value)
- static void `wdt_disable` (void)

## 23.38 wdt.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, 2004 Marek Michalkiewicz
00002    Copyright (c) 2005, 2006, 2007 Eric B. Weddington
00003    All rights reserved.
00004
00005    Redistribution and use in source and binary forms, with or without
00006    modification, are permitted provided that the following conditions are met:
00007
00008    * Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010
00011    * Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in
00013    the documentation and/or other materials provided with the
00014    distribution.
00015
00016    * Neither the name of the copyright holders nor the names of
00017    contributors may be used to endorse or promote products derived
00018    from this software without specific prior written permission.
00019
00020    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00022    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030    POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033
00034 /*
00035    avr/wdt.h - macros for AVR watchdog timer
00036 */
00037
00038 #ifndef _AVR_WDT_H_
00039 #define _AVR_WDT_H_
00040
00041 #include <avr/io.h>
00042 #include <stdint.h>
00043
00044 /** \file */
00045 /** \defgroup avr_watchdog <avr/wdt.h>: Watchdog timer handling
00046     \code #include <avr/wdt.h> \endcode
00047
00048     This header file declares the interface to some inline macros
00049     handling the watchdog timer present in many AVR devices. In order
00050     to prevent the watchdog timer configuration from being
00051     accidentally altered by a crashing application, a special timed
00052     sequence is required in order to change it. The macros within
00053     this header file handle the required sequence automatically
00054     before changing any value. Interrupts will be disabled during
00055     the manipulation.
00056
00057     \note Depending on the fuse configuration of the particular
00058     device, further restrictions might apply, in particular it might

```

```

00059     be disallowed to turn off the watchdog timer.
00060
00061     Note that for newer devices (ATmega88 and newer, effectively any
00062     AVR that has the option to also generate interrupts), the watchdog
00063     timer remains active even after a system reset (except a power-on
00064     condition), using the fastest prescaler value (approximately 15
00065     ms). It is therefore required to turn off the watchdog early
00066     during program startup, the datasheet recommends a sequence like
00067     the following:
00068
00069     \code
00070     #include <stdint.h>
00071     #include <avr/wdt.h>
00072
00073     uint8_t mcusr_mirror __attribute__ ((section (".noinit")));
00074
00075     __attribute__((used, unused, naked, section(".init3")))
00076     static void get_mcusr (void);
00077
00078     void get_mcusr (void)
00079     {
00080         mcusr_mirror = MCUSR;
00081         MCUSR = 0;
00082         wdt_disable();
00083     }
00084     \endcode
00085
00086     Saving the value of MCUSR in \c mcusr_mirror is only needed if the
00087     application later wants to examine the reset source, but in particular,
00088     clearing the watchdog reset flag before disabling the
00089     watchdog is required, according to the datasheet.
00090 */
00091
00092 /**
00093     \ingroup avr_watchdog
00094     Reset the watchdog timer. When the watchdog timer is enabled,
00095     a call to this instruction is required before the timer expires,
00096     otherwise a watchdog-initiated device reset will occur.
00097 */
00098
00099 #define wdt_reset() __asm__ __volatile__ ("wdr")
00100
00101 #ifndef __DOXYGEN__
00102
00103 #ifndef __ATTR_ALWAYS_INLINE__
00104 #define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00105 #endif
00106
00107 #if defined(WDP3)
00108 # define _WD_PS3_MASK      _BV(WDP3)
00109 #else
00110 # define _WD_PS3_MASK      0x00
00111 #endif
00112
00113 #if defined(WDTCSR)
00114 # define _WD_CONTROL_REG   WDTCSR
00115 #elif defined(WDTCR)
00116 # define _WD_CONTROL_REG   WDTCSR
00117 #else
00118 # define _WD_CONTROL_REG   WDT
00119 #endif
00120
00121 #if defined(WDTOE)
00122 #define _WD_CHANGE_BIT     WDTOE
00123 #else
00124 #define _WD_CHANGE_BIT     WDCE
00125 #endif
00126
00127 #endif /* !__DOXYGEN__ */
00128
00129 #ifdef __DOXYGEN__
00130 /**
00131     \ingroup avr_watchdog
00132     Enable the watchdog timer, configuring it for expiry after
00133     \c timeout (which is a combination of the \c WDP0 through
00134     \c WDP2 bits to write into the \c WDTCSR register; For those devices
00135     that have a \c WDTCSR register, it uses the combination of the \c WDP0
00136     through \c WDP3 bits).
00137
00138     See also the symbolic constants \c WDTO_15MS et al.
00139 */
00140 #define wdt_enable(timeout)
00141 #endif /* __DOXYGEN__ */
00142
00143
00144 #if defined(__AVR_XMEGA__)
00145

```

```

00146 #if defined (WDT_CTRLA) && !defined(RAMPD)
00147
00148 #define wdt_enable(timeout) \
00149 do { \
00150     uint8_t __temp; \
00151     __asm__ __volatile__ ( \
00152         "wdr"                                     "\n\t" \
00153         "out %[ccp_reg], %[ioreg_cen_mask]"        "\n\t" \
00154         "lds %[tmp], %[wdt_reg]"                  "\n\t" \
00155         "sbr %[tmp], %[wdt_enable_timeout]"       "\n\t" \
00156         "sts %[wdt_reg], %[tmp]"                   "\n\t" \
00157         "l:lds %[tmp], %[wdt_status_reg]"          "\n\t" \
00158         "sbrc %[tmp], %[wdt_syncbusy_bit]"        "\n\t" \
00159         "rjmp 1b"                                   \
00160         : [tmp]                                     "=d" (__temp) \
00161         : [ccp_reg]                                 "I"  (_SFR_IO_ADDR(CCP)), \
00162           [ioreg_cen_mask]                          "r"  ((uint8_t)CCP_IOREG_gc), \
00163           [wdt_reg]                                  "n"  (_SFR_MEM_ADDR(WDT_CTRLA)), \
00164           [wdt_enable_timeout]                      "M"  (timeout), \
00165           [wdt_status_reg]                          "n"  (_SFR_MEM_ADDR(WDT_STATUS)), \
00166           [wdt_syncbusy_bit]                        "I"  (WDT_SYNCBUSY_bm) \
00167     ); \
00168 } while(0)
00169
00170 #define wdt_disable() \
00171 do { \
00172     uint8_t __temp; \
00173     __asm__ __volatile__ ( \
00174         "wdr"                                     "\n\t" \
00175         "out %[ccp_reg], %[ioreg_cen_mask]"        "\n\t" \
00176         "lds %[tmp], %[wdt_reg]"                  "\n\t" \
00177         "cbr %[tmp], %[timeout_mask]"             "\n\t" \
00178         "sts %[wdt_reg], %[tmp]"                   \
00179         : [tmp]                                     "=d" (__temp) \
00180         : [ccp_reg]                                 "I"  (_SFR_IO_ADDR(CCP)), \
00181           [ioreg_cen_mask]                          "r"  ((uint8_t)CCP_IOREG_gc), \
00182           [wdt_reg]                                  "n"  (_SFR_MEM_ADDR(WDT_CTRLA)), \
00183           [timeout_mask]                            "I"  (WDT_PERIOD_gm) \
00184     ); \
00185 } while(0)
00186
00187 #else // defined (WDT_CTRLA) && !defined(RAMPD)
00188
00189 /*
00190     wdt_enable(timeout) for xmega devices
00191 ** write signature (CCP_IOREG_gc) that enables change of protected I/O
00192 registers to the CCP register
00193 ** At the same time,
00194 1) set WDT change enable (WDT_CEN_bm)
00195 2) enable WDT (WDT_ENABLE_bm)
00196 3) set timeout (timeout)
00197 ** Synchronization starts when ENABLE bit of WDT is set. So, wait till it
00198 finishes (SYNCBUSY of STATUS register is automatically cleared after the
00199 sync is finished).
00200 */
00201 #define wdt_enable(timeout) \
00202 do { \
00203     uint8_t __temp; \
00204     __asm__ __volatile__ ( \
00205         "in __tmp_reg__, %[rampd]"                 "\n\t" \
00206         "out %[rampd], __zero_reg__"               "\n\t" \
00207         "out %[ccp_reg], %[ioreg_cen_mask]"        "\n\t" \
00208         "sts %[wdt_reg], %[wdt_enable_timeout]"   "\n\t" \
00209         "l:lds %[tmp], %[wdt_status_reg]"          "\n\t" \
00210         "sbrc %[tmp], %[wdt_syncbusy_bit]"        "\n\t" \
00211         "rjmp 1b"                                   "\n\t" \
00212         "out %[rampd], __tmp_reg__"               \
00213         : [tmp]                                     "=r" (__temp) \
00214         : [rampd]                                   "I"  (_SFR_IO_ADDR(RAMPD)), \
00215           [ccp_reg]                                 "I"  (_SFR_IO_ADDR(CCP)), \
00216           [ioreg_cen_mask]                          "r"  ((uint8_t)CCP_IOREG_gc), \
00217           [wdt_reg]                                  "n"  (_SFR_MEM_ADDR(WDT_CTRLA)), \
00218           [wdt_enable_timeout]                      "r"  ((uint8_t)(WDT_CEN_bm | WDT_ENABLE_bm | timeout)), \
00219           [wdt_status_reg]                          "n"  (_SFR_MEM_ADDR(WDT_STATUS)), \
00220           [wdt_syncbusy_bit]                        "I"  (WDT_SYNCBUSY_bm) \
00221         : "r0" \
00222     ); \
00223 } while(0)
00224
00225 #define wdt_disable() \
00226 __asm__ __volatile__ ( \
00227     "in __tmp_reg__, %[rampd]"                     "\n\t" \
00228     "out %[rampd], __zero_reg__"                   "\n\t" \
00229     "out %[ccp_reg], %[ioreg_cen_mask]"            "\n\t" \
00230     "sts %[wdt_reg], %[disable_mask]"              "\n\t" \
00231     "out %[rampd], __tmp_reg__"                    \
00232     : /* no outputs */ \

```



```

00233     : [rampd]           "I" (_SFR_IO_ADDR(RAMPD)), \
00234     [ccp_reg]         "I" (_SFR_IO_ADDR(CCP)), \
00235     [ioreg_cen_mask]  "r" ((uint8_t)CCP_IOREG_gc), \
00236     [wdt_reg]         "n" (_SFR_MEM_ADDR(WDT_CTRL)), \
00237     [disable_mask]   "r" ((uint8_t)((-WDT_ENABLE_bm) | WDT_CEN_bm)) \
00238     : "r0" \
00239 )
00240
00241 #endif // defined(WDT_CTRLA) && !defined(RAMPD)
00242
00243 #elif defined(__AVR_TINY__)
00244
00245 #define wdt_enable(value) \
00246 __asm__ __volatile__ ( \
00247     "in __tmp_reg__, __SREG__" "\n\t" \
00248     "cli" "\n\t" \
00249     "wdr" "\n\t" \
00250     "out %[CCPADDRESS], %[SIGNATURE]" "\n\t" \
00251     "out %[WDTREG], %[WDVALUE]" "\n\t" \
00252     "out __SREG__, __tmp_reg__" \
00253     : /* no outputs */ \
00254     : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)), \
00255     [SIGNATURE] "r" ((uint8_t)0xD8), \
00256     [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)), \
00257     [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00) \
00258     | _BV(WDE) | (value & 0x07) )) \
00259     : "r16" \
00260 )
00261
00262 #define wdt_disable() \
00263 do { \
00264     uint8_t __temp_wd; \
00265     __asm__ __volatile__ ( \
00266         "in __tmp_reg__, __SREG__" "\n\t" \
00267         "cli" "\n\t" \
00268         "wdr" "\n\t" \
00269         "out %[CCPADDRESS], %[SIGNATURE]" "\n\t" \
00270         "in %[TEMP_WD], %[WDTREG]" "\n\t" \
00271         "cbr %[TEMP_WD], %[WDVALUE]" "\n\t" \
00272         "out %[WDTREG], %[TEMP_WD]" "\n\t" \
00273         "out __SREG__, __tmp_reg__" \
00274         : [TEMP_WD] "=d" (__temp_wd) \
00275         : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)), \
00276         [SIGNATURE] "r" ((uint8_t)0xD8), \
00277         [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)), \
00278         [WDVALUE] "n" (1 << WDE) \
00279         : "r16" \
00280     ); \
00281 } while(0)
00282
00283 #elif defined(CCP)
00284
00285 static __ATTR_ALWAYS_INLINE__
00286 void wdt_enable (const uint8_t value)
00287 {
00288     if (!_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
00289     {
00290         __asm__ __volatile__ (
00291             "in __tmp_reg__, __SREG__" "\n\t"
00292             "cli" "\n\t"
00293             "wdr" "\n\t"
00294             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
00295             "sts %[WDTREG], %[WDVALUE]" "\n\t"
00296             "out __SREG__, __tmp_reg__"
00297             : /* no outputs */
00298             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
00299             [SIGNATURE] "x" ((uint8_t)0xD8),
00300             [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
00301             [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00)
00302             | _BV(WDE) | (value & 0x07) ))
00303             : "r0"
00304         );
00305     }
00306     else if (!_SFR_IO_REG_P (CCP) && _SFR_IO_REG_P (_WD_CONTROL_REG))
00307     {
00308         __asm__ __volatile__ (
00309             "in __tmp_reg__, __SREG__" "\n\t"
00310             "cli" "\n\t"
00311             "wdr" "\n\t"
00312             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
00313             "out %[WDTREG], %[WDVALUE]" "\n\t"
00314             "out __SREG__, __tmp_reg__"
00315             : /* no outputs */
00316             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
00317             [SIGNATURE] "x" ((uint8_t)0xD8),
00318             [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
00319             [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00)

```

```

00320         | _BV(WDE) | (value & 0x07) ))
00321         : "r0"
00322     );
00323 }
00324 else if (_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
00325 {
00326     __asm__ __volatile__ (
00327         "in __tmp_reg__, __SREG__" "\n\t"
00328         "cli" "\n\t"
00329         "wdr" "\n\t"
00330         "out %[CCPADDRESS], %[SIGNATURE]" "\n\t"
00331         "sts %[WDTREG], %[WDVALUE]" "\n\t"
00332         "out __SREG__, __tmp_reg__"
00333         : /* no outputs */
00334         : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
00335         [SIGNATURE] "r" ((uint8_t)0xD8),
00336         [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
00337         [WDVALUE] "r" ((uint8_t)(value & 0x08 ? _WD_PS3_MASK : 0x00)
00338         | _BV(WDE) | (value & 0x07) ))
00339         : "r0"
00340     );
00341 }
00342 else
00343 {
00344     __asm__ __volatile__ (
00345         "in __tmp_reg__, __SREG__" "\n\t"
00346         "cli" "\n\t"
00347         "wdr" "\n\t"
00348         "out %[CCPADDRESS], %[SIGNATURE]" "\n\t"
00349         "out %[WDTREG], %[WDVALUE]" "\n\t"
00350         "out __SREG__, __tmp_reg__"
00351         : /* no outputs */
00352         : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
00353         [SIGNATURE] "r" ((uint8_t)0xD8),
00354         [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
00355         [WDVALUE] "r" ((uint8_t)(value & 0x08 ? _WD_PS3_MASK : 0x00)
00356         | _BV(WDE) | (value & 0x07) ))
00357         : "r0"
00358     );
00359 }
00360 }
00361
00362 static __ATTR_ALWAYS_INLINE__
00363 void wdt_disable (void)
00364 {
00365     if (!_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
00366     {
00367         uint8_t __temp_wd;
00368         __asm__ __volatile__ (
00369             "in __tmp_reg__, __SREG__" "\n\t"
00370             "cli" "\n\t"
00371             "wdr" "\n\t"
00372             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
00373             "lds %[TEMP_WD], %[WDTREG]" "\n\t"
00374             "cbr %[TEMP_WD], %[WDVALUE]" "\n\t"
00375             "sts %[WDTREG], %[TEMP_WD]" "\n\t"
00376             "out __SREG__, __tmp_reg__"
00377             : [TEMP_WD] "=d" (__temp_wd)
00378             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
00379             [SIGNATURE] "r" ((uint8_t)0xD8),
00380             [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
00381             [WDVALUE] "n" (1 << WDE)
00382             : "r0"
00383         );
00384     }
00385     else if (!_SFR_IO_REG_P (CCP) && _SFR_IO_REG_P (_WD_CONTROL_REG))
00386     {
00387         uint8_t __temp_wd;
00388         __asm__ __volatile__ (
00389             "in __tmp_reg__, __SREG__" "\n\t"
00390             "cli" "\n\t"
00391             "wdr" "\n\t"
00392             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
00393             "in %[TEMP_WD], %[WDTREG]" "\n\t"
00394             "cbr %[TEMP_WD], %[WDVALUE]" "\n\t"
00395             "out %[WDTREG], %[TEMP_WD]" "\n\t"
00396             "out __SREG__, __tmp_reg__"
00397             : [TEMP_WD] "=d" (__temp_wd)
00398             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
00399             [SIGNATURE] "r" ((uint8_t)0xD8),
00400             [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
00401             [WDVALUE] "n" (1 << WDE)
00402             : "r0"
00403         );
00404     }
00405     else if (_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
00406     {

```

```

00407     uint8_t __temp_wd;
00408     __asm__ __volatile__ (
00409         "in __tmp_reg__, __SREG__ "\n\t"
00410         "cli "\n\t"
00411         "wdr "\n\t"
00412         "out %[CCPADDRESS], %[SIGNATURE] "\n\t"
00413         "lds %[TEMP_WD], %[WDTREG] "\n\t"
00414         "cbr %[TEMP_WD], %[WDVALUE] "\n\t"
00415         "sts %[WDTREG], %[TEMP_WD] "\n\t"
00416         "out __SREG__, __tmp_reg__"
00417         : [TEMP_WD] "=d" (__temp_wd)
00418         : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
00419         [SIGNATURE] "r" ((uint8_t)0xD8),
00420         [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
00421         [WDVALUE] "n" (1 << WDE)
00422         : "r0"
00423         );
00424     }
00425     else
00426     {
00427         uint8_t __temp_wd;
00428         __asm__ __volatile__ (
00429             "in __tmp_reg__, __SREG__ "\n\t"
00430             "cli "\n\t"
00431             "wdr "\n\t"
00432             "out %[CCPADDRESS], %[SIGNATURE] "\n\t"
00433             "in %[TEMP_WD], %[WDTREG] "\n\t"
00434             "cbr %[TEMP_WD], %[WDVALUE] "\n\t"
00435             "out %[WDTREG], %[TEMP_WD] "\n\t"
00436             "out __SREG__, __tmp_reg__"
00437             : [TEMP_WD] "=d" (__temp_wd)
00438             : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
00439             [SIGNATURE] "r" ((uint8_t)0xD8),
00440             [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
00441             [WDVALUE] "n" (1 << WDE)
00442             : "r0"
00443             );
00444     }
00445 }
00446
00447 #else
00448
00449 static __ATTR_ALWAYS_INLINE__
00450 void wdt_enable (const uint8_t value)
00451 {
00452     if (_SFR_IO_REG_P (_WD_CONTROL_REG))
00453     {
00454         __asm__ __volatile__ (
00455             "in __tmp_reg__, __SREG__ "\n\t"
00456             "cli "\n\t"
00457             "wdr "\n\t"
00458             "out %0, %1 "\n\t"
00459             "out __SREG__, __tmp_reg__ "\n\t"
00460             "out %0, %2"
00461             : /* no outputs */
00462             : "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
00463             "r" ((uint8_t)(_BV(_WD_CHANGE_BIT) | _BV(WDE))),
00464             "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00) |
00465             _BV(WDE) | (value & 0x07)) )
00466             : "r0"
00467             );
00468     }
00469     else
00470     {
00471         __asm__ __volatile__ (
00472             "in __tmp_reg__, __SREG__ "\n\t"
00473             "cli "\n\t"
00474             "wdr "\n\t"
00475             "sts %0, %1 "\n\t"
00476             "out __SREG__, __tmp_reg__ "\n\t"
00477             "sts %0, %2"
00478             : /* no outputs */
00479             : "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
00480             "r" ((uint8_t)(_BV(_WD_CHANGE_BIT) | _BV(WDE))),
00481             "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00) |
00482             _BV(WDE) | (value & 0x07)) )
00483             : "r0"
00484             );
00485     }
00486 }
00487
00488 static __ATTR_ALWAYS_INLINE__
00489 void wdt_disable (void)
00490 {
00491     if (_SFR_IO_REG_P (_WD_CONTROL_REG))
00492     {
00493         uint8_t __temp_reg;

```

```

00494     __asm__ __volatile__ (
00495         "in __tmp_reg__, __SREG__"      "\n\t"
00496         "cli"                            "\n\t"
00497         "wdr"                            "\n\t"
00498         "in %[TEMPREG], %[WDTREG]"      "\n\t"
00499         "ori %[TEMPREG], %[WDCE_WDE]"   "\n\t"
00500         "out %[WDTREG], %[TEMPREG]"     "\n\t"
00501         "out %[WDTREG], __zero_reg__"   "\n\t"
00502         "out __SREG__, __tmp_reg__"
00503         : [TEMPREG] "=d" (__tmp_reg)
00504         : [WDTREG] "I"  (_SFR_IO_ADDR(_WD_CONTROL_REG)),
00505           [WDCE_WDE] "n" ((uint8_t) (_BV(_WD_CHANGE_BIT) | _BV(WDE)))
00506         : "r0"
00507     );
00508 }
00509 else
00510 {
00511     uint8_t __tmp_reg;
00512     __asm__ __volatile__ (
00513         "in __tmp_reg__, __SREG__"      "\n\t"
00514         "cli"                            "\n\t"
00515         "wdr"                            "\n\t"
00516         "lds %[TEMPREG], %[WDTREG]"      "\n\t"
00517         "ori %[TEMPREG], %[WDCE_WDE]"   "\n\t"
00518         "sts %[WDTREG], %[TEMPREG]"     "\n\t"
00519         "sts %[WDTREG], __zero_reg__"   "\n\t"
00520         "out __SREG__, __tmp_reg__"
00521         : [TEMPREG] "=d" (__tmp_reg)
00522         : [WDTREG] "n"  (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
00523           [WDCE_WDE] "n" ((uint8_t) (_BV(_WD_CHANGE_BIT) | _BV(WDE)))
00524         : "r0"
00525     );
00526 }
00527 }
00528
00529 #endif
00530
00531
00532 /**
00533  \ingroup avr_watchdog
00534  Symbolic constants for the watchdog timeout. Since the watchdog
00535  timer is based on a free-running RC oscillator, the times are
00536  approximate only and apply to a supply voltage of 5 V. At lower
00537  supply voltages, the times will increase. For older devices, the
00538  times will be as large as three times when operating at Vcc = 3 V,
00539  while the newer devices (e. g. ATmega128, ATmega8) only experience
00540  a negligible change.
00541
00542  Possible timeout values are: 15 ms, 30 ms, 60 ms, 120 ms, 250 ms,
00543  500 ms, 1 s, 2 s. (Some devices also allow for 4 s and 8 s.)
00544  Symbolic constants are formed by the prefix
00545  \c WDTO_, followed by the time.
00546
00547  Example that would select a watchdog timer expiry of approximately
00548  500 ms:
00549  \code
00550  wdt_enable(WDTO_500MS);
00551  \endcode
00552  */
00553 #define WDTO_15MS    0
00554
00555 /** \ingroup avr_watchdog
00556     See \c WDTO_15MS */
00557 #define WDTO_30MS    1
00558
00559 /** \ingroup avr_watchdog
00560     See \c WDTO_15MS */
00561 #define WDTO_60MS    2
00562
00563 /** \ingroup avr_watchdog
00564     See \c WDTO_15MS */
00565 #define WDTO_120MS   3
00566
00567 /** \ingroup avr_watchdog
00568     See \c WDTO_15MS */
00569 #define WDTO_250MS   4
00570
00571 /** \ingroup avr_watchdog
00572     See \c WDTO_15MS */
00573 #define WDTO_500MS   5
00574
00575 /** \ingroup avr_watchdog
00576     See \c WDTO_15MS */
00577 #define WDTO_1S      6
00578
00579 /** \ingroup avr_watchdog
00580     See \c WDTO_15MS */

```

```

00581 #define WDTO_2S      7
00582
00583 #if defined(__DOXYGEN__) || defined(WDP3)
00584
00585 /** \ingroup avr_watchdog
00586     See \c WDTO_15MS
00587     Note: This is only available on the
00588     ATtiny2313,
00589     ATtiny24, ATtiny44, ATtiny84, ATtiny84A,
00590     ATtiny25, ATtiny45, ATtiny85,
00591     ATtiny261, ATtiny461, ATtiny861,
00592     ATmega48*, ATmega88*, ATmega168*, ATmega328*,
00593     ATmega164P, ATmega324P, ATmega324PB, ATmega644P, ATmega644,
00594     ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561,
00595     ATmega8HVA, ATmega16HVA, ATmega32HVB,
00596     ATmega406, ATmega1284P,
00597     AT90PWM1, AT90PWM2, AT90PWM2B, AT90PWM3, AT90PWM3B, AT90PWM216, AT90PWM316,
00598     AT90PWM81, AT90PWM161,
00599     AT90USB82, AT90USB162,
00600     AT90USB646, AT90USB647, AT90USB1286, AT90USB1287,
00601     ATtiny48, ATtiny88.
00602
00603     Note: This value does <em>not</em> match the bit pattern of the
00604     respective control register. It is solely meant to be used together
00605     with wdt_enable().
00606     */
00607 #define WDTO_4S      8
00608
00609 /** \ingroup avr_watchdog
00610     See \c WDTO_15MS
00611     Note: This is only available on the
00612     ATtiny2313,
00613     ATtiny24, ATtiny44, ATtiny84, ATtiny84A,
00614     ATtiny25, ATtiny45, ATtiny85,
00615     ATtiny261, ATtiny461, ATtiny861,
00616     ATmega48*, ATmega88*, ATmega168*, ATmega328*,
00617     ATmega164P, ATmega324P, ATmega324PB, ATmega644P, ATmega644,
00618     ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561,
00619     ATmega8HVA, ATmega16HVA, ATmega32HVB,
00620     ATmega406, ATmega1284P,
00621     ATmega2564RFR2, ATmega256RFR2, ATmega1284RFR2, ATmega128RFR2, ATmega644RFR2, ATmega64RFR2
00622     AT90PWM1, AT90PWM2, AT90PWM2B, AT90PWM3, AT90PWM3B, AT90PWM216, AT90PWM316,
00623     AT90PWM81, AT90PWM161,
00624     AT90USB82, AT90USB162,
00625     AT90USB646, AT90USB647, AT90USB1286, AT90USB1287,
00626     ATtiny48, ATtiny88,
00627     ATxmegal6a4u, ATxmega32a4u,
00628     ATxmegal6c4, ATxmega32c4,
00629     ATxmegal28c3, ATxmegal92c3, ATxmega256c3.
00630
00631     Note: This value does <em>not</em> match the bit pattern of the
00632     respective control register. It is solely meant to be used together
00633     with wdt_enable().
00634     */
00635 #define WDTO_8S      9
00636
00637 #endif /* defined(__DOXYGEN__) || defined(WDP3) */
00638
00639
00640 #endif /* _AVR_WDT_H_ */

```

## 23.39 xmega.h

```

00001 /* Copyright (c) 2012 Joerg Wunsch
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011     notice, this list of conditions and the following disclaimer in
00012     the documentation and/or other materials provided with the
00013     distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016     contributors may be used to endorse or promote products derived
00017     from this software without specific prior written permission.
00018
00019     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

```

```

00022 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029 POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /*
00034 * This file is included by <avr/io.h> whenever compiling for an Xmega
00035 * device. It abstracts certain features common to the Xmega device
00036 * families.
00037 */
00038
00039 #ifndef _AVR_XMEGA_H
00040 #define _AVR_XMEGA_H
00041
00042 #ifdef __DOXYGEN__
00043 /**
00044  \def _PROTECTED_WRITE
00045  \ingroup avr_io
00046
00047  Write value \c value to IO register \c reg that is protected through
00048  the Xmega configuration change protection (CCP) mechanism. This
00049  implements the timed sequence that is required for CCP.
00050
00051  Example to modify the CPU clock:
00052  \code
00053  #include <avr/io.h>
00054
00055  _PROTECTED_WRITE(CLK_PSCTRL, CLK_PSADIV0_bm);
00056  _PROTECTED_WRITE(CLK_CTRL, CLK_SCLKSEL0_bm);
00057  \endcode
00058  */
00059 #define _PROTECTED_WRITE(reg, value)
00060
00061 /**
00062  \def _PROTECTED_WRITE_SPM
00063  \ingroup avr_io
00064
00065  Write value \c value to register \c reg that is protected through
00066  the Xmega configuration change protection (CCP) key for self
00067  programming (SPM). This implements the timed sequence that is
00068  required for CCP.
00069
00070  Example to modify the CPU clock:
00071  \code
00072  #include <avr/io.h>
00073
00074  _PROTECTED_WRITE_SPM(NVMCTRL_CTRLA, NVMCTRL_CMD_PAGEERASEWRITE_gc);
00075  \endcode
00076  */
00077 #define _PROTECTED_WRITE_SPM(reg, value)
00078
00079 #else /* !__DOXYGEN__ */
00080
00081 #define _PROTECTED_WRITE(reg, value)
00082   __asm__ __volatile__ ("out %[ccp], %[ccp_ioreg]" "\n\t" \
00083       "sts %[ioreg], %[val]" \
00084       : \
00085       : [ccp] "I" (_SFR_IO_ADDR(CCP)), \
00086       [ccp_ioreg] "d" ((uint8_t)CCP_IOREG_gc), \
00087       [ioreg] "n" (_SFR_MEM_ADDR(reg)), \
00088       [val] "r" ((uint8_t)value))
00089
00090 #define _PROTECTED_WRITE_SPM(reg, value) \
00091   __asm__ __volatile__ ("out %[ccp], %[ccp_spm_mask]" "\n\t" \
00092       "sts %[ioreg], %[val]" \
00093       : \
00094       : [ccp] \
00095       : [ccp_spm_mask] "I" (_SFR_IO_ADDR(CCP)), \
00096       [ioreg] "d" ((uint8_t)CCP_SPM_gc), \
00097       [val] "r" ((uint8_t)value))
00098 #endif /* DOXYGEN */
00099
00100 #endif /* _AVR_XMEGA_H */

```

## 23.40 deprecated.h

```

00001 /* Copyright (c) 2005,2006 Joerg Wunsch
00002    All rights reserved.

```

```

00003
00004 Redistribution and use in source and binary forms, with or without
00005 modification, are permitted provided that the following conditions are met:
00006
00007 * Redistributions of source code must retain the above copyright
00008 notice, this list of conditions and the following disclaimer.
00009
00010 * Redistributions in binary form must reproduce the above copyright
00011 notice, this list of conditions and the following disclaimer in
00012 the documentation and/or other materials provided with the
00013 distribution.
00014
00015 * Neither the name of the copyright holders nor the names of
00016 contributors may be used to endorse or promote products derived
00017 from this software without specific prior written permission.
00018
00019 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029 POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 #ifndef _COMPAT_DEPRECATED_H_
00034 #define _COMPAT_DEPRECATED_H_
00035
00036 /** \defgroup deprecated_items <compat/deprecated.h>: Deprecated items
00037
00038 This header file contains several items that used to be available
00039 in previous versions of this library, but have eventually been
00040 deprecated over time.
00041
00042 \code #include <compat/deprecated.h> \endcode
00043
00044 These items are supplied within that header file for backward
00045 compatibility reasons only, so old source code that has been
00046 written for previous library versions could easily be maintained
00047 until its end-of-life. Use of any of these items in new code is
00048 strongly discouraged.
00049 */
00050
00051 /** \name Allowing specific system-wide interrupts
00052
00053 In addition to globally enabling interrupts, each device's particular
00054 interrupt needs to be enabled separately if interrupts for this device are
00055 desired. While some devices maintain their interrupt enable bit inside
00056 the device's register set, external and timer interrupts have system-wide
00057 configuration registers.
00058
00059 Example:
00060
00061 \code
00062 // Enable timer 1 overflow interrupts.
00063 timer_enable_int(_BV(TOIE1));
00064
00065 // Do some work...
00066
00067 // Disable all timer interrupts.
00068 timer_enable_int(0);
00069 \endcode
00070
00071 \note Be careful when you use these functions. If you already have a
00072 different interrupt enabled, you could inadvertently disable it by
00073 enabling another interrupt. */
00074
00075 /**@{*/
00076
00077 /** \ingroup deprecated_items
00078 \def enable_external_int(mask)
00079 \deprecated
00080
00081 This macro gives access to the \c GIMSK register (or \c EIMSK register
00082 if using an AVR Mega device or \c GICR register for others). Although this
00083 macro is essentially the same as assigning to the register, it does
00084 adapt slightly to the type of device being used. This macro is
00085 unavailable if none of the registers listed above are defined. */
00086
00087 /* Define common register definition if available. */
00088 #if defined(EIMSK)
00089 # define __EICR EIMSK

```

```

00090 #elif defined(GIMSK)
00091 # define __EICR GIMSK
00092 #elif defined(GICR)
00093 # define __EICR GICR
00094 #endif
00095
00096 /* If common register defined, define macro. */
00097 #if defined(__EICR) || defined(__DOXYGEN__)
00098 #define enable_external_int(mask)          (__EICR = mask)
00099 #endif
00100
00101 /** \ingroup deprecated_items
00102     \deprecated
00103
00104     This function modifies the \c timsk register.
00105     The value you pass via \c ints is device specific. */
00106
00107 static __inline__ void timer_enable_int (unsigned char ints)
00108 {
00109 #ifdef TIMSK
00110     TIMSK = ints;
00111 #endif
00112 }
00113
00114 /** \def INTERRUPT(signame)
00115     \ingroup deprecated_items
00116     \deprecated
00117
00118     Introduces an interrupt handler function that runs with global interrupts
00119     initially enabled. This allows interrupt handlers to be interrupted.
00120
00121     As this macro has been used by too many unsuspecting people in the
00122     past, it has been deprecated, and will be removed in a future
00123     version of the library. Users who want to legitimately re-enable
00124     interrupts in their interrupt handlers as quickly as possible are
00125     encouraged to explicitly declare their handlers as described
00126     \ref attr_interrupt "above".
00127 */
00128
00129 #if (__GNUC__ == 4 && __GNUC_MINOR__ >= 1) || (__GNUC__ > 4)
00130 # define __INTR_ATTRS __used__, __externally_visible__
00131 #else /* GCC < 4.1 */
00132 # define __INTR_ATTRS __used__
00133 #endif
00134
00135 #ifdef __cplusplus
00136 #define INTERRUPT(signame) \
00137 extern "C" void signame(void); \
00138 void signame (void) __attribute__ ((__interrupt__, __INTR_ATTRS)); \
00139 void signame (void)
00140 #else
00141 #define INTERRUPT(signame) \
00142 void signame (void) __attribute__ ((__interrupt__, __INTR_ATTRS)); \
00143 void signame (void)
00144 #endif
00145
00146 /**@}*/
00147
00148 /**
00149     \name Obsolete IO macros
00150
00151     Back in a time when AVR-GCC and AVR-LibC could not handle IO port
00152     access in the direct assignment form as they are handled now, all
00153     IO port access had to be done through specific macros that
00154     eventually resulted in inline assembly instructions performing the
00155     desired action.
00156
00157     These macros became obsolete, as reading and writing IO ports can
00158     be done by simply using the IO port name in an expression, and all
00159     bit manipulation (including those on IO ports) can be done using
00160     generic C bit manipulation operators.
00161
00162     The macros in this group simulate the historical behaviour. While
00163     they are supposed to be applied to IO ports, the emulation actually
00164     uses standard C methods, so they could be applied to arbitrary
00165     memory locations as well.
00166 */
00167
00168 /**@{*/
00169
00170 /**
00171     \ingroup deprecated_items
00172     \def inp(port)
00173     \deprecated
00174
00175     Read a value from an IO port \c port.
00176 */

```



```

00177 #define inp(port) (port)
00178
00179 /**
00180  \ingroup deprecated_items
00181  \def outp(val, port)
00182  \deprecated
00183
00184  Write \c val to IO port \c port.
00185 */
00186 #define outp(val, port) (port) = (val)
00187
00188 /**
00189  \ingroup deprecated_items
00190  \def inb(port)
00191  \deprecated
00192
00193  Read a value from an IO port \c port.
00194 */
00195 #define inb(port) (port)
00196
00197 /**
00198  \ingroup deprecated_items
00199  \def outb(port, val)
00200  \deprecated
00201
00202  Write \c val to IO port \c port.
00203 */
00204 #define outb(port, val) (port) = (val)
00205
00206 /**
00207  \ingroup deprecated_items
00208  \def sbi(port, bit)
00209  \deprecated
00210
00211  Set \c bit in IO port \c port.
00212 */
00213 #define sbi(port, bit) (port) |= (1 << (bit))
00214
00215 /**
00216  \ingroup deprecated_items
00217  \def cbi(port, bit)
00218  \deprecated
00219
00220  Clear \c bit in IO port \c port.
00221 */
00222 #define cbi(port, bit) (port) &= ~(1 << (bit))
00223
00224 /**@}*/
00225
00226 #endif /* _COMPAT_DEPRECATED_H_ */

```

## 23.41 ina90.h

```

00001 /* Copyright (c) 2002,2004 Marek Michalkiewicz
00002  All rights reserved.
00003
00004  Redistribution and use in source and binary forms, with or without
00005  modification, are permitted provided that the following conditions are met:
00006
00007  * Redistributions of source code must retain the above copyright
00008  notice, this list of conditions and the following disclaimer.
00009
00010  * Redistributions in binary form must reproduce the above copyright
00011  notice, this list of conditions and the following disclaimer in
00012  the documentation and/or other materials provided with the
00013  distribution.
00014
00015  * Neither the name of the copyright holders nor the names of
00016  contributors may be used to endorse or promote products derived
00017  from this software without specific prior written permission.
00018
00019  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029  POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */

```

```

00032 /* copied from: Id: avr/ina90.h,v 1.8 2004/11/09 19:16:09 arcanum Exp */
00033
00034 /*
00035     ina90.h
00036
00037     Contributors:
00038     Created by Marek Michalkiewicz <marekm@linux.org.pl>
00039 */
00040
00041 /**
00042     \defgroup compat_ina90 <compat/ina90.h>: Compatibility with IAR EWB 3.x
00043
00044     \code #include <compat/ina90.h> \endcode
00045
00046     This is an attempt to provide some compatibility with
00047     header files that come with IAR C, to make porting applications
00048     between different compilers easier. No 100% compatibility though.
00049
00050     \note For actual documentation, please see the IAR manual.
00051 */
00052
00053 #ifndef _INA90_H_
00054 #define _INA90_H_ 1
00055
00056 #define _CLI() do { __asm__ __volatile__ ("cli"); } while (0)
00057 #define _SEI() do { __asm__ __volatile__ ("sei"); } while (0)
00058 #define _NOP() do { __asm__ __volatile__ ("nop"); } while (0)
00059 #define _WDR() do { __asm__ __volatile__ ("wdr"); } while (0)
00060 #define _SLEEP() do { __asm__ __volatile__ ("sleep"); } while (0)
00061 #define _OPC(op) do { __asm__ __volatile__ (".word %0" : : "n" (op)); } while (0)
00062
00063 /* _LPM, _ELPM */
00064 #include <avr/pgmspace.h>
00065 #define _LPM(x) do { __LPM(x); } while (0)
00066 #define _ELPM(x) do { __ELPM(x); } while (0)
00067
00068 /* _EGET, _EEPWRITE */
00069 #include <avr/eeprom.h>
00070
00071 #define input(port) (port)
00072 #define output(port, val) do { (port) = (val); } while (0)
00073
00074 #define __inp_blk__(port, addr, cnt, op) do { \
00075     unsigned char __i = (cnt); \
00076     unsigned char *__addr = (addr); \
00077     while (__i) { \
00078         *(__addr op) = input(port); \
00079         __i--; \
00080     } \
00081 } while (0)
00082
00083 #define input_block_inc(port, addr, cnt) __inp_blk__(port, addr, cnt, ++)
00084 #define input_block_dec(port, addr, cnt) __inp_blk__(port, addr, cnt, --)
00085
00086 #define __out_blk__(port, addr, cnt, op) do { \
00087     unsigned char __i = (cnt); \
00088     const unsigned char *__addr = (addr); \
00089     while (__i) { \
00090         output(port, *(__addr op)); \
00091         __i--; \
00092     } \
00093 } while (0)
00094
00095 #define output_block_inc(port, addr, cnt) __out_blk__(port, addr, cnt, ++)
00096 #define output_block_dec(port, addr, cnt) __out_blk__(port, addr, cnt, --)
00097
00098 #endif
00099

```

## 23.42 ctype.h File Reference

### Functions

#### Character classification routines

These functions perform character classification. They return true or false status depending whether the character passed to the function falls into the function's classification (i.e. `isdigit()` returns true if its argument is any value '0' though '9', inclusive). If the input is not an unsigned char value, all of this function return false.

- `int isalnum (int __c)`

- `int isalpha (int __c)`
- `int isascii (int __c)`
- `int isblank (int __c)`
- `int iscntrl (int __c)`
- `int isdigit (int __c)`
- `int isgraph (int __c)`
- `int islower (int __c)`
- `int isprint (int __c)`
- `int ispunct (int __c)`
- `int isspace (int __c)`
- `int isupper (int __c)`
- `int isxdigit (int __c)`

### Character conversion routines

This realization permits all possible values of integer argument. The `toascii()` function clears all highest bits. The `tolower()` and `toupper()` functions return an input argument as is, if it is not an unsigned char value.

- `int toascii (int __c)`
- `int tolower (int __c)`
- `int toupper (int __c)`

## 23.43 ctype.h

[Go to the documentation of this file.](#)

```
00001 /* Copyright (c) 2002,2007 Michael Stumpf
00002    All rights reserved.
00003
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions are met:
00006
00007    * Redistributions of source code must retain the above copyright
00008    notice, this list of conditions and the following disclaimer.
00009
00010    * Redistributions in binary form must reproduce the above copyright
00011    notice, this list of conditions and the following disclaimer in
00012    the documentation and/or other materials provided with the
00013    distribution.
00014
00015    * Neither the name of the copyright holders nor the names of
00016    contributors may be used to endorse or promote products derived
00017    from this software without specific prior written permission.
00018
00019    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029    POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /*
00034    ctype.h - character conversion macros and ctype macros
00035
00036    Author : Michael Stumpf
00037            Michael.Stumpf@t-online.de
00038 */
00039
00040 #ifndef __CTYPE_H_
00041 #define __CTYPE_H_ 1
00042
00043 #ifdef __cplusplus
00044 extern "C" {
00045 #endif
00046
00047 /** \file */
00048 /** \defgroup ctype <ctype.h>: Character Operations
00049     These functions perform various operations on characters.
00050
```

```
00051     \code #include <ctype.h>\endcode
00052
00053 */
00054
00055 /** \name Character classification routines
00056
00057     These functions perform character classification. They return true or
00058     false status depending whether the character passed to the function falls
00059     into the function's classification (i.e. isdigit() returns true if its
00060     argument is any value '0' though '9', inclusive). If the input is not
00061     an unsigned char value, all of this function return false. */
00062
00063 /**@{*/
00064
00065 /** \ingroup ctype
00066
00067     Checks for an alphanumeric character. It is equivalent to <tt>(isalpha(c)
00068     || isdigit(c)</tt>. */
00069
00070 extern int isalnum(int __c);
00071
00072 /** \ingroup ctype
00073
00074     Checks for an alphabetic character. It is equivalent to <tt>(isupper(c) ||
00075     islower(c)</tt>. */
00076
00077 extern int isalpha(int __c);
00078
00079 /** \ingroup ctype
00080
00081     Checks whether \c c is a 7-bit unsigned char value that fits into the
00082     ASCII character set. */
00083
00084 extern int isascii(int __c);
00085
00086 /** \ingroup ctype
00087
00088     Checks for a blank character, that is, a space or a tab. */
00089
00090 extern int isblank(int __c);
00091
00092 /** \ingroup ctype
00093
00094     Checks for a control character. */
00095
00096 extern int iscntrl(int __c);
00097
00098 /** \ingroup ctype
00099
00100     Checks for a digit (0 through 9). */
00101
00102 extern int isdigit(int __c);
00103
00104 /** \ingroup ctype
00105
00106     Checks for any printable character except space. */
00107
00108 extern int isgraph(int __c);
00109
00110 /** \ingroup ctype
00111
00112     Checks for a lower-case character. */
00113
00114 extern int islower(int __c);
00115
00116 /** \ingroup ctype
00117
00118     Checks for any printable character including space. */
00119
00120 extern int isprint(int __c);
00121
00122 /** \ingroup ctype
00123
00124     Checks for any printable character which is not a space or an alphanumeric
00125     character. */
00126
00127 extern int ispunct(int __c);
00128
00129 /** \ingroup ctype
00130
00131     Checks for white-space characters. For the AVR-LibC library, these are:
00132     space, form-feed ('\f'), newline ('\n'), carriage return ('\r'),
00133     horizontal tab ('\t'), and vertical tab ('\v'). */
00134
00135 extern int isspace(int __c);
00136
00137 /** \ingroup ctype
```

```

00138
00139     Checks for an uppercase letter. */
00140
00141 extern int isupper(int __c);
00142
00143 /** \ingroup ctype
00144
00145     Checks for a hexadecimal digits, i.e. one of 0 1 2 3 4 5 6 7 8 9 a b c d e
00146     f A B C D E F. */
00147
00148 extern int isxdigit(int __c);
00149
00150 /**@}*/
00151
00152 /** \name Character conversion routines
00153
00154     This realization permits all possible values of integer argument.
00155     The toascii() function clears all highest bits. The tolower() and
00156     toupper() functions return an input argument as is, if it is not an
00157     unsigned char value. */
00158
00159 /**@{*/
00160
00161 /** \ingroup ctype
00162
00163     Converts \c c to a 7-bit unsigned char value that fits into the ASCII
00164     character set, by clearing the high-order bits.
00165
00166     \warning Many people will be unhappy if you use this function. This
00167     function will convert accented letters into random characters. */
00168
00169 extern int toascii(int __c);
00170
00171 /** \ingroup ctype
00172
00173     Converts the letter \c c to lower case, if possible. */
00174
00175 extern int tolower(int __c);
00176
00177 /** \ingroup ctype
00178
00179     Converts the letter \c c to upper case, if possible. */
00180
00181 extern int toupper(int __c);
00182
00183 /**@}*/
00184
00185 #ifdef __cplusplus
00186 }
00187 #endif
00188
00189 #endif

```

## 23.44 errno.h File Reference

### Macros

- #define [EDOM](#) 33
- #define [ERANGE](#) 34

### Variables

- int [errno](#)

## 23.45 errno.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002,2007 Marek Michalkiewicz
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:

```

```

00006
00007 * Redistributions of source code must retain the above copyright
00008 notice, this list of conditions and the following disclaimer.
00009
00010 * Redistributions in binary form must reproduce the above copyright
00011 notice, this list of conditions and the following disclaimer in
00012 the documentation and/or other materials provided with the
00013 distribution.
00014
00015 * Neither the name of the copyright holders nor the names of
00016 contributors may be used to endorse or promote products derived
00017 from this software without specific prior written permission.
00018
00019 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029 POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 #ifndef __ERRNO_H_
00034 #define __ERRNO_H_ 1
00035
00036 /** \file */
00037 /** \defgroup avr_errno <errno.h>: System Errors
00038
00039 \code #include <errno.h>\endcode
00040
00041 Some functions in the library set the global variable \c errno when an
00042 error occurs. The file, \c <errno.h>, provides symbolic names for various
00043 error codes.
00044 */
00045
00046 #ifdef __cplusplus
00047 extern "C" {
00048 #endif
00049
00050 /** \ingroup avr_errno
00051 \brief Error code for last error encountered by library
00052
00053 The variable \c errno holds the last error code encountered by
00054 a library function. This variable must be cleared by the
00055 user prior to calling a library function.
00056
00057 \warning The \c errno global variable is not safe to use in a threaded or
00058 multi-task system. A race condition can occur if a task is interrupted
00059 between the call which sets \c error and when the task examines \c
00060 errno. If another task changes \c errno during this time, the result will
00061 be incorrect for the interrupted task. */
00062 extern int errno;
00063
00064 #ifdef __cplusplus
00065 }
00066 #endif
00067
00068 /** \ingroup avr_errno
00069 \def EDOM
00070
00071 Domain error. */
00072 #define EDOM 33
00073
00074 /** \ingroup avr_errno
00075 \def ERANGE
00076
00077 Range error. */
00078 #define ERANGE 34
00079
00080 #ifndef __DOXYGEN__
00081
00082 /* (((('E'-64)*26+('N'-64))*26+('O'-64))*26+('S'-64))*26+('Y'-64))*26+'S'-64 */
00083 #define ENOSYS ((int)(66081697 & 0x7fff))
00084
00085 /* (((('E'-64)*26+('I'-64))*26+('N'-64))*26+('T'-64))*26+('R'-64) */
00086 #define EINTR ((int)(2453066 & 0x7fff))
00087
00088 #define E2BIG ENOERR
00089 #define EACCES ENOERR
00090 #define EADDRINUSE ENOERR
00091 #define EADDRNOTAVAIL ENOERR
00092 #define EAFNOSUPPORT ENOERR

```

```

00093 #define EAGAIN ENOERR
00094 #define EALREADY ENOERR
00095 #define EBADF ENOERR
00096 #define EBUSY ENOERR
00097 #define ECHILD ENOERR
00098 #define ECONNABORTED ENOERR
00099 #define ECONNREFUSED ENOERR
00100 #define ECONNRESET ENOERR
00101 #define EDEADLK ENOERR
00102 #define EDESTADDRREQ ENOERR
00103 #define EEXIST ENOERR
00104 #define EFAULT ENOERR
00105 #define EFBIG ENOERR
00106 #define EHOSTUNREACH ENOERR
00107 #define EILSEQ ENOERR
00108 #define EINPROGRESS ENOERR
00109 #define EINVAL ENOERR
00110 #define EIO ENOERR
00111 #define EISCONN ENOERR
00112 #define EISDIR ENOERR
00113 #define ELOOP ENOERR
00114 #define EMFILE ENOERR
00115 #define EMLINK ENOERR
00116 #define EMSGSIZE ENOERR
00117 #define ENAMETOOLONG ENOERR
00118 #define ENETDOWN ENOERR
00119 #define ENETRESET ENOERR
00120 #define ENETUNREACH ENOERR
00121 #define ENFILE ENOERR
00122 #define ENOBUFS ENOERR
00123 #define ENODEV ENOERR
00124 #define ENOENT ENOERR
00125 #define ENOEXEC ENOERR
00126 #define ENOLCK ENOERR
00127 #define ENOMEM ENOERR
00128 #define ENOMSG ENOERR
00129 #define ENOPROTOPT ENOERR
00130 #define ENOSPC ENOERR
00131 #define ENOTCONN ENOERR
00132 #define ENOTDIR ENOERR
00133 #define ENOTEMPTY ENOERR
00134 #define ENOTSOCK ENOERR
00135 #define ENOTTY ENOERR
00136 #define ENXIO ENOERR
00137 #define EOPNOTSUPP ENOERR
00138 #define EPERM ENOERR
00139 #define EPIPE ENOERR
00140 #define EPROTONOSUPPORT ENOERR
00141 #define EPROTOTYPE ENOERR
00142 #define EROFS ENOERR
00143 #define ESPIPE ENOERR
00144 #define ESRCH ENOERR
00145 #define ETIMEDOUT ENOERR
00146 #define EWOULDBLOCK ENOERR
00147 #define EXDEV ENOERR
00148
00149 /* (((('E'-64)*26+('N'-64))*26+('O'-64))*26+('E'-64))*26+('R'-64))*26+'R'-64 */
00150 #define ENOERR ((int)(66072050 & 0xffff))
00151
00152 #endif /* !__DOXYGEN__ */
00153
00154 #endif

```

## 23.46 inttypes.h File Reference

### Macros

#### macros for printf and scanf format specifiers

For C++, these are only included if `__STDC_LIMIT_MACROS` is defined before including `<inttypes.h>`.

- `#define PRId8 "d"`
- `#define PRIdLEAST8 "d"`
- `#define PRIdFAST8 "d"`
- `#define PRIi8 "i"`
- `#define PRIiLEAST8 "i"`
- `#define PRIiFAST8 "i"`
- `#define PRId16 "d"`

- #define [PRIdLEAST16](#) "d"
- #define [PRIdFAST16](#) "d"
- #define [PRIi16](#) "i"
- #define [PRIiLEAST16](#) "i"
- #define [PRIiFAST16](#) "i"
- #define [PRId32](#) "ld"
- #define [PRIdLEAST32](#) "ld"
- #define [PRIdFAST32](#) "ld"
- #define [PRIi32](#) "li"
- #define [PRIiLEAST32](#) "li"
- #define [PRIiFAST32](#) "li"
- #define [PRIdPTR](#) [PRId16](#)
- #define [PRIiPTR](#) [PRIi16](#)
- #define [PRIo8](#) "o"
- #define [PRIoLEAST8](#) "o"
- #define [PRIoFAST8](#) "o"
- #define [PRIu8](#) "u"
- #define [PRIuLEAST8](#) "u"
- #define [PRIuFAST8](#) "u"
- #define [PRIx8](#) "x"
- #define [PRIxLEAST8](#) "x"
- #define [PRIxFAST8](#) "x"
- #define [PRIX8](#) "X"
- #define [PRIXLEAST8](#) "X"
- #define [PRIXFAST8](#) "X"
- #define [PRIo16](#) "o"
- #define [PRIoLEAST16](#) "o"
- #define [PRIoFAST16](#) "o"
- #define [PRIu16](#) "u"
- #define [PRIuLEAST16](#) "u"
- #define [PRIuFAST16](#) "u"
- #define [PRIx16](#) "x"
- #define [PRIxLEAST16](#) "x"
- #define [PRIxFAST16](#) "x"
- #define [PRIX16](#) "X"
- #define [PRIXLEAST16](#) "X"
- #define [PRIXFAST16](#) "X"
- #define [PRIo32](#) "lo"
- #define [PRIoLEAST32](#) "lo"
- #define [PRIoFAST32](#) "lo"
- #define [PRIu32](#) "lu"
- #define [PRIuLEAST32](#) "lu"
- #define [PRIuFAST32](#) "lu"
- #define [PRIx32](#) "lx"
- #define [PRIxLEAST32](#) "lx"
- #define [PRIxFAST32](#) "lx"
- #define [PRIX32](#) "lX"
- #define [PRIXLEAST32](#) "lX"
- #define [PRIXFAST32](#) "lX"
- #define [PRIoPTR](#) [PRIo16](#)
- #define [PRIuPTR](#) [PRIu16](#)
- #define [PRIXPTR](#) [PRIX16](#)
- #define [PRIXPTR](#) [PRIX16](#)
- #define [SCNd8](#) "hhd"
- #define [SCNdLEAST8](#) "hhd"
- #define [SCNdFAST8](#) "hhd"
- #define [SCNi8](#) "hhi"
- #define [SCNiLEAST8](#) "hhi"
- #define [SCNiFAST8](#) "hhi"
- #define [SCNd16](#) "d"
- #define [SCNdLEAST16](#) "d"
- #define [SCNdFAST16](#) "d"
- #define [SCNi16](#) "i"
- #define [SCNiLEAST16](#) "i"



- #define SCNiFAST16 "i"
- #define SCNd32 "ld"
- #define SCNdLEAST32 "ld"
- #define SCNdFAST32 "ld"
- #define SCNi32 "li"
- #define SCNiLEAST32 "li"
- #define SCNiFAST32 "li"
- #define SCNdPTR SCNd16
- #define SCNiPTR SCNi16
- #define SCNo8 "hho"
- #define SCNoLEAST8 "hho"
- #define SCNoFAST8 "hho"
- #define SCNu8 "hhu"
- #define SCNuLEAST8 "hhu"
- #define SCNuFAST8 "hhu"
- #define SCNx8 "hhx"
- #define SCNxLEAST8 "hhx"
- #define SCNxFAST8 "hhx"
- #define SCNo16 "o"
- #define SCNoLEAST16 "o"
- #define SCNoFAST16 "o"
- #define SCNu16 "u"
- #define SCNuLEAST16 "u"
- #define SCNuFAST16 "u"
- #define SCNx16 "x"
- #define SCNxLEAST16 "x"
- #define SCNxFAST16 "x"
- #define SCNo32 "lo"
- #define SCNoLEAST32 "lo"
- #define SCNoFAST32 "lo"
- #define SCNu32 "lu"
- #define SCNuLEAST32 "lu"
- #define SCNuFAST32 "lu"
- #define SCNx32 "lx"
- #define SCNxLEAST32 "lx"
- #define SCNxFAST32 "lx"
- #define SCNoPTR SCNo16
- #define SCNuPTR SCNu16
- #define SCNxPTR SCNx16

## Typedefs

### Far pointers for memory access > 64K

- typedef `int32_t int_farptr_t`
- typedef `uint32_t uint_farptr_t`

## 23.47 inttypes.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2004,2005,2007,2012 Joerg Wunsch
00002 Copyright (c) 2005, Carlos Lamas
00003 All rights reserved.
00004
00005 Redistribution and use in source and binary forms, with or without
00006 modification, are permitted provided that the following conditions are met:
00007
00008 * Redistributions of source code must retain the above copyright
00009 notice, this list of conditions and the following disclaimer.
00010
00011 * Redistributions in binary form must reproduce the above copyright
00012 notice, this list of conditions and the following disclaimer in
00013 the documentation and/or other materials provided with the
00014 distribution.
```

```

00015
00016 * Neither the name of the copyright holders nor the names of
00017 contributors may be used to endorse or promote products derived
00018 from this software without specific prior written permission.
00019
00020 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00022 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030 POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033
00034 #ifndef __INTTYPES_H_
00035 #define __INTTYPES_H_
00036
00037 #include <stdint.h>
00038
00039 /** \file */
00040 /** \defgroup avr_inttypes <inttypes.h>: Integer Type conversions
00041     \code #include <inttypes.h> \endcode
00042
00043     This header file includes the exact-width integer definitions from
00044     <tt><stdint.h></tt>, and extends them with additional facilities
00045     provided by the implementation.
00046
00047     Currently, the extensions include two additional integer types
00048     that could hold a "far" pointer (i.e. a code pointer that can
00049     address more than 64 KB), as well as standard names for all printf
00050     and scanf formatting options that are supported by the \ref avr_stdio.
00051     As the library does not support the full range of conversion
00052     specifiers from ISO 9899:1999, only those conversions that are
00053     actually implemented will be listed here.
00054
00055     The idea behind these conversion macros is that, for each of the
00056     types defined by <stdint.h>, a macro will be supplied that portably
00057     allows formatting an object of that type in printf() or scanf()
00058     operations. Example:
00059
00060     \code
00061     #include <inttypes.h>
00062
00063     uint8_t smallval;
00064     int32_t longval;
00065     ...
00066     printf("The hexadecimal value of smallval is %" PRIx8
00067           ", the decimal value of longval is %" PRId32 ".\n",
00068           smallval, longval);
00069     \endcode
00070 */
00071
00072 /** \name Far pointers for memory access > 64K */
00073
00074 /**@{*/
00075 /** \ingroup avr_inttypes
00076     signed integer type that can hold a pointer > 64 KiB */
00077 typedef int32_t int_farptr_t;
00078
00079 /** \ingroup avr_inttypes
00080     unsigned integer type that can hold a pointer > 64 KiB,
00081     see also pgm_get_far_address()
00082 */
00083 typedef uint32_t uint_farptr_t;
00084 /**@}*/
00085
00086 #if !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS)
00087
00088
00089 /** \name macros for printf and scanf format specifiers
00090
00091     For C++, these are only included if __STDC_LIMIT_MACROS
00092     is defined before including <inttypes.h>.
00093 */
00094
00095 /**@{*/
00096 /** \ingroup avr_inttypes
00097     decimal printf format for int8_t */
00098 #define PRId8 "d"
00099 /** \ingroup avr_inttypes
00100     decimal printf format for int_least8_t */
00101 #define PRIdLEAST8 "d"

```

```
00102 /** \ingroup avr_inttypes
00103     decimal printf format for int_fast8_t */
00104 #define     PRI_FAST8      "d"
00105
00106 /** \ingroup avr_inttypes
00107     integer printf format for int8_t */
00108 #define     PRI_i8        "i"
00109 /** \ingroup avr_inttypes
00110     integer printf format for int_least8_t */
00111 #define     PRI_LEAST8    "i"
00112 /** \ingroup avr_inttypes
00113     integer printf format for int_fast8_t */
00114 #define     PRI_FAST8     "i"
00115
00116
00117 /** \ingroup avr_inttypes
00118     decimal printf format for int16_t */
00119 #define     PRI_d16       "d"
00120 /** \ingroup avr_inttypes
00121     decimal printf format for int_least16_t */
00122 #define     PRI_LEAST16   "d"
00123 /** \ingroup avr_inttypes
00124     decimal printf format for int_fast16_t */
00125 #define     PRI_FAST16    "d"
00126
00127 /** \ingroup avr_inttypes
00128     integer printf format for int16_t */
00129 #define     PRI_i16       "i"
00130 /** \ingroup avr_inttypes
00131     integer printf format for int_least16_t */
00132 #define     PRI_LEAST16   "i"
00133 /** \ingroup avr_inttypes
00134     integer printf format for int_fast16_t */
00135 #define     PRI_FAST16    "i"
00136
00137
00138 /** \ingroup avr_inttypes
00139     decimal printf format for int32_t */
00140 #define     PRI_d32       "ld"
00141 /** \ingroup avr_inttypes
00142     decimal printf format for int_least32_t */
00143 #define     PRI_LEAST32   "ld"
00144 /** \ingroup avr_inttypes
00145     decimal printf format for int_fast32_t */
00146 #define     PRI_FAST32    "ld"
00147
00148 /** \ingroup avr_inttypes
00149     integer printf format for int32_t */
00150 #define     PRI_i32       "li"
00151 /** \ingroup avr_inttypes
00152     integer printf format for int_least32_t */
00153 #define     PRI_LEAST32   "li"
00154 /** \ingroup avr_inttypes
00155     integer printf format for int_fast32_t */
00156 #define     PRI_FAST32    "li"
00157
00158
00159 #ifdef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
00160
00161 #define     PRI_d64        "lld"
00162 #define     PRI_LEAST64    "lld"
00163 #define     PRI_FAST64     "lld"
00164
00165 #define     PRI_i64        "lli"
00166 #define     PRI_LEAST64    "lli"
00167 #define     PRI_FAST64     "lli"
00168
00169
00170 #define     PRI_dMAX       "lld"
00171 #define     PRI_iMAX       "lli"
00172
00173 #endif
00174
00175 /** \ingroup avr_inttypes
00176     decimal printf format for intptr_t */
00177 #define     PRI_PTR        PRI_d16
00178 /** \ingroup avr_inttypes
00179     integer printf format for intptr_t */
00180 #define     PRI_IPTR       PRI_i16
00181
00182 /** \ingroup avr_inttypes
00183     octal printf format for uint8_t */
00184 #define     PRI_o8         "o"
00185 /** \ingroup avr_inttypes
00186     octal printf format for uint_least8_t */
00187 #define     PRI_OLEAST8    "o"
00188 /** \ingroup avr_inttypes
```

```
00189     octal printf format for uint_fast8_t */
00190 #define     PRIoFAST8     "o"
00191
00192 /** \ingroup avr_inttypes
00193     decimal printf format for uint8_t */
00194 #define     PRIu8     "u"
00195 /** \ingroup avr_inttypes
00196     decimal printf format for uint_least8_t */
00197 #define     PRIuLEAST8     "u"
00198 /** \ingroup avr_inttypes
00199     decimal printf format for uint_fast8_t */
00200 #define     PRIuFAST8     "u"
00201
00202 /** \ingroup avr_inttypes
00203     hexadecimal printf format for uint8_t */
00204 #define     PRIx8     "x"
00205 /** \ingroup avr_inttypes
00206     hexadecimal printf format for uint_least8_t */
00207 #define     PRIxLEAST8     "x"
00208 /** \ingroup avr_inttypes
00209     hexadecimal printf format for uint_fast8_t */
00210 #define     PRIxFAST8     "x"
00211
00212 /** \ingroup avr_inttypes
00213     uppercase hexadecimal printf format for uint8_t */
00214 #define     PRIX8     "X"
00215 /** \ingroup avr_inttypes
00216     uppercase hexadecimal printf format for uint_least8_t */
00217 #define     PRIXLEAST8     "X"
00218 /** \ingroup avr_inttypes
00219     uppercase hexadecimal printf format for uint_fast8_t */
00220 #define     PRIXFAST8     "X"
00221
00222
00223 /** \ingroup avr_inttypes
00224     octal printf format for uint16_t */
00225 #define     PRIo16     "o"
00226 /** \ingroup avr_inttypes
00227     octal printf format for uint_least16_t */
00228 #define     PRIoLEAST16     "o"
00229 /** \ingroup avr_inttypes
00230     octal printf format for uint_fast16_t */
00231 #define     PRIoFAST16     "o"
00232
00233 /** \ingroup avr_inttypes
00234     decimal printf format for uint16_t */
00235 #define     PRIu16     "u"
00236 /** \ingroup avr_inttypes
00237     decimal printf format for uint_least16_t */
00238 #define     PRIuLEAST16     "u"
00239 /** \ingroup avr_inttypes
00240     decimal printf format for uint_fast16_t */
00241 #define     PRIuFAST16     "u"
00242
00243 /** \ingroup avr_inttypes
00244     hexadecimal printf format for uint16_t */
00245 #define     PRIx16     "x"
00246 /** \ingroup avr_inttypes
00247     hexadecimal printf format for uint_least16_t */
00248 #define     PRIxLEAST16     "x"
00249 /** \ingroup avr_inttypes
00250     hexadecimal printf format for uint_fast16_t */
00251 #define     PRIXFAST16     "X"
00252
00253 /** \ingroup avr_inttypes
00254     uppercase hexadecimal printf format for uint16_t */
00255 #define     PRIX16     "X"
00256 /** \ingroup avr_inttypes
00257     uppercase hexadecimal printf format for uint_least16_t */
00258 #define     PRIXLEAST16     "X"
00259 /** \ingroup avr_inttypes
00260     uppercase hexadecimal printf format for uint_fast16_t */
00261 #define     PRIXFAST16     "X"
00262
00263
00264 /** \ingroup avr_inttypes
00265     octal printf format for uint32_t */
00266 #define     PRIo32     "lo"
00267 /** \ingroup avr_inttypes
00268     octal printf format for uint_least32_t */
00269 #define     PRIoLEAST32     "lo"
00270 /** \ingroup avr_inttypes
00271     octal printf format for uint_fast32_t */
00272 #define     PRIoFAST32     "lo"
00273
00274 /** \ingroup avr_inttypes
00275     decimal printf format for uint32_t */
```

```
00276 #define      PRIu32          "lu"
00277 /** \ingroup avr_inttypes
00278     decimal printf format for uint_least32_t */
00279 #define      PRIuLEAST32      "lu"
00280 /** \ingroup avr_inttypes
00281     decimal printf format for uint_fast32_t */
00282 #define      PRIuFAST32       "lu"
00283
00284 /** \ingroup avr_inttypes
00285     hexadecimal printf format for uint32_t */
00286 #define      PRIx32           "lx"
00287 /** \ingroup avr_inttypes
00288     hexadecimal printf format for uint_least32_t */
00289 #define      PRIxLEAST32      "lx"
00290 /** \ingroup avr_inttypes
00291     hexadecimal printf format for uint_fast32_t */
00292 #define      PRIxFAST32       "lx"
00293
00294 /** \ingroup avr_inttypes
00295     uppercase hexadecimal printf format for uint32_t */
00296 #define      PRIX32           "lX"
00297 /** \ingroup avr_inttypes
00298     uppercase hexadecimal printf format for uint_least32_t */
00299 #define      PRIxLEAST32      "lX"
00300 /** \ingroup avr_inttypes
00301     uppercase hexadecimal printf format for uint_fast32_t */
00302 #define      PRIxFAST32       "lX"
00303
00304
00305 #ifndef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
00306
00307 #define      PRIO64           "llo"
00308 #define      PRIOLEAST64      "llo"
00309 #define      PRIOFAST64       "llo"
00310
00311 #define      PRIu64           "llu"
00312 #define      PRIuLEAST64      "llu"
00313 #define      PRIuFAST64       "llu"
00314
00315 #define      PRIx64           "llx"
00316 #define      PRIxLEAST64      "llx"
00317 #define      PRIxFAST64       "llx"
00318
00319 #define      PRIX64           "llX"
00320 #define      PRIxLEAST64      "llX"
00321 #define      PRIxFAST64       "llX"
00322
00323 #define      PRIOMAX          "llo"
00324 #define      PRIUMAX          "llu"
00325 #define      PRIXMAX          "llx"
00326 #define      PRIUMAX          "llX"
00327
00328 #endif
00329
00330 /** \ingroup avr_inttypes
00331     octal printf format for uintptr_t */
00332 #define      PRIOPTR          PRIO16
00333 /** \ingroup avr_inttypes
00334     decimal printf format for uintptr_t */
00335 #define      PRIuPTR          PRIu16
00336 /** \ingroup avr_inttypes
00337     hexadecimal printf format for uintptr_t */
00338 #define      PRIXPTR          PRIx16
00339 /** \ingroup avr_inttypes
00340     uppercase hexadecimal printf format for uintptr_t */
00341 #define      PRIXPTR          PRIX16
00342
00343
00344 /** \ingroup avr_inttypes
00345     decimal scanf format for int8_t */
00346 #define      SCNd8            "hhd"
00347 /** \ingroup avr_inttypes
00348     decimal scanf format for int_least8_t */
00349 #define      SCNdLEAST8       "hhd"
00350 /** \ingroup avr_inttypes
00351     decimal scanf format for int_fast8_t */
00352 #define      SCNdFAST8        "hhd"
00353
00354 /** \ingroup avr_inttypes
00355     generic-integer scanf format for int8_t */
00356 #define      SCNi8            "hhi"
00357 /** \ingroup avr_inttypes
00358     generic-integer scanf format for int_least8_t */
00359 #define      SCNiLEAST8       "hhi"
00360 /** \ingroup avr_inttypes
00361     generic-integer scanf format for int_fast8_t */
00362 #define      SCNiFAST8        "hhi"
```

```

00363
00364
00365 /** \ingroup avr_inttypes
00366     decimal scanf format for int16_t */
00367 #define      SCNd16          "d"
00368 /** \ingroup avr_inttypes
00369     decimal scanf format for int_least16_t */
00370 #define      SCNLEAST16     "d"
00371 /** \ingroup avr_inttypes
00372     decimal scanf format for int_fast16_t */
00373 #define      SCNFAST16      "d"
00374
00375 /** \ingroup avr_inttypes
00376     generic-integer scanf format for int16_t */
00377 #define      SCNi16         "i"
00378 /** \ingroup avr_inttypes
00379     generic-integer scanf format for int_least16_t */
00380 #define      SCNiLEAST16   "i"
00381 /** \ingroup avr_inttypes
00382     generic-integer scanf format for int_fast16_t */
00383 #define      SCNiFAST16    "i"
00384
00385
00386 /** \ingroup avr_inttypes
00387     decimal scanf format for int32_t */
00388 #define      SCNd32          "ld"
00389 /** \ingroup avr_inttypes
00390     decimal scanf format for int_least32_t */
00391 #define      SCNLEAST32     "ld"
00392 /** \ingroup avr_inttypes
00393     decimal scanf format for int_fast32_t */
00394 #define      SCNFAST32      "ld"
00395
00396 /** \ingroup avr_inttypes
00397     generic-integer scanf format for int32_t */
00398 #define      SCNi32         "li"
00399 /** \ingroup avr_inttypes
00400     generic-integer scanf format for int_least32_t */
00401 #define      SCNiLEAST32   "li"
00402 /** \ingroup avr_inttypes
00403     generic-integer scanf format for int_fast32_t */
00404 #define      SCNiFAST32    "li"
00405
00406
00407 #ifndef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
00408
00409 #define      SCNd64          "lld"
00410 #define      SCNLEAST64     "lld"
00411 #define      SCNFAST64      "lld"
00412
00413 #define      SCNi64         "lli"
00414 #define      SCNiLEAST64   "lli"
00415 #define      SCNiFAST64    "lli"
00416
00417 #define      SCNMAX         "lld"
00418 #define      SCNiMAX        "lli"
00419
00420 #endif
00421
00422 /** \ingroup avr_inttypes
00423     decimal scanf format for intptr_t */
00424 #define      SCNdPTR         SCNd16
00425 /** \ingroup avr_inttypes
00426     generic-integer scanf format for intptr_t */
00427 #define      SCNiPTR         SCNi16
00428
00429 /** \ingroup avr_inttypes
00430     octal scanf format for uint8_t */
00431 #define      SCNo8          "hho"
00432 /** \ingroup avr_inttypes
00433     octal scanf format for uint_least8_t */
00434 #define      SCNoLEAST8     "hho"
00435 /** \ingroup avr_inttypes
00436     octal scanf format for uint_fast8_t */
00437 #define      SCNoFAST8      "hho"
00438
00439 /** \ingroup avr_inttypes
00440     decimal scanf format for uint8_t */
00441 #define      SCNu8          "hhu"
00442 /** \ingroup avr_inttypes
00443     decimal scanf format for uint_least8_t */
00444 #define      SCNuLEAST8     "hhu"
00445 /** \ingroup avr_inttypes
00446     decimal scanf format for uint_fast8_t */
00447 #define      SCNuFAST8      "hhu"
00448
00449 /** \ingroup avr_inttypes

```

```
00450     hexadecimal scanf format for uint8_t */
00451 #define     SCNx8      "hhx"
00452 /** \ingroup avr_inttypes
00453     hexadecimal scanf format for uint_least8_t */
00454 #define     SCNxLEAST8  "hhx"
00455 /** \ingroup avr_inttypes
00456     hexadecimal scanf format for uint_fast8_t */
00457 #define     SCNxFAST8   "hhx"
00458
00459 /** \ingroup avr_inttypes
00460     octal scanf format for uint16_t */
00461 #define     SCNo16     "o"
00462 /** \ingroup avr_inttypes
00463     octal scanf format for uint_least16_t */
00464 #define     SCNoLEAST16 "o"
00465 /** \ingroup avr_inttypes
00466     octal scanf format for uint_fast16_t */
00467 #define     SCNoFAST16  "o"
00468
00469 /** \ingroup avr_inttypes
00470     decimal scanf format for uint16_t */
00471 #define     SCNu16     "u"
00472 /** \ingroup avr_inttypes
00473     decimal scanf format for uint_least16_t */
00474 #define     SCNuLEAST16 "u"
00475 /** \ingroup avr_inttypes
00476     decimal scanf format for uint_fast16_t */
00477 #define     SCNuFAST16  "u"
00478
00479 /** \ingroup avr_inttypes
00480     hexadecimal scanf format for uint16_t */
00481 #define     SCNx16     "x"
00482 /** \ingroup avr_inttypes
00483     hexadecimal scanf format for uint_least16_t */
00484 #define     SCNxLEAST16 "x"
00485 /** \ingroup avr_inttypes
00486     hexadecimal scanf format for uint_fast16_t */
00487 #define     SCNxFAST16  "x"
00488
00489
00490 /** \ingroup avr_inttypes
00491     octal scanf format for uint32_t */
00492 #define     SCNo32     "lo"
00493 /** \ingroup avr_inttypes
00494     octal scanf format for uint_least32_t */
00495 #define     SCNoLEAST32 "lo"
00496 /** \ingroup avr_inttypes
00497     octal scanf format for uint_fast32_t */
00498 #define     SCNoFAST32  "lo"
00499
00500 /** \ingroup avr_inttypes
00501     decimal scanf format for uint32_t */
00502 #define     SCNu32     "lu"
00503 /** \ingroup avr_inttypes
00504     decimal scanf format for uint_least32_t */
00505 #define     SCNuLEAST32 "lu"
00506 /** \ingroup avr_inttypes
00507     decimal scanf format for uint_fast32_t */
00508 #define     SCNuFAST32  "lu"
00509
00510 /** \ingroup avr_inttypes
00511     hexadecimal scanf format for uint32_t */
00512 #define     SCNx32     "lx"
00513 /** \ingroup avr_inttypes
00514     hexadecimal scanf format for uint_least32_t */
00515 #define     SCNxLEAST32 "lx"
00516 /** \ingroup avr_inttypes
00517     hexadecimal scanf format for uint_fast32_t */
00518 #define     SCNxFAST32  "lx"
00519
00520
00521 #ifndef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
00522
00523 #define     SCNo64      "llo"
00524 #define     SCNoLEAST64 "llo"
00525 #define     SCNoFAST64  "llo"
00526
00527 #define     SCNu64      "llu"
00528 #define     SCNuLEAST64 "llu"
00529 #define     SCNuFAST64  "llu"
00530
00531 #define     SCNx64      "llx"
00532 #define     SCNxLEAST64 "llx"
00533 #define     SCNxFAST64  "llx"
00534
00535 #define     SCNoMAX     "llo"
00536 #define     SCNuMAX     "llu"
```

```

00537 #define      SCNxMAX      "llx"
00538
00539 #endif
00540
00541 /** \ingroup avr_inttypes
00542     octal scanf format for uintptr_t */
00543 #define      SCNoPTR      SCNo16
00544 /** \ingroup avr_inttypes
00545     decimal scanf format for uintptr_t */
00546 #define      SCNuPTR      SCNu16
00547 /** \ingroup avr_inttypes
00548     hexadecimal scanf format for uintptr_t */
00549 #define      SCNxPTR      SCNx16
00550
00551 /** @} */
00552
00553
00554 #endif /* !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS) */
00555
00556
00557 #endif /* __INTTYPES_H_ */

```

## 23.48 math.h File Reference

### Macros

- #define [M\\_E](#) 2.7182818284590452354
- #define [M\\_LOG2E](#) 1.4426950408889634074
- #define [M\\_LOG10E](#) 0.43429448190325182765
- #define [M\\_LN2](#) 0.69314718055994530942
- #define [M\\_LN10](#) 2.30258509299404568402
- #define [M\\_PI](#) 3.14159265358979323846
- #define [M\\_PI\\_2](#) 1.57079632679489661923
- #define [M\\_PI\\_4](#) 0.78539816339744830962
- #define [M\\_1\\_PI](#) 0.31830988618379067154
- #define [M\\_2\\_PI](#) 0.63661977236758134308
- #define [M\\_2\\_SQRTPI](#) 1.12837916709551257390
- #define [M\\_SQRT2](#) 1.41421356237309504880
- #define [M\\_SQRT1\\_2](#) 0.70710678118654752440
- #define [NAN](#) \_\_builtin\_nan("")
- #define [nanf](#)(\_\_tag) \_\_builtin\_nanf(\_\_tag)
- #define [nan](#)(\_\_tag) \_\_builtin\_nan(\_\_tag)
- #define [nanl](#)(\_\_tag) \_\_builtin\_nanl(\_\_tag)
- #define [INFINITY](#) \_\_builtin\_inf()
- #define [HUGE\\_VALF](#) \_\_builtin\_huge\_valf()
- #define [HUGE\\_VAL](#) \_\_builtin\_huge\_val()
- #define [HUGE\\_VALL](#) \_\_builtin\_huge\_vall()

### Functions

- float [cosf](#) (float x)
- double [cos](#) (double x)
- long double [cosl](#) (long double x)
- float [sinf](#) (float x)
- double [sin](#) (double x)
- long double [sinl](#) (long double x)
- float [tanf](#) (float x)
- double [tan](#) (double x)
- long double [tanl](#) (long double x)
- static float [fabsf](#) (float \_\_x)



- static double `fabs` (double `__x`)
- static long double `fabsl` (long double `__x`)
- float `fmodf` (float `x`, float `y`)
- double `fmod` (double `x`, double `y`)
- long double `fmodl` (long double `x`, long double `y`)
- float `modff` (float `x`, float `*iptr`)
- double `modf` (double `x`, double `*iptr`)
- long double `modfl` (long double `x`, long double `*iptr`)
- float `sqrtf` (float `x`)
- double `sqrt` (double `x`)
- long double `sqrtl` (long double `x`)
- float `cbrtf` (float `x`)
- double `cbrt` (double `x`)
- long double `cbrtl` (long double `x`)
- float `hypotf` (float `x`, float `y`)
- double `hypot` (double `x`, double `y`)
- long double `hypotl` (long double `x`, long double `y`)
- float `floorf` (float `x`)
- double `floor` (double `x`)
- long double `floorl` (long double `x`)
- float `ceilf` (float `x`)
- double `ceil` (double `x`)
- long double `ceilf` (long double `x`)
- float `frexpf` (float `x`, int `*pexp`)
- double `frexp` (double `x`, int `*pexp`)
- long double `frexpl` (long double `x`, int `*pexp`)
- float `ldexpf` (float `x`, int `iexp`)
- double `ldexp` (double `x`, int `iexp`)
- long double `ldexpl` (long double `x`, int `iexp`)
- float `expf` (float `x`)
- double `exp` (double `x`)
- long double `expl` (long double `x`)
- float `coshf` (float `x`)
- double `cosh` (double `x`)
- long double `coshl` (long double `x`)
- float `sinhf` (float `x`)
- double `sinh` (double `x`)
- long double `sinhl` (long double `x`)
- float `tanhf` (float `x`)
- double `tanh` (double `x`)
- long double `tanhf` (long double `x`)
- float `acosf` (float `x`)
- double `acos` (double `x`)
- long double `acosl` (long double `x`)
- float `asinf` (float `x`)
- double `asin` (double `x`)
- long double `asinl` (long double `x`)
- float `atanf` (float `x`)
- double `atan` (double `x`)
- long double `atanf` (long double `x`)
- float `atan2f` (float `y`, float `x`)
- double `atan2` (double `y`, double `x`)
- long double `atan2f` (long double `y`, long double `x`)
- float `logf` (float `x`)
- double `log` (double `x`)

- long double [logl](#) (long double x)
- float [log10f](#) (float x)
- double [log10](#) (double x)
- long double [log10l](#) (long double x)
- float [powf](#) (float x, float y)
- double [pow](#) (double x, double y)
- long double [powl](#) (long double x, long double y)
- int [isnanf](#) (float x)
- int [isnan](#) (double x)
- int [isnanl](#) (long double x)
- int [isinf](#) (float x)
- int [isinf](#) (double x)
- int [isinfl](#) (long double x)
- static int [isfinitef](#) (float \_\_x)
- static int [isfinite](#) (double \_\_x)
- static int [isfinitel](#) (long double \_\_x)
- static float [copysignf](#) (float \_\_x, float \_\_y)
- static double [copysign](#) (double \_\_x, double \_\_y)
- static long double [copysignl](#) (long double \_\_x, long double \_\_y)
- int [signbitf](#) (float x)
- int [signbit](#) (double x)
- int [signbitl](#) (long double x)
- float [fdimf](#) (float x, float y)
- double [fdim](#) (double x, double y)
- long double [fdiml](#) (long double x, long double y)
- float [fmaf](#) (float x, float y, float z)
- double [fma](#) (double x, double y, double z)
- long double [fmal](#) (long double x, long double y, long double z)
- float [fmaxf](#) (float x, float y)
- double [fmax](#) (double x, double y)
- long double [fmaxl](#) (long double x, long double y)
- float [fminf](#) (float x, float y)
- double [fmin](#) (double x, double y)
- long double [fminl](#) (long double x, long double y)
- float [truncf](#) (float x)
- double [trunc](#) (double x)
- long double [truncl](#) (long double x)
- float [roundf](#) (float x)
- double [round](#) (double x)
- long double [roundl](#) (long double x)
- long [lroundf](#) (float x)
- long [lround](#) (double x)
- long [lroundl](#) (long double x)
- long [lrintf](#) (float x)
- long [lrint](#) (double x)
- long [lrintl](#) (long double x)

### Non-Standard Math Functions

- float [squaref](#) (float x)
- double [square](#) (double x)
- long double [squarel](#) (long double x)

## 23.49 math.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002,2007-2009 Michael Stumpf
00002
00003     Portions of documentation Copyright (c) 1990 - 1994
00004     The Regents of the University of California.
00005
00006     All rights reserved.
00007
00008     Redistribution and use in source and binary forms, with or without
00009     modification, are permitted provided that the following conditions are met:
00010
00011     * Redistributions of source code must retain the above copyright
00012     notice, this list of conditions and the following disclaimer.
00013
00014     * Redistributions in binary form must reproduce the above copyright
00015     notice, this list of conditions and the following disclaimer in
00016     the documentation and/or other materials provided with the
00017     distribution.
00018
00019     * Neither the name of the copyright holders nor the names of
00020     contributors may be used to endorse or promote products derived
00021     from this software without specific prior written permission.
00022
00023     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00024     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00025     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00026     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00027     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00028     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00029     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00030     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00031     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00032     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00033     POSSIBILITY OF SUCH DAMAGE. */
00034
00035 /* $Id$ */
00036
00037 /*
00038     math.h - mathematical functions
00039
00040     Author : Michael Stumpf
00041             Michael.Stumpf@t-online.de
00042
00043     __ATTR_CONST__ added by marekm@linux.org.pl for functions
00044     that "do not examine any values except their arguments, and have
00045     no effects except the return value", for better optimization by gcc.
00046 */
00047
00048 #ifndef __MATH_H
00049 #define __MATH_H
00050
00051 #ifdef __cplusplus
00052 extern "C" {
00053 #endif
00054
00055 /** \file */
00056 /** \defgroup avr_math <math.h>: Mathematics
00057     \code #include <math.h> \endcode
00058
00059     This header file declares basic mathematics constants and
00060     functions.
00061
00062     \par Notes:
00063     - Math functions do not raise exceptions and do not change the
00064       \c errno variable. Therefore the majority of them are declared
00065       with \c const attribute, for better optimization by GCC.
00066     - 64-bit floating-point arithmetic is only available in
00067       <a href="https://gcc.gnu.org/gcc-10/changes.html#avr">avr-gcc v10</a>
00068       and up.
00069       The size of the \c double and \c long \c double type can be selected
00070       at compile-time with options like <tt>-mdouble=64</tt> and
00071       <tt>-mlong-double=32</tt>. Whether such options are available,
00072       and their default values,
00073       depend on how the compiler has been configured.
00074     - The implementation of 64-bit floating-point arithmetic has some
00075       shortcomings and limitations, see the
00076       <a href="https://gcc.gnu.org/wiki/avr-gcc#Libf7">avr-gcc Wiki</a>
00077       for details.
00078     - In order to access the <tt>float</tt> functions,
00079       in avr-gcc v4.6 and older it is usually
00080       also required to link with \c -lm. In avr-gcc v4.7 and up, \c -lm
00081       is added automatically to all linker invocations.
00082 */
00083

```

```

00084
00085 /** \ingroup avr_math */
00086 /**@{*/
00087
00088 /** The constant Euler's number  $e$ . */
00089 #define M_E 2.7182818284590452354
00090
00091 /** The constant logarithm of Euler's number  $e$  to base 2. */
00092 #define M_LOG2E 1.4426950408889634074
00093
00094 /** The constant logarithm of Euler's number  $e$  to base 10. */
00095 #define M_LOG10E 0.43429448190325182765
00096
00097 /** The constant natural logarithm of 2. */
00098 #define M_LN2 0.69314718055994530942
00099
00100 /** The constant natural logarithm of 10. */
00101 #define M_LN10 2.30258509299404568402
00102
00103 /** The constant  $\pi$ . */
00104 #define M_PI 3.14159265358979323846
00105
00106 /** The constant  $\pi/2$ . */
00107 #define M_PI_2 1.57079632679489661923
00108
00109 /** The constant  $\pi/4$ . */
00110 #define M_PI_4 0.78539816339744830962
00111
00112 /** The constant  $1/\pi$ . */
00113 #define M_1_PI 0.31830988618379067154
00114
00115 /** The constant  $2/\pi$ . */
00116 #define M_2_PI 0.63661977236758134308
00117
00118 /** The constant  $2/\sqrt{\pi}$ . */
00119 #define M_2_SQRTPI 1.12837916709551257390
00120
00121 /** The square root of 2. */
00122 #define M_SQRT2 1.41421356237309504880
00123
00124 /** The constant  $1/\sqrt{2}$ . */
00125 #define M_SQRT1_2 0.70710678118654752440
00126
00127 /** The  $\text{c}$  double representation of a constant quiet NaN. */
00128 #define NAN __builtin_nan("")
00129
00130 /** The  $\text{c}$  float representation of a constant quiet NaN.
00131     \p __tag is a string constant like  $\text{c} ""$  or  $\text{c} "123"$ . */
00132 #define nanf(__tag) __builtin_nanf(__tag)
00133
00134 /** The  $\text{c}$  double representation of a constant quiet NaN.
00135     \p __tag is a string constant like  $\text{c} ""$  or  $\text{c} "123"$ . */
00136 #define nan(__tag) __builtin_nan(__tag)
00137
00138 /** The  $\text{c}$  long  $\text{c}$  double representation of a constant quiet NaN.
00139     \p __tag is a string constant like  $\text{c} ""$  or  $\text{c} "123"$ . */
00140 #define nanl(__tag) __builtin_nanl(__tag)
00141
00142 /**  $\text{c}$  double infinity constant. */
00143 #define INFINITY __builtin_inf()
00144
00145 /**  $\text{c}$  float infinity constant. */
00146 #define HUGE_VALF __builtin_huge_valf()
00147
00148 /**  $\text{c}$  double infinity constant. */
00149 #define HUGE_VAL __builtin_huge_val()
00150
00151 /**  $\text{c}$  long  $\text{c}$  double infinity constant. */
00152 #define HUGE_VALL __builtin_huge_vall()
00153
00154 #ifndef __DOXYGEN__
00155 #ifndef __ATTR_CONST__
00156 # define __ATTR_CONST__ __attribute__((__const__))
00157 #endif
00158
00159 #ifndef __ATTR_ALWAYS_INLINE__
00160 #define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00161 #endif
00162 #endif /* ! DOXYGEN */
00163
00164 /** The  $\text{cosf}()$  function returns the cosine of  $x$ , measured in radians. */
00165 __ATTR_CONST__ extern float cosf(float x);
00166 /** The  $\text{cos}()$  function returns the cosine of  $x$ , measured in radians. */
00167 __ATTR_CONST__ extern double cos(double x);
00168 /** The  $\text{cosl}()$  function returns the cosine of  $x$ , measured in radians. */
00169 __ATTR_CONST__ extern long double cosl(long double x);
00170

```

```

00171 /** The sinf() function returns the sine of \a x, measured in radians. */
00172 __ATTR_CONST__ extern float sinf (float x);
00173 /** The sin() function returns the sine of \a x, measured in radians. */
00174 __ATTR_CONST__ extern double sin (double x);
00175 /** The sinl() function returns the sine of \a x, measured in radians. */
00176 __ATTR_CONST__ extern long double sinl (long double x);
00177
00178 /** The tanf() function returns the tangent of \a x, measured in radians. */
00179 __ATTR_CONST__ extern float tanf (float x);
00180 /** The tan() function returns the tangent of \a x, measured in radians. */
00181 __ATTR_CONST__ extern double tan (double x);
00182 /** The tanl() function returns the tangent of \a x, measured in radians. */
00183 __ATTR_CONST__ extern long double tanl (long double x);
00184
00185 /** The fabsf() function computes the absolute value of a floating-point number \a x. */
00186 static __ATTR_ALWAYS_INLINE__ float fabsf (float __x)
00187 {
00188     return __builtin_fabsf (__x);
00189 }
00190
00191 /** The fabs() function computes the absolute value of a floating-point number \a x. */
00192 static __ATTR_ALWAYS_INLINE__ double fabs (double __x)
00193 {
00194     return __builtin_fabs (__x);
00195 }
00196
00197 /** The fabsl() function computes the absolute value of a floating-point number \a x. */
00198 static __ATTR_ALWAYS_INLINE__ long double fabsl (long double __x)
00199 {
00200     return __builtin_fabsl (__x);
00201 }
00202
00203 /** The function fmodf() returns the floating-point remainder of <em>x / y</em>. */
00204 __ATTR_CONST__ extern float fmodf (float x, float y);
00205 /** The function fmod() returns the floating-point remainder of <em>x / y</em>. */
00206 __ATTR_CONST__ extern double fmod (double x, double y);
00207 /** The function fmodl() returns the floating-point remainder of <em>x / y</em>. */
00208 __ATTR_CONST__ extern long double fmodl (long double x, long double y);
00209
00210 /** The modff() function breaks the argument \a x into integral and
00211     fractional parts, each of which has the same sign as the argument.
00212     It stores the integral part as a \c float in the object pointed to by
00213     \a iptr.
00214
00215     The modff() function returns the signed fractional part of \a x.
00216
00217     \note This implementation skips writing by zero pointer. However,
00218     the GCC 4.3 can replace this function with inline code that does not
00219     permit to use NULL address for the avoiding of storing. */
00220 extern float modff (float x, float *iptr);
00221 /** The modf() function breaks the argument \a x into integral and
00222     fractional parts, each of which has the same sign as the argument.
00223     It stores the integral part as a \c double in the object pointed to by
00224     \a iptr.
00225
00226     The modf() function returns the signed fractional part of \a x. */
00227 extern double modf (double x, double *iptr);
00228 /** The modfl() function breaks the argument \a x into integral and
00229     fractional parts, each of which has the same sign as the argument.
00230     It stores the integral part as a \c long \c double in the object pointed to by
00231     \a iptr.
00232
00233     The modf() function returns the signed fractional part of \a x. */
00234 extern long double modfl (long double x, long double *iptr);
00235
00236 /** The sqrtf() function returns the non-negative square root of \a x. */
00237 __ATTR_CONST__ extern float sqrtf (float x);
00238 /** The sqrt() function returns the non-negative square root of \a x. */
00239 __ATTR_CONST__ extern double sqrt (double x);
00240 /** The sqrtl() function returns the non-negative square root of \a x. */
00241 __ATTR_CONST__ extern long double sqrtl (long double x);
00242
00243 /** The cbrtf() function returns the cube root of \a x. */
00244 __ATTR_CONST__ extern float cbrtf (float x);
00245 /** The cbrt() function returns the cube root of \a x. */
00246 __ATTR_CONST__ extern double cbrt (double x);
00247 /** The cbrtl() function returns the cube root of \a x. */
00248 __ATTR_CONST__ extern long double cbrtl (long double x);
00249
00250 /** The hypotf() function returns <em>sqrtf(x*x + y*y)</em>. This
00251     is the length of the hypotenuse of a right triangle with sides of
00252     length \a x and \a y, or the distance of the point (\a x, \a
00253     y) from the origin. Using this function instead of the direct
00254     formula is wise, since the error is much smaller. No underflow with
00255     small \a x and \a y. No overflow if result is in range. */
00256 __ATTR_CONST__ extern float hypotf (float x, float y);
00257 /** The hypot() function returns <em>sqrt(x*x + y*y)</em>. This

```

```

00258     is the length of the hypotenuse of a right triangle with sides of
00259     length  $\sqrt{a^2 + b^2}$ , or the distance of the point  $(x, y)$ 
00260     from the origin. Using this function instead of the direct
00261     formula is wise, since the error is much smaller. No underflow with
00262     small  $x$  and  $y$ . No overflow if result is in range. */
00263     __ATTR_CONST__ extern double hypot (double x, double y);
00264 /** The hypotl() function returns  $\sqrt{x^2 + y^2}$ . This
00265     is the length of the hypotenuse of a right triangle with sides of
00266     length  $\sqrt{a^2 + b^2}$ , or the distance of the point  $(x, y)$ 
00267     from the origin. Using this function instead of the direct
00268     formula is wise, since the error is much smaller. No underflow with
00269     small  $x$  and  $y$ . No overflow if result is in range. */
00270     __ATTR_CONST__ extern long double hypotl (long double x, long double y);
00271
00272 /** The floorf() function returns the largest integral value less than or
00273     equal to  $x$ , expressed as a floating-point number. */
00274     __ATTR_CONST__ extern float floorf (float x);
00275 /** The floor() function returns the largest integral value less than or
00276     equal to  $x$ , expressed as a floating-point number. */
00277     __ATTR_CONST__ extern double floor (double x);
00278 /** The floorl() function returns the largest integral value less than or
00279     equal to  $x$ , expressed as a floating-point number. */
00280     __ATTR_CONST__ extern long double floorl (long double x);
00281
00282 /** The ceilf() function returns the smallest integral value greater than
00283     or equal to  $x$ , expressed as a floating-point number. */
00284     __ATTR_CONST__ extern float ceilf (float x);
00285 /** The ceil() function returns the smallest integral value greater than
00286     or equal to  $x$ , expressed as a floating-point number. */
00287     __ATTR_CONST__ extern double ceil (double x);
00288 /** The ceill() function returns the smallest integral value greater than
00289     or equal to  $x$ , expressed as a floating-point number. */
00290     __ATTR_CONST__ extern long double ceill (long double x);
00291
00292 /** The frexpf() function breaks a floating-point number into a normalized
00293     fraction and an integral power of 2. It stores the integer in the  $\text{int}$ 
00294     object pointed to by  $\text{pexp}$ .
00295
00296     If  $x$  is a normal float point number, the frexpf() function
00297     returns the value  $v$ , such that  $v$  has a magnitude in the
00298     interval  $[1/2, 1)$  or zero, and  $x$  equals  $v$  times 2 raised to
00299     the power  $\text{pexp}$ . If  $x$  is zero, both parts of the result are
00300     zero. If  $x$  is not a finite number, the frexpf() returns  $x$  as
00301     is and stores 0 by  $\text{pexp}$ .
00302
00303     \note This implementation permits a zero pointer as a directive to
00304     skip a storing the exponent.
00305     */
00306     extern float frexpf (float x, int *pexp);
00307 /** The frexp() function breaks a floating-point number into a normalized
00308     fraction and an integral power of 2. It stores the integer in the  $\text{int}$ 
00309     object pointed to by  $\text{pexp}$ .
00310
00311     If  $x$  is a normal float point number, the frexp() function
00312     returns the value  $v$ , such that  $v$  has a magnitude in the
00313     interval  $[1/2, 1)$  or zero, and  $x$  equals  $v$  times 2 raised to
00314     the power  $\text{pexp}$ . If  $x$  is zero, both parts of the result are
00315     zero. If  $x$  is not a finite number, the frexp() returns  $x$  as
00316     is and stores 0 by  $\text{pexp}$ . */
00317     extern double frexp (double x, int *pexp);
00318 /** The frexpl() function breaks a floating-point number into a normalized
00319     fraction and an integral power of 2. It stores the integer in the  $\text{int}$ 
00320     object pointed to by  $\text{pexp}$ .
00321
00322     If  $x$  is a normal float point number, the frexpl() function
00323     returns the value  $v$ , such that  $v$  has a magnitude in the
00324     interval  $[1/2, 1)$  or zero, and  $x$  equals  $v$  times 2 raised to
00325     the power  $\text{pexp}$ . If  $x$  is zero, both parts of the result are
00326     zero. If  $x$  is not a finite number, the frexpl() returns  $x$  as
00327     is and stores 0 by  $\text{pexp}$ . */
00328     extern long double frexpl (long double x, int *pexp);
00329
00330 /** The ldexpf() function multiplies a floating-point number by an integral
00331     power of 2. It returns the value of  $x$  times 2 raised to the power
00332      $\text{iexp}$ . */
00333     __ATTR_CONST__ extern float ldexpf (float x, int iexp);
00334 /** The ldexp() function multiplies a floating-point number by an integral
00335     power of 2. It returns the value of  $x$  times 2 raised to the power
00336      $\text{iexp}$ . */
00337     __ATTR_CONST__ extern double ldexp (double x, int iexp);
00338 /** The ldexpl() function multiplies a floating-point number by an integral
00339     power of 2. It returns the value of  $x$  times 2 raised to the power
00340      $\text{iexp}$ . */
00341     __ATTR_CONST__ extern long double ldexpl (long double x, int iexp);
00342
00343 /** The expf() function returns the exponential value of  $x$ . */
00344     __ATTR_CONST__ extern float expf (float x);

```

```

00345 /** The exp() function returns the exponential value of \a x. */
00346 __ATTR_CONST__ extern double exp (double x);
00347 /** The expl() function returns the exponential value of \a x. */
00348 __ATTR_CONST__ extern long double expl (long double x);
00349
00350 /** The coshf() function returns the hyperbolic cosine of \a x. */
00351 __ATTR_CONST__ extern float coshf (float x);
00352 /** The cosh() function returns the hyperbolic cosine of \a x. */
00353 __ATTR_CONST__ extern double cosh (double x);
00354 /** The coshl() function returns the hyperbolic cosine of \a x. */
00355 __ATTR_CONST__ extern long double coshl (long double x);
00356
00357 /** The sinhf() function returns the hyperbolic sine of \a x. */
00358 __ATTR_CONST__ extern float sinhf (float x);
00359 /** The sinh() function returns the hyperbolic sine of \a x. */
00360 __ATTR_CONST__ extern double sinh (double x);
00361 /** The sinhl() function returns the hyperbolic sine of \a x. */
00362 __ATTR_CONST__ extern long double sinhl (long double x);
00363
00364 /** The tanhf() function returns the hyperbolic tangent of \a x. */
00365 __ATTR_CONST__ extern float tanhf (float x);
00366 /** The tanh() function returns the hyperbolic tangent of \a x. */
00367 __ATTR_CONST__ extern double tanh (double x);
00368 /** The tanhl() function returns the hyperbolic tangent of \a x. */
00369 __ATTR_CONST__ extern long double tanhl (long double x);
00370
00371 /** The acosf() function computes the principal value of the arc cosine of
00372     \a x. The returned value is in the range [0, pi] radians. A domain
00373     error occurs for arguments not in the range [&minus;1, +1]. */
00374 __ATTR_CONST__ extern float acosf (float x);
00375 /** The acos() function computes the principal value of the arc cosine of
00376     \a x. The returned value is in the range [0, pi] radians or NaN. */
00377 __ATTR_CONST__ extern double acos (double x);
00378 /** The acosl() function computes the principal value of the arc cosine of
00379     \a x. The returned value is in the range [0, pi] radians or NaN. */
00380 __ATTR_CONST__ extern long double acosl (long double x);
00381
00382 /** The asinf() function computes the principal value of the arc sine of
00383     \a x. The returned value is in the range [&minus;pi/2, pi/2] radians. A
00384     domain error occurs for arguments not in the range [&minus;1, +1]. */
00385 __ATTR_CONST__ extern float asinf (float x);
00386 /** The asin() function computes the principal value of the arc sine of
00387     \a x. The returned value is in the range [&minus;pi/2, pi/2] radians or NaN. */
00388 __ATTR_CONST__ extern double asin (double x);
00389 /** The asinl() function computes the principal value of the arc sine of
00390     \a x. The returned value is in the range [&minus;pi/2, pi/2] radians or NaN. */
00391 __ATTR_CONST__ extern long double asinl (long double x);
00392
00393 /** The atanf() function computes the principal value of the arc tangent
00394     of \a x. The returned value is in the range [&minus;pi/2, pi/2] radians. */
00395 __ATTR_CONST__ extern float atanf (float x);
00396 /** The atan() function computes the principal value of the arc tangent
00397     of \a x. The returned value is in the range [&minus;pi/2, pi/2] radians. */
00398 __ATTR_CONST__ extern double atan (double x);
00399 /** The atanl() function computes the principal value of the arc tangent
00400     of \a x. The returned value is in the range [&minus;pi/2, pi/2] radians. */
00401 __ATTR_CONST__ extern long double atanl (long double x);
00402
00403 /** The atan2f() function computes the principal value of the arc tangent
00404     of <em>y / x</em>, using the signs of both arguments to determine
00405     the quadrant of the return value. The returned value is in the range
00406     [&minus;pi, +pi] radians. */
00407 __ATTR_CONST__ extern float atan2f (float y, float x);
00408 /** The atan2() function computes the principal value of the arc tangent
00409     of <em>y / x</em>, using the signs of both arguments to determine
00410     the quadrant of the return value. The returned value is in the range
00411     [&minus;pi, +pi] radians. */
00412 __ATTR_CONST__ extern double atan2 (double y, double x);
00413 /** The atan2l() function computes the principal value of the arc tangent
00414     of <em>y / x</em>, using the signs of both arguments to determine
00415     the quadrant of the return value. The returned value is in the range
00416     [&minus;pi, +pi] radians. */
00417 __ATTR_CONST__ extern long double atan2l (long double y, long double x);
00418
00419 /** The logf() function returns the natural logarithm of argument \a x. */
00420 __ATTR_CONST__ extern float logf (float x);
00421 /** The log() function returns the natural logarithm of argument \a x. */
00422 __ATTR_CONST__ extern double log (double x);
00423 /** The logl() function returns the natural logarithm of argument \a x. */
00424 __ATTR_CONST__ extern long double logl (long double x);
00425
00426 /** The log10f() function returns the logarithm of argument \a x to base 10. */
00427 __ATTR_CONST__ extern float log10f (float x);
00428 /** The log10() function returns the logarithm of argument \a x to base 10. */
00429 __ATTR_CONST__ extern double log10 (double x);
00430 /** The log10l() function returns the logarithm of argument \a x to base 10. */
00431 __ATTR_CONST__ extern long double log10l (long double x);

```

```

00432
00433 /** The function powf() returns the value of \a x to the exponent \a y.
00434     \n Notice that for integer exponents, there is the more efficient
00435     <code>float __builtin_powif(float x, int y)</code>. */
00436 __ATTR_CONST__ extern float powf (float x, float y);
00437 /** The function pow() returns the value of \a x to the exponent \a y.
00438     \n Notice that for integer exponents, there is the more efficient
00439     <code>double __builtin_powi(double x, int y)</code>. */
00440 __ATTR_CONST__ extern double pow (double x, double y);
00441 /** The function powl() returns the value of \a x to the exponent \a y.
00442     \n Notice that for integer exponents, there is the more efficient
00443     <code>long double __builtin_powil(long double x, int y)</code>. */
00444 __ATTR_CONST__ extern long double powl (long double x, long double y);
00445
00446 /** The function isnanf() returns 1 if the argument \a x represents a
00447     "not-a-number" (NaN) object, otherwise 0. */
00448 __ATTR_CONST__ extern int isnanf (float x);
00449 /** The function isnan() returns 1 if the argument \a x represents a
00450     "not-a-number" (NaN) object, otherwise 0. */
00451 __ATTR_CONST__ extern int isnan (double x);
00452 /** The function isnanl() returns 1 if the argument \a x represents a
00453     "not-a-number" (NaN) object, otherwise 0. */
00454 __ATTR_CONST__ extern int isnanl (long double x);
00455
00456 /** The function isinff() returns 1 if the argument \a x is positive
00457     infinity, &minus;1 if \a x is negative infinity, and 0 otherwise. */
00458 __ATTR_CONST__ extern int isinff (float x);
00459 /** The function isinf() returns 1 if the argument \a x is positive
00460     infinity, &minus;1 if \a x is negative infinity, and 0 otherwise. */
00461 __ATTR_CONST__ extern int isinf (double x);
00462 /** The function isinfl() returns 1 if the argument \a x is positive
00463     infinity, &minus;1 if \a x is negative infinity, and 0 otherwise. */
00464 __ATTR_CONST__ extern int isinfl (long double x);
00465
00466 /** The isfinitef() function returns a nonzero value if \a __x is finite:
00467     not plus or minus infinity, and not NaN. */
00468 __ATTR_CONST__ static __ATTR_ALWAYS_INLINE__ int isfinitef (float __x)
00469 {
00470     unsigned char __exp;
00471     __asm__ (
00472         "mov    %0, %C1"      "\n\t"
00473         "lsl    %0"          "\n\t"
00474         "mov    %0, %D1"      "\n\t"
00475         "rol    %0"
00476         : "=r" (__exp)
00477         : "r" (__x) );
00478     return __exp != 0xff;
00479 }
00480
00481 /** The isfinite() function returns a nonzero value if \a __x is finite:
00482     not plus or minus infinity, and not NaN. */
00483 #ifdef __DOXYGEN__
00484 static __ATTR_ALWAYS_INLINE__ int isfinite (double __x);
00485 #elif __SIZEOF_DOUBLE__ == __SIZEOF_FLOAT__
00486 static __ATTR_ALWAYS_INLINE__ int isfinite (double __x)
00487 {
00488     return isfinitef (__x);
00489 }
00490 #else
00491 int isfinite (double __x);
00492 #endif /* double = float */
00493
00494 /** The isfinite() function returns a nonzero value if \a __x is finite:
00495     not plus or minus infinity, and not NaN. */
00496 #ifdef __DOXYGEN__
00497 static __ATTR_ALWAYS_INLINE__ int isfinitel (long double __x);
00498 #elif __SIZEOF_LONG_DOUBLE__ == __SIZEOF_FLOAT__
00499 static __ATTR_ALWAYS_INLINE__ int isfinitel (long double __x)
00500 {
00501     return isfinitef (__x);
00502 }
00503 #else
00504 int isfinitel (long double __x);
00505 #endif /* long double = float */
00506
00507 /** The copysignf() function returns \a x but with the sign of \a y.
00508     They work even if \a x or \a y are NaN or zero. */
00509 __ATTR_CONST__ static __ATTR_ALWAYS_INLINE__ float copysignf (float __x, float __y)
00510 {
00511     __asm__ (
00512         "bst    %D2, 7"      "\n\t"
00513         "bld    %D0, 7"
00514         : "=r" (__x)
00515         : "0" (__x), "r" (__y));
00516     return __x;
00517 }
00518

```



```

00519 /** The copysign() function returns \a __x but with the sign of \a __y.
00520     They work even if \a __x or \a __y are NaN or zero. */
00521 __ATTR_CONST__ static __ATTR_ALWAYS_INLINE__ double copysign (double __x, double __y)
00522 {
00523     __asm__ (
00524         "bst    %r1,%2-1, 7" "\n\t"
00525         "bld    %r0,%2-1, 7"
00526         : "+r" (__x)
00527         : "r" (__y), "n" (__SIZEOF_DOUBLE__);
00528     return __x;
00529 }
00530
00531 /** The copysignl() function returns \a __x but with the sign of \a __y.
00532     They work even if \a __x or \a __y are NaN or zero. */
00533 __ATTR_CONST__ static __ATTR_ALWAYS_INLINE__ long double copysignl (long double __x, long double __y)
00534 {
00535     __asm__ (
00536         "bst    %r1,%2-1, 7" "\n\t"
00537         "bld    %r0,%2-1, 7"
00538         : "+r" (__x)
00539         : "r" (__y), "n" (__SIZEOF_LONG_DOUBLE__);
00540     return __x;
00541 }
00542
00543 /** The signbitf() function returns a nonzero value if the value of \a x
00544     has its sign bit set. This is not the same as '\a x < 0.0',
00545     because IEEE 754 floating point allows zero to be signed. The
00546     comparison '&minus;0.0 < 0.0' is false, but 'signbit (&minus;0.0)' will return a
00547     nonzero value. */
00548 __ATTR_CONST__ extern int signbitf (float x);
00549 /** The signbit() function returns a nonzero value if the value of \a x
00550     has its sign bit set. This is not the same as '\a x < 0.0',
00551     because IEEE 754 floating point allows zero to be signed. The
00552     comparison '&minus;0.0 < 0.0' is false, but 'signbit (&minus;0.0)' will return a
00553     nonzero value. */
00554 __ATTR_CONST__ extern int signbit (double x);
00555 /** The signbitl() function returns a nonzero value if the value of \a x
00556     has its sign bit set. This is not the same as '\a x < 0.0',
00557     because IEEE 754 floating point allows zero to be signed. The
00558     comparison '&minus;0.0 < 0.0' is false, but 'signbit (&minus;0.0)' will return a
00559     nonzero value. */
00560 __ATTR_CONST__ extern int signbitl (long double x);
00561
00562 /** The fdimf() function returns <em>max(x &minus; y, 0)</em>. If \a x or
00563     \a y or both are NaN, NaN is returned. */
00564 __ATTR_CONST__ extern float fdimf (float x, float y);
00565 /** The fdim() function returns <em>max(x &minus; y, 0)</em>. If \a x or
00566     \a y or both are NaN, NaN is returned. */
00567 __ATTR_CONST__ extern double fdim (double x, double y);
00568 /** The fdiml() function returns <em>max(x &minus; y, 0)</em>. If \a x or
00569     \a y or both are NaN, NaN is returned. */
00570 __ATTR_CONST__ extern long double fdiml (long double x, long double y);
00571
00572 /** The fmaf() function performs floating-point multiply-add. This is the
00573     operation <em>(x * y) + z</em>, but the intermediate result is
00574     not rounded to the destination type. This can sometimes improve the
00575     precision of a calculation. */
00576 __ATTR_CONST__ extern float fmaf (float x, float y, float z);
00577 /** The fma() function performs floating-point multiply-add. This is the
00578     operation <em>(x * y) + z</em>, but the intermediate result is
00579     not rounded to the destination type. This can sometimes improve the
00580     precision of a calculation. */
00581 __ATTR_CONST__ extern double fma (double x, double y, double z);
00582 /** The fmal() function performs floating-point multiply-add. This is the
00583     operation <em>(x * y) + z</em>, but the intermediate result is
00584     not rounded to the destination type. This can sometimes improve the
00585     precision of a calculation. */
00586 __ATTR_CONST__ extern long double fmal (long double x, long double y, long double z);
00587
00588 /** The fmaxf() function returns the greater of the two values \a x and
00589     \a y. If an argument is NaN, the other argument is returned. If
00590     both arguments are NaN, NaN is returned. */
00591 __ATTR_CONST__ extern float fmaxf (float x, float y);
00592 /** The fmax() function returns the greater of the two values \a x and
00593     \a y. If an argument is NaN, the other argument is returned. If
00594     both arguments are NaN, NaN is returned. */
00595 __ATTR_CONST__ extern double fmax (double x, double y);
00596 /** The fmaxl() function returns the greater of the two values \a x and
00597     \a y. If an argument is NaN, the other argument is returned. If
00598     both arguments are NaN, NaN is returned. */
00599 __ATTR_CONST__ extern long double fmaxl (long double x, long double y);
00600
00601 /** The fminf() function returns the lesser of the two values \a x and
00602     \a y. If an argument is NaN, the other argument is returned. If
00603     both arguments are NaN, NaN is returned. */
00604 __ATTR_CONST__ extern float fminf (float x, float y);
00605 /** The fmin() function returns the lesser of the two values \a x and

```

```

00606     \a y. If an argument is NaN, the other argument is returned. If
00607     both arguments are NaN, NaN is returned. */
00608 __ATTR_CONST__ extern double fmin (double x, double y);
00609 /** The fminl() function returns the lesser of the two values \a x and
00610     \a y. If an argument is NaN, the other argument is returned. If
00611     both arguments are NaN, NaN is returned. */
00612 __ATTR_CONST__ extern long double fminl (long double x, long double y);
00613
00614 /** The truncf() function rounds \a x to the nearest integer not larger
00615     in absolute value. */
00616 __ATTR_CONST__ extern float truncf (float x);
00617 /** The trunc() function rounds \a x to the nearest integer not larger
00618     in absolute value. */
00619 __ATTR_CONST__ extern double trunc (double x);
00620 /** The truncl() function rounds \a x to the nearest integer not larger
00621     in absolute value. */
00622 __ATTR_CONST__ extern long double trunc (long double x);
00623
00624 /** The roundf() function rounds \a x to the nearest integer, but rounds
00625     halfway cases away from zero (instead of to the nearest even integer).
00626     Overflow is impossible.
00627
00628     \return The rounded value. If \a x is an integral or infinite, \a
00629     x itself is returned. If \a x is \c NaN, then \c NaN is returned. */
00630 __ATTR_CONST__ extern float roundf (float x);
00631 /** The round() function rounds \a x to the nearest integer, but rounds
00632     halfway cases away from zero (instead of to the nearest even integer).
00633     Overflow is impossible.
00634
00635     \return The rounded value. If \a x is an integral or infinite, \a
00636     x itself is returned. If \a x is \c NaN, then \c NaN is returned. */
00637 __ATTR_CONST__ extern double round (double x);
00638 /** The roundl() function rounds \a x to the nearest integer, but rounds
00639     halfway cases away from zero (instead of to the nearest even integer).
00640     Overflow is impossible.
00641
00642     \return The rounded value. If \a x is an integral or infinite, \a
00643     x itself is returned. If \a x is \c NaN, then \c NaN is returned. */
00644 __ATTR_CONST__ extern long double roundl (long double x);
00645
00646 /** The lroundf() function rounds \a x to the nearest integer, but rounds
00647     halfway cases away from zero (instead of to the nearest even integer).
00648     This function is similar to round() function, but it differs in type of
00649     return value and in that an overflow is possible.
00650
00651     \return The rounded long integer value. If \a x is not a finite number
00652     or an overflow was, this realization returns the \c LONG_MIN value
00653     (0x80000000). */
00654 __ATTR_CONST__ extern long lroundf (float x);
00655 /** The lround() function rounds \a x to the nearest integer, but rounds
00656     halfway cases away from zero (instead of to the nearest even integer).
00657     This function is similar to round() function, but it differs in type of
00658     return value and in that an overflow is possible.
00659
00660     \return The rounded long integer value. If \a x is not a finite number
00661     or an overflow was, this realization returns the \c LONG_MIN value
00662     (0x80000000). */
00663 __ATTR_CONST__ extern long lround (double x);
00664 /** The lroundl() function rounds \a x to the nearest integer, but rounds
00665     halfway cases away from zero (instead of to the nearest even integer).
00666     This function is similar to round() function, but it differs in type of
00667     return value and in that an overflow is possible.
00668
00669     \return The rounded long integer value. If \a x is not a finite number
00670     or an overflow was, this realization returns the \c LONG_MIN value
00671     (0x80000000). */
00672 __ATTR_CONST__ extern long lroundl (long double x);
00673
00674 /** The lrintf() function rounds \a x to the nearest integer, rounding the
00675     halfway cases to the even integer direction. (That is both 1.5 and 2.5
00676     values are rounded to 2). This function is similar to rintf() function,
00677     but it differs in type of return value and in that an overflow is
00678     possible.
00679
00680     \return The rounded long integer value. If \a x is not a finite
00681     number or an overflow was, this realization returns the \c LONG_MIN
00682     value (0x80000000). */
00683 __ATTR_CONST__ extern long lrintf (float x);
00684 /** The lrint() function rounds \a x to the nearest integer, rounding the
00685     halfway cases to the even integer direction. (That is both 1.5 and 2.5
00686     values are rounded to 2). This function is similar to rint() function,
00687     but it differs in type of return value and in that an overflow is
00688     possible.
00689
00690     \return The rounded long integer value. If \a x is not a finite
00691     number or an overflow was, this realization returns the \c LONG_MIN
00692     value (0x80000000). */

```

```

00693 __ATTR_CONST__ extern long lrint (double x);
00694 /** The lrintl() function rounds \a x to the nearest integer, rounding the
00695     halfway cases to the even integer direction. (That is both 1.5 and 2.5
00696     values are rounded to 2). This function is similar to rintl() function,
00697     but it differs in type of return value and in that an overflow is
00698     possible.
00699
00700     \return The rounded long integer value. If \a x is not a finite
00701     number or an overflow was, this realization returns the \c LONG_MIN
00702     value (0x80000000). */
00703 __ATTR_CONST__ extern long lrintl (long double x);
00704
00705 /**@}*/
00706
00707 /**@{*/
00708 /**
00709     \name Non-Standard Math Functions
00710     */
00711
00712 /** \ingroup avr_math
00713     The function squaref() returns <em>x * x</em>.
00714     \note This function does not belong to the C standard definition. */
00715 __ATTR_CONST__ extern float squaref (float x);
00716
00717 /** The function square() returns <em>x * x</em>.
00718     \note This function does not belong to the C standard definition. */
00719
00720 #if defined(__DOXYGEN__) || __SIZEOF_DOUBLE__ == __SIZEOF_FLOAT__
00721 __ATTR_CONST__ extern double square (double x);
00722 #elif defined(__WITH_LIBF7_MATH__)
00723 __ATTR_CONST__ extern double square (double x) __asm("__f7_square");
00724 #endif
00725
00726 /** The function squarel() returns <em>x * x</em>.
00727     \note This function does not belong to the C standard definition. */
00728 #if defined(__DOXYGEN__) || __SIZEOF_LONG_DOUBLE__ == __SIZEOF_FLOAT__
00729 __ATTR_CONST__ extern long double squarel (long double x);
00730 #elif defined(__WITH_LIBF7_MATH__)
00731 __ATTR_CONST__ extern long double squarel (long double x) __asm("__f7_square");
00732 #endif
00733
00734 /**@}*/
00735
00736 #ifdef __cplusplus
00737 }
00738 #endif
00739
00740 #endif /* !__MATH_H */

```

## 23.50 setjmp.h File Reference

### Functions

- int [setjmp](#) (jmp\_buf \_\_jmpb)
- void [longjmp](#) (jmp\_buf \_\_jmpb, int \_\_ret)

### 23.51 setjmp.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002,2007 Marek Michalkiewicz
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011     notice, this list of conditions and the following disclaimer in
00012     the documentation and/or other materials provided with the
00013     distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016     contributors may be used to endorse or promote products derived
00017     from this software without specific prior written permission.

```

```

00018
00019 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029 POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 #ifndef __SETJMP_H_
00034 #define __SETJMP_H_ 1
00035
00036 #ifdef __cplusplus
00037 extern "C" {
00038 #endif
00039
00040 /*
00041 jmp_buf:
00042 offset size description
00043 0 16/2 call-saved registers (r2-r17)
00044 (AVR_TINY arch has only 2 call saved registers (r18,r19))
00045 16/2 2 frame pointer (r29:r28)
00046 18/4 2 stack pointer (SPH:SPL)
00047 20/6 1 status register (SREG)
00048 21/7 2/3 return address (PC) (2 bytes used for <=128Kw flash)
00049 23/24/9 = total size (AVR_TINY arch always has 2 bytes PC)
00050 */
00051
00052 #if !defined(__DOXYGEN__)
00053
00054 #if defined(__AVR_TINY__)
00055 # define _JBLEN 9
00056 #elif defined(__AVR_3_BYTE_PC__) && __AVR_3_BYTE_PC__
00057 # define _JBLEN 24
00058 #else
00059 # define _JBLEN 23
00060 #endif
00061 typedef struct _jmp_buf { unsigned char _jb[_JBLEN]; } jmp_buf[1];
00062
00063 #endif /* not __DOXYGEN__ */
00064
00065 /** \file */
00066 /** \defgroup setjmp <setjmp.h>: Non-local goto
00067
00068 While the C language has the dreaded \c goto statement, it can only be
00069 used to jump to a label in the same (local) function. In order to jump
00070 directly to another (non-local) function, the C library provides the
00071 #setjmp and #longjmp functions. setjmp and longjmp are useful for
00072 dealing with errors and interrupts encountered in a low-level subroutine
00073 of a program.
00074
00075 \note #setjmp and #longjmp make programs hard to understand and maintain.
00076 If possible, an alternative should be used.
00077
00078 \note longjmp can destroy changes made to global register
00079 variables (see \ref faq_regbind).
00080
00081 For a very detailed discussion of setjmp/longjmp, see Chapter 7 of
00082 <em>Advanced Programming in the UNIX Environment</em>, by W. Richard
00083 Stevens.
00084
00085 Example:
00086
00087 \code
00088 #include <setjmp.h>
00089
00090 jmp_buf env;
00091
00092 int main (void)
00093 {
00094     if (setjmp (env))
00095     {
00096         // Handle error ...
00097     }
00098
00099     while (1)
00100     {
00101         // Main processing loop which calls foo() somewhere ...
00102     }
00103 }
00104

```

```

00105     void foo (void)
00106     {
00107         // blah, blah, blah ...
00108
00109         if (err)
00110         {
00111             longjmp (env, 1);
00112         }
00113     }
00114     \endcode */
00115
00116 #ifndef __DOXYGEN__
00117 #ifndef __ATTR_NORETURN__
00118 #define __ATTR_NORETURN__ __attribute__((__noreturn__))
00119 #endif
00120 #endif /* ! DOXYGEN */
00121
00122 /** \ingroup setjmp
00123     \brief Save stack context for non-local goto.
00124
00125     \code #include <setjmp.h>\endcode
00126
00127     setjmp() saves the stack context/environment in \e __jmpb for later use by
00128     longjmp(). The stack context will be invalidated if the function which
00129     called setjmp() returns.
00130
00131     \param __jmpb Variable of type \c jmp_buf which holds the stack
00132     information such that the environment can be restored.
00133
00134     \returns setjmp() returns 0 if returning directly, and
00135     non-zero when returning from longjmp() using the saved context. */
00136 extern int setjmp(jmp_buf __jmpb);
00137
00138
00139 /** \ingroup setjmp
00140     \brief Non-local jump to a saved stack context.
00141
00142     \code #include <setjmp.h>\endcode
00143
00144     longjmp() restores the environment saved by the last call of setjmp() with
00145     the corresponding \e __jmpb argument. After longjmp() is completed,
00146     program execution continues as if the corresponding call of setjmp() had
00147     just returned the value \e __ret.
00148
00149     \note longjmp() cannot cause 0 to be returned. If longjmp() is invoked
00150     with a second argument of 0, 1 will be returned instead.
00151
00152     \param __jmpb Information saved by a previous call to setjmp().
00153     \param __ret Value to return to the caller of setjmp().
00154
00155     \returns This function never returns. */
00156 extern void longjmp(jmp_buf __jmpb, int __ret) __ATTR_NORETURN__;
00157
00158 #ifdef __cplusplus
00159 }
00160 #endif
00161 #endif /* !__SETJMP_H_ */

```

## 23.52 stdint.h File Reference

### Macros

#### Limits of specified-width integer types

*C++ implementations should define these macros only when `__STDC_LIMIT_MACROS` is defined before `<stdint.h>` is included*

- `#define INT8_MAX 0x7f`
- `#define INT8_MIN (-INT8_MAX - 1)`
- `#define UINT8_MAX (INT8_MAX * 2 + 1)`
- `#define INT16_MAX 0x7fff`
- `#define INT16_MIN (-INT16_MAX - 1)`
- `#define UINT16_MAX (__CONCAT(INT16_MAX, U) * 2U + 1U)`
- `#define INT32_MAX 0x7fffffffL`
- `#define INT32_MIN (-INT32_MAX - 1L)`
- `#define UINT32_MAX (__CONCAT(INT32_MAX, U) * 2UL + 1UL)`

- #define `INT64_MAX` `0x7fffffffffffffffLL`
- #define `INT64_MIN` `(-INT64_MAX - 1LL)`
- #define `UINT64_MAX` `(__CONCAT(INT64_MAX, U) * 2ULL + 1ULL)`

#### Limits of minimum-width integer types

- #define `INT_LEAST8_MAX` `INT8_MAX`
- #define `INT_LEAST8_MIN` `INT8_MIN`
- #define `UINT_LEAST8_MAX` `UINT8_MAX`
- #define `INT_LEAST16_MAX` `INT16_MAX`
- #define `INT_LEAST16_MIN` `INT16_MIN`
- #define `UINT_LEAST16_MAX` `UINT16_MAX`
- #define `INT_LEAST32_MAX` `INT32_MAX`
- #define `INT_LEAST32_MIN` `INT32_MIN`
- #define `UINT_LEAST32_MAX` `UINT32_MAX`
- #define `INT_LEAST64_MAX` `INT64_MAX`
- #define `INT_LEAST64_MIN` `INT64_MIN`
- #define `UINT_LEAST64_MAX` `UINT64_MAX`

#### Limits of fastest minimum-width integer types

- #define `INT_FAST8_MAX` `INT8_MAX`
- #define `INT_FAST8_MIN` `INT8_MIN`
- #define `UINT_FAST8_MAX` `UINT8_MAX`
- #define `INT_FAST16_MAX` `INT16_MAX`
- #define `INT_FAST16_MIN` `INT16_MIN`
- #define `UINT_FAST16_MAX` `UINT16_MAX`
- #define `INT_FAST32_MAX` `INT32_MAX`
- #define `INT_FAST32_MIN` `INT32_MIN`
- #define `UINT_FAST32_MAX` `UINT32_MAX`
- #define `INT_FAST64_MAX` `INT64_MAX`
- #define `INT_FAST64_MIN` `INT64_MIN`
- #define `UINT_FAST64_MAX` `UINT64_MAX`

#### Limits of integer types capable of holding object pointers

- #define `INTPTR_MAX` `INT16_MAX`
- #define `INTPTR_MIN` `INT16_MIN`
- #define `UINTPTR_MAX` `UINT16_MAX`

#### Limits of greatest-width integer types

- #define `INTMAX_MAX` `INT64_MAX`
- #define `INTMAX_MIN` `INT64_MIN`
- #define `UINTMAX_MAX` `UINT64_MAX`

#### Limits of other integer types

*C++ implementations should define these macros only when `__STDC_LIMIT_MACROS` is defined before `<stdint.h>` is included*

- #define `PTRDIFF_MAX` `INT16_MAX`
- #define `PTRDIFF_MIN` `INT16_MIN`
- #define `SIG_ATOMIC_MAX` `INT8_MAX`
- #define `SIG_ATOMIC_MIN` `INT8_MIN`
- #define `SIZE_MAX` `UINT16_MAX`
- #define `WCHAR_MAX` `__WCHAR_MAX__`
- #define `WCHAR_MIN` `__WCHAR_MIN__`
- #define `WINT_MAX` `__WINT_MAX__`
- #define `WINT_MIN` `__WINT_MIN__`

### Macros for integer constants

C++ implementations should define these macros only when `__STDC_CONSTANT_MACROS` is defined before `<stdint.h>` is included.

These definitions are valid for integer constants without suffix and for macros defined as integer constant without suffix

- `#define INT8_C(value) ((int8_t) value)`
- `#define UINT8_C(value) ((uint8_t) __CONCAT(value, U))`
- `#define INT16_C(value) value`
- `#define UINT16_C(value) __CONCAT(value, U)`
- `#define INT32_C(value) __CONCAT(value, L)`
- `#define UINT32_C(value) __CONCAT(value, UL)`
- `#define INT64_C(value) __CONCAT(value, LL)`
- `#define UINT64_C(value) __CONCAT(value, ULL)`
- `#define INTMAX_C(value) __CONCAT(value, LL)`
- `#define UINTMAX_C(value) __CONCAT(value, ULL)`

### Typedefs

#### Exact-width integer types

Integer types having exactly the specified width

- typedef signed char `int8_t`
- typedef unsigned char `uint8_t`
- typedef signed int `int16_t`
- typedef unsigned int `uint16_t`
- typedef signed long int `int32_t`
- typedef unsigned long int `uint32_t`
- typedef signed long long int `int64_t`
- typedef unsigned long long int `uint64_t`

#### Integer types capable of holding object pointers

These allow you to declare variables of the same size as a pointer.

- typedef `int16_t` `intptr_t`
- typedef `uint16_t` `uintptr_t`

#### Minimum-width integer types

Integer types having at least the specified width

- typedef `int8_t` `int_least8_t`
- typedef `uint8_t` `uint_least8_t`
- typedef `int16_t` `int_least16_t`
- typedef `uint16_t` `uint_least16_t`
- typedef `int32_t` `int_least32_t`
- typedef `uint32_t` `uint_least32_t`
- typedef `int64_t` `int_least64_t`
- typedef `uint64_t` `uint_least64_t`

#### Fastest minimum-width integer types

Integer types being usually fastest having at least the specified width

- typedef `int8_t` `int_fast8_t`
- typedef `uint8_t` `uint_fast8_t`
- typedef `int16_t` `int_fast16_t`
- typedef `uint16_t` `uint_fast16_t`
- typedef `int32_t` `int_fast32_t`
- typedef `uint32_t` `uint_fast32_t`

- typedef `int64_t int_fast64_t`
- typedef `uint64_t uint_fast64_t`

### Greatest-width integer types

*Types designating integer data capable of representing any value of any integer type in the corresponding signed or unsigned category*

- typedef `int64_t intmax_t`
- typedef `uint64_t uintmax_t`

## 23.53 stdint.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002,2004,2005 Marek Michalkiewicz
00002 Copyright (c) 2005, Carlos Lamas
00003 Copyright (c) 2005,2007 Joerg Wunsch
00004 Copyright (c) 2013 Embecosm
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without
00008 modification, are permitted provided that the following conditions are met:
00009
00010 * Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 * Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in
00015 the documentation and/or other materials provided with the
00016 distribution.
00017
00018 * Neither the name of the copyright holders nor the names of
00019 contributors may be used to endorse or promote products derived
00020 from this software without specific prior written permission.
00021
00022 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00023 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00024 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00025 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00026 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00027 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00028 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00029 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00030 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00031 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00032 POSSIBILITY OF SUCH DAMAGE. */
00033
00034 /* $Id$ */
00035
00036 /*
00037 * ISO/IEC 9899:1999 7.18 Integer types <stdint.h>
00038 */
00039
00040 #ifndef __STDINT_H_
00041 #define __STDINT_H_
00042
00043 /** \file */
00044 /** \defgroup avr_stdint <stdint.h>: Standard Integer Types
00045     \code #include <stdint.h> \endcode
00046
00047     Use [u]intN_t if you need exactly N bits.
00048
00049     Since these typedefs are mandated by the C99 standard, they are preferred
00050     over rolling your own typedefs. */
00051
00052 #ifndef __DOXYGEN__
00053 /*
00054 * __USING_MINT8 is defined to 1 if the -mint8 option is in effect.
00055 */
00056 #if __INT_MAX__ == 127
00057 # define __USING_MINT8 1
00058 #else
00059 # define __USING_MINT8 0
00060 #endif
00061
00062 #endif /* !__DOXYGEN__ */
00063
00064 /* Integer types */

```



```

00065
00066 #if defined(__DOXYGEN__)
00067
00068 /* doxygen gets confused by the __attribute__ stuff */
00069
00070 /** \name Exact-width integer types
00071     Integer types having exactly the specified width */
00072
00073 /** @{ */
00074
00075 /** \ingroup avr_stdint
00076     8-bit signed type. */
00077
00078 typedef signed char int8_t;
00079
00080 /** \ingroup avr_stdint
00081     8-bit unsigned type. */
00082
00083 typedef unsigned char uint8_t;
00084
00085 /** \ingroup avr_stdint
00086     16-bit signed type. */
00087
00088 typedef signed int int16_t;
00089
00090 /** \ingroup avr_stdint
00091     16-bit unsigned type. */
00092
00093 typedef unsigned int uint16_t;
00094
00095 /** \ingroup avr_stdint
00096     32-bit signed type. */
00097
00098 typedef signed long int int32_t;
00099
00100 /** \ingroup avr_stdint
00101     32-bit unsigned type. */
00102
00103 typedef unsigned long int uint32_t;
00104
00105 /** \ingroup avr_stdint
00106     64-bit signed type.
00107     \note This type is not available when the compiler
00108     option -mint8 is in effect. */
00109
00110 typedef signed long long int int64_t;
00111
00112 /** \ingroup avr_stdint
00113     64-bit unsigned type.
00114     \note This type is not available when the compiler
00115     option -mint8 is in effect. */
00116
00117 typedef unsigned long long int uint64_t;
00118
00119 /** @} */
00120
00121 #else /* !defined(__DOXYGEN__) */
00122
00123 /* actual implementation goes here */
00124
00125 typedef signed int int8_t __attribute__((__mode__(__QI__)));
00126 typedef unsigned int uint8_t __attribute__((__mode__(__QI__)));
00127 typedef signed int int16_t __attribute__((__mode__(__HI__)));
00128 typedef unsigned int uint16_t __attribute__((__mode__(__HI__)));
00129 typedef signed int int32_t __attribute__((__mode__(__SI__)));
00130 typedef unsigned int uint32_t __attribute__((__mode__(__SI__)));
00131 #if !__USING_MINT8
00132 typedef signed int int64_t __attribute__((__mode__(__DI__)));
00133 typedef unsigned int uint64_t __attribute__((__mode__(__DI__)));
00134 #endif
00135
00136 #endif /* defined(__DOXYGEN__) */
00137
00138 /** \name Integer types capable of holding object pointers
00139     These allow you to declare variables of the same size as a pointer. */
00140
00141 /** @{ */
00142
00143 /** \ingroup avr_stdint
00144     Signed pointer compatible type. */
00145
00146 typedef int16_t intptr_t;
00147
00148 /** \ingroup avr_stdint
00149     Unsigned pointer compatible type. */
00150
00151 typedef uint16_t uintptr_t;

```

```
00152
00153 /**@}*/
00154
00155 /** \name Minimum-width integer types
00156     Integer types having at least the specified width */
00157
00158 /**@{*/
00159
00160 /** \ingroup avr_stdint
00161     signed int with at least 8 bits. */
00162
00163 typedef int8_t  int_least8_t;
00164
00165 /** \ingroup avr_stdint
00166     unsigned int with at least 8 bits. */
00167
00168 typedef uint8_t uint_least8_t;
00169
00170 /** \ingroup avr_stdint
00171     signed int with at least 16 bits. */
00172
00173 typedef int16_t int_least16_t;
00174
00175 /** \ingroup avr_stdint
00176     unsigned int with at least 16 bits. */
00177
00178 typedef uint16_t uint_least16_t;
00179
00180 /** \ingroup avr_stdint
00181     signed int with at least 32 bits. */
00182
00183 typedef int32_t int_least32_t;
00184
00185 /** \ingroup avr_stdint
00186     unsigned int with at least 32 bits. */
00187
00188 typedef uint32_t uint_least32_t;
00189
00190 #if !__USING_MINT8 || defined(__DOXYGEN__)
00191 /** \ingroup avr_stdint
00192     signed int with at least 64 bits.
00193     \note This type is not available when the compiler
00194     option -mint8 is in effect. */
00195
00196 typedef int64_t int_least64_t;
00197
00198 /** \ingroup avr_stdint
00199     unsigned int with at least 64 bits.
00200     \note This type is not available when the compiler
00201     option -mint8 is in effect. */
00202
00203 typedef uint64_t uint_least64_t;
00204 #endif
00205
00206 /**@}*/
00207
00208
00209 /** \name Fastest minimum-width integer types
00210     Integer types being usually fastest having at least the specified width */
00211
00212 /**@{*/
00213
00214 /** \ingroup avr_stdint
00215     fastest signed int with at least 8 bits. */
00216
00217 typedef int8_t int_fast8_t;
00218
00219 /** \ingroup avr_stdint
00220     fastest unsigned int with at least 8 bits. */
00221
00222 typedef uint8_t uint_fast8_t;
00223
00224 /** \ingroup avr_stdint
00225     fastest signed int with at least 16 bits. */
00226
00227 typedef int16_t int_fast16_t;
00228
00229 /** \ingroup avr_stdint
00230     fastest unsigned int with at least 16 bits. */
00231
00232 typedef uint16_t uint_fast16_t;
00233
00234 /** \ingroup avr_stdint
00235     fastest signed int with at least 32 bits. */
00236
00237 typedef int32_t int_fast32_t;
00238
```

```

00239 /** \ingroup avr_stdint
00240     fastest unsigned int with at least 32 bits. */
00241
00242 typedef uint32_t uint_fast32_t;
00243
00244 #if !__USING_MINT8 || defined(__DOXYGEN__)
00245 /** \ingroup avr_stdint
00246     fastest signed int with at least 64 bits.
00247     \note This type is not available when the compiler
00248     option -mint8 is in effect. */
00249
00250 typedef int64_t int_fast64_t;
00251
00252 /** \ingroup avr_stdint
00253     fastest unsigned int with at least 64 bits.
00254     \note This type is not available when the compiler
00255     option -mint8 is in effect. */
00256
00257 typedef uint64_t uint_fast64_t;
00258 #endif
00259
00260 /** @} */
00261
00262
00263 /** \name Greatest-width integer types
00264     Types designating integer data capable of representing any value of
00265     any integer type in the corresponding signed or unsigned category */
00266
00267 /** @{ */
00268
00269 #if __USING_MINT8
00270 typedef int32_t intmax_t;
00271
00272 typedef uint32_t uintmax_t;
00273 #else /* !__USING_MINT8 */
00274 /** \ingroup avr_stdint
00275     largest signed int available. */
00276
00277 typedef int64_t intmax_t;
00278
00279 /** \ingroup avr_stdint
00280     largest unsigned int available. */
00281
00282 typedef uint64_t uintmax_t;
00283 #endif /* __USING_MINT8 */
00284
00285 /** @} */
00286
00287 #ifndef __DOXYGEN__
00288 /* Helping macro */
00289 #ifndef __CONCAT
00290 #define __CONCATenate(left, right) left ## right
00291 #define __CONCAT(left, right) __CONCATenate(left, right)
00292 #endif
00293
00294 #endif /* !__DOXYGEN__ */
00295
00296 #if !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS)
00297
00298 /** \name Limits of specified-width integer types
00299     C++ implementations should define these macros only when
00300     __STDC_LIMIT_MACROS is defined before <stdint.h> is included */
00301
00302 /** @{ */
00303
00304 /** \ingroup avr_stdint
00305     largest positive value an int8_t can hold. */
00306
00307 #define INT8_MAX 0x7f
00308
00309 /** \ingroup avr_stdint
00310     smallest negative value an int8_t can hold. */
00311
00312 #define INT8_MIN (-INT8_MAX - 1)
00313
00314 #if __USING_MINT8
00315
00316 #define UINT8_MAX (__CONCAT(INT8_MAX, U) * 2U + 1U)
00317
00318 #define INT16_MAX 0x7fffL
00319 #define INT16_MIN (-INT16_MAX - 1L)
00320 #define UINT16_MAX (__CONCAT(INT16_MAX, U) * 2UL + 1UL)
00321
00322 #define INT32_MAX 0x7fffffffLL
00323 #define INT32_MIN (-INT32_MAX - 1LL)
00324 #define UINT32_MAX (__CONCAT(INT32_MAX, U) * 2ULL + 1ULL)
00325

```

```
00326 #else /* !__USING_MINT8 */
00327
00328 /** \ingroup avr_stdint
00329     largest value an uint8_t can hold. */
00330
00331 #define UINT8_MAX (INT8_MAX * 2 + 1)
00332
00333 /** \ingroup avr_stdint
00334     largest positive value an int16_t can hold. */
00335
00336 #define INT16_MAX 0x7fff
00337
00338 /** \ingroup avr_stdint
00339     smallest negative value an int16_t can hold. */
00340
00341 #define INT16_MIN (-INT16_MAX - 1)
00342
00343 /** \ingroup avr_stdint
00344     largest value an uint16_t can hold. */
00345
00346 #define UINT16_MAX (__CONCAT(INT16_MAX, U) * 2U + 1U)
00347
00348 /** \ingroup avr_stdint
00349     largest positive value an int32_t can hold. */
00350
00351 #define INT32_MAX 0x7fffffffL
00352
00353 /** \ingroup avr_stdint
00354     smallest negative value an int32_t can hold. */
00355
00356 #define INT32_MIN (-INT32_MAX - 1L)
00357
00358 /** \ingroup avr_stdint
00359     largest value an uint32_t can hold. */
00360
00361 #define UINT32_MAX (__CONCAT(INT32_MAX, U) * 2UL + 1UL)
00362
00363 #endif /* __USING_MINT8 */
00364
00365 /** \ingroup avr_stdint
00366     largest positive value an int64_t can hold. */
00367
00368 #define INT64_MAX 0x7fffffffffffffffLL
00369
00370 /** \ingroup avr_stdint
00371     smallest negative value an int64_t can hold. */
00372
00373 #define INT64_MIN (-INT64_MAX - 1LL)
00374
00375 /** \ingroup avr_stdint
00376     largest value an uint64_t can hold. */
00377
00378 #define UINT64_MAX (__CONCAT(INT64_MAX, U) * 2ULL + 1ULL)
00379
00380 /** @} */
00381
00382 /** \name Limits of minimum-width integer types */
00383 /** @{ */
00384
00385 /** \ingroup avr_stdint
00386     largest positive value an int_least8_t can hold. */
00387
00388 #define INT_LEAST8_MAX INT8_MAX
00389
00390 /** \ingroup avr_stdint
00391     smallest negative value an int_least8_t can hold. */
00392
00393 #define INT_LEAST8_MIN INT8_MIN
00394
00395 /** \ingroup avr_stdint
00396     largest value an uint_least8_t can hold. */
00397
00398 #define UINT_LEAST8_MAX UINT8_MAX
00399
00400 /** \ingroup avr_stdint
00401     largest positive value an int_least16_t can hold. */
00402
00403 #define INT_LEAST16_MAX INT16_MAX
00404
00405 /** \ingroup avr_stdint
00406     smallest negative value an int_least16_t can hold. */
00407
00408 #define INT_LEAST16_MIN INT16_MIN
00409
00410 /** \ingroup avr_stdint
00411     largest value an uint_least16_t can hold. */
00412
```

```
00413 #define UINT_LEAST16_MAX UINT16_MAX
00414
00415 /** \ingroup avr_stdint
00416     largest positive value an int_least32_t can hold. */
00417
00418 #define INT_LEAST32_MAX INT32_MAX
00419
00420 /** \ingroup avr_stdint
00421     smallest negative value an int_least32_t can hold. */
00422
00423 #define INT_LEAST32_MIN INT32_MIN
00424
00425 /** \ingroup avr_stdint
00426     largest value an uint_least32_t can hold. */
00427
00428 #define UINT_LEAST32_MAX UINT32_MAX
00429
00430 /** \ingroup avr_stdint
00431     largest positive value an int_least64_t can hold. */
00432
00433 #define INT_LEAST64_MAX INT64_MAX
00434
00435 /** \ingroup avr_stdint
00436     smallest negative value an int_least64_t can hold. */
00437
00438 #define INT_LEAST64_MIN INT64_MIN
00439
00440 /** \ingroup avr_stdint
00441     largest value an uint_least64_t can hold. */
00442
00443 #define UINT_LEAST64_MAX UINT64_MAX
00444
00445 /**}*/
00446
00447 /** \name Limits of fastest minimum-width integer types */
00448
00449 /**@{*/
00450
00451 /** \ingroup avr_stdint
00452     largest positive value an int_fast8_t can hold. */
00453
00454 #define INT_FAST8_MAX INT8_MAX
00455
00456 /** \ingroup avr_stdint
00457     smallest negative value an int_fast8_t can hold. */
00458
00459 #define INT_FAST8_MIN INT8_MIN
00460
00461 /** \ingroup avr_stdint
00462     largest value an uint_fast8_t can hold. */
00463
00464 #define UINT_FAST8_MAX UINT8_MAX
00465
00466 /** \ingroup avr_stdint
00467     largest positive value an int_fast16_t can hold. */
00468
00469 #define INT_FAST16_MAX INT16_MAX
00470
00471 /** \ingroup avr_stdint
00472     smallest negative value an int_fast16_t can hold. */
00473
00474 #define INT_FAST16_MIN INT16_MIN
00475
00476 /** \ingroup avr_stdint
00477     largest value an uint_fast16_t can hold. */
00478
00479 #define UINT_FAST16_MAX UINT16_MAX
00480
00481 /** \ingroup avr_stdint
00482     largest positive value an int_fast32_t can hold. */
00483
00484 #define INT_FAST32_MAX INT32_MAX
00485
00486 /** \ingroup avr_stdint
00487     smallest negative value an int_fast32_t can hold. */
00488
00489 #define INT_FAST32_MIN INT32_MIN
00490
00491 /** \ingroup avr_stdint
00492     largest value an uint_fast32_t can hold. */
00493
00494 #define UINT_FAST32_MAX UINT32_MAX
00495
00496 /** \ingroup avr_stdint
00497     largest positive value an int_fast64_t can hold. */
00498
00499 #define INT_FAST64_MAX INT64_MAX
```

```
00500
00501 /** \ingroup avr_stdint
00502     smallest negative value an int_fast64_t can hold. */
00503
00504 #define INT_FAST64_MIN INT64_MIN
00505
00506 /** \ingroup avr_stdint
00507     largest value an uint_fast64_t can hold. */
00508
00509 #define UINT_FAST64_MAX UINT64_MAX
00510
00511 /**@{*/
00512
00513 /** \name Limits of integer types capable of holding object pointers */
00514
00515 /**@{*/
00516
00517 /** \ingroup avr_stdint
00518     largest positive value an intptr_t can hold. */
00519
00520 #define INTPTR_MAX INT16_MAX
00521
00522 /** \ingroup avr_stdint
00523     smallest negative value an intptr_t can hold. */
00524
00525 #define INTPTR_MIN INT16_MIN
00526
00527 /** \ingroup avr_stdint
00528     largest value an uintptr_t can hold. */
00529
00530 #define UINTPTR_MAX UINT16_MAX
00531
00532 /**@{*/
00533
00534 /** \name Limits of greatest-width integer types */
00535
00536 /**@{*/
00537
00538 /** \ingroup avr_stdint
00539     largest positive value an intmax_t can hold. */
00540
00541 #define INTMAX_MAX INT64_MAX
00542
00543 /** \ingroup avr_stdint
00544     smallest negative value an intmax_t can hold. */
00545
00546 #define INTMAX_MIN INT64_MIN
00547
00548 /** \ingroup avr_stdint
00549     largest value an uintmax_t can hold. */
00550
00551 #define UINTMAX_MAX UINT64_MAX
00552
00553 /**@{*/
00554
00555 /** \name Limits of other integer types
00556     C++ implementations should define these macros only when
00557     __STDC_LIMIT_MACROS is defined before <stdint.h> is included */
00558
00559 /**@{*/
00560
00561 /** \ingroup avr_stdint
00562     largest positive value a ptrdiff_t can hold. */
00563
00564 #define PTRDIFF_MAX INT16_MAX
00565
00566 /** \ingroup avr_stdint
00567     smallest negative value a ptrdiff_t can hold. */
00568
00569 #define PTRDIFF_MIN INT16_MIN
00570
00571
00572 /* Limits of sig_atomic_t */
00573 /* signal.h is currently not implemented (not avr/signal.h) */
00574
00575 /** \ingroup avr_stdint
00576     largest positive value a sig_atomic_t can hold. */
00577
00578 #define SIG_ATOMIC_MAX INT8_MAX
00579
00580 /** \ingroup avr_stdint
00581     smallest negative value a sig_atomic_t can hold. */
00582
00583 #define SIG_ATOMIC_MIN INT8_MIN
00584
00585
00586 /** \ingroup avr_stdint
```

```

00587     largest value a size_t can hold. */
00588
00589 #define SIZE_MAX UINT16_MAX
00590
00591
00592 /* Limits of wchar_t */
00593 /* wchar.h is currently not implemented */
00594 /* #define WCHAR_MAX */
00595 /* #define WCHAR_MIN */
00596
00597
00598 /* Limits of wint_t */
00599 /* wchar.h is currently not implemented */
00600 #ifndef WCHAR_MAX
00601 #define WCHAR_MAX __WCHAR_MAX__
00602 #define WCHAR_MIN __WCHAR_MIN__
00603 #endif
00604 #ifndef WINT_MAX
00605 #define WINT_MAX __WINT_MAX__
00606 #define WINT_MIN __WINT_MIN__
00607 #endif
00608
00609
00610 #endif /* !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS) */
00611
00612 #if (!defined __cplusplus || __cplusplus >= 201103L \
00613     || defined __STDC_CONSTANT_MACROS)
00614
00615 /** \name Macros for integer constants
00616     C++ implementations should define these macros only when
00617     __STDC_CONSTANT_MACROS is defined before <stdint.h> is included.
00618
00619     These definitions are valid for integer constants without suffix and
00620     for macros defined as integer constant without suffix */
00621
00622 /* The GNU C preprocessor defines special macros in the implementation
00623     namespace to allow a definition that works in #if expressions. */
00624 #ifdef __INT8_C
00625 #define INT8_C(c) __INT8_C(c)
00626 #define INT16_C(c) __INT16_C(c)
00627 #define INT32_C(c) __INT32_C(c)
00628 #define INT64_C(c) __INT64_C(c)
00629 #define UINT8_C(c) __UINT8_C(c)
00630 #define UINT16_C(c) __UINT16_C(c)
00631 #define UINT32_C(c) __UINT32_C(c)
00632 #define UINT64_C(c) __UINT64_C(c)
00633 #define INTMAX_C(c) __INTMAX_C(c)
00634 #define UINTMAX_C(c) __UINTMAX_C(c)
00635 #else
00636 /** \ingroup avr_stdint
00637     define a constant of type int8_t */
00638
00639 #define INT8_C(value) ((int8_t) value)
00640
00641 /** \ingroup avr_stdint
00642     define a constant of type uint8_t */
00643
00644 #define UINT8_C(value) ((uint8_t) __CONCAT(value, U))
00645
00646 #if __USING_MINT8
00647
00648 #define INT16_C(value) __CONCAT(value, L)
00649 #define UINT16_C(value) __CONCAT(value, UL)
00650
00651 #define INT32_C(value) ((int32_t) __CONCAT(value, LL))
00652 #define UINT32_C(value) ((uint32_t) __CONCAT(value, ULL))
00653
00654 #else /* !__USING_MINT8 */
00655
00656 /** \ingroup avr_stdint
00657     define a constant of type int16_t */
00658
00659 #define INT16_C(value) value
00660
00661 /** \ingroup avr_stdint
00662     define a constant of type uint16_t */
00663
00664 #define UINT16_C(value) __CONCAT(value, U)
00665
00666 /** \ingroup avr_stdint
00667     define a constant of type int32_t */
00668
00669 #define INT32_C(value) __CONCAT(value, L)
00670
00671 /** \ingroup avr_stdint
00672     define a constant of type uint32_t */
00673

```

```

00674 #define UINT32_C(value) __CONCAT(value, UL)
00675
00676 #endif /* __USING_MINT8 */
00677
00678 /** \ingroup avr_stdint
00679     define a constant of type int64_t */
00680
00681 #define INT64_C(value) __CONCAT(value, LL)
00682
00683 /** \ingroup avr_stdint
00684     define a constant of type uint64_t */
00685
00686 #define UINT64_C(value) __CONCAT(value, ULL)
00687
00688 /** \ingroup avr_stdint
00689     define a constant of type intmax_t */
00690
00691 #define INTMAX_C(value) __CONCAT(value, LL)
00692
00693 /** \ingroup avr_stdint
00694     define a constant of type uintmax_t */
00695
00696 #define UINTMAX_C(value) __CONCAT(value, ULL)
00697
00698 #endif /* !__INT8_C */
00699
00700 /**@}*/
00701
00702 #endif /* (!defined __cplusplus || __cplusplus >= 201103L \
00703         || defined __STDC_CONSTANT_MACROS) */
00704
00705
00706 #endif /* _STDINT_H_ */

```

## 23.54 `stdio.h` File Reference

### Macros

- #define `stdin` (`__iob[0]`)
- #define `stdout` (`__iob[1]`)
- #define `stderr` (`__iob[2]`)
- #define `EOF` (-1)
- #define `fdev_set_udata`(stream, u) do { (stream)->udata = u; } while(0)
- #define `fdev_get_udata`(stream) ((stream)->udata)
- #define `fdev_setup_stream`(stream, put, get, rwflag)
- #define `_FDEV_SETUP_READ` `__SRD`
- #define `_FDEV_SETUP_WRITE` `__SWR`
- #define `_FDEV_SETUP_RW` (`__SRD|__SWR`)
- #define `_FDEV_ERR` (-1)
- #define `_FDEV_EOF` (-2)
- #define `FDEV_SETUP_STREAM`(put, get, rwflag)
- #define `fdev_close`()
- #define `putc`(`__c`, `__stream`) `fputc`(`__c`, `__stream`)
- #define `putchar`(`__c`) `fputc`(`__c`, `stdout`)
- #define `getc`(`__stream`) `fgetc`(`__stream`)
- #define `getchar`() `fgetc`(`stdin`)

### Typedefs

- typedef struct `__file` `FILE`



## Functions

- `int fclose` (`FILE *__stream`)
- `int vfprintf` (`FILE *__stream`, `const char *__fmt`, `va_list __ap`)
- `int vfprintf_P` (`FILE *__stream`, `const char *__fmt`, `va_list __ap`)
- `int fputc` (`int __c`, `FILE *__stream`)
- `int printf` (`const char *__fmt`,...)
- `int printf_P` (`const char *__fmt`,...)
- `int vprintf` (`const char *__fmt`, `va_list __ap`)
- `int sprintf` (`char *__s`, `const char *__fmt`,...)
- `int sprintf_P` (`char *__s`, `const char *__fmt`,...)
- `int snprintf` (`char *__s`, `size_t __n`, `const char *__fmt`,...)
- `int snprintf_P` (`char *__s`, `size_t __n`, `const char *__fmt`,...)
- `int vsprintf` (`char *__s`, `const char *__fmt`, `va_list ap`)
- `int vsprintf_P` (`char *__s`, `const char *__fmt`, `va_list ap`)
- `int vsnprintf` (`char *__s`, `size_t __n`, `const char *__fmt`, `va_list ap`)
- `int vsnprintf_P` (`char *__s`, `size_t __n`, `const char *__fmt`, `va_list ap`)
- `int fprintf` (`FILE *__stream`, `const char *__fmt`,...)
- `int fprintf_P` (`FILE *__stream`, `const char *__fmt`,...)
- `int fputs` (`const char *__str`, `FILE *__stream`)
- `int fputs_P` (`const char *__str`, `FILE *__stream`)
- `int puts` (`const char *__str`)
- `int puts_P` (`const char *__str`)
- `size_t fwrite` (`const void *__ptr`, `size_t __size`, `size_t __nmemb`, `FILE *__stream`)
- `int fgetc` (`FILE *__stream`)
- `int ungetc` (`int __c`, `FILE *__stream`)
- `char * fgets` (`char *__str`, `int __size`, `FILE *__stream`)
- `char * gets` (`char *__str`)
- `size_t fread` (`void *__ptr`, `size_t __size`, `size_t __nmemb`, `FILE *__stream`)
- `void clearerr` (`FILE *__stream`)
- `int feof` (`FILE *__stream`)
- `int ferror` (`FILE *__stream`)
- `int vfscanf` (`FILE *__stream`, `const char *__fmt`, `va_list __ap`)
- `int vfscanf_P` (`FILE *__stream`, `const char *__fmt`, `va_list __ap`)
- `int fscanf` (`FILE *__stream`, `const char *__fmt`,...)
- `int fscanf_P` (`FILE *__stream`, `const char *__fmt`,...)
- `int scanf` (`const char *__fmt`,...)
- `int scanf_P` (`const char *__fmt`,...)
- `int vscanf` (`const char *__fmt`, `va_list __ap`)
- `int sscanf` (`const char *__buf`, `const char *__fmt`,...)
- `int sscanf_P` (`const char *__buf`, `const char *__fmt`,...)
- `int fflush` (`FILE *stream`)

## 23.55 stdio.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, 2005, 2007 Joerg Wunsch
00002    All rights reserved.
00003
00004    Portions of documentation Copyright (c) 1990, 1991, 1993
00005    The Regents of the University of California.
00006
00007    All rights reserved.
00008
00009    Redistribution and use in source and binary forms, with or without
00010    modification, are permitted provided that the following conditions are met:
00011
00012    * Redistributions of source code must retain the above copyright

```

```

00013     notice, this list of conditions and the following disclaimer.
00014
00015     * Redistributions in binary form must reproduce the above copyright
00016     notice, this list of conditions and the following disclaimer in
00017     the documentation and/or other materials provided with the
00018     distribution.
00019
00020     * Neither the name of the copyright holders nor the names of
00021     contributors may be used to endorse or promote products derived
00022     from this software without specific prior written permission.
00023
00024     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00025     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00026     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00027     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00028     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00029     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00030     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00031     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00032     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00033     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00034     POSSIBILITY OF SUCH DAMAGE.
00035
00036     $Id$
00037 */
00038
00039 #ifndef _STDIO_H_
00040 #define _STDIO_H_ 1
00041
00042 #ifndef __ASSEMBLER__
00043
00044 #include <inttypes.h>
00045 #include <stdarg.h>
00046
00047 #ifndef __DOXYGEN__
00048 #define __need_NULL
00049 #define __need_size_t
00050 #include <stddef.h>
00051 #endif /* !__DOXYGEN__ */
00052
00053 /** \file */
00054 /** \defgroup avr_stdio <stdio.h>: Standard IO facilities
00055     \code #include <stdio.h> \endcode
00056
00057     <h3>Introduction to the Standard IO facilities</h3>
00058
00059     This file declares the standard IO facilities that are implemented
00060     in AVR-LibC. Due to the nature of the underlying hardware,
00061     only a limited subset of standard IO is implemented. There is no
00062     actual file implementation available, so only device IO can be
00063     performed. Since there's no operating system, the application
00064     needs to provide enough details about their devices in order to
00065     make them usable by the standard IO facilities.
00066
00067     Due to space constraints, some functionality has not been
00068     implemented at all (like some of the \c printf conversions that
00069     have been left out). Nevertheless, potential users of this
00070     implementation should be warned: the \c printf and \c scanf families of functions, although
00071     usually associated with presumably simple things like the
00072     famous "Hello, world!" program, are actually fairly complex
00073     which causes their inclusion to eat up a fair amount of code space.
00074     Also, they are not fast due to the nature of interpreting the
00075     format string at run-time. Whenever possible, resorting to the
00076     (sometimes non-standard) predetermined conversion facilities that are
00077     offered by AVR-LibC will usually cost much less in terms of speed
00078     and code size.
00079
00080     <h3>Tunable options for code size vs. feature set</h3>
00081
00082     In order to allow programmers a code size vs. functionality tradeoff,
00083     the function vfprintf() which is the heart of the printf family can be
00084     selected in different flavours using linker options. See the
00085     documentation of vfprintf() for a detailed description. The same
00086     applies to vscanf() and the \c scanf family of functions.
00087
00088     <h3>Outline of the chosen API</h3>
00089
00090     The standard streams \c stdin, \c stdout, and \c stderr are
00091     provided, but contrary to the C standard, since AVR-LibC has no
00092     knowledge about applicable devices, these streams are not already
00093     pre-initialized at application startup. Also, since there is no
00094     notion of "file" whatsoever to AVR-LibC, there is no function
00095     \c fopen() that could be used to associate a stream to some device.
00096     (See \ref stdio_note1 "note 1".) Instead, the function \c fdevopen()
00097     is provided to associate a stream to a device, where the device
00098     needs to provide a function to send a character, to receive a
00099     character, or both. There is no differentiation between "text" and

```

00100 "binary" streams inside AVR-LibC. Character `\c \n` is sent  
00101 literally down to the device's `\c put()` function. If the device  
00102 requires a carriage return (`\c \r`) character to be sent before  
00103 the linefeed, its `\c put()` routine must implement this (see  
00104 `\ref stdio_note2 "note 2"`).  
00105  
00106 As an alternative method to `fdevopen()`, the macro  
00107 `fdev_setup_stream()` might be used to setup a user-supplied FILE  
00108 structure.  
00109  
00110 It should be noted that the automatic conversion of a newline  
00111 character into a carriage return - newline sequence breaks binary  
00112 transfers. If binary transfers are desired, no automatic  
00113 conversion should be performed, but instead any string that aims  
00114 to issue a CR-LF sequence must use `<tt>"\r\n"</tt>` explicitly.  
00115  
00116 For convenience, the first call to `\c fdevopen()` that opens a  
00117 stream for reading will cause the resulting stream to be aliased  
00118 to `\c stdin`. Likewise, the first call to `\c fdevopen()` that opens  
00119 a stream for writing will cause the resulting stream to be aliased  
00120 to both, `\c stdout`, and `\c stderr`. Thus, if the open was done  
00121 with both, read and write intent, all three standard streams will  
00122 be identical. Note that these aliases are indistinguishable from  
00123 each other, thus calling `\c fclose()` on such a stream will also  
00124 effectively close all of its aliases (`\ref stdio_note3 "note 3"`).  
00125  
00126 It is possible to tie additional user data to a stream, using  
00127 `fdev_set_udata()`. The backend put and get functions can then  
00128 extract this user data using `fdev_get_udata()`, and act  
00129 appropriately. For example, a single put function could be used  
00130 to talk to two different UARTs that way, or the put and get  
00131 functions could keep internal state between calls there.  
00132  
00133 <h3>Format strings in flash ROM</h3>  
00134  
00135 All the `\c printf` and `\c scanf` family functions come in two flavours: the  
00136 standard name, where the format string is expected to be in  
00137 SRAM, as well as a version with the suffix `"_P"` where the format  
00138 string is expected to reside in the flash ROM. The macro  
00139 `#PSTR` (explained in `\ref avr_pgmspace`) becomes very handy  
00140 for declaring these format strings.  
00141  
00142 `\anchor stdio_without_malloc`  
00143 <h3>Running stdio without malloc()</h3>  
00144  
00145 By default, `fdevopen()` requires `malloc()`. As this is often  
00146 not desired in the limited environment of a microcontroller, an  
00147 alternative option is provided to run completely without `malloc()`.  
00148  
00149 The macro `fdev_setup_stream()` is provided to prepare a  
00150 user-supplied FILE buffer for operation with stdio.  
00151  
00152 <h4>Example</h4>  
00153  
00154 `\code`  
00155 `#include <stdio.h>`  
00156  
00157 `static int uart_putchar(char c, FILE *stream);`  
00158  
00159 `static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,`  
00160 `_FDEV_SETUP_WRITE);`  
00161  
00162 `static int`  
00163 `uart_putchar(char c, FILE *stream)`  
00164 `{`  
00165 `if (c == '\n')`  
00166 `uart_putchar('\r', stream);`  
00167 `loop_until_bit_is_set(UCSRA, UDRE);`  
00168 `UDR = c;`  
00169 `return 0;`  
00170 `}`  
00171  
00172  
00173 `int`  
00174 `main(void)`  
00175 `{`  
00176 `init_uart();`  
00177 `stdout = &mystdout;`  
00178 `printf("Hello, world!\n");`  
00179  
00180 `return 0;`  
00181 `}`  
00182 `\endcode`  
00183  
00184 This example uses the initializer form `FDEV_SETUP_STREAM()` rather  
00185 than the function-like `fdev_setup_stream()`, so all data  
00186 initialization happens during C start-up.

```

00187
00188     If streams initialized that way are no longer needed, they can be
00189     destroyed by first calling the macro fdev_close(), and then
00190     destroying the object itself. No call to fclose() should be
00191     issued for these streams. While calling fclose() itself is
00192     harmless, it will cause an undefined reference to free() and thus
00193     cause the linker to link the malloc module into the application.
00194
00195     <h3>Notes</h3>
00196
00197     <dl>
00198     <dt>\anchor stdio_note1 Note 1:</dt>
00199     <dd>
00200         It might have been possible to implement a device abstraction that
00201         is compatible with \c fopen() but since this would have required
00202         to parse a string, and to take all the information needed either
00203         out of this string, or out of an additional table that would need to be
00204         provided by the application, this approach was not taken.
00205     </dd>
00206     <dt>\anchor stdio_note2 Note 2:</dt>
00207     <dd>
00208         This basically follows the Unix approach: if a device such as a
00209         terminal needs special handling, it is in the domain of the
00210         terminal device driver to provide this functionality. Thus, a
00211         simple function suitable as \c put() for \c fdevopen() that talks
00212         to a UART interface might look like this:
00213
00214         \code
00215         int
00216         uart_putchar(char c, FILE *stream)
00217         {
00218             if (c == '\n')
00219                 uart_putchar('\r', stream);
00220             loop_until_bit_is_set(UCSRA, UDRE);
00221             UDR = c;
00222             return 0;
00223         }
00224         \endcode
00225     </dd>
00226     <dt>\anchor stdio_note3 Note 3:</dt>
00227     <dd>
00228         This implementation has been chosen because the cost of maintaining
00229         an alias is considerably smaller than the cost of maintaining full
00230         copies of each stream. Yet, providing an implementation that offers
00231         the complete set of standard streams was deemed to be useful. Not
00232         only that writing \c printf() instead of <tt>fprintf(mystream, ...)</tt>
00233         saves typing work, but since avr-gcc needs to resort to pass all
00234         arguments of variadic functions on the stack (as opposed to passing
00235         them in registers for functions that take a fixed number of
00236         parameters), the ability to pass one parameter less by implying
00237         \c stdin or stdout will also save some execution time.
00238     </dd>
00239 </dl>
00240 </dl>
00241 */
00242
00243 #if !defined(__DOXYGEN__)
00244
00245 /*
00246  * This is an internal structure of the library that is subject to be
00247  * changed without warnings at any time. Please do *never* reference
00248  * elements of it beyond by using the official interfaces provided.
00249  */
00250 struct __file {
00251     char *buf; /* buffer pointer */
00252     unsigned char ungetc; /* ungetc() buffer */
00253     uint8_t flags; /* flags, see below */
00254     #define __SRD 0x0001 /* OK to read */
00255     #define __SWR 0x0002 /* OK to write */
00256     #define __SSBF 0x0004 /* this is an sprintf/snprintf string */
00257     #define __SPGM 0x0008 /* fmt string is in progmem */
00258     #define __SERR 0x0010 /* found error */
00259     #define __SEOF 0x0020 /* found EOF */
00260     #define __SUNGET 0x0040 /* ungetc() happened */
00261     #define __SMALLOC 0x0080 /* handle is malloc()ed */
00262     #if 0
00263     /* possible future extensions, will require uint16_t flags */
00264     #define __SRW 0x0100 /* open for reading & writing */
00265     #define __SLBF 0x0200 /* line buffered */
00266     #define __SNBF 0x0400 /* unbuffered */
00267     #define __SMBF 0x0800 /* buf is from malloc */
00268     #endif
00269     int size; /* size of buffer */
00270     int len; /* characters read or written so far */
00271     int (*put)(char, struct __file *); /* function to write one char to device */
00272     int (*get)(struct __file *); /* function to read one char from device */
00273     void *udata; /* User defined and accessible data. */

```

```
00274 };
00275
00276 #endif /* not __DOXYGEN__ */
00277
00278 /**@{*/
00279 /**
00280  \c FILE is the opaque structure that is passed around between the
00281  various standard IO functions.
00282  */
00283 typedef struct __file FILE;
00284
00285 /**
00286  Stream that will be used as an input stream by the simplified
00287  functions that don't take a \c stream argument.
00288
00289  The first stream opened with read intent using \c fdevopen()
00290  will be assigned to \c stdin.
00291  */
00292 #define stdin (__iob[0])
00293
00294 /**
00295  Stream that will be used as an output stream by the simplified
00296  functions that don't take a \c stream argument.
00297
00298  The first stream opened with write intent using \c fdevopen()
00299  will be assigned to both, \c stdin, and \c stderr.
00300  */
00301 #define stdout (__iob[1])
00302
00303 /**
00304  Stream destined for error output. Unless specifically assigned,
00305  identical to \c stdout.
00306
00307  If \c stderr should point to another stream, the result of
00308  another \c fdevopen() must be explicitly assigned to it without
00309  closing the previous \c stderr (since this would also close
00310  \c stdout).
00311  */
00312 #define stderr (__iob[2])
00313
00314 /**
00315  \c EOF declares the value that is returned by various standard IO
00316  functions in case of an error. Since the AVR platform (currently)
00317  doesn't contain an abstraction for actual files, its origin as
00318  "end of file" is somewhat meaningless here.
00319  */
00320 #define EOF (-1)
00321
00322 /** This macro inserts a pointer to user defined data into a FILE
00323  stream object.
00324
00325  The user data can be useful for tracking state in the put and get
00326  functions supplied to the fdevopen() function. */
00327 #define fdev_set_ufdata(stream, u) do { (stream)->ufdata = u; } while(0)
00328
00329 /** This macro retrieves a pointer to user defined data from a FILE
00330  stream object. */
00331 #define fdev_get_ufdata(stream) ((stream)->ufdata)
00332
00333 #if defined(__DOXYGEN__)
00334 /**
00335  \brief Setup a user-supplied buffer as an stdio stream
00336
00337  This macro takes a user-supplied buffer \c stream, and sets it up
00338  as a stream that is valid for stdio operations, similar to one that
00339  has been obtained dynamically from fdevopen(). The buffer to setup
00340  must be of type #FILE.
00341
00342  The arguments \c put and \c get are identical to those that need to
00343  be passed to fdevopen().
00344
00345  The \c rflag argument can take one of the values #_FDEV_SETUP_READ,
00346  #_FDEV_SETUP_WRITE, or #_FDEV_SETUP_RW, for read, write, or read/write
00347  intent, respectively.
00348
00349  \note No assignments to the standard streams will be performed by
00350  fdev_setup_stream(). If standard streams are to be used, these
00351  need to be assigned by the user. See also under
00352  \ref stdio_without_malloc "Running stdio without malloc()".
00353  */
00354 #define fdev_setup_stream(stream, put, get, rflag)
00355 #else /* !DOXYGEN */
00356 #define fdev_setup_stream(stream, p, g, f) \
00357 do { \
00358     (stream)->put = p; \
00359     (stream)->get = g; \
00360     (stream)->flags = f; \

```

```

00361         (stream)->udata = 0; \
00362     } while(0)
00363 #endif /* DOXYGEN */
00364
00365 #define _FDEV_SETUP_READ  __SRD /**< fdev_setup_stream() with read intent */
00366 #define _FDEV_SETUP_WRITE __SWR /**< fdev_setup_stream() with write intent */
00367 #define _FDEV_SETUP_RW   (__SRD|__SWR) /**< fdev_setup_stream() with read/write intent */
00368
00369 /**
00370  * Return code for an error condition during device read.
00371  *
00372  * To be used in the get function of fdevopen().
00373  */
00374 #define _FDEV_ERR (-1)
00375
00376 /**
00377  * Return code for an end-of-file condition during device read.
00378  *
00379  * To be used in the get function of fdevopen().
00380  */
00381 #define _FDEV_EOF (-2)
00382
00383 #if defined(__DOXYGEN__)
00384 /**
00385  \brief Initializer for a user-supplied stdio stream
00386
00387  This macro acts similar to fdev_setup_stream(), but it is to be
00388  used as the initializer of a variable of type FILE.
00389
00390  The remaining arguments are to be used as explained in
00391  fdev_setup_stream().
00392  */
00393 #define FDEV_SETUP_STREAM(put, get, rwflag)
00394 #else /* !DOXYGEN */
00395 /* In order to work with C++, we have to mention the fields in the order
00396  as they appear in struct __file. Also, designated initializers are
00397  only supported since C++20. */
00398 #define FDEV_SETUP_STREAM(PU, GE, FL) \
00399     { \
00400         (char*) 0 /* buf */, \
00401         0u /* unget */, \
00402         FL /* flags */, \
00403         0 /* size */, \
00404         0 /* len */, \
00405         PU /* put */, \
00406         GE /* get */, \
00407         (void*) 0 /* udata */ \
00408     } \
00409 #endif /* DOXYGEN */
00410
00411 #ifndef __cplusplus
00412 extern "C" {
00413 #endif
00414
00415 #if !defined(__DOXYGEN__)
00416 /*
00417  * Doxygen documentation can be found in fdevopen.c.
00418  */
00419
00420 extern struct __file *__iob[];
00421
00422 #if defined(__STDIO_FDEVOPEN_COMPAT_12)
00423 /*
00424  * Declare prototype for the discontinued version of fdevopen() that
00425  * has been in use up to AVR-LibC 1.2.x. The new implementation has
00426  * some backwards compatibility with the old version.
00427  */
00428 extern FILE *fdevopen(int (*__put)(char), int (*__get)(void),
00429                      int __opts __attribute__((unused)));
00430 #else /* !defined(__STDIO_FDEVOPEN_COMPAT_12) */
00431 /* New prototype for AVR-LibC 1.4 and above. */
00432 extern FILE *fdevopen(int (*__put)(char, FILE*), int (*__get)(FILE*));
00433 #endif /* defined(__STDIO_FDEVOPEN_COMPAT_12) */
00434
00435 #endif /* not __DOXYGEN__ */
00436
00437 /**
00438  This function closes \c stream, and disallows and further
00439  IO to and from it.
00440
00441  When using fdevopen() to setup the stream, a call to fclose() is
00442  needed in order to free the internal resources allocated.
00443
00444  If the stream has been set up using fdev_setup_stream() or
00445  FDEV_SETUP_STREAM(), use fdev_close() instead.
00446
00447  It currently always returns 0 (for success).

```

```

00448 */
00449 extern int fclose(FILE *__stream);
00450
00451 /**
00452     This macro frees up any library resources that might be associated
00453     with \c stream. It should be called if \c stream is no longer
00454     needed, right before the application is going to destroy the
00455     \c stream object itself.
00456
00457     (Currently, this macro evaluates to nothing, but this might change
00458     in future versions of the library.)
00459 */
00460 #if defined(__DOXYGEN__)
00461 # define fdev_close()
00462 #else
00463 # define fdev_close() ((void)0)
00464 #endif
00465
00466 /**
00467     \c fprintf is the central facility of the \c printf family of
00468     functions. It outputs values to \c stream under control of a
00469     format string passed in \c fmt. The actual values to print are
00470     passed as a variable argument list \c ap.
00471
00472     \c fprintf returns the number of characters written to \c stream,
00473     or \c EOF in case of an error. Currently, this will only happen
00474     if \c stream has not been opened with write intent.
00475
00476     The format string is composed of zero or more directives: ordinary
00477     characters (not \c %), which are copied unchanged to the output
00478     stream; and conversion specifications, each of which results in
00479     fetching zero or more subsequent arguments. Each conversion
00480     specification is introduced by the \c % character. The arguments must
00481     properly correspond (after type promotion) with the conversion
00482     specifier. After the \c %, the following appear in sequence:
00483
00484     - Zero or more of the following flags:
00485       <ul>
00486         <li> \c # The value should be converted to an "alternate form". For
00487           c, d, i, s, and u conversions, this option has no effect.
00488           For o conversions, the precision of the number is
00489           increased to force the first character of the output
00490           string to a zero (except if a zero value is printed with
00491           an explicit precision of zero). For x and X conversions,
00492           a non-zero result has the string '0x' (or '0X' for X
00493           conversions) prepended to it.</li>
00494         <li> \c 0 (zero) Zero padding. For all conversions, the converted
00495           value is padded on the left with zeros rather than blanks.
00496           If a precision is given with a numeric conversion (d, i,
00497           o, u, i, x, and X), the 0 flag is ignored.</li>
00498         <li> \c - A negative field width flag; the converted value is to be
00499           left adjusted on the field boundary. The converted value
00500           is padded on the right with blanks, rather than on the
00501           left with blanks or zeros. A - overrides a 0 if both are
00502           given.</li>
00503         <li> ' ' (space) A blank should be left before a positive number
00504           produced by a signed conversion (d, or i).</li>
00505         <li> \c + A sign must always be placed before a number produced by a
00506           signed conversion. A + overrides a space if both are
00507           used.</li>
00508       </ul>
00509
00510     - An optional decimal digit string specifying a minimum field width.
00511       If the converted value has fewer characters than the field width, it
00512       will be padded with spaces on the left (or right, if the left-adjustment
00513       flag has been given) to fill out the field width.
00514     - An optional precision, in the form of a period . followed by an
00515       optional digit string. If the digit string is omitted, the
00516       precision is taken as zero. This gives the minimum number of
00517       digits to appear for d, i, o, u, x, and X conversions, or the
00518       maximum number of characters to be printed from a string for \c s
00519       conversions.
00520     - An optional \c l or \c h length modifier, that specifies that the
00521       argument for the d, i, o, u, x, or X conversion is a \c "long int"
00522       rather than \c int. The \c h is ignored, as \c "short int" is
00523       equivalent to \c int.
00524     - A character that specifies the type of conversion to be applied.
00525
00526     The conversion specifiers and their meanings are:
00527
00528     - \c diouxX The int (or appropriate variant) argument is converted
00529       to signed decimal (d and i), unsigned octal (o), unsigned
00530       decimal (u), or unsigned hexadecimal (x and X) notation.
00531       The letters "abcdef" are used for x conversions; the
00532       letters "ABCDEF" are used for X conversions. The
00533       precision, if any, gives the minimum number of digits that
00534       must appear; if the converted value requires fewer digits,

```

```

00535     it is padded on the left with zeros.
00536 - \c p The <tt>void *</tt> argument is taken as an unsigned integer,
00537     and converted similarly as a <tt>%#x</tt> command would do.
00538 - \c c The \c int argument is converted to an \c "unsigned char", and the
00539     resulting character is written.
00540 - \c s The \c "char *" argument is expected to be a pointer to an array
00541     of character type (pointer to a string). Characters from
00542     the array are written up to (but not including) a
00543     terminating NUL character; if a precision is specified, no
00544     more than the number specified are written. If a precision
00545     is given, no null character need be present; if the
00546     precision is not specified, or is greater than the size of
00547     the array, the array must contain a terminating NUL
00548     character.
00549 - \c % A \c % is written. No argument is converted. The complete
00550     conversion specification is "%%".
00551 - \c eE The double argument is rounded and converted in the format
00552     \c "[-]d.ddd±dd" where there is one digit before the
00553     decimal-point character and the number of digits after it
00554     is equal to the precision; if the precision is missing, it
00555     is taken as 6; if the precision is zero, no decimal-point
00556     character appears. An \c E conversion uses the letter \c 'E'
00557     (rather than \c 'e') to introduce the exponent. The exponent
00558     always contains two digits; if the value is zero,
00559     the exponent is 00.
00560 - \c fF The double argument is rounded and converted to decimal notation
00561     in the format \c "[-]ddd.ddd", where the number of digits after the
00562     decimal-point character is equal to the precision specification.
00563     If the precision is missing, it is taken as 6; if the precision
00564     is explicitly zero, no decimal-point character appears. If a
00565     decimal point appears, at least one digit appears before it.
00566 - \c gG The double argument is converted in style \c f or \c e (or
00567     \c F or \c E for \c G conversions). The precision
00568     specifies the number of significant digits. If the
00569     precision is missing, 6 digits are given; if the precision
00570     is zero, it is treated as 1. Style \c e is used if the
00571     exponent from its conversion is less than -4 or greater
00572     than or equal to the precision. Trailing zeros are removed
00573     from the fractional part of the result; a decimal point
00574     appears only if it is followed by at least one digit.
00575 - \c S Similar to the \c s format, except the pointer is expected to
00576     point to a program-memory (ROM) string instead of a RAM string.
00577
00578 In no case does a non-existent or small field width cause truncation of a
00579 numeric field; if the result of a conversion is wider than the field
00580 width, the field is expanded to contain the conversion result.
00581
00582 Since the full implementation of all the mentioned features becomes
00583 fairly large, three different flavours of vfprintf() can be
00584 selected using linker options. The default vfprintf() implements
00585 all the mentioned functionality except floating point conversions.
00586 A minimized version of vfprintf() is available that only implements
00587 the very basic integer and string conversion facilities, but only
00588 the \c # additional option can be specified using conversion
00589 flags (these flags are parsed correctly from the format
00590 specification, but then simply ignored). This version can be
00591 requested using the following \ref gcc\_minusW "compiler options":
00592
00593 \code
00594 -Wl,-u,vfprintf -lprintf_min
00595 \endcode
00596
00597 If the full functionality including the floating point conversions
00598 is required, the following options should be used:
00599
00600 \code
00601 -Wl,-u,vfprintf -lprintf_float -lm
00602 \endcode
00603
00604 \par Limitations:
00605 - The specified width and precision can be at most 255.
00606
00607 \par Notes:
00608 - For floating-point conversions, if you link default or minimized
00609 version of vfprintf(), the symbol \c ? will be output and double
00610 argument will be skipped. So you output below will not be crashed.
00611 For default version the width field and the "pad to left" ( symbol
00612 minus ) option will work in this case.
00613 - The \c hh length modifier is ignored (\c char argument is
00614 promoted to \c int). More exactly, this realization does not check
00615 the number of \c h symbols.
00616 - But the \c ll length modifier will to abort the output, as this
00617 realization does not operate \c long \c long arguments.
00618 - The variable width or precision field (an asterisk \c * symbol)
00619 is not realized and will to abort the output.
00620
00621 */

```



```
00622
00623 extern int  vfprintf(FILE *__stream, const char *__fmt, va_list __ap);
00624
00625 /**
00626     Variant of \c vfprintf() that uses a \c fmt string that resides
00627     in program memory.
00628 */
00629 extern int  vfprintf_P(FILE *__stream, const char *__fmt, va_list __ap);
00630
00631 /**
00632     The function \c fputc sends the character \c c (though given as type
00633     \c int) to \c stream. It returns the character, or \c EOF in case
00634     an error occurred.
00635 */
00636 extern int  fputc(int __c, FILE *__stream);
00637
00638 #if !defined(__DOXYGEN__)
00639
00640 /* putc() function implementation, required by standard */
00641 extern int  putc(int __c, FILE *__stream);
00642
00643 /* putchar() function implementation, required by standard */
00644 extern int  putchar(int __c);
00645
00646 #endif /* not __DOXYGEN__ */
00647
00648 /**
00649     The macro \c putc used to be a "fast" macro implementation with a
00650     functionality identical to fputc(). For space constraints, in
00651     AVR-LibC, it is just an alias for \c fputc.
00652 */
00653 #define putc(__c, __stream) fputc(__c, __stream)
00654
00655 /**
00656     The macro \c putchar sends character \c c to \c stdout.
00657 */
00658 #define putchar(__c) fputc(__c, stdout)
00659
00660 /**
00661     The function \c printf performs formatted output to stream
00662     \c stdout. See \c vfprintf() for details.
00663 */
00664 extern int  printf(const char *__fmt, ...);
00665
00666 /**
00667     Variant of \c printf() that uses a \c fmt string that resides
00668     in program memory.
00669 */
00670 extern int  printf_P(const char *__fmt, ...);
00671
00672 /**
00673     The function \c vprintf performs formatted output to stream
00674     \c stdout, taking a variable argument list as in vfprintf().
00675
00676     See vfprintf() for details.
00677 */
00678 extern int  vprintf(const char *__fmt, va_list __ap);
00679
00680 /**
00681     Variant of \c printf() that sends the formatted characters
00682     to string \c s.
00683 */
00684 extern int  sprintf(char *__s, const char *__fmt, ...);
00685
00686 /**
00687     Variant of \c sprintf() that uses a \c fmt string that resides
00688     in program memory.
00689 */
00690 extern int  sprintf_P(char *__s, const char *__fmt, ...);
00691
00692 /**
00693     Like \c sprintf(), but instead of assuming \c s to be of infinite
00694     size, no more than \c n characters (including the trailing NUL
00695     character) will be converted to \c s.
00696
00697     Returns the number of characters that would have been written to
00698     \c s if there were enough space.
00699 */
00700 extern int  snprintf(char *__s, size_t __n, const char *__fmt, ...);
00701
00702 /**
00703     Variant of \c snprintf() that uses a \c fmt string that resides
00704     in program memory.
00705 */
00706 extern int  snprintf_P(char *__s, size_t __n, const char *__fmt, ...);
00707
00708 /**
```

```
00709     Like \c sprintf() but takes a variable argument list for the
00710     arguments.
00711 */
00712 extern int vsprintf(char *__s, const char *__fmt, va_list ap);
00713
00714 /**
00715     Variant of \c vsprintf() that uses a \c fmt string that resides
00716     in program memory.
00717 */
00718 extern int vsprintf_P(char *__s, const char *__fmt, va_list ap);
00719
00720 /**
00721     Like \c vsprintf(), but instead of assuming \c s to be of infinite
00722     size, no more than \c n characters (including the trailing NUL
00723     character) will be converted to \c s.
00724
00725     Returns the number of characters that would have been written to
00726     \c s if there were enough space.
00727 */
00728 extern int vsnprintf(char *__s, size_t __n, const char *__fmt, va_list ap);
00729
00730 /**
00731     Variant of \c vsnprintf() that uses a \c fmt string that resides
00732     in program memory.
00733 */
00734 extern int vsnprintf_P(char *__s, size_t __n, const char *__fmt, va_list ap);
00735 /**
00736     The function \c fprintf performs formatted output to \c stream.
00737     See \c vfprintf() for details.
00738 */
00739 extern int fprintf(FILE *__stream, const char *__fmt, ...);
00740
00741 /**
00742     Variant of \c fprintf() that uses a \c fmt string that resides
00743     in program memory.
00744 */
00745 extern int fprintf_P(FILE *__stream, const char *__fmt, ...);
00746
00747 /**
00748     Write the string pointed to by \c str to stream \c stream.
00749
00750     Returns 0 on success and EOF on error.
00751 */
00752 extern int fputs(const char *__str, FILE *__stream);
00753
00754 /**
00755     Variant of fputs() where \c str resides in program memory.
00756 */
00757 extern int fputs_P(const char *__str, FILE *__stream);
00758
00759 /**
00760     Write the string pointed to by \c str, and a trailing newline
00761     character, to \c stdout.
00762 */
00763 extern int puts(const char *__str);
00764
00765 /**
00766     Variant of puts() where \c str resides in program memory.
00767 */
00768 extern int puts_P(const char *__str);
00769
00770 /**
00771     Write \c nmemb objects, \c size bytes each, to \c stream.
00772     The first byte of the first object is referenced by \c ptr.
00773
00774     Returns the number of objects successfully written, i. e.
00775     \c nmemb unless an output error occurred.
00776 */
00777 extern size_t fwrite(const void *__ptr, size_t __size, size_t __nmemb,
00778                     FILE *__stream);
00779
00780 /**
00781     The function \c fgetc reads a character from \c stream. It returns
00782     the character, or \c EOF in case end-of-file was encountered or an
00783     error occurred. The routines feof() or ferror() must be used to
00784     distinguish between both situations.
00785 */
00786 extern int fgetc(FILE *__stream);
00787
00788 #if !defined(__DOXYGEN__)
00789
00790 /* getc() function implementation, required by standard */
00791 extern int getc(FILE *__stream);
00792
00793 /* getchar() function implementation, required by standard */
00794 extern int getchar(void);
00795
```

```

00796 #endif /* not __DOXYGEN__ */
00797
00798 /**
00799     The macro \c getc used to be a "fast" macro implementation with a
00800     functionality identical to fgetc(). For space constraints, in
00801     AVR-LibC, it is just an alias for \c fgetc.
00802 */
00803 #define getc(__stream) fgetc(__stream)
00804
00805 /**
00806     The macro \c getchar reads a character from \c stdin. Return
00807     values and error handling is identical to fgetc().
00808 */
00809 #define getchar() fgetc(stdin)
00810
00811 /**
00812     The ungetc() function pushes the character \c c (converted to an
00813     unsigned char) back onto the input stream pointed to by \c stream.
00814     The pushed-back character will be returned by a subsequent read on
00815     the stream.
00816
00817     Currently, only a single character can be pushed back onto the
00818     stream.
00819
00820     The ungetc() function returns the character pushed back after the
00821     conversion, or \c EOF if the operation fails. If the value of the
00822     argument \c c character equals \c EOF, the operation will fail and
00823     the stream will remain unchanged.
00824 */
00825 extern int ungetc(int __c, FILE *__stream);
00826
00827 /**
00828     Read at most <tt>size - 1</tt> bytes from \c stream, until a
00829     newline character was encountered, and store the characters in the
00830     buffer pointed to by \c str. Unless an error was encountered while
00831     reading, the string will then be terminated with a \c NUL
00832     character.
00833
00834     If an error was encountered, the function returns NULL and sets the
00835     error flag of \c stream, which can be tested using ferror().
00836     Otherwise, a pointer to the string will be returned. */
00837 extern char *fgets(char *__str, int __size, FILE *__stream);
00838
00839 /**
00840     Similar to fgets() except that it will operate on stream \c stdin,
00841     and the trailing newline (if any) will not be stored in the string.
00842     It is the caller's responsibility to provide enough storage to hold
00843     the characters read. */
00844 extern char *gets(char *__str);
00845
00846 /**
00847     Read \c nmemb objects, \c size bytes each, from \c stream,
00848     to the buffer pointed to by \c ptr.
00849
00850     Returns the number of objects successfully read, i. e.
00851     \c nmemb unless an input error occurred or end-of-file was
00852     encountered. feof() and ferror() must be used to distinguish
00853     between these two conditions.
00854 */
00855 extern size_t fread(void *__ptr, size_t __size, size_t __nmemb,
00856 FILE *__stream);
00857
00858 /**
00859     Clear the error and end-of-file flags of \c stream.
00860 */
00861 extern void clearerr(FILE *__stream);
00862
00863 #if !defined(__DOXYGEN__)
00864 /* fast inlined version of clearerr() */
00865 #define clearerror(s) do { (s)->flags &= ~(__SERR | __SEOF); } while(0)
00866 #endif /* !defined(__DOXYGEN__) */
00867
00868 /**
00869     Test the end-of-file flag of \c stream. This flag can only be cleared
00870     by a call to clearerr().
00871 */
00872 extern int feof(FILE *__stream);
00873
00874 #if !defined(__DOXYGEN__)
00875 /* fast inlined version of feof() */
00876 #define feof(s) ((s)->flags & __SEOF)
00877 #endif /* !defined(__DOXYGEN__) */
00878
00879 /**
00880     Test the error flag of \c stream. This flag can only be cleared
00881     by a call to clearerr().
00882 */

```

```

00883 extern int  ferror(FILE *__stream);
00884
00885 #if !defined(__DOXYGEN__)
00886 /* fast inlined version of ferror() */
00887 #define ferror(s) ((s)->flags & __SERR)
00888 #endif /* !defined(__DOXYGEN__) */
00889
00890 extern int  vfscanf(FILE *__stream, const char *__fmt, va_list __ap);
00891
00892 /**
00893  Variant of vfscanf() using a \c fmt string in program memory.
00894  */
00895 extern int  vfscanf_P(FILE *__stream, const char *__fmt, va_list __ap);
00896
00897 /**
00898  The function \c fscanf performs formatted input, reading the
00899  input data from \c stream.
00900
00901  See vfscanf() for details.
00902  */
00903 extern int  fscanf(FILE *__stream, const char *__fmt, ...);
00904
00905 /**
00906  Variant of fscanf() using a \c fmt string in program memory.
00907  */
00908 extern int  fscanf_P(FILE *__stream, const char *__fmt, ...);
00909
00910 /**
00911  The function \c scanf performs formatted input from stream \c stdin.
00912
00913  See vfscanf() for details.
00914  */
00915 extern int  scanf(const char *__fmt, ...);
00916
00917 /**
00918  Variant of scanf() where \c fmt resides in program memory.
00919  */
00920 extern int  scanf_P(const char *__fmt, ...);
00921
00922 /**
00923  The function \c vscanf performs formatted input from stream
00924  \c stdin, taking a variable argument list as in vfscanf().
00925
00926  See vfscanf() for details.
00927  */
00928 extern int  vscanf(const char *__fmt, va_list __ap);
00929
00930 /**
00931  The function \c sscanf performs formatted input, reading the
00932  input data from the buffer pointed to by \c buf.
00933
00934  See vfscanf() for details.
00935  */
00936 extern int  sscanf(const char *__buf, const char *__fmt, ...);
00937
00938 /**
00939  Variant of sscanf() using a \c fmt string in program memory.
00940  */
00941 extern int  sscanf_P(const char *__buf, const char *__fmt, ...);
00942
00943 #if defined(__DOXYGEN__)
00944 /**
00945  Flush \c stream.
00946
00947  This is a null operation provided for source-code compatibility
00948  only, as the standard IO implementation currently does not perform
00949  any buffering.
00950  */
00951 extern int  fflush(FILE *stream);
00952 #else
00953 static __inline__ int fflush(FILE *stream __attribute__((unused)))
00954 {
00955     return 0;
00956 }
00957 #endif
00958
00959 #ifndef __DOXYGEN__
00960 /* only mentioned for libstdc++ support, not implemented in library */
00961 #define BUFSIZ 1024
00962 #define _IONBF 0
00963 __extension__ typedef long long fpos_t;
00964 extern int  fgetpos(FILE *stream, fpos_t *pos);
00965 extern FILE *fopen(const char *path, const char *mode);
00966 extern FILE *freopen(const char *path, const char *mode, FILE *stream);
00967 extern FILE *fdopen(int, const char *);
00968 extern int  fseek(FILE *stream, long offset, int whence);
00969 extern int  fsetpos(FILE *stream, fpos_t *pos);

```

```

00970 extern long ftell(FILE *stream);
00971 extern int fileno(FILE *);
00972 extern void perror(const char *s);
00973 extern int remove(const char *pathname);
00974 extern int rename(const char *oldpath, const char *newpath);
00975 extern void rewind(FILE *stream);
00976 extern void setbuf(FILE *stream, char *buf);
00977 extern int setvbuf(FILE *stream, char *buf, int mode, size_t size);
00978 extern FILE *tmpfile(void);
00979 extern char *tmpnam(char *s);
00980 #endif /* !__DOXYGEN__ */
00981
00982 #ifdef __cplusplus
00983 }
00984 #endif
00985
00986 /** @} */
00987
00988 #ifndef __DOXYGEN__
00989 /*
00990  * The following constants are currently not used by AVR-LibC's
00991  * stdio subsystem. They are defined here since the gcc build
00992  * environment expects them to be here.
00993  */
00994 #define SEEK_SET 0
00995 #define SEEK_CUR 1
00996 #define SEEK_END 2
00997
00998 #endif
00999
01000 #endif /* __ASSEMBLER__ */
01001
01002 #endif /* _STDIO_H_ */

```

## 23.56 stdlib.h File Reference

### Data Structures

- struct [div\\_t](#)
- struct [ldiv\\_t](#)

### Macros

- #define [RAND\\_MAX](#) 0x7FFF

### Typedefs

- typedef int(\* [\\_\\_compar\\_fn\\_t](#)) (const void \*, const void \*)

### Functions

- void [abort](#) (void)
- int [abs](#) (int \_\_i)
- long [labs](#) (long \_\_i)
- void \* [bsearch](#) (const void \* \_\_key, const void \* \_\_base, size\_t \_\_nmemb, size\_t \_\_size, int(\* \_\_compar)(const void \*, const void \*))
- [div\\_t](#) [div](#) (int \_\_num, int \_\_denom) \_\_asm\_\_("\_\_divmodhi4")
- [ldiv\\_t](#) [ldiv](#) (long \_\_num, long \_\_denom) \_\_asm\_\_("\_\_divmodsi4")
- void [qsort](#) (void \* \_\_base, size\_t \_\_nmemb, size\_t \_\_size, [\\_\\_compar\\_fn\\_t](#) \_\_compar)
- long [strtol](#) (const char \* \_\_nptr, char \*\* \_\_endptr, int \_\_base)
- unsigned long [strtoul](#) (const char \* \_\_nptr, char \*\* \_\_endptr, int \_\_base)
- long [atol](#) (const char \* \_\_s)
- int [atoi](#) (const char \* \_\_s)

- void `exit` (int `__status`)
- void \* `malloc` (size\_t `__size`)
- void `free` (void \* `__ptr`)
- void \* `calloc` (size\_t `__nele`, size\_t `__size`)
- void \* `realloc` (void \* `__ptr`, size\_t `__size`)
- float `strtof` (const char \* `__nptr`, char \*\* `__endptr`)
- double `strtod` (const char \* `__nptr`, char \*\* `__endptr`)
- long double `strtold` (const char \* `__nptr`, char \*\* `__endptr`)
- int `atexit` (void(\*`func`)(void))
- float `atoff` (const char \* `__nptr`)
- double `atof` (const char \* `__nptr`)
- long double `atofl` (const char \* `__nptr`)
- int `rand` (void)
- void `srand` (unsigned int `__seed`)
- int `rand_r` (unsigned long \* `__ctx`)

### Variables

- size\_t `__malloc_margin`
- char \* `__malloc_heap_start`
- char \* `__malloc_heap_end`

### Non-standard (i.e. non-ISO C) functions.

- #define `RANDOM_MAX` 0x7FFFFFFF
- char \* `itoa` (int `val`, char \* `s`, int `radix`)
- char \* `ltoa` (long `val`, char \* `s`, int `radix`)
- char \* `utoa` (unsigned int `val`, char \* `s`, int `radix`)
- char \* `ultoa` (unsigned long `val`, char \* `s`, int `radix`)
- long `random` (void)
- void `srandom` (unsigned long `__seed`)
- long `random_r` (unsigned long \* `__ctx`)

### Conversion functions for double arguments.

- #define `DTOSTR_ALWAYS_SIGN` 0x01 /\* put '+' or '-' for positives \*/
- #define `DTOSTR_PLUS_SIGN` 0x02 /\* put '+' rather than '-' \*/
- #define `DTOSTR_UPPERCASE` 0x04 /\* put 'E' rather 'e' \*/
- #define `EXIT_SUCCESS` 0
- #define `EXIT_FAILURE` 1
- char \* `fstre` (float `__val`, char \* `__s`, unsigned char `__prec`, unsigned char `__flags`)
- char \* `dstre` (double `__val`, char \* `__s`, unsigned char `__prec`, unsigned char `__flags`)
- char \* `ldstre` (long double `__val`, char \* `__s`, unsigned char `__prec`, unsigned char `__flags`)
- char \* `fstfr` (float `__val`, signed char `__width`, unsigned char `__prec`, char \* `__s`)
- char \* `dstfr` (double `__val`, signed char `__width`, unsigned char `__prec`, char \* `__s`)
- char \* `ldstfr` (long double `__val`, signed char `__width`, unsigned char `__prec`, char \* `__s`)

## 23.57 stdlib.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, Marek Michalkiewicz
00002    Copyright (c) 2004,2007 Joerg Wunsch
00003
00004    Portions of documentation Copyright (c) 1990, 1991, 1993, 1994
00005    The Regents of the University of California.
00006
00007    All rights reserved.
00008
00009    Redistribution and use in source and binary forms, with or without
00010    modification, are permitted provided that the following conditions are met:
00011
00012    * Redistributions of source code must retain the above copyright
00013      notice, this list of conditions and the following disclaimer.
00014
00015    * Redistributions in binary form must reproduce the above copyright
00016      notice, this list of conditions and the following disclaimer in
00017      the documentation and/or other materials provided with the
00018      distribution.
00019
00020    * Neither the name of the copyright holders nor the names of
00021      contributors may be used to endorse or promote products derived
00022      from this software without specific prior written permission.
00023
00024    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00025    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00026    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00027    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00028    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00029    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00030    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00031    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00032    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00033    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00034    POSSIBILITY OF SUCH DAMAGE.
00035
00036    $Id$
00037 */
00038
00039 #ifndef _STDLIB_H_
00040 #define _STDLIB_H_ 1
00041
00042 #ifndef __ASSEMBLER__
00043
00044 #ifndef __DOXYGEN__
00045 #define __need_NULL
00046 #define __need_size_t
00047 #define __need_wchar_t
00048 #include <stddef.h>
00049
00050 #ifndef __ptr_t
00051 #define __ptr_t void *
00052 #endif
00053 #endif /* !__DOXYGEN__ */
00054
00055 #ifdef __cplusplus
00056 extern "C" {
00057 #endif
00058
00059 /** \file */
00060
00061 /** \defgroup avr_stdlib <stdlib.h>: General utilities
00062     \code #include <stdlib.h> \endcode
00063
00064     This file declares some basic C macros and functions as
00065     defined by the ISO standard, plus some AVR-specific extensions.
00066 */
00067
00068 /** \ingroup avr_stdlib */
00069 /**@{*/
00070 /** Result type for function div(). */
00071 typedef struct {
00072     int quot;           /**< The Quotient. */
00073     int rem;           /**< The Remainder. */
00074 } div_t;
00075
00076 /** Result type for function ldiv(). */
00077 typedef struct {
00078     long quot;         /**< The Quotient. */
00079     long rem;         /**< The Remainder. */
00080 } ldiv_t;
00081
00082 /** Comparison function type for qsort(), just for convenience. */
00083 typedef int (*__compar_fn_t)(const void *, const void *);

```

```

00084
00085 #ifndef __DOXYGEN__
00086
00087 #ifndef __ATTR_CONST__
00088 # define __ATTR_CONST__ __attribute__((__const__))
00089 #endif
00090
00091 #ifndef __ATTR_MALLOC__
00092 # define __ATTR_MALLOC__ __attribute__((__malloc__))
00093 #endif
00094
00095 #ifndef __ATTR_NORETURN__
00096 # define __ATTR_NORETURN__ __attribute__((__noreturn__))
00097 #endif
00098
00099 #ifndef __ATTR_PURE__
00100 # define __ATTR_PURE__ __attribute__((__pure__))
00101 #endif
00102
00103 #ifndef __ATTR_GNU_INLINE__
00104 # ifdef __GNUC_STDC_INLINE__
00105 # define __ATTR_GNU_INLINE__ __attribute__((__gnu_inline__))
00106 # else
00107 # define __ATTR_GNU_INLINE__
00108 # endif
00109 #endif
00110
00111 #ifndef __ATTR_ALWAYS_INLINE__
00112 #define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00113 #endif
00114
00115 #endif
00116
00117 /** The abort() function causes abnormal program termination to occur.
00118     This realization disables interrupts and jumps to _exit() function
00119     with argument equal to 1. In the limited AVR environment, execution is
00120     effectively halted by entering an infinite loop. */
00121 extern void abort(void) __ATTR_NORETURN__;
00122
00123 #ifndef __DOXYGEN__
00124 static __ATTR_ALWAYS_INLINE__
00125 int abs (int __i)
00126 {
00127     return __builtin_abs (__i);
00128 }
00129 #endif
00130 /** The abs() function computes the absolute value of the integer \c i.
00131     \note The abs() and labs() functions are builtins of gcc.
00132 */
00133 extern int abs(int __i) __ATTR_CONST__;
00134
00135 #ifndef __DOXYGEN__
00136 static __ATTR_ALWAYS_INLINE__
00137 long labs (long __i)
00138 {
00139     return __builtin_labs (__i);
00140 }
00141 #endif
00142 /** The labs() function computes the absolute value of the long integer
00143     \c i.
00144     \note The abs() and labs() functions are builtins of gcc.
00145 */
00146 extern long labs(long __i) __ATTR_CONST__;
00147
00148 /**
00149     The bsearch() function searches an array of \c nmemb objects, the
00150     initial member of which is pointed to by \c base, for a member
00151     that matches the object pointed to by \c key. The size of each
00152     member of the array is specified by \c size.
00153
00154     The contents of the array should be in ascending sorted order
00155     according to the comparison function referenced by \c compar.
00156     The \c compar routine is expected to have two arguments which
00157     point to the key object and to an array member, in that order,
00158     and should return an integer less than, equal to, or greater than
00159     zero if the key object is found, respectively, to be less than,
00160     to match, or be greater than the array member.
00161
00162     The bsearch() function returns a pointer to a matching member of
00163     the array, or a null pointer if no match is found. If two
00164     members compare as equal, which member is matched is unspecified.
00165 */
00166 extern void *bsearch(const void *__key, const void *__base, size_t __nmemb,
00167                     size_t __size, int (*__compar)(const void *, const void *));
00168
00169 /* __divmodhi4 and __divmodsi4 from libgcc.a */
00170 /**

```



```

00171     The div() function computes the value \c num/denom and returns
00172     the quotient and remainder in a structure named \c div_t that
00173     contains two int members named \c quot and \c rem.
00174 */
00175 extern div_t div(int __num, int __denom) __asm__("__divmodhi4") __ATTR_CONST__;
00176 /**
00177     The ldiv() function computes the value \c num/denom and returns
00178     the quotient and remainder in a structure named \c ldiv_t that
00179     contains two long integer members named \c quot and \c rem.
00180 */
00181 extern ldiv_t ldiv(long __num, long __denom) __asm__("__divmodsi4") __ATTR_CONST__;
00182
00183 /**
00184     The qsort() function is a modified partition-exchange sort, or
00185     quicksort.
00186
00187     The qsort() function sorts an array of \c nmemb objects, the
00188     initial member of which is pointed to by \c base. The size of
00189     each object is specified by \c size. The contents of the array
00190     base are sorted in ascending order according to a comparison
00191     function pointed to by \c compar, which requires two arguments
00192     pointing to the objects being compared.
00193
00194     The comparison function must return an integer less than, equal
00195     to, or greater than zero if the first argument is considered to
00196     be respectively less than, equal to, or greater than the second.
00197 */
00198 extern void qsort(void *__base, size_t __nmemb, size_t __size,
00199                 __compar_fn_t __compar);
00200
00201 /**
00202     The strtol() function converts the string in \c nptr to a long
00203     value. The conversion is done according to the given base, which
00204     must be between 2 and 36 inclusive, or be the special value 0.
00205
00206     The string may begin with an arbitrary amount of white space (as
00207     determined by isspace()) followed by a single optional \c '+' or \c '-'
00208     sign. If \c base is zero or 16, the string may then include a
00209     \c "0x" prefix, and the number will be read in base 16; otherwise,
00210     a zero base is taken as 10 (decimal) unless the next character is
00211     \c '0', in which case it is taken as 8 (octal).
00212
00213     The remainder of the string is converted to a long value in the
00214     obvious manner, stopping at the first character which is not a
00215     valid digit in the given base. (In bases above 10, the letter \c 'A'
00216     in either upper or lower case represents 10, \c 'B' represents 11,
00217     and so forth, with \c 'Z' representing 35.)
00218
00219     If \c endptr is not NULL, strtol() stores the address of the first
00220     invalid character in \c *endptr. If there were no digits at all,
00221     however, strtol() stores the original value of \c nptr in \c
00222     *endptr. (Thus, if \c *nptr is not \c '\\0' but \c **endptr is \c '\\0'
00223     on return, the entire string was valid.)
00224
00225     The strtol() function returns the result of the conversion, unless
00226     the value would underflow or overflow. If no conversion could be
00227     performed, 0 is returned. If an overflow or underflow occurs, \c
00228     errno is set to \ref avr_errno "ERANGE" and the function return value
00229     is clamped to \c LONG_MIN or \c LONG_MAX, respectively.
00230 */
00231 extern long strtol(const char *__nptr, char **__endptr, int __base);
00232
00233 /**
00234     The strtoul() function converts the string in \c nptr to an
00235     unsigned long value. The conversion is done according to the
00236     given base, which must be between 2 and 36 inclusive, or be the
00237     special value 0.
00238
00239     The string may begin with an arbitrary amount of white space (as
00240     determined by isspace()) followed by a single optional \c '+' or \c '-'
00241     sign. If \c base is zero or 16, the string may then include a
00242     \c "0x" prefix, and the number will be read in base 16; otherwise,
00243     a zero base is taken as 10 (decimal) unless the next character is
00244     \c '0', in which case it is taken as 8 (octal).
00245
00246     The remainder of the string is converted to an unsigned long value
00247     in the obvious manner, stopping at the first character which is
00248     not a valid digit in the given base. (In bases above 10, the
00249     letter \c 'A' in either upper or lower case represents 10, \c 'B'
00250     represents 11, and so forth, with \c 'Z' representing 35.)
00251
00252     If \c endptr is not NULL, strtoul() stores the address of the first
00253     invalid character in \c *endptr. If there were no digits at all,
00254     however, strtoul() stores the original value of \c nptr in \c
00255     *endptr. (Thus, if \c *nptr is not \c '\\0' but \c **endptr is \c '\\0'
00256     on return, the entire string was valid.)
00257
00257

```

```

00258     The strtoul() function return either the result of the conversion
00259     or, if there was a leading minus sign, the negation of the result
00260     of the conversion, unless the original (non-negated) value would
00261     overflow; in the latter case, strtoul() returns ULONG_MAX, and \c
00262     errno is set to \ref avr_errno "ERANGE". If no conversion could
00263     be performed, 0 is returned.
00264 */
00265 extern unsigned long strtoul(const char *__nptr, char **__endptr, int __base);
00266
00267 /**
00268     The atol() function converts the initial portion of the string
00269     pointed to by \p s to long integer representation. In contrast to
00270
00271     \code strtol(s, (char **)NULL, 10); \endcode
00272
00273     this function does not detect overflow (\c errno is not changed and
00274     the result value is not predictable), uses smaller memory (flash and
00275     stack) and works more quickly.
00276 */
00277 extern long atol(const char *__s) __ATTR_PURE__;
00278
00279 /**
00280     The atoi() function converts the initial portion of the string
00281     pointed to by \p s to integer representation. In contrast to
00282
00283     \code (int)strtol(s, (char **)NULL, 10); \endcode
00284
00285     this function does not detect overflow (\c errno is not changed and
00286     the result value is not predictable), uses smaller memory (flash and
00287     stack) and works more quickly.
00288 */
00289 extern int atoi(const char *__s) __ATTR_PURE__;
00290
00291 /**
00292     The exit() function terminates the application. Since there is no
00293     environment to return to, \c status is ignored, and code execution
00294     will eventually reach an infinite loop, thereby effectively halting
00295     all code processing. Before entering the infinite loop, interrupts
00296     are globally disabled.
00297
00298     Global destructors will be called before halting
00299     execution, see the \ref sec_dot_fini ".fini" sections.
00300 */
00301 extern void exit(int __status) __ATTR_NORETURN__;
00302
00303 /**
00304     The malloc() function allocates \c size bytes of memory.
00305     If malloc() fails, a NULL pointer is returned.
00306
00307     Note that malloc() does \e not initialize the returned memory to
00308     zero bytes.
00309
00310     See the chapter about \ref malloc "malloc() usage" for implementation
00311     details.
00312 */
00313 extern void *malloc(size_t __size) __ATTR_MALLOC__;
00314
00315 /**
00316     The free() function causes the allocated memory referenced by \c
00317     ptr to be made available for future allocations. If \c ptr is
00318     NULL, no action occurs.
00319 */
00320 extern void free(void *__ptr);
00321
00322 /**
00323     \c malloc() \ref malloc_tunables "tunable".
00324 */
00325 extern size_t __malloc_margin;
00326
00327 /**
00328     \c malloc() \ref malloc_tunables "tunable".
00329 */
00330 extern char *__malloc_heap_start;
00331
00332 /**
00333     \c malloc() \ref malloc_tunables "tunable".
00334 */
00335 extern char *__malloc_heap_end;
00336
00337 /**
00338     Allocate \c nele elements of \c size each. Identical to calling
00339     \c malloc() using <tt>nele * size</tt> as argument, except the
00340     allocated memory will be cleared to zero.
00341 */
00342 extern void *calloc(size_t __nele, size_t __size) __ATTR_MALLOC__;
00343
00344 /**

```

```

00345     The realloc() function tries to change the size of the region
00346     allocated at \c ptr to the new \c size value. It returns a
00347     pointer to the new region. The returned pointer might be the
00348     same as the old pointer, or a pointer to a completely different
00349     region.
00350
00351     The contents of the returned region up to either the old or the new
00352     size value (whatever is less) will be identical to the contents of
00353     the old region, even in case a new region had to be allocated.
00354
00355     It is acceptable to pass \c ptr as NULL, in which case realloc()
00356     will behave identical to malloc().
00357
00358     If the new memory cannot be allocated, realloc() returns NULL, and
00359     the region at \c ptr will not be changed.
00360 */
00361 extern void *realloc(void *__ptr, size_t __size) __ATTR_MALLOC__;
00362
00363 extern float strttof(const char *__nptr, char **__endptr);
00364 /** \ingroup avr_stdlib
00365     The strtod() function is similar to strttof(), except that the conversion
00366     result is of type \c double instead of \c float.
00367
00368     strtod() is currently only supported when \c double is a 32-bit type. */
00369 extern double strtod(const char *__nptr, char **__endptr);
00370 /** \ingroup avr_stdlib
00371     The strtold() function is similar to strttof(), except that the conversion
00372     result is of type \c long \c double instead of \c float.
00373
00374     strtold() is currently only supported when \c long \c double is a
00375     32-bit type. */
00376 extern long double strtold(const char *__nptr, char **__endptr);
00377
00378 /**
00379     \ingroup avr_stdlib
00380     The atexit() function registers function \a func to be run as part of
00381     the \c exit() function during \ref sec_dot_fini ".fini8".
00382     atexit() calls malloc().
00383 */
00384 extern int atexit(void (*func)(void));
00385
00386 /** \ingroup avr_stdlib
00387     \fn float atoff (const char *nptr)
00388
00389     The atoff() function converts the initial portion of the string pointed
00390     to by \a nptr to \c float representation.
00391
00392     It is equivalent to calling
00393     \code strttof(nptr, (char**) 0); \endcode */
00394 extern float atoff(const char *__nptr);
00395 /** \ingroup avr_stdlib
00396     \fn double atof (const char *nptr)
00397
00398     The atof() function converts the initial portion of the string pointed
00399     to by \a nptr to \c double representation.
00400
00401     It is equivalent to calling
00402     \code strtod(nptr, (char**) 0); \endcode */
00403 extern double atof(const char *__nptr);
00404 /** \ingroup avr_stdlib
00405     \fn long double atofl (const char *nptr)
00406
00407     The atofl() function converts the initial portion of the string pointed
00408     to by \a nptr to \c long \c double representation.
00409
00410     It is equivalent to calling
00411     \code strtold(nptr, (char**) 0); \endcode */
00412 extern long double atofl(const char *__nptr);
00413
00414 /** Highest number that can be generated by rand(). */
00415 #define RAND_MAX 0x7FFF
00416
00417 /**
00418     The rand() function computes a sequence of pseudo-random integers in the
00419     range of 0 to \c RAND_MAX (as defined by the header file <stdlib.h>).
00420
00421     The srand() function sets its argument \c seed as the seed for a new
00422     sequence of pseudo-random numbers to be returned by rand(). These
00423     sequences are repeatable by calling srand() with the same seed value.
00424
00425     If no seed value is provided, the functions are automatically seeded with
00426     a value of 1.
00427
00428     In compliance with the C standard, these functions operate on
00429     \c int arguments. Since the underlying algorithm already uses
00430     32-bit calculations, this causes a loss of precision. See
00431     \c random() for an alternate set of functions that retains full

```

```

00432     32-bit precision.
00433 */
00434 extern int rand(void);
00435 /**
00436     Pseudo-random number generator seeding; see rand().
00437 */
00438 extern void srand(unsigned int __seed);
00439
00440 /**
00441     Variant of rand() that stores the context in the user-supplied
00442     variable located at \c ctx instead of a static library variable
00443     so the function becomes re-entrant.
00444 */
00445 extern int rand_r(unsigned long *__ctx);
00446 /**@}*/
00447
00448 /**@{*/
00449 /** \name Non-standard (i.e. non-ISO C) functions.
00450     \ingroup avr_stdlib
00451 */
00452 /**
00453     \brief Convert an integer to a string.
00454
00455     The function itoa() converts the integer value from \c val into an
00456     ASCII representation that will be stored under \c s. The caller
00457     is responsible for providing sufficient storage in \c s.
00458
00459     \note The minimal size of the buffer \c s depends on the choice of
00460     radix. For example, if the radix is 2 (binary), you need to supply a buffer
00461     with a minimal length of 8 * sizeof(int) + 1 characters, i.e. one
00462     character for each bit plus one for the string terminator. Using a larger
00463     radix will require a smaller minimal buffer size.
00464
00465     \warning If the buffer is too small, you risk a buffer overflow.
00466
00467     Conversion is done using the \c radix as base, which may be a
00468     number between 2 (binary conversion) and up to 36. If \c radix
00469     is greater than 10, the next digit after \c '9' will be the letter
00470     \c 'a'.
00471
00472     If radix is 10 and val is negative, a minus sign will be prepended.
00473
00474     The itoa() function returns the pointer passed as \c s.
00475 */
00476 #ifdef __DOXYGEN__
00477 extern char *itoa(int val, char *s, int radix);
00478 #else
00479 extern __inline__ __ATTR_GNU_INLINE__
00480 char *itoa (int __val, char *__s, int __radix)
00481 {
00482     if (!__builtin_constant_p (__radix)) {
00483         extern char *__itoa (int, char *, int);
00484         return __itoa (__val, __s, __radix);
00485     } else if (__radix < 2 || __radix > 36) {
00486         *__s = 0;
00487         return __s;
00488     } else {
00489         extern char *__itoa_ncheck (int, char *, unsigned char);
00490         return __itoa_ncheck (__val, __s, __radix);
00491     }
00492 }
00493 #endif
00494
00495 /**
00496     \ingroup avr_stdlib
00497
00498     \brief Convert a long integer to a string.
00499
00500     The function ltoa() converts the long integer value from \c val into an
00501     ASCII representation that will be stored under \c s. The caller
00502     is responsible for providing sufficient storage in \c s.
00503
00504     \note The minimal size of the buffer \c s depends on the choice of
00505     radix. For example, if the radix is 2 (binary), you need to supply a buffer
00506     with a minimal length of 8 * sizeof(long int) + 1 characters, i.e. one
00507     character for each bit plus one for the string terminator. Using a larger
00508     radix will require a smaller minimal buffer size.
00509
00510     \warning If the buffer is too small, you risk a buffer overflow.
00511
00512     Conversion is done using the \c radix as base, which may be a
00513     number between 2 (binary conversion) and up to 36. If \c radix
00514     is greater than 10, the next digit after \c '9' will be the letter
00515     \c 'a'.
00516
00517     If radix is 10 and val is negative, a minus sign will be prepended.
00518

```

```

00519     The ltoa() function returns the pointer passed as \c s.
00520 */
00521 #ifdef __DOXYGEN__
00522 extern char *ltoa(long val, char *s, int radix);
00523 #else
00524 extern __inline__ __ATTR_GNU_INLINE__
00525 char *ltoa (long __val, char *__s, int __radix)
00526 {
00527     if (!__builtin_constant_p (__radix))
00528     {
00529         extern char *__ltoa (long, char *, int);
00530         return __ltoa (__val, __s, __radix);
00531     }
00532     else if (__radix < 2 || __radix > 36)
00533     {
00534         *__s = 0;
00535         return __s;
00536     }
00537     else
00538     {
00539         extern char *__ltoa_ncheck (long, char *, unsigned char);
00540         return __ltoa_ncheck (__val, __s, __radix);
00541     }
00542 }
00543 #endif
00544
00545 /**
00546  \ingroup avr_stdlib
00547
00548  \brief Convert an unsigned integer to a string.
00549
00550  The function utoa() converts the unsigned integer value from \c val into an
00551  ASCII representation that will be stored under \c s. The caller
00552  is responsible for providing sufficient storage in \c s.
00553
00554  \note The minimal size of the buffer \c s depends on the choice of
00555  radix. For example, if the radix is 2 (binary), you need to supply a buffer
00556  with a minimal length of 8 * sizeof (unsigned int) + 1 characters, i.e. one
00557  character for each bit plus one for the string terminator. Using a larger
00558  radix will require a smaller minimal buffer size.
00559
00560  \warning If the buffer is too small, you risk a buffer overflow.
00561
00562  Conversion is done using the \c radix as base, which may be a
00563  number between 2 (binary conversion) and up to 36. If \c radix
00564  is greater than 10, the next digit after \c '9' will be the letter
00565  \c 'a'.
00566
00567  The utoa() function returns the pointer passed as \c s.
00568 */
00569 #ifdef __DOXYGEN__
00570 extern char *utoa(unsigned int val, char *s, int radix);
00571 #else
00572 extern __inline__ __ATTR_GNU_INLINE__
00573 char *utoa (unsigned int __val, char *__s, int __radix)
00574 {
00575     if (!__builtin_constant_p (__radix))
00576     {
00577         extern char *__utoa (unsigned int, char *, int);
00578         return __utoa (__val, __s, __radix);
00579     }
00580     else if (__radix < 2 || __radix > 36)
00581     {
00582         *__s = 0;
00583         return __s;
00584     }
00585     else
00586     {
00587         extern char *__utoa_ncheck (unsigned int, char *, unsigned char);
00588         return __utoa_ncheck (__val, __s, __radix);
00589     }
00590 }
00591 #endif
00592
00593 /**
00594  \ingroup avr_stdlib
00595  \brief Convert an unsigned long integer to a string.
00596
00597  The function ultoa() converts the unsigned long integer value from
00598  \c val into an ASCII representation that will be stored under \c s.
00599  The caller is responsible for providing sufficient storage in \c s.
00600
00601  \note The minimal size of the buffer \c s depends on the choice of
00602  radix. For example, if the radix is 2 (binary), you need to supply a buffer
00603  with a minimal length of 8 * sizeof (unsigned long int) + 1 characters,
00604  i.e. one character for each bit plus one for the string terminator. Using a
00605  larger radix will require a smaller minimal buffer size.

```

```

00606
00607 \warning If the buffer is too small, you risk a buffer overflow.
00608
00609 Conversion is done using the \c radix as base, which may be a
00610 number between 2 (binary conversion) and up to 36. If \c radix
00611 is greater than 10, the next digit after \c '9' will be the letter
00612 \c 'a'.
00613
00614 The ultoa() function returns the pointer passed as \c s.
00615 */
00616 #ifdef __DOXYGEN__
00617 extern char *ultoa(unsigned long val, char *s, int radix);
00618 #else
00619 extern __inline__ __ATTR_GNU_INLINE__
00620 char *ultoa(unsigned long __val, char *__s, int __radix)
00621 {
00622     if (!__builtin_constant_p(__radix)) {
00623         extern char *__ultoa(unsigned long, char *, int);
00624         return __ultoa(__val, __s, __radix);
00625     } else if (__radix < 2 || __radix > 36) {
00626         *__s = 0;
00627         return __s;
00628     } else {
00629         extern char *__ultoa_ncheck(unsigned long, char *, unsigned char);
00630         return __ultoa_ncheck(__val, __s, __radix);
00631     }
00632 }
00633 #endif
00634
00635 /** \ingroup avr_stdlib
00636 Highest number that can be generated by random(). */
00637 #define RANDOM_MAX 0x7FFFFFFF
00638
00639 /**
00640 \ingroup avr_stdlib
00641 The random() function computes a sequence of pseudo-random integers in the
00642 range of 0 to \c RANDOM_MAX (as defined by the header file <stdlib.h>).
00643
00644 The srandom() function sets its argument \c seed as the seed for a new
00645 sequence of pseudo-random numbers to be returned by rand(). These
00646 sequences are repeatable by calling srandom() with the same seed value.
00647
00648 If no seed value is provided, the functions are automatically seeded with
00649 a value of 1.
00650 */
00651 extern long random(void);
00652 /**
00653 \ingroup avr_stdlib
00654 Pseudo-random number generator seeding; see random().
00655 */
00656 extern void srandom(unsigned long __seed);
00657
00658 /**
00659 \ingroup avr_stdlib
00660 Variant of random() that stores the context in the user-supplied
00661 variable located at \c ctx instead of a static library variable
00662 so the function becomes re-entrant.
00663 */
00664 extern long random_r(unsigned long *__ctx);
00665 #endif /* __ASSEMBLER */
00666 /**@}*/
00667
00668 /**@{*/
00669 /** \name Conversion functions for double arguments. */
00670 /** \ingroup avr_stdlib
00671 Bit value that can be passed in \c flags to ftostre(),
00672 dtostre() and ldtostr(). */
00673 #define DTOSTR_ALWAYS_SIGN 0x01 /* put '+' or '-' for positives */
00674 /** \ingroup avr_stdlib
00675 Bit value that can be passed in \c flags to ftostre(),
00676 dtostre() and ldtostr(). */
00677 #define DTOSTR_PLUS_SIGN 0x02 /* put '+' rather than '-' */
00678 /** \ingroup avr_stdlib
00679 Bit value that can be passed in \c flags to ftostre(),
00680 dtostre() and ldtostr(). */
00681 #define DTOSTR_UPPERCASE 0x04 /* put 'E' rather 'e' */
00682
00683 #ifndef __ASSEMBLER__
00684
00685 /**
00686 \ingroup avr_stdlib
00687 The ftostre() function converts the \c float value passed in \c val into
00688 an ASCII representation that will be stored under \c s. The caller
00689 is responsible for providing sufficient storage in \c s.
00690
00691 Conversion is done in the format
00692 <tt>"[-]d.ddde[plusmn];dd</tt> where there is

```

```

00693     one digit before the decimal-point character and the number of
00694     digits after it is equal to the precision \c prec; if the precision
00695     is zero, no decimal-point character appears. If \c flags has the
00696     #DOSTR_UPPERCASE bit set, the letter \c 'E' (rather than \c 'e' ) will be
00697     used to introduce the exponent. The exponent always contains two
00698     digits; if the value is zero, the exponent is \c "00".
00699
00700     If \c flags has the #DOSTR_ALWAYS_SIGN bit set, a space character
00701     will be placed into the leading position for positive numbers.
00702
00703     If \c flags has the #DOSTR_PLUS_SIGN bit set, a plus sign will be
00704     used instead of a space character in this case.
00705
00706     The ftostr() function returns the pointer to the converted string \c s.
00707 */
00708 extern char *ftostr(float __val, char *__s, unsigned char __prec,
00709                    unsigned char __flags);
00710 /**
00711     \ingroup avr_stdlib
00712     The dtostr() function is similar to the ftostr() function, except that
00713     it converts a \c double value instead of a \c float value.
00714
00715     dtostr() is currently only supported when \c double is a 32-bit type. */
00716 extern char *dtostr(double __val, char *__s, unsigned char __prec,
00717                    unsigned char __flags);
00718 /**
00719     \ingroup avr_stdlib
00720     The ldtostr() function is similar to the ftostr() function, except that
00721     it converts a \c long \c double value instead of a \c float value.
00722
00723     ldtostr() is currently only supported when \c long \c double is a
00724     32-bit type. */
00725 extern char *ldtostr(long double __val, char *__s, unsigned char __prec,
00726                     unsigned char __flags);
00727
00728 /**
00729     \ingroup avr_stdlib
00730     The ftostrf() function converts the \c float value passed in \c val into
00731     an ASCII representation that will be stored in \c s. The caller
00732     is responsible for providing sufficient storage in \c s.
00733
00734     Conversion is done in the format \c "[-]d.ddd". The minimum field
00735     width of the output string (including the possible \c '.' and the possible
00736     sign for negative values) is given in \c width, and \c prec determines
00737     the number of digits after the decimal sign. \c width is signed value,
00738     negative for left adjustment.
00739
00740     The ftostrf() function returns the pointer to the converted string \c s.
00741 */
00742 extern char *ftostrf(float __val, signed char __width,
00743                     unsigned char __prec, char *__s);
00744 /**
00745     \ingroup avr_stdlib
00746     The dtostrf() function is similar to the ftostrf() function, except that
00747     converts a \c double value instead of a \c float value.
00748
00749     ldtostrf() is currently only supported when \c double is a 32-bit type. */
00750 extern char *dtostrf(double __val, signed char __width,
00751                     unsigned char __prec, char *__s);
00752 /**
00753     \ingroup avr_stdlib
00754     The ldtostrf() function is similar to the ftostrf() function, except that
00755     converts a \c long \c double value instead of a \c float value.
00756
00757     ldtostrf() is currently only supported when \c long \c double is a
00758     32-bit type. */
00759 extern char *ldtostrf(long double __val, signed char __width,
00760                       unsigned char __prec, char *__s);
00761
00762 /**
00763     \ingroup avr_stdlib
00764     Successful termination for exit(); evaluates to 0.
00765 */
00766 #define EXIT_SUCCESS 0
00767
00768 /**
00769     \ingroup avr_stdlib
00770     Unsuccessful termination for exit(); evaluates to a non-zero value.
00771 */
00772 #define EXIT_FAILURE 1
00773
00774 /** @} */
00775
00776 #ifndef __DOXYGEN__
00777 /* dummy declarations for libstdc++ compatibility */
00778 extern int system (const char *);
00779 extern char *getenv (const char *);

```

```
00780 #endif /* __DOXYGEN__ */
00781
00782 #ifdef __cplusplus
00783 }
00784 #endif
00785
00786 #endif /* __ASSEMBLER__ */
00787
00788 #endif /* _STDLIB_H_ */
```

## 23.58 string.h File Reference

### Macros

- #define `_FFS(x)`

### Functions

- int `ffs` (int `__val`)
- int `ffsl` (long `__val`)
- int `ffsll` (long long `__val`)
- void \* `memccpy` (void \*, const void \*, int, size\_t)
- void \* `memchr` (const void \*, int, size\_t)
- int `memcmp` (const void \*, const void \*, size\_t)
- void \* `memcpy` (void \*, const void \*, size\_t)
- void \* `memmem` (const void \*, size\_t, const void \*, size\_t)
- void \* `memmove` (void \*, const void \*, size\_t)
- void \* `memrchr` (const void \*, int, size\_t)
- void \* `memset` (void \*, int, size\_t)
- char \* `strcat` (char \*, const char \*)
- char \* `strchr` (const char \*, int)
- char \* `strchrnul` (const char \*, int)
- int `strcmp` (const char \*, const char \*)
- char \* `strcpy` (char \*, const char \*)
- int `strcasecmp` (const char \*, const char \*)
- char \* `strcasestr` (const char \*, const char \*)
- size\_t `strcspn` (const char \* `__s`, const char \* `__reject`)
- char \* `strdup` (const char \* `s1`)
- char \* `strndup` (const char \* `s`, size\_t `n`)
- size\_t `strlcat` (char \*, const char \*, size\_t)
- size\_t `strlcpy` (char \*, const char \*, size\_t)
- size\_t `strlen` (const char \*)
- char \* `strlwr` (char \*)
- char \* `strncat` (char \*, const char \*, size\_t)
- int `strncmp` (const char \*, const char \*, size\_t)
- char \* `strncpy` (char \*, const char \*, size\_t)
- int `strncasecmp` (const char \*, const char \*, size\_t)
- size\_t `strnlen` (const char \*, size\_t)
- char \* `strpbrk` (const char \* `__s`, const char \* `__accept`)
- char \* `strrchr` (const char \*, int)
- char \* `strrev` (char \*)
- char \* `strsep` (char \*\*, const char \*)
- size\_t `strspn` (const char \* `__s`, const char \* `__accept`)
- char \* `strstr` (const char \*, const char \*)
- char \* `strtok` (char \*, const char \*)
- char \* `strtok_r` (char \*, const char \*, char \*\*)
- char \* `strupr` (char \*)



## 23.59 string.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002,2007 Marek Michalkiewicz
00002    All rights reserved.
00003
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions are met:
00006
00007    * Redistributions of source code must retain the above copyright
00008    notice, this list of conditions and the following disclaimer.
00009
00010    * Redistributions in binary form must reproduce the above copyright
00011    notice, this list of conditions and the following disclaimer in
00012    the documentation and/or other materials provided with the
00013    distribution.
00014
00015    * Neither the name of the copyright holders nor the names of
00016    contributors may be used to endorse or promote products derived
00017    from this software without specific prior written permission.
00018
00019    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029    POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 /*
00034    string.h
00035
00036    Contributors:
00037    Created by Marek Michalkiewicz <marekm@linux.org.pl>
00038 */
00039
00040 #ifndef _STRING_H_
00041 #define _STRING_H_ 1
00042
00043 #ifndef __DOXYGEN__
00044 #define __need_NULL
00045 #define __need_size_t
00046 #include <stddef.h>
00047
00048 #ifndef __ATTR_PURE__
00049 #define __ATTR_PURE__ __attribute__((__pure__))
00050 #endif
00051
00052 #ifndef __ATTR_CONST__
00053 #define __ATTR_CONST__ __attribute__((__const__))
00054 #endif
00055 #endif /* !__DOXYGEN__ */
00056
00057 #ifdef __cplusplus
00058 extern "C" {
00059 #endif
00060
00061 /** \file */
00062 /** \defgroup avr_string <string.h>: Strings
00063     \code #include <string.h> \endcode
00064
00065     The string functions perform string operations on \c NULL terminated
00066     strings.
00067
00068     \note If the strings you are working on resident in program space (flash),
00069     you will need to use the string functions described in \ref avr_pgmspace. */
00070
00071 /** \ingroup avr_string
00072
00073     This macro finds the first (least significant) bit set in the
00074     input value.
00075
00076     This macro is very similar to the function ffs() except that
00077     it evaluates its argument at compile-time, so it should only
00078     be applied to compile-time constant expressions where it will
00079     reduce to a constant itself.
00080     Application of this macro to expressions that are not constant
00081     at compile-time is not recommended, and might result in a huge
00082     amount of code generated.
00083

```

```

00084
00085 \returns The _FFS() macro returns the position of the first
00086 (least significant) bit set in the word val, or 0 if no bits are set.
00087 The least significant bit is position 1. Only 16 bits of argument
00088 are evaluated.
00089 */
00090 #if defined(__DOXYGEN__)
00091 #define _FFS(x)
00092 #else /* !DOXYGEN */
00093 #define _FFS(x) \
00094     (1 \
00095      + (((x) & 1) == 0) \
00096      + (((x) & 3) == 0) \
00097      + (((x) & 7) == 0) \
00098      + (((x) & 017) == 0) \
00099      + (((x) & 037) == 0) \
00100      + (((x) & 077) == 0) \
00101      + (((x) & 0177) == 0) \
00102      + (((x) & 0377) == 0) \
00103      + (((x) & 0777) == 0) \
00104      + (((x) & 01777) == 0) \
00105      + (((x) & 03777) == 0) \
00106      + (((x) & 07777) == 0) \
00107      + (((x) & 017777) == 0) \
00108      + (((x) & 037777) == 0) \
00109      + (((x) & 077777) == 0) \
00110      - (((x) & 0177777) == 0) * 16)
00111 #endif /* DOXYGEN */
00112
00113 /** \ingroup avr_string
00114     \fn int ffs(int val);
00115
00116     \brief This function finds the first (least significant) bit set in the input value.
00117
00118     \returns The ffs() function returns the position of the first
00119     (least significant) bit set in the word \p val, or 0 if no bits are set.
00120     The least significant bit is position 1.
00121
00122     \note For expressions that are constant at compile time, consider
00123     using the \ref _FFS macro instead.
00124 */
00125 extern int ffs(int __val) __ATTR_CONST__;
00126
00127 /** \ingroup avr_string
00128     \fn int ffsl(long val);
00129
00130     \brief Same as ffs(), for an argument of type long. */
00131 extern int ffsl(long __val) __ATTR_CONST__;
00132
00133 /** \ingroup avr_string
00134     \fn int ffsll(long long val);
00135
00136     \brief Same as ffs(), for an argument of type <tt>long long</tt>. */
00137 __extension__ extern int ffsll(long long __val) __ATTR_CONST__;
00138
00139 /** \ingroup avr_string
00140     \fn void *memcpy(void *dest, const void *src, int val, size_t len)
00141     \brief Copy memory area.
00142
00143     The memcpy() function copies no more than \p len bytes from memory
00144     area \p src to memory area \p dest, stopping when the character \p val
00145     is found.
00146
00147     \returns The memcpy() function returns a pointer to the next character
00148     in \p dest after \p val, or \c NULL if \p val was not found in the first
00149     \p len characters of \p src. */
00150 extern void *memcpy(void *, const void *, int, size_t);
00151
00152 /** \ingroup avr_string
00153     \fn void *memchr(const void *src, int val, size_t len)
00154     \brief Scan memory for a character.
00155
00156     The memchr() function scans the first len bytes of the memory area pointed
00157     to by src for the character val. The first byte to match val (interpreted
00158     as an unsigned character) stops the operation.
00159
00160     \returns The memchr() function returns a pointer to the matching byte or
00161     NULL if the character does not occur in the given memory area. */
00162 extern void *memchr(const void *, int, size_t) __ATTR_PURE__;
00163
00164 /** \ingroup avr_string
00165     \fn int memcmp(const void *s1, const void *s2, size_t len)
00166     \brief Compare memory areas
00167
00168     The memcmp() function compares the first len bytes of the memory areas s1
00169     and s2. The comparison is performed using unsigned char operations.
00170

```

```

00171     \returns The memcmp() function returns an integer less than, equal to, or
00172     greater than zero if the first len bytes of s1 is found, respectively, to be
00173     less than, to match, or be greater than the first len bytes of s2.
00174
00175     \note Be sure to store the result in a 16 bit variable since you may get
00176     incorrect results if you use an unsigned char or char due to truncation.
00177
00178     \warning This function is not -mint8 compatible, although if you only care
00179     about testing for equality, this function should be safe to use. */
00180 extern int memcmp(const void *, const void *, size_t) __ATTR_PURE__;
00181
00182 /** \ingroup avr_string
00183     \fn void *memcpy(void *dest, const void *src, size_t len)
00184     \brief Copy a memory area.
00185
00186     The memcpy() function copies len bytes from memory area src to memory area
00187     dest. The memory areas may not overlap. Use memmove() if the memory
00188     areas do overlap.
00189
00190     \returns The memcpy() function returns a pointer to dest. */
00191 extern void *memcpy(void *, const void *, size_t);
00192
00193 /** \ingroup avr_string
00194     \fn void *mемmem(const void *s1, size_t len1, const void *s2, size_t len2)
00195
00196     The memmem() function finds the start of the first occurrence of the
00197     substring \p s2 of length \p len2 in the memory area \p s1 of length
00198     \p len1.
00199
00200     \return The memmem() function returns a pointer to the beginning of
00201     the substring, or \c NULL if the substring is not found. If \p len2
00202     is zero, the function returns \p s1. */
00203 extern void *memmem(const void *, size_t, const void *, size_t) __ATTR_PURE__;
00204
00205 /** \ingroup avr_string
00206     \fn void *memmove(void *dest, const void *src, size_t len)
00207     \brief Copy memory area.
00208
00209     The memmove() function copies len bytes from memory area src to memory area
00210     dest. The memory areas may overlap.
00211
00212     \returns The memmove() function returns a pointer to dest. */
00213 extern void *memmove(void *, const void *, size_t);
00214
00215 /** \ingroup avr_string
00216     \fn void *memrchr(const void *src, int val, size_t len)
00217
00218     The memrchr() function is like the memchr() function, except that it
00219     searches backwards from the end of the \p len bytes pointed to by \p
00220     src instead of forwards from the front. (Glibc, GNU extension.)
00221
00222     \return The memrchr() function returns a pointer to the matching
00223     byte or \c NULL if the character does not occur in the given memory
00224     area. */
00225 extern void *memrchr(const void *, int, size_t) __ATTR_PURE__;
00226
00227 /** \ingroup avr_string
00228     \fn void *memset(void *dest, int val, size_t len)
00229     \brief Fill memory with a constant byte.
00230
00231     The memset() function fills the first len bytes of the memory area pointed
00232     to by dest with the constant byte val.
00233
00234     \returns The memset() function returns a pointer to the memory area dest. */
00235 extern void *memset(void *, int, size_t);
00236
00237 /** \ingroup avr_string
00238     \fn char *strcat(char *dest, const char *src)
00239     \brief Concatenate two strings.
00240
00241     The strcat() function appends the src string to the dest string
00242     overwriting the '\\0' character at the end of dest, and then adds a
00243     terminating '\\0' character. The strings may not overlap, and the dest
00244     string must have enough space for the result.
00245
00246     \returns The strcat() function returns a pointer to the resulting string
00247     dest. */
00248 extern char *strcat(char *, const char *);
00249
00250 /** \ingroup avr_string
00251     \fn char *strchr(const char *src, int val)
00252     \brief Locate character in string.
00253
00254     \returns The strchr() function returns a pointer to the first occurrence of
00255     the character \p val in the string \p src, or \c NULL if the character
00256     is not found.
00257     <br><br>

```

```

00258     Here "character" means "byte" -- these functions do not work with
00259     wide or multi-byte characters. */
00260 extern char *strchr(const char *, int) __ATTR_PURE__;
00261
00262 /** \ingroup avr_string
00263     \fn char *strchrnul(const char *s, int c)
00264
00265     The strchrnul() function is like strchr() except that if \p c is not
00266     found in \p s, then it returns a pointer to the null byte at the end
00267     of \p s, rather than \c NULL. (Glibc, GNU extension.)
00268
00269     \return The strchrnul() function returns a pointer to the matched
00270     character, or a pointer to the null byte at the end of \p s (i.e.,
00271     \c s+strlen(s)) if the character is not found. */
00272 extern char *strchrnul(const char *, int) __ATTR_PURE__;
00273
00274 /** \ingroup avr_string
00275     \fn int strcmp(const char *s1, const char *s2)
00276     \brief Compare two strings.
00277
00278     The strcmp() function compares the two strings \p s1 and \p s2.
00279
00280     \returns The strcmp() function returns an integer less than, equal
00281     to, or greater than zero if \p s1 is found, respectively, to be less
00282     than, to match, or be greater than \p s2. A consequence of the
00283     ordering used by strcmp() is that if \p s1 is an initial substring
00284     of \p s2, then \p s1 is considered to be "less than" \p s2. */
00285 extern int strcmp(const char *, const char *) __ATTR_PURE__;
00286
00287 /** \ingroup avr_string
00288     \fn char *strcpy(char *dest, const char *src)
00289     \brief Copy a string.
00290
00291     The strcpy() function copies the string pointed to by src (including the
00292     terminating '\0' character) to the array pointed to by dest. The strings
00293     may not overlap, and the destination string dest must be large enough to
00294     receive the copy.
00295
00296     \returns The strcpy() function returns a pointer to the destination
00297     string dest.
00298
00299     \note If the destination string of a strcpy() is not large enough (that
00300     is, if the programmer was stupid/lazy, and failed to check the size before
00301     copying) then anything might happen. Overflowing fixed length strings is
00302     a favourite cracker technique. */
00303 extern char *strcpy(char *, const char *);
00304
00305 /** \ingroup avr_string
00306     \fn int strcasecmp(const char *s1, const char *s2)
00307     \brief Compare two strings ignoring case.
00308
00309     The strcasecmp() function compares the two strings \p s1 and \p s2,
00310     ignoring the case of the characters.
00311
00312     \returns The strcasecmp() function returns an integer less than,
00313     equal to, or greater than zero if \p s1 is found, respectively, to
00314     be less than, to match, or be greater than \p s2. A consequence of
00315     the ordering used by strcasecmp() is that if \p s1 is an initial
00316     substring of \p s2, then \p s1 is considered to be "less than"
00317     \p s2. */
00318 extern int strcasecmp(const char *, const char *) __ATTR_PURE__;
00319
00320 /** \ingroup avr_string
00321     \fn char *strcasestr(const char *s1, const char *s2)
00322
00323     The strcasestr() function finds the first occurrence of the
00324     substring \p s2 in the string \p s1. This is like strstr(), except
00325     that it ignores case of alphabetic symbols in searching for the
00326     substring. (Glibc, GNU extension.)
00327
00328     \return The strcasestr() function returns a pointer to the beginning
00329     of the substring, or \c NULL if the substring is not found. If \p s2
00330     points to a string of zero length, the function returns \p s1. */
00331 extern char *strcasestr(const char *, const char *) __ATTR_PURE__;
00332
00333 /** \ingroup avr_string
00334     \fn size_t strcspn(const char *s, const char *reject)
00335
00336     The strcspn() function calculates the length of the initial segment
00337     of \p s which consists entirely of characters not in \p reject.
00338
00339     \return The strcspn() function returns the number of characters in
00340     the initial segment of \p s which are not in the string \p reject.
00341     The terminating zero is not considered as a part of string. */
00342 extern size_t strcspn(const char *__s, const char *__reject) __ATTR_PURE__;
00343
00344 /** \ingroup avr_string

```

```

00345     \fn char *strdup(const char *s1)
00346     \brief Duplicate a string.
00347
00348     The strdup() function allocates memory and copies into it the string
00349     addressed by \p s1, including the terminating null character.
00350
00351     \warning The strdup() function calls malloc() to allocate the memory
00352     for the duplicated string! The user is responsible for freeing the
00353     memory by calling free().
00354
00355     \returns The strdup() function returns a pointer to the resulting string
00356     dest. If malloc() cannot allocate enough storage for the string, strdup()
00357     will return \c NULL.
00358
00359     \warning Be sure to check the return value of the strdup() function to
00360     make sure that the function has succeeded in allocating the memory!
00361 */
00362 extern char *strdup(const char *s1);
00363
00364 /** \ingroup avr_string
00365     \fn char *strndup(const char *s, size_t len)
00366     \brief Duplicate a string.
00367
00368     The strndup() function is similar to strdup(), but copies at most
00369     \p len bytes. If \p s is longer than \p len, only \p len bytes are copied,
00370     and a terminating null byte (<tt>'\\0'</tt>) is added.
00371
00372     Memory for the new string is obtained with malloc(), and can be freed
00373     with free().
00374
00375     \returns The strndup() function returns the location of the newly malloc'ed
00376     memory, or \c NULL if the allocation failed.
00377 */
00378 extern char *strndup(const char *s, size_t n);
00379
00380 /** \ingroup avr_string
00381     \fn size_t strlcat(char *dst, const char *src, size_t siz)
00382     \brief Concatenate two strings.
00383
00384     Appends \p src to string \p dst of size \p siz (unlike strcat(),
00385     \p siz is the full size of \p dst, not space left). At most \p siz-1
00386     characters will be copied. Always \p '\\0' terminated (unless \p siz <=
00387     \p strlen(dst)).
00388
00389     \returns The strlcat() function returns strlen(src) + MIN(siz,
00390     strlen(initial dst)). If retval >= siz, truncation occurred. */
00391 extern size_t strlcat(char *, const char *, size_t);
00392
00393 /** \ingroup avr_string
00394     \fn size_t strlcpy(char *dst, const char *src, size_t siz)
00395     \brief Copy a string.
00396
00397     Copy \p src to string \p dst of size \p siz. At most \p siz-1
00398     characters will be copied. Always '\\0' terminated (unless \p siz == 0).
00399
00400     \returns The strlcpy() function returns strlen(src). If retval >= siz,
00401     truncation occurred. */
00402 extern size_t strlcpy(char *, const char *, size_t);
00403
00404 /** \ingroup avr_string
00405     \fn size_t strlen(const char *src)
00406     \brief Calculate the length of a string.
00407
00408     The strlen() function calculates the length of the string \p src, not
00409     including the terminating '\\0' character.
00410
00411     \returns The strlen() function returns the number of characters in
00412     \p src. */
00413 extern size_t strlen(const char *) __ATTR_PURE__;
00414
00415 /** \ingroup avr_string
00416     \fn char *strlwr(char *s)
00417     \brief Convert a string to lower case.
00418
00419     The strlwr() function will convert a string to lower case. Only the upper
00420     case alphabetic characters [A .. Z] are converted. Non-alphabetic
00421     characters will not be changed.
00422
00423     \returns The strlwr() function returns a pointer to the converted
00424     string. Conversion is performed in-place. */
00425 extern char *strlwr(char *);
00426
00427 /** \ingroup avr_string
00428     \fn char *strncat(char *dest, const char *src, size_t len)
00429     \brief Concatenate two strings.
00430
00431     The strncat() function is similar to strcat(), except that only the first

```

```

00432     \p len characters of \p src are appended to \p dest.
00433
00434     \returns The strncat() function returns a pointer to the resulting string
00435     \p dest. */
00436 extern char *strncat(char *, const char *, size_t);
00437
00438 /** \ingroup avr_string
00439     \fn int strncmp(const char *s1, const char *s2, size_t len)
00440     \brief Compare two strings.
00441
00442     The strncmp() function is similar to strcmp(), except it only compares the
00443     first (at most) \p len characters of \p s1 and \p s2.
00444
00445     \returns The strncmp() function returns an integer less than, equal to, or
00446     greater than zero if \p s1 (or the first \p len bytes thereof) is found,
00447     respectively, to be less than, to match, or be greater than \p s2. */
00448 extern int strncmp(const char *, const char *, size_t) __ATTR_PURE__;
00449
00450 /** \ingroup avr_string
00451     \fn char *strncpy(char *dest, const char *src, size_t len)
00452     \brief Copy a string.
00453
00454     The strncpy() function is similar to strcpy(), except that not more than
00455     \p len bytes of \p src are copied. Thus, if there is no null byte among
00456     the first \p len bytes of \p src, the result will not be null-terminated.
00457
00458     In the case where the length of \p src is less than that of \p len,
00459     the remainder of \p dest will be padded with nulls (<tt>'0'</tt>).
00460
00461     \returns The strncpy() function returns a pointer to the destination
00462     string \p dest. */
00463 extern char *strncpy(char *, const char *, size_t);
00464
00465 /** \ingroup avr_string
00466     \fn int strncasecmp(const char *s1, const char *s2, size_t len)
00467     \brief Compare two strings ignoring case.
00468
00469     The strncasecmp() function is similar to strcasecmp(), except it
00470     only compares the first \p len characters of \p s1.
00471
00472     \returns The strncasecmp() function returns an integer less than,
00473     equal to, or greater than zero if \p s1 (or the first \p len bytes
00474     thereof) is found, respectively, to be less than, to match, or be
00475     greater than \p s2. A consequence of the ordering used by
00476     strncasecmp() is that if \p s1 is an initial substring of \p s2,
00477     then \p s1 is considered to be "less than" \p s2. */
00478 extern int strncasecmp(const char *, const char *, size_t) __ATTR_PURE__;
00479
00480 /** \ingroup avr_string
00481     \fn size_t strlen(const char *src, size_t len)
00482     \brief Determine the length of a fixed-size string.
00483
00484     The strlen() function returns the number of characters in the string
00485     pointed to by \p src, not including the terminating '\\0' character, but at
00486     most \p len. In doing this, strlen() looks only at the first \p len
00487     characters at \p src and never beyond \p src + \p len.
00488
00489     \returns The strlen function returns strlen(src), if that is less than
00490     \p len, or \p len if there is no '\\0' character among the first \p len
00491     characters pointed to by \p src. */
00492 extern size_t strlen(const char *, size_t) __ATTR_PURE__;
00493
00494 /** \ingroup avr_string
00495     \fn char *strpbrk(const char *s, const char *accept)
00496
00497     The strpbrk() function locates the first occurrence in the string
00498     \p s of any of the characters in the string \p accept.
00499
00500     \return The strpbrk() function returns a pointer to the character
00501     in \p s that matches one of the characters in \p accept, or \c NULL
00502     if no such character is found. The terminating zero is not
00503     considered as a part of string: if one or both args are empty, the
00504     result will be \c NULL. */
00505 extern char *strpbrk(const char *__s, const char *__accept) __ATTR_PURE__;
00506
00507 /** \ingroup avr_string
00508     \fn char *strrchr(const char *src, int val)
00509     \brief Locate character in string.
00510
00511     The strrchr() function returns a pointer to the last occurrence of the
00512     character val in the string src.
00513
00514     Here "character" means "byte" -- these functions do not work with wide or
00515     multi-byte characters.
00516
00517     \returns The strrchr() function returns a pointer to the matched character
00518     or \c NULL if the character is not found. */

```

```

00519 extern char *strchr(const char *, int) __ATTR_PURE__;
00520
00521 /** \ingroup avr_string
00522     \fn char *strrev(char *s)
00523     \brief Reverse a string.
00524
00525     The strrev() function reverses the order of the string.
00526
00527     \returns The strrev() function returns a pointer to the beginning of the
00528     reversed string. */
00529 extern char *strrev(char *);
00530
00531 /** \ingroup avr_string
00532     \fn char *strsep(char **sp, const char *delim)
00533     \brief Parse a string into tokens.
00534
00535     The strsep() function locates, in the string referenced by \p *sp,
00536     the first occurrence of any character in the string \p delim (or the
00537     terminating '\\0' character) and replaces it with a '\\0'. The
00538     location of the next character after the delimiter character (or \c
00539     NULL, if the end of the string was reached) is stored in \p *sp. An
00540     "empty" field, i.e. one caused by two adjacent delimiter
00541     characters, can be detected by comparing the location referenced by
00542     the pointer returned in \p *sp to '\\0'.
00543
00544     \return The strsep() function returns a pointer to the original
00545     value of \p *sp. If \p *sp is initially \c NULL, strsep() returns
00546     \c NULL. */
00547 extern char *strsep(char **, const char *);
00548
00549 /** \ingroup avr_string
00550     \fn size_t strspn(const char *s, const char *accept)
00551
00552     The strspn() function calculates the length of the initial segment
00553     of \p s which consists entirely of characters in \p accept.
00554
00555     \return The strspn() function returns the number of characters in
00556     the initial segment of \p s which consist only of characters from \p
00557     accept. The terminating zero is not considered as a part of string. */
00558 extern size_t strspn(const char *__s, const char *__accept) __ATTR_PURE__;
00559
00560 /** \ingroup avr_string
00561     \fn char *strstr(const char *s1, const char *s2)
00562     \brief Locate a substring.
00563
00564     The strstr() function finds the first occurrence of the substring \p
00565     s2 in the string \p s1. The terminating '\\0' characters are not
00566     compared.
00567
00568     \returns The strstr() function returns a pointer to the beginning of
00569     the substring, or \c NULL if the substring is not found. If \p s2
00570     points to a string of zero length, the function returns \p s1. */
00571 extern char *strstr(const char *, const char *) __ATTR_PURE__;
00572
00573 /** \ingroup avr_string
00574     \fn char *strtok(char *s, const char *delim)
00575     \brief Parses the string s into tokens.
00576
00577     strtok parses the string s into tokens. The first call to strtok
00578     should have s as its first argument. Subsequent calls should have
00579     the first argument set to \c NULL. If a token ends with a delimiter, this
00580     delimiting character is overwritten with a '\\0' and a pointer to the next
00581     character is saved for the next call to strtok. The delimiter string
00582     delim may be different for each call.
00583
00584     \returns The strtok() function returns a pointer to the next token or
00585     \c NULL when no more tokens are found.
00586
00587     \note strtok() is NOT reentrant. For a reentrant version of this function
00588     see \c strtok_r().
00589     */
00590 extern char *strtok(char *, const char *);
00591
00592 /** \ingroup avr_string
00593     \fn char *strtok_r(char *string, const char *delim, char **last)
00594     \brief Parses string into tokens.
00595
00596     strtok_r parses string into tokens. The first call to strtok_r
00597     should have string as its first argument. Subsequent calls should have
00598     the first argument set to \c NULL. If a token ends with a delimiter, this
00599     delimiting character is overwritten with a '\\0' and a pointer to the next
00600     character is saved for the next call to strtok_r. The delimiter string
00601     \p delim may be different for each call. \p last is a user allocated char*
00602     pointer. It must be the same while parsing the same string. strtok_r is
00603     a reentrant version of strtok().
00604
00605     \returns The strtok_r() function returns a pointer to the next token or

```

```
00606     \c NULL when no more tokens are found. */
00607 extern char *strtok_r(char *, const char *, char **);
00608
00609 /** \ingroup avr_string
00610     \fn char *strupr(char *s)
00611     \brief Convert a string to upper case.
00612
00613     The strupr() function will convert a string to upper case. Only the lower
00614     case alphabetic characters [a .. z] are converted. Non-alphabetic
00615     characters will not be changed.
00616
00617     \returns The strupr() function returns a pointer to the converted
00618     string. The pointer is the same as that passed in since the operation is
00619     perform in place. */
00620 extern char *strupr(char *);
00621
00622 #ifndef __DOXYGEN__
00623 /* libstdc++ compatibility, dummy declarations */
00624 extern int strcoll(const char *s1, const char *s2);
00625 extern char *strerror(int errnum);
00626 extern size_t strxfrm(char *dest, const char *src, size_t n);
00627 #endif /* !__DOXYGEN__ */
00628
00629 #ifdef __cplusplus
00630 }
00631 #endif
00632
00633 #endif /* _STRING_H_ */
00634
```

## 23.60 time.h File Reference

### Data Structures

- struct [tm](#)
- struct [week\\_date](#)

### Macros

- #define [ONE\\_HOUR](#) 3600
- #define [ONE\\_DEGREE](#) 3600
- #define [ONE\\_DAY](#) 86400
- #define [UNIX\\_OFFSET](#) 946684800
- #define [NTP\\_OFFSET](#) 3155673600

### Typedefs

- typedef [uint32\\_t](#) [time\\_t](#)

### Enumerations

- enum [\\_WEEK\\_DAYS\\_](#) {  
    [SUNDAY](#) , [MONDAY](#) , [TUESDAY](#) , [WEDNESDAY](#) ,  
    [THURSDAY](#) , [FRIDAY](#) , [SATURDAY](#) }
- enum [\\_MONTHS\\_](#) {  
    [JANUARY](#) , [FEBRUARY](#) , [MARCH](#) , [APRIL](#) ,  
    [MAY](#) , [JUNE](#) , [JULY](#) , [AUGUST](#) ,  
    [SEPTEMBER](#) , [OCTOBER](#) , [NOVEMBER](#) , [DECEMBER](#) }



## Functions

- [time\\_t time](#) ([time\\_t](#) \*timer)
- [int32\\_t difftime](#) ([time\\_t](#) time1, [time\\_t](#) time0)
- [time\\_t mktime](#) ([struct tm](#) \*timeptr)
- [time\\_t mk\\_gmtime](#) ([const struct tm](#) \*timeptr)
- [struct tm \\* gmtime](#) ([const time\\_t](#) \*timer)
- [void gmtime\\_r](#) ([const time\\_t](#) \*timer, [struct tm](#) \*timeptr)
- [struct tm \\* localtime](#) ([const time\\_t](#) \*timer)
- [void localtime\\_r](#) ([const time\\_t](#) \*timer, [struct tm](#) \*timeptr)
- [char \\* asctime](#) ([const struct tm](#) \*timeptr)
- [void asctime\\_r](#) ([const struct tm](#) \*timeptr, [char](#) \*buf)
- [char \\* ctime](#) ([const time\\_t](#) \*timer)
- [void ctime\\_r](#) ([const time\\_t](#) \*timer, [char](#) \*buf)
- [char \\* isotime](#) ([const struct tm](#) \*timeptr)
- [void isotime\\_r](#) ([const struct tm](#) \*, [char](#) \*)
- [size\\_t strftime](#) ([char](#) \*s, [size\\_t](#) maxsize, [const char](#) \*format, [const struct tm](#) \*timeptr)
- [void set\\_dst](#) ([int\(\\*\)](#)([const time\\_t](#) \*, [int32\\_t](#) \*)
- [void set\\_zone](#) ([int32\\_t](#))
- [void set\\_system\\_time](#) ([time\\_t](#) timestamp)
- [void system\\_tick](#) ([void](#))
- [uint8\\_t is\\_leap\\_year](#) ([int16\\_t](#) year)
- [uint8\\_t month\\_length](#) ([int16\\_t](#) year, [uint8\\_t](#) month)
- [uint8\\_t week\\_of\\_year](#) ([const struct tm](#) \*timeptr, [uint8\\_t](#) start)
- [uint8\\_t week\\_of\\_month](#) ([const struct tm](#) \*timeptr, [uint8\\_t](#) start)
- [struct week\\_date \\* iso\\_week\\_date](#) ([int](#) year, [int](#) yday)
- [void iso\\_week\\_date\\_r](#) ([int](#) year, [int](#) yday, [struct week\\_date](#) \*)
- [uint32\\_t fatfs\\_time](#) ([const struct tm](#) \*timeptr)
- [void set\\_position](#) ([int32\\_t](#) latitude, [int32\\_t](#) longitude)
- [int16\\_t equation\\_of\\_time](#) ([const time\\_t](#) \*timer)
- [int32\\_t daylight\\_seconds](#) ([const time\\_t](#) \*timer)
- [time\\_t solar\\_noon](#) ([const time\\_t](#) \*timer)
- [time\\_t sun\\_rise](#) ([const time\\_t](#) \*timer)
- [time\\_t sun\\_set](#) ([const time\\_t](#) \*timer)
- [float solar\\_declinationf](#) ([const time\\_t](#) \*timer)
- [double solar\\_declination](#) ([const time\\_t](#) \*timer)
- [long double solar\\_declinationl](#) ([const time\\_t](#) \*timer)
- [int8\\_t moon\\_phase](#) ([const time\\_t](#) \*timer)
- [unsigned long gm\\_sidereal](#) ([const time\\_t](#) \*timer)
- [unsigned long lm\\_sidereal](#) ([const time\\_t](#) \*timer)

### 23.61 time.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * (C)2012 Michael Duane Rice All rights reserved.
00003  *
00004  * Redistribution and use in source and binary forms, with or without
00005  * modification, are permitted provided that the following conditions are
00006  * met:
00007  *
00008  * Redistributions of source code must retain the above copyright notice, this
00009  * list of conditions and the following disclaimer. Redistributions in binary
00010  * form must reproduce the above copyright notice, this list of conditions
00011  * and the following disclaimer in the documentation and/or other materials
00012  * provided with the distribution. Neither the name of the copyright holders
00013  * nor the names of contributors may be used to endorse or promote products
00014  * derived from this software without specific prior written permission.
00015  *

```

```

00016 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00017 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00018 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00019 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00020 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00021 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00022 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00023 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00024 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00025 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00026 * POSSIBILITY OF SUCH DAMAGE.
00027 */
00028
00029 /* $Id$ */
00030
00031 #ifndef TIME_H
00032 #define TIME_H
00033
00034 #ifdef __cplusplus
00035 extern "C" {
00036 #endif
00037
00038 #include <stdint.h>
00039 #include <stdlib.h>
00040
00041 /** \file */
00042
00043 /** \defgroup avr_time <time.h>: Time
00044     \code #include <time.h> \endcode
00045     <h3>Introduction to the Time functions</h3>
00046     This file declares the time functions implemented in AVR-LibC.
00047
00048     The implementation aspires to conform with ISO/IEC 9899 (C90). However, due to limitations of the
00049     target processor and the nature of its development environment, a practical implementation must
00050     of necessity deviate from the standard.
00051
00052     Section 7.23.2.1 clock()
00053     The type clock_t, the macro CLOCKS_PER_SEC, and the function clock() are not implemented. We
00054     consider these items belong to operating system code, or to application code when no operating
00055     system is present.
00056
00057     Section 7.23.2.3 mktime()
00058     The standard specifies that mktime() should return (time_t) -1, if the time cannot be represented.
00059     This implementation always returns a 'best effort' representation.
00060
00061     Section 7.23.2.4 time()
00062     The standard specifies that time() should return (time_t) -1, if the time is not available.
00063     Since the application must initialize the time system, this functionality is not implemented.
00064
00065     Section 7.23.2.2, difftime()
00066     Due to the lack of a 64 bit double, the function difftime() returns a long integer. In most cases
00067     this change will be invisible to the user, handled automatically by the compiler.
00068
00069     Section 7.23.1.4 struct tm
00070     Per the standard, struct tm->tm_isdst is greater than zero when Daylight Saving time is in effect.
00071     This implementation further specifies that, when positive, the value of tm_isdst represents
00072     the amount time is advanced during Daylight Saving time.
00073
00074     Section 7.23.3.5 strftime()
00075     Only the 'C' locale is supported, therefore the modifiers 'E' and 'O' are ignored.
00076     The 'Z' conversion is also ignored, due to the lack of time zone name.
00077
00078     In addition to the above departures from the standard, there are some behaviors which are
    different
00079     from what is often expected, though allowed under the standard.
00080
00081     There is no 'platform standard' method to obtain the current time, time zone, or
00082     daylight savings 'rules' in the AVR environment. Therefore the application must initialize
00083     the time system with this information. The functions set_zone(), set_dst(), and
00084     set_system_time() are provided for initialization. Once initialized, system time is maintained by
00085     calling the function system_tick() at one second intervals.
00086
00087     Though not specified in the standard, it is often expected that time_t is a signed integer
00088     representing an offset in seconds from Midnight Jan 1 1970... i.e. 'Unix time'. This
    implementation
00089     uses an unsigned 32 bit integer offset from Midnight Jan 1 2000. The use of this 'epoch' helps to
00090     simplify the conversion functions, while the 32 bit value allows time to be properly represented
00091     until Tue Feb 7 06:28:15 2136 UTC. The macros UNIX_OFFSET and NTP_OFFSET are defined to assist in
00092     converting to and from Unix and NTP time stamps.
00093
00094     Unlike desktop counterparts, it is impractical to implement or maintain the 'zoneinfo' database.
00095     Therefore no attempt is made to account for time zone, daylight saving, or leap seconds in past
    dates.
00096     All calculations are made according to the currently configured time zone and daylight saving
    'rule'.
00097
00098     In addition to C standard functions, re-entrant versions of ctime(), asctime(), gmtime() and

```

```

00099     localtime() are provided which, in addition to being re-entrant, have the property of claiming
00100     less permanent storage in RAM. An additional time conversion, isotime() and its re-entrant
        version,
00101     uses far less storage than either ctime() or asctime().
00102
00103     Along with the usual smattering of utility functions, such as is_leap_year(), this library
        includes
00104     a set of functions related the sun and moon, as well as sidereal time functions.
00105 */
00106
00107 /** \ingroup avr_time */
00108 /**@{*/
00109
00110 /**
00111     time_t represents seconds elapsed from Midnight, Jan 1 2000 UTC (the Y2K 'epoch').
00112     Its range allows this implementation to represent time up to Tue Feb 7 06:28:15 2136 UTC.
00113 */
00114 typedef uint32_t time_t;
00115
00116 /**
00117     The time function returns the systems current time stamp.
00118     If timer is not a null pointer, the return value is also assigned to the object it points to.
00119 */
00120 time_t time(time_t *timer);
00121
00122 /**
00123     The difftime function returns the difference between two binary time stamps,
00124     time1 - time0.
00125 */
00126 int32_t difftime(time_t time1, time_t time0);
00127
00128
00129 /**
00130     The tm structure contains a representation of time 'broken down' into components of the
00131     Gregorian calendar.
00132
00133     The value of tm_isdst is zero if Daylight Saving Time is not in effect, and is negative if
00134     the information is not available.
00135
00136     When Daylight Saving Time is in effect, the value represents the number of
00137     seconds the clock is advanced.
00138
00139     See the set_dst() function for more information about Daylight Saving.
00140 */
00141 struct tm {
00142     int8_t      tm_sec; /**< seconds after the minute - [ 0 to 59 ] */
00143     int8_t      tm_min; /**< minutes after the hour - [ 0 to 59 ] */
00144     int8_t      tm_hour; /**< hours since midnight - [ 0 to 23 ] */
00145     int8_t      tm_mday; /**< day of the month - [ 1 to 31 ] */
00146     int8_t      tm_wday; /**< days since Sunday - [ 0 to 6 ] */
00147     int8_t      tm_mon; /**< months since January - [ 0 to 11 ] */
00148     int16_t     tm_year; /**< years since 1900 */
00149     int16_t     tm_yday; /**< days since January 1 - [ 0 to 365 ] */
00150     int16_t     tm_isdst; /**< Daylight Saving Time flag */
00151 };
00152
00153 #ifndef __DOXYGEN__
00154 /* We have to provide clock_t / CLOCKS_PER_SEC so that libstdc++-v3 can
00155     be built. We define CLOCKS_PER_SEC via a symbol _CLOCKS_PER_SEC_
00156     so that the user can provide the value on the link line, which should
00157     result in little or no run-time overhead compared with a constant. */
00158 typedef unsigned long clock_t;
00159 extern char *_CLOCKS_PER_SEC_;
00160 #define CLOCKS_PER_SEC ((clock_t) _CLOCKS_PER_SEC_)
00161 extern clock_t clock(void);
00162 #endif /* !__DOXYGEN__ */
00163
00164 /**
00165     This function 'compiles' the elements of a broken-down time structure, returning a binary time
        stamp.
00166     The elements of timeptr are interpreted as representing Local Time.
00167
00168     The original values of the tm_wday and tm_yday elements of the structure are ignored,
00169     and the original values of the other elements are not restricted to the ranges stated for struct
        tm.
00170
00171     The element tm_isdst is used for input and output. If set to 0 or a positive value on input, this
00172     requests calculation for Daylight Savings Time being off or on, respectively. If set to a negative
00173     value on input, it requests calculation to return whether Daylight Savings Time is in effect or
00174     not according to the other values.
00175
00176     On successful completion, the values of all elements of timeptr are set to the appropriate range.
00177 */
00178 time_t      mktime(struct tm * timeptr);
00179
00180 /**
00181     This function 'compiles' the elements of a broken-down time structure, returning a binary time

```

```

stamp.
00182     The elements of timeptr are interpreted as representing UTC.
00183
00184     The original values of the tm_wday and tm_yday elements of the structure are ignored,
00185     and the original values of the other elements are not restricted to the ranges stated for struct
tm.
00186
00187     Unlike mktime(), this function DOES NOT modify the elements of timeptr.
00188 */
00189 time_t      mk_gmtime(const struct tm * timeptr);
00190
00191 /**
00192     The gmtime function converts the time stamp pointed to by timer into broken-down time,
00193     expressed as UTC.
00194 */
00195 struct tm    *gmtime(const time_t * timer);
00196
00197 /**
00198     Re entrant version of gmtime().
00199 */
00200 void         localtime_r(const time_t * timer, struct tm * timeptr);
00201
00202 /**
00203     The localtime function converts the time stamp pointed to by timer into broken-down time,
00204     expressed as Local time.
00205 */
00206 struct tm    *localtime(const time_t * timer);
00207
00208 /**
00209     Re entrant version of localtime().
00210 */
00211 void         localtime_r(const time_t * timer, struct tm * timeptr);
00212
00213 /**
00214     The asctime function converts the broken-down time of timeptr, into an ascii string in the form
00215     Sun Mar 23 01:03:52 2013
00216     Sun Mar 23 01:03:52 2013
00217 */
00218 char         *asctime(const struct tm * timeptr);
00219
00220 /**
00221     Re entrant version of asctime().
00222 */
00223 void         asctime_r(const struct tm * timeptr, char *buf);
00224
00225 /**
00226     The ctime function is equivalent to asctime(localtime(timer))
00227 */
00228 char         *ctime(const time_t * timer);
00229
00230 /**
00231     Re entrant version of ctime().
00232 */
00233 void         ctime_r(const time_t * timer, char *buf);
00234
00235 /**
00236     The isotime function constructs an ascii string in the form
00237     \code2013-03-23 01:03:52\endcode
00238 */
00239 char         *isotime(const struct tm * tmpr);
00240
00241 /**
00242     Re entrant version of isotime()
00243 */
00244 void         isotime_r(const struct tm *, char *);
00245
00246 /**
00247     A complete description of strftime() is beyond the pale of this document.
00248     Refer to ISO/IEC document 9899 for details.
00249
00250     All conversions are made using the 'C Locale', ignoring the E or O modifiers. Due to the lack of
00251     a time zone 'name', the 'Z' conversion is also ignored.
00252 */
00253 size_t      strftime(char *s, size_t maxsize, const char *format, const struct tm * timeptr);
00254
00255 /**
00256     Specify the Daylight Saving function.
00257
00258     The Daylight Saving function should examine its parameters to determine whether
00259     Daylight Saving is in effect, and return a value appropriate for tm_isdst.
00260
00261     Working examples for the USA and the EU are available..
00262
00263     \code #include <util/eu_dst.h>\endcode
00264     for the European Union, and
00265     \code #include <util/usa_dst.h>\endcode
00266     for the United States

```

```

00267
00268     If a Daylight Saving function is not specified, the system will ignore Daylight Saving.
00269 */
00270 void             set_dst(int (*) (const time_t *, int32_t *));
00271
00272 /**
00273     Set the 'time zone'. The parameter is given in seconds East of the Prime Meridian.
00274     Example for New York City:
00275     \code set_zone(-5 * ONE_HOUR);\endcode
00276
00277     If the time zone is not set, the time system will operate in UTC only.
00278 */
00279 void             set_zone(int32_t);
00280
00281 /**
00282     Initialize the system time. Examples are...
00283
00284     From a Clock / Calendar type RTC:
00285     \code
00286     struct tm rtc_time;
00287
00288     read_rtc(&rtc_time);
00289     rtc_time.tm_isdst = 0;
00290     set_system_time( mktime(&rtc_time) );
00291     \endcode
00292
00293     From a Network Time Protocol time stamp:
00294     \code
00295     set_system_time(ntp_timestamp - NTP_OFFSET);
00296     \endcode
00297
00298     From a UNIX time stamp:
00299     \code
00300     set_system_time(unix_timestamp - UNIX_OFFSET);
00301     \endcode
00302
00303 */
00304 void             set_system_time(time_t timestamp);
00305
00306 /**
00307     Maintain the system time by calling this function at a rate of 1 Hertz.
00308
00309     It is anticipated that this function will typically be called from within an
00310     Interrupt Service Routine, (though that is not required). It therefore includes code which
00311     makes it simple to use from within a 'Naked' ISR, avoiding the cost of saving and restoring
00312     all the cpu registers.
00313
00314     Such an ISR may resemble the following example...
00315     \code
00316     ISR(RTC_OVF_vect, ISR_NAKED)
00317     {
00318         system_tick();
00319         reti();
00320     }
00321     \endcode
00322 */
00323 void             system_tick(void);
00324
00325 /**
00326     Enumerated labels for the days of the week.
00327 */
00328 enum _WEEK_DAYS_ {
00329     SUNDAY,
00330     MONDAY,
00331     TUESDAY,
00332     WEDNESDAY,
00333     THURSDAY,
00334     FRIDAY,
00335     SATURDAY
00336 };
00337
00338 /**
00339     Enumerated labels for the months.
00340 */
00341 enum _MONTHS_ {
00342     JANUARY,
00343     FEBRUARY,
00344     MARCH,
00345     APRIL,
00346     MAY,
00347     JUNE,
00348     JULY,
00349     AUGUST,
00350     SEPTEMBER,
00351     OCTOBER,
00352     NOVEMBER,
00353     DECEMBER

```

```
00354 };
00355
00356 /**
00357     Return 1 if year is a leap year, zero if it is not.
00358 */
00359 uint8_t      is_leap_year(int16_t year);
00360
00361 /**
00362     Return the length of month, given the year and month, where month is in the range 1 to 12.
00363 */
00364 uint8_t      month_length(int16_t year, uint8_t month);
00365
00366 /**
00367     Return the calendar week of year, where week 1 is considered to begin on the
00368     day of week specified by 'start'. The returned value may range from zero to 52.
00369 */
00370 uint8_t      week_of_year(const struct tm * timeptr, uint8_t start);
00371
00372 /**
00373     Return the calendar week of month, where the first week is considered to begin on the
00374     day of week specified by 'start'. The returned value may range from zero to 5.
00375 */
00376 uint8_t      week_of_month(const struct tm * timeptr, uint8_t start);
00377
00378 /**
00379     Structure which represents a date as a year, week number of that year, and day of week.
00380     See http://en.wikipedia.org/wiki/ISO\_week\_date for more information.
00381 */
00382 struct week_date {
00383     int year; /**< year number (Gregorian calendar) */
00384     int week; /**< week number (#1 is where first Thursday is in) */
00385     int day; /**< day within week */
00386 };
00387
00388 /**
00389     Return a week_date structure with the ISO_8601 week based date corresponding to the given
00390     year and day of year. See http://en.wikipedia.org/wiki/ISO\_week\_date for more
00391     information.
00392 */
00393 struct week_date * iso_week_date( int year, int yday);
00394
00395 /**
00396     Re-entrant version of iso-week_date.
00397 */
00398 void iso_week_date_r( int year, int yday, struct week_date *);
00399
00400 /**
00401     Convert a Y2K time stamp into a FAT file system time stamp.
00402 */
00403 uint32_t      fatfs_time(const struct tm * timeptr);
00404
00405 /** One hour, expressed in seconds */
00406 #define ONE_HOUR 3600
00407
00408 /** Angular degree, expressed in arc seconds */
00409 #define ONE_DEGREE 3600
00410
00411 /** One day, expressed in seconds */
00412 #define ONE_DAY 86400
00413
00414 /** Difference between the Y2K and the UNIX epochs, in seconds. To convert a Y2K
00415     timestamp to UNIX...
00416     \code
00417     long unix;
00418     time_t y2k;
00419
00420     y2k = time(NULL);
00421     unix = y2k + UNIX_OFFSET;
00422     \endcode
00423 */
00424 #define UNIX_OFFSET 946684800
00425
00426 /** Difference between the Y2K and the NTP epochs, in seconds. To convert a Y2K
00427     timestamp to NTP...
00428     \code
00429     unsigned long ntp;
00430     time_t y2k;
00431
00432     y2k = time(NULL);
00433     ntp = y2k + NTP_OFFSET;
00434     \endcode
00435 */
00436 #define NTP_OFFSET 3155673600
00437
00438 /*
00439 * =====
00440 *                                     Ephemera
```

```

00441 */
00442
00443 /**
00444     Set the geographic coordinates of the 'observer', for use with several of the
00445     following functions. Parameters are passed as seconds of North Latitude, and seconds
00446     of East Longitude.
00447
00448     For New York City...
00449     \code set_position( 40.7142 * ONE_DEGREE, -74.0064 * ONE_DEGREE); \endcode
00450 */
00451 void set_position(int32_t latitude, int32_t longitude);
00452
00453 /**
00454     Computes the difference between apparent solar time and mean solar time.
00455     The returned value is in seconds.
00456 */
00457 int16_t equation_of_time(const time_t * timer);
00458
00459 /**
00460     Computes the amount of time the sun is above the horizon, at the location of the observer.
00461
00462     NOTE: At observer locations inside a polar circle, this value can be zero during the winter,
00463     and can exceed ONE_DAY during the summer.
00464
00465     The returned value is in seconds.
00466 */
00467 int32_t daylight_seconds(const time_t * timer);
00468
00469 /**
00470     Computes the time of solar noon, at the location of the observer.
00471 */
00472 time_t solar_noon(const time_t * timer);
00473
00474 /**
00475     Return the time of sunrise, at the location of the observer. See the note about
    daylight_seconds().
00476 */
00477 time_t sun_rise(const time_t * timer);
00478
00479 /**
00480     Return the time of sunset, at the location of the observer. See the note about daylight_seconds().
00481 */
00482 time_t sun_set(const time_t * timer);
00483
00484 /**
00485     Returns the declination of the sun in radians.
00486 */
00487 float solar_declinationf(const time_t * timer);
00488
00489 /**
00490     Returns the declination of the sun in radians.
00491
00492     This implementation is only available when \c double is a 32-bit type.
00493 */
00494 double solar_declination(const time_t * timer);
00495
00496 /**
00497     Returns the declination of the sun in radians.
00498
00499     This implementation is only available when <tt>long double</tt> is
00500     a 32-bit type.
00501 */
00502 long double solar_declinationl(const time_t * timer);
00503
00504 /**
00505     Returns an approximation to the phase of the moon.
00506     The sign of the returned value indicates a waning or waxing phase.
00507     The magnitude of the returned value indicates the percentage illumination.
00508 */
00509 int8_t moon_phase(const time_t * timer);
00510
00511 /**
00512     Returns Greenwich Mean Sidereal Time, as seconds into the sidereal day.
00513     The returned value will range from 0 through 86399 seconds.
00514 */
00515 unsigned long gm_sidereal(const time_t * timer);
00516
00517 /**
00518     Returns Local Mean Sidereal Time, as seconds into the sidereal day.
00519     The returned value will range from 0 through 86399 seconds.
00520 */
00521 unsigned long lm_sidereal(const time_t * timer);
00522
00523 /**@}*/
00524 #ifdef __cplusplus
00525 }
00526 #endif

```

```
00527
00528 #endif /* TIME_H */
```

## 23.62 atomic.h File Reference

### Macros

- #define [ATOMIC\\_BLOCK](#)(type)
- #define [NONATOMIC\\_BLOCK](#)(type)
- #define [ATOMIC\\_RESTORESTATE](#)
- #define [ATOMIC\\_FORCEON](#)
- #define [NONATOMIC\\_RESTORESTATE](#)
- #define [NONATOMIC\\_FORCEOFF](#)

## 23.63 atomic.h

[Go to the documentation of this file.](#)

```
00001 /* Copyright (c) 2007 Dean Camera
00002    All rights reserved.
00003
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions are met:
00006
00007    * Redistributions of source code must retain the above copyright
00008    notice, this list of conditions and the following disclaimer.
00009
00010    * Redistributions in binary form must reproduce the above copyright
00011    notice, this list of conditions and the following disclaimer in
00012    the documentation and/or other materials provided with the
00013    distribution.
00014
00015    * Neither the name of the copyright holders nor the names of
00016    contributors may be used to endorse or promote products derived
00017    from this software without specific prior written permission.
00018
00019    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029    POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 /* $Id$ */
00033
00034 #ifndef _UTIL_ATOMIC_H_
00035 #define _UTIL_ATOMIC_H_ 1
00036
00037 #include <avr/io.h>
00038 #include <avr/interrupt.h>
00039
00040 #if !defined(__DOXYGEN__)
00041 /* Internal helper functions. */
00042 static __inline__ uint8_t __iSeiRetVal(void)
00043 {
00044     sei();
00045     return 1;
00046 }
00047
00048 static __inline__ uint8_t __iCliRetVal(void)
00049 {
00050     cli();
00051     return 1;
00052 }
00053
00054 static __inline__ void __iSeiParam(const uint8_t *__s)
00055 {
00056     sei();
```



```

00057     __asm__ volatile (" ::: \"memory\");
00058     (void)__s;
00059 }
00060
00061 static __inline__ void __iCliParam(const uint8_t *__s)
00062 {
00063     cli();
00064     __asm__ volatile (" ::: \"memory\");
00065     (void)__s;
00066 }
00067
00068 static __inline__ void __iRestore(const uint8_t *__s)
00069 {
00070     SREG = *__s;
00071     __asm__ volatile (" ::: \"memory\");
00072 }
00073 #endif /* !__DOXYGEN__ */
00074
00075 /** \file */
00076 /** \defgroup util_atomic <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks
00077
00078     \code
00079     #include <util/atomic.h>
00080     \endcode
00081
00082     \note The macros in this header file require the ISO/IEC 9899:1999
00083     ("ISO C99") feature of for loop variables that are declared inside
00084     the for loop itself. For that reason, this header file can only
00085     be used if the standard level of the compiler (option --std=) is
00086     set to either \c c99, \c gnu99 or higher.
00087
00088     The macros in this header file deal with code blocks that are
00089     guaranteed to be executed Atomically or Non-Atomically. The term
00090     "Atomic" in this context refers to the inability of the respective
00091     code to be interrupted.
00092
00093     These macros operate via automatic manipulation of the Global
00094     Interrupt Status (I) bit of the SREG register. Exit paths from
00095     both block types are all managed automatically without the need
00096     for special considerations, i.e. the interrupt status will be
00097     restored to the same value it had when entering the respective
00098     block (unless ATOMIC_FORCEON or NONATOMIC_FORCEOFF are used).
00099     \warning The features in this header are implemented by means of
00100     a for loop. This means that commands like \c break and \c continue
00101     that are located in an atomic block refer to the atomic for loop,
00102     not to a loop construct that hosts the atomic block.
00103
00104     A typical example that requires atomic access is a 16 (or more)
00105     bit variable that is shared between the main execution path and an
00106     ISR. While declaring such a variable as volatile ensures that the
00107     compiler will not optimize accesses to it away, it does not
00108     guarantee atomic access to it. Assuming the following example:
00109
00110     \code
00111     #include <stdint.h>
00112     #include <avr/interrupt.h>
00113     #include <avr/io.h>
00114
00115     volatile uint16_t ctr;
00116
00117     ISR(TIMER1_OVF_vect)
00118     {
00119         ctr--;
00120     }
00121
00122     ...
00123     int
00124     main(void)
00125     {
00126         ...
00127         ctr = 0x200;
00128         start_timer();
00129         while (ctr != 0)
00130             // wait
00131             ;
00132         ...
00133     }
00134     \endcode
00135
00136     There is a chance where the main context will exit its wait loop
00137     when the variable \c ctr just reached the value 0xFF. This happens
00138     because the compiler cannot natively access a 16-bit variable
00139     atomically in an 8-bit CPU. So the variable is for example at
00140     0x100, the compiler then tests the low byte for 0, which succeeds.
00141     It then proceeds to test the high byte, but that moment the ISR
00142     triggers, and the main context is interrupted. The ISR will
00143     decrement the variable from 0x100 to 0xFF, and the main context

```

```

00144     proceeds. It now tests the high byte of the variable which is
00145     (now) also 0, so it concludes the variable has reached 0, and
00146     terminates the loop.
00147
00148     Using the macros from this header file, the above code can be
00149     rewritten like:
00150
00151     \code
00152 #include <stdint.h>
00153 #include <avr/interrupt.h>
00154 #include <avr/io.h>
00155 #include <util/atomic.h>
00156
00157 volatile uint16_t ctr;
00158
00159 ISR(TIMER1_OVF_vect)
00160 {
00161     ctr--;
00162 }
00163
00164 ...
00165 int
00166 main(void)
00167 {
00168     ...
00169     ctr = 0x200;
00170     start_timer();
00171     sei();
00172     uint16_t ctr_copy;
00173     do
00174     {
00175         ATOMIC_BLOCK(ATOMIC_FORCEON)
00176         {
00177             ctr_copy = ctr;
00178         }
00179     }
00180     while (ctr_copy != 0);
00181     ...
00182 }
00183     \endcode
00184
00185     This will install the appropriate interrupt protection before
00186     accessing variable \c ctr, so it is guaranteed to be consistently
00187     tested. If the global interrupt state were uncertain before
00188     entering the #ATOMIC_BLOCK, it should be executed with the
00189     parameter #ATOMIC_RESTORESTATE rather than #ATOMIC_FORCEON.
00190
00191     See \ref optim_code_reorder for things to be taken into account
00192     with respect to compiler optimizations.
00193 */
00194
00195 /** \def ATOMIC_BLOCK(type)
00196     \ingroup util_atomic
00197
00198     Creates a block of code that is guaranteed to be executed
00199     atomically. Upon entering the block the Global Interrupt Status
00200     flag in SREG is disabled, and re-enabled upon exiting the block
00201     from any exit path.
00202
00203     Two possible macro parameters are permitted, #ATOMIC_RESTORESTATE
00204     and #ATOMIC_FORCEON.
00205 */
00206 #if defined(__DOXYGEN__)
00207 #define ATOMIC_BLOCK(type)
00208 #else
00209 #define ATOMIC_BLOCK(type) for ( type, __ToDo = __iCliRetVal(); \
00210     __ToDo ; __ToDo = 0 )
00211 #endif /* __DOXYGEN__ */
00212
00213 /** \def NONATOMIC_BLOCK(type)
00214     \ingroup util_atomic
00215
00216     Creates a block of code that is executed non-atomically. Upon
00217     entering the block the Global Interrupt Status flag in SREG is
00218     enabled, and disabled upon exiting the block from any exit
00219     path. This is useful when nested inside ATOMIC_BLOCK sections,
00220     allowing for non-atomic execution of small blocks of code while
00221     maintaining the atomic access of the other sections of the parent
00222     ATOMIC_BLOCK.
00223
00224     Two possible macro parameters are permitted,
00225     #NONATOMIC_RESTORESTATE and #NONATOMIC_FORCEOFF.
00226 */
00227 #if defined(__DOXYGEN__)
00228 #define NONATOMIC_BLOCK(type)
00229 #else
00230 #define NONATOMIC_BLOCK(type) for ( type, __ToDo = __iSeiRetVal(); \

```

```

00231                                     __ToDo ; __ToDo = 0 )
00232 #endif /* __DOXYGEN__ */
00233
00234 /** \def ATOMIC_RESTORESTATE
00235     \ingroup util_atomic
00236
00237     This is a possible parameter for #ATOMIC_BLOCK. When used, it will
00238     cause the ATOMIC_BLOCK to restore the previous state of the SREG
00239     register, saved before the Global Interrupt Status flag bit was
00240     disabled. The net effect of this is to make the ATOMIC_BLOCK's
00241     contents guaranteed atomic, without changing the state of the
00242     Global Interrupt Status flag when execution of the block
00243     completes.
00244 */
00245 #if defined(__DOXYGEN__)
00246 #define ATOMIC_RESTORESTATE
00247 #else
00248 #define ATOMIC_RESTORESTATE uint8_t sreg_save \
00249     __attribute__((__cleanup__(iRestore))) = SREG
00250 #endif /* __DOXYGEN__ */
00251
00252 /** \def ATOMIC_FORCEON
00253     \ingroup util_atomic
00254
00255     This is a possible parameter for #ATOMIC_BLOCK. When used, it will
00256     cause the ATOMIC_BLOCK to force the state of the SREG register on
00257     exit, enabling the Global Interrupt Status flag bit. This saves a
00258     small amount of flash space, a register, and one or more processor
00259     cycles, since the previous value of the SREG register does not need
00260     to be saved at the start of the block.
00261
00262     Care should be taken that ATOMIC_FORCEON is only used when it is
00263     known that interrupts are enabled before the block's execution or
00264     when the side effects of enabling global interrupts at the block's
00265     completion are known and understood.
00266 */
00267 #if defined(__DOXYGEN__)
00268 #define ATOMIC_FORCEON
00269 #else
00270 #define ATOMIC_FORCEON uint8_t sreg_save \
00271     __attribute__((__cleanup__(iSeiParam))) = 0
00272 #endif /* __DOXYGEN__ */
00273
00274 /** \def NONATOMIC_RESTORESTATE
00275     \ingroup util_atomic
00276
00277     This is a possible parameter for #NONATOMIC_BLOCK. When used, it
00278     will cause the NONATOMIC_BLOCK to restore the previous state of
00279     the SREG register, saved before the Global Interrupt Status flag
00280     bit was enabled. The net effect of this is to make the
00281     NONATOMIC_BLOCK's contents guaranteed non-atomic, without changing
00282     the state of the Global Interrupt Status flag when execution of
00283     the block completes.
00284 */
00285 #if defined(__DOXYGEN__)
00286 #define NONATOMIC_RESTORESTATE
00287 #else
00288 #define NONATOMIC_RESTORESTATE uint8_t sreg_save \
00289     __attribute__((__cleanup__(iRestore))) = SREG
00290 #endif /* __DOXYGEN__ */
00291
00292 /** \def NONATOMIC_FORCEOFF
00293     \ingroup util_atomic
00294
00295     This is a possible parameter for #NONATOMIC_BLOCK. When used, it
00296     will cause the NONATOMIC_BLOCK to force the state of the SREG
00297     register on exit, disabling the Global Interrupt Status flag
00298     bit. This saves a small amount of flash space, a register, and one
00299     or more processor cycles, since the previous value of the SREG
00300     register does not need to be saved at the start of the block.
00301
00302     Care should be taken that NONATOMIC_FORCEOFF is only used when it
00303     is known that interrupts are disabled before the block's execution
00304     or when the side effects of disabling global interrupts at the
00305     block's completion are known and understood.
00306 */
00307 #if defined(__DOXYGEN__)
00308 #define NONATOMIC_FORCEOFF
00309 #else
00310 #define NONATOMIC_FORCEOFF uint8_t sreg_save \
00311     __attribute__((__cleanup__(iCliParam))) = 0
00312 #endif /* __DOXYGEN__ */
00313
00314 #endif

```

## 23.64 crc16.h File Reference

### Functions

- `static uint16_t crc16_update (uint16_t __crc, uint8_t __data)`
- `static uint16_t crc_xmodem_update (uint16_t __crc, uint8_t __data)`
- `static uint16_t crc_ccitt_update (uint16_t __crc, uint8_t __data)`
- `static uint8_t crc_ibutton_update (uint8_t __crc, uint8_t __data)`
- `static uint8_t crc8_ccitt_update (uint8_t __crc, uint8_t __data)`

## 23.65 crc16.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, 2003, 2004 Marek Michalkiewicz
00002 Copyright (c) 2005, 2007 Joerg Wunsch
00003 Copyright (c) 2013 Dave Hylands
00004 Copyright (c) 2013 Frederic Nadeau
00005 All rights reserved.
00006
00007 Redistribution and use in source and binary forms, with or without
00008 modification, are permitted provided that the following conditions are met:
00009
00010 * Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 * Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in
00015 the documentation and/or other materials provided with the
00016 distribution.
00017
00018 * Neither the name of the copyright holders nor the names of
00019 contributors may be used to endorse or promote products derived
00020 from this software without specific prior written permission.
00021
00022 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00023 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00024 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00025 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00026 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00027 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00028 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00029 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00030 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00031 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00032 POSSIBILITY OF SUCH DAMAGE. */
00033
00034 /* $Id$ */
00035
00036 #ifndef _UTIL_CRC16_H_
00037 #define _UTIL_CRC16_H_
00038
00039 #include <stdint.h>
00040
00041 #ifndef __DOXYGEN__
00042 #ifndef __ATTR_ALWAYS_INLINE__
00043 #define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00044 #endif
00045 #endif /* ! DOXYGEN */
00046
00047 /** \file */
00048 /** \defgroup util_crc <util/crc16.h>: CRC Computations
00049 \code#include <util/crc16.h>\endcode
00050
00051 This header file provides a optimized inline functions for calculating
00052 cyclic redundancy checks (CRC) using common polynomials.
00053
00054 \par References:
00055
00056 \par
00057
00058 See the Dallas Semiconductor app note 27 for 8051 assembler example and
00059 general CRC optimization suggestions. The table on the last page of the
00060 app note is the key to understanding these implementations.
00061
00062 \par
00063
00064 Jack Crenshaw's "Implementing CRCs" article in the January 1992 issue of \e

```

```

00065 Embedded \e Systems \e Programming. This may be difficult to find, but it
00066 explains CRC's in very clear and concise terms. Well worth the effort to
00067 obtain a copy.
00068
00069 A typical application would look like:
00070
00071 \code
00072 // Dallas iButton test vector.
00073 uint8_t serno[] = { 0x02, 0x1c, 0xb8, 0x01, 0, 0, 0, 0xa2 };
00074
00075 int
00076 checkcrc (void)
00077 {
00078     uint8_t crc = 0, i;
00079
00080     for (i = 0; i < sizeof serno / sizeof serno[0]; i++)
00081         crc = _crc_ibutton_update (crc, serno[i]);
00082
00083     return crc; // must be 0
00084 }
00085 \endcode
00086 */
00087
00088 /** \ingroup util_crc
00089 Optimized CRC-16 calculation.
00090
00091 Polynomial:  $x^{16} + x^{15} + x^2 + 1$  (0xa001)
00092 Initial value:  $0xffff$ 
00093
00094 This CRC is normally used in disk-drive controllers.
00095
00096 The following is the equivalent functionality written in C.
00097
00098 \code
00099 uint16_t
00100 crc16_update (uint16_t crc, uint8_t a)
00101 {
00102     crc ^= a;
00103     for (int i = 0; i < 8; ++i)
00104     {
00105         if (crc & 1)
00106             crc = (crc >> 1) ^ 0xA001;
00107         else
00108             crc = crc >> 1;
00109     }
00110
00111     return crc;
00112 }
00113 \endcode */
00114
00115 static __ATTR_ALWAYS_INLINE__ uint16_t
00116 _crc16_update(uint16_t __crc, uint8_t __data)
00117 {
00118     uint8_t __tmp;
00119     uint16_t __ret;
00120
00121     __asm__ __volatile__ (
00122         "eor %A0,%2" "\n\t"
00123         "mov %1,%A0" "\n\t"
00124         "swap %1" "\n\t"
00125         "eor %1,%A0" "\n\t"
00126         "mov __tmp_reg__,%1" "\n\t"
00127         "lsr %1" "\n\t"
00128         "lsr %1" "\n\t"
00129         "eor %1,__tmp_reg__" "\n\t"
00130         "mov __tmp_reg__,%1" "\n\t"
00131         "lsr %1" "\n\t"
00132         "eor %1,__tmp_reg__" "\n\t"
00133         "andi %1,0x07" "\n\t"
00134         "mov __tmp_reg__,%A0" "\n\t"
00135         "mov %A0,%B0" "\n\t"
00136         "lsr %1" "\n\t"
00137         "ror __tmp_reg__" "\n\t"
00138         "ror %1" "\n\t"
00139         "mov %B0,__tmp_reg__" "\n\t"
00140         "eor %A0,%1" "\n\t"
00141         "lsr __tmp_reg__" "\n\t"
00142         "ror %1" "\n\t"
00143         "eor %B0,__tmp_reg__" "\n\t"
00144         "eor %A0,%1"
00145         : "=r" (__ret), "=d" (__tmp)
00146         : "r" (__data), "0" (__crc)
00147         : "r0"
00148     );
00149     return __ret;
00150 }
00151

```

```

00152 /** \ingroup util_crc
00153     Optimized CRC-XMODEM calculation.
00154
00155     Polynomial:  $x^{16} + x^{12} + x^5 + 1$  (0x1021)<br>
00156     Initial value:  $\backslash c 0x0$ 
00157
00158     This is the CRC used by the Xmodem-CRC protocol.
00159
00160     The following is the equivalent functionality written in C.
00161
00162     \code
00163     uint16_t
00164     crc_xmodem_update (uint16_t crc, uint8_t data)
00165     {
00166         crc = crc ^ ((uint16_t)data << 8);
00167         for (int i = 0; i < 8; i++)
00168         {
00169             if (crc & 0x8000)
00170                 crc = (crc << 1) ^ 0x1021;
00171             else
00172                 crc <<= 1;
00173         }
00174
00175         return crc;
00176     }
00177     \endcode */
00178
00179 static __ATTR_ALWAYS_INLINE__ uint16_t
00180 _crc_xmodem_update (uint16_t __crc, uint8_t __data)
00181 {
00182     uint16_t __ret;          /* %B0:%A0 (alias for __crc) */
00183     uint8_t __tmp1;         /* %1 */
00184     uint8_t __tmp2;         /* %2 */
00185                             /* %3 __data */
00186
00187     __asm__ __volatile__ (
00188         "eor    %B0,%3"      "\n\t"
00189         "mov    %1,%A0"      "\n\t"
00190         "mov    %2,%B0"      "\n\t"
00191
00192         "mov    %A0,%B0"     "\n\t"
00193         "swap   %B0"         "\n\t"
00194         "eor    %A0,%B0"     "\n\t"
00195
00196         "andi   %A0,0xf0"    "\n\t"
00197         "andi   %B0,0x0f"    "\n\t"
00198
00199         "eor    %1,%A0"      "\n\t"
00200         "eor    %2,%B0"      "\n\t"
00201
00202         "lsl   %A0"          "\n\t"
00203         "rol   %B0"          "\n\t"
00204
00205         "eor    %B0,%1"      "\n\t"
00206         "eor    %A0,%2"
00207         : "=d" (__ret), "=r" (__tmp1), "=r" (__tmp2)
00208         : "r" (__data), "0" (__crc)
00209     );
00210     return __ret;
00211 }
00212
00213 /** \ingroup util_crc
00214     Optimized CRC-CCITT calculation.
00215
00216     Polynomial:  $x^{16} + x^{12} + x^5 + 1$  (0x8408)<br>
00217     Initial value:  $\backslash c 0xffff$ 
00218
00219     This is the CRC used by PPP and IrDA.
00220
00221     See RFC1171 (PPP protocol) and IrDA IrLAP 1.1
00222
00223     \note Although the CCITT polynomial is the same as that used by the Xmodem
00224     protocol, they are quite different. The difference is in how the bits are
00225     shifted through the algorithm. Xmodem shifts the MSB of the CRC and the
00226     input first, while CCITT shifts the LSB of the CRC and the input first.
00227
00228     The following is the equivalent functionality written in C.
00229
00230     \code
00231     uint16_t
00232     crc_ccitt_update (uint16_t crc, uint8_t data)
00233     {
00234         data ^= lo8 (crc);
00235         data ^= data << 4;
00236
00237         return (((uint16_t)data << 8) | hi8 (crc)) ^ (uint8_t)(data >> 4)
00238             ^ ((uint16_t)data << 3);

```

```

00239     }
00240     \endcode */
00241
00242 static __ATTR_ALWAYS_INLINE__ uint16_t
00243 _crc_ccitt_update (uint16_t __crc, uint8_t __data)
00244 {
00245     uint16_t __ret;
00246
00247     __asm__ __volatile__ (
00248         "eor    %A0,%1"           "\n\t"
00249
00250         "mov    __tmp_reg__,%A0" "\n\t"
00251         "swap  %A0"               "\n\t"
00252         "andi  %A0,0xf0"         "\n\t"
00253         "eor    %A0,__tmp_reg__" "\n\t"
00254
00255         "mov    __tmp_reg__,%B0" "\n\t"
00256
00257         "mov    %B0,%A0"         "\n\t"
00258
00259         "swap  %A0"               "\n\t"
00260         "andi  %A0,0x0f"         "\n\t"
00261         "eor    __tmp_reg__,%A0" "\n\t"
00262
00263         "lsr   %A0"               "\n\t"
00264         "eor    %B0,%A0"         "\n\t"
00265
00266         "eor    %A0,%B0"         "\n\t"
00267         "lsl   %A0"               "\n\t"
00268         "lsl   %A0"               "\n\t"
00269         "lsl   %A0"               "\n\t"
00270         "eor    %A0,__tmp_reg__"
00271
00272         : "=d" (__ret)
00273         : "r" (__data), "0" (__crc)
00274         : "r0"
00275     );
00276     return __ret;
00277 }
00278
00279 /** \ingroup util_crc
00280     Optimized Dallas (now Maxim) iButton 8-bit CRC calculation.
00281
00282     Polynomial:  $x^8 + x^5 + x^4 + 1$  (0x8C)
00283     Initial value:  $\backslash c 0x0$ 
00284
00285     See http://www.maxim-ic.com/appnotes.cfm/appnote\_number/27
00286
00287     The following is the equivalent functionality written in C.
00288
00289     \code
00290     uint8_t
00291     _crc_ibutton_update (uint8_t crc, uint8_t data)
00292     {
00293         crc = crc ^ data;
00294         for (uint8_t i = 0; i < 8; i++)
00295         {
00296             if (crc & 0x01)
00297                 crc = (crc » 1) ^ 0x8C;
00298             else
00299                 crc »= 1;
00300         }
00301         return crc;
00302     }
00303     \endcode
00304 */
00305
00306 static __ATTR_ALWAYS_INLINE__ uint8_t
00307 _crc_ibutton_update (uint8_t __crc, uint8_t __data)
00308 {
00309     uint8_t __i, __pattern;
00310     __asm__ __volatile__ (
00311         "eor    %0, %4"           "\n\t"
00312         "ldi    %1, 8"           "\n\t"
00313         "ldi    %2, 0x8C"        "\n\t"
00314         "1: lsr %0"               "\n\t"
00315         "brcc  2f"               "\n\t"
00316         "eor    %0, %2"           "\n\t"
00317         "2: dec %1"              "\n\t"
00318         "brne  1b"
00319
00320         : "=r" (__crc), "=d" (__i), "=d" (__pattern)
00321         : "0" (__crc), "r" (__data);
00322     return __crc;
00323 }
00324
00325 /** \ingroup util_crc

```

```

00326     Optimized CRC-8-CCITT calculation.
00327
00328     Polynomial:  $x^8 + x^2 + x + 1$  (0xE0)
00329
00330     For use with simple CRC-8
00331     Initial value: 0x0
00332
00333     For use with CRC-8-ROHC
00334     Initial value: 0xff
00335     Reference: http://tools.ietf.org/html/rfc3095#section-5.9.1
00336
00337     For use with CRC-8-ATM/ITU
00338     Initial value: 0xff
00339     Final XOR value: 0x55
00340     Reference: http://www.itu.int/rec/T-REC-I.432.1-199902-I/en
00341
00342     The C equivalent has been originally written by Dave Hylands.
00343     Assembly code is based on _crc_ibutton_update optimization.
00344
00345     The following is the equivalent functionality written in C.
00346
00347     \code
00348     uint8_t
00349     _crc8_ccitt_update (uint8_t inCrc, uint8_t inData)
00350     {
00351         uint8_t data = inCrc ^ inData;
00352
00353         for (int i = 0; i < 8; i++)
00354         {
00355             if ((data & 0x80) != 0)
00356             {
00357                 data <<= 1;
00358                 data ^= 0x07;
00359             }
00360             else
00361             {
00362                 data <<= 1;
00363             }
00364         }
00365         return data;
00366     }
00367     \endcode
00368 */
00369
00370 static __ATTR_ALWAYS_INLINE__ uint8_t
00371 _crc8_ccitt_update(uint8_t __crc, uint8_t __data)
00372 {
00373     uint8_t __i, __pattern;
00374     __asm__ __volatile__ (
00375         "eor    %0, %4" "\n\t"
00376         "ldi    %1, 8" "\n\t"
00377         "ldi    %2, 0x07" "\n\t"
00378         "1: lsl    %0" "\n\t"
00379         "brcc   2f" "\n\t"
00380         "eor    %0, %2" "\n\t"
00381         "2: dec    %1" "\n\t"
00382         "brne   1b"
00383         : "=r" (__crc), "=d" (__i), "=d" (__pattern)
00384         : "0" (__crc), "r" (__data);
00385         return __crc;
00386     }
00387
00388 #endif /* _UTIL_CRC16_H_ */

```

## 23.66 delay.h File Reference

### Macros

- `#define F_CPU 1000000UL`

### Functions

- static void `_delay_ms` (double \_\_ms)
- static void `_delay_us` (double \_\_us)



## 23.67 delay.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, Marek Michalkiewicz
00002    Copyright (c) 2004,2005,2007 Joerg Wunsch
00003    Copyright (c) 2007 Florin-Viorel Petrov
00004    All rights reserved.
00005
00006    Redistribution and use in source and binary forms, with or without
00007    modification, are permitted provided that the following conditions are met:
00008
00009    * Redistributions of source code must retain the above copyright
00010    notice, this list of conditions and the following disclaimer.
00011
00012    * Redistributions in binary form must reproduce the above copyright
00013    notice, this list of conditions and the following disclaimer in
00014    the documentation and/or other materials provided with the
00015    distribution.
00016
00017    * Neither the name of the copyright holders nor the names of
00018    contributors may be used to endorse or promote products derived
00019    from this software without specific prior written permission.
00020
00021    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00022    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00023    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00024    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00025    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00026    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00027    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00028    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00029    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00030    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00031    POSSIBILITY OF SUCH DAMAGE. */
00032
00033 /* $Id$ */
00034
00035 #ifndef _UTIL_DELAY_H_
00036 #define _UTIL_DELAY_H_ 1
00037
00038 #ifndef __DOXYGEN__
00039 #   ifndef __HAS_DELAY_CYCLES
00040 #       define __HAS_DELAY_CYCLES 1
00041 #   endif
00042
00043 #   ifndef __ATTR_ALWAYS_INLINE__
00044 #       define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00045 #   endif
00046
00047 #endif /* __DOXYGEN__ */
00048
00049 #include <stdint.h>
00050 #include <util/delay_basic.h>
00051
00052 /** \file */
00053 /** \defgroup util_delay <util/delay.h>: Convenience functions for busy-wait delay loops
00054     \code
00055     #define F_CPU 1000000UL // 1 MHz
00056     // #define F_CPU 14.7456e6
00057     #include <util/delay.h>
00058     \endcode
00059
00060     \note As an alternative method, it is possible to pass the
00061     F_CPU macro down to the compiler from the Makefile.
00062     Obviously, in that case, no \c #define statement should be
00063     used.
00064
00065     The functions in this header file are wrappers around the basic
00066     busy-wait functions from <tt>\<util/delay_basic.h>\</tt>. They are meant as
00067     convenience functions where actual time values can be specified
00068     rather than a number of cycles to wait for. The idea behind is
00069     that compile-time constant expressions will be eliminated by
00070     compiler optimization so floating-point expressions can be used
00071     to calculate the number of delay cycles needed based on the CPU
00072     frequency passed by the macro F_CPU.
00073
00074     \note In order for these functions to work as intended, compiler
00075     optimizations <em>must</em> be enabled, and the delay time
00076     <em>must</em> be an expression that is a known constant at
00077     compile-time. If these requirements are not met, the resulting
00078     delay will be much longer (and basically unpredictable), and
00079     applications that otherwise do not use floating-point calculations
00080     will experience severe code bloat by the floating-point library
00081     routines linked into the application.
00082
00083     The functions available allow the specification of microsecond, and

```

```

00084     millisecond delays directly, using the application-supplied macro
00085     F_CPU as the CPU clock frequency (in Hertz).
00086 */
00087
00088
00089 #ifndef F_CPU
00090 /* prevent compiler error by supplying a default */
00091 # warning "F_CPU not defined for <util/delay.h>"
00092 /** \ingroup util_delay
00093     \def F_CPU
00094     \brief CPU frequency in Hz
00095
00096     The macro F_CPU specifies the CPU frequency to be considered by
00097     the delay macros. This macro is normally supplied by the
00098     environment (e.g. from within a project header, or the project's
00099     Makefile). The value 1 MHz here is only provided as a "vanilla"
00100     fallback if no such user-provided definition could be found.
00101
00102     In terms of the delay functions, the CPU frequency can be given as
00103     a floating-point constant (e.g. 3.6864e6 for 3.6864 MHz).
00104     However, the macros in <util/setbaud.h> require it to be an
00105     integer value.
00106 */
00107 # define F_CPU 1000000UL
00108 #endif
00109
00110 #ifndef __OPTIMIZE__
00111 # warning "Compiler optimizations disabled; functions from <util/delay.h> won't work as designed"
00112 #endif
00113
00114 /**
00115     \ingroup util_delay
00116
00117     Perform a delay of \c __ms milliseconds, using _delay_loop_2().
00118
00119     The macro #F_CPU is supposed to be defined to a
00120     constant defining the CPU clock frequency (in Hertz).
00121
00122     The maximal possible delay is 262.14 ms / F_CPU in MHz.
00123
00124     When the user request delay which exceed the maximum possible one,
00125     _delay_ms() provides a decreased resolution functionality. In this
00126     mode _delay_ms() will work with a resolution of 1/10&nbsp;ms, providing
00127     delays up to 6.5535 seconds (independent from CPU frequency). The
00128     user will not be informed about decreased resolution.
00129
00130     If the avr-gcc toolchain has \c __builtin_avr_delay_cycles()
00131     support, maximal possible delay is 4294967.295 ms/ F_CPU in MHz. For
00132     values greater than the maximal possible delay, overflow may result in
00133     no delay i.e., 0&nbsp;ms.
00134
00135     Conversion of \c __ms into clock cycles may not always result in
00136     an integral value. By default, the clock cycles are rounded up to the next
00137     integer. This ensures that the user gets at least \c __ms
00138     microseconds of delay.
00139
00140     Alternatively, by defining the macro \c __DELAY_ROUND_DOWN__, or
00141     \c __DELAY_ROUND_CLOSEST__, before including this header file, the
00142     algorithm can be made to round down, or round to closest integer,
00143     respectively.
00144
00145     \note The implementation of _delay_ms() based on
00146     \c __builtin_avr_delay_cycles() is not backward compatible with older
00147     implementations. In order to get functionality backward compatible
00148     with previous versions, the macro \c __DELAY_BACKWARD_COMPATIBLE__
00149     must be defined before including this header file.
00150 */
00151 static __ATTR_ALWAYS_INLINE__ void _delay_ms(double __ms);
00152
00153 void
00154 _delay_ms(double __ms)
00155 {
00156     double __tmp ;
00157     #if __HAS_DELAY_CYCLES && defined(__OPTIMIZE__) \
00158     && !defined(__DELAY_BACKWARD_COMPATIBLE__)
00159         uint32_t __ticks_dc;
00160         extern void __builtin_avr_delay_cycles(uint32_t);
00161         __tmp = ((F_CPU) / 1e3) * __ms;
00162
00163         #if defined(__DELAY_ROUND_DOWN__)
00164             __ticks_dc = (uint32_t)__builtin_fabs(__tmp);
00165
00166         #elif defined(__DELAY_ROUND_CLOSEST__)
00167             __ticks_dc = (uint32_t)(__builtin_fabs(__tmp)+0.5);
00168
00169         #else
00170             //round up by default

```

```

00171     __ticks_dc = (uint32_t)(__builtin_ceil(__builtin_fabs(__tmp)));
00172 #endif
00173
00174     __builtin_avr_delay_cycles(__ticks_dc);
00175
00176 #else
00177     uint16_t __ticks;
00178     __tmp = ((F_CPU) / 4e3) * __ms;
00179     if (__tmp < 1.0)
00180         __ticks = 1;
00181     else if (__tmp > 65535)
00182     {
00183         // __ticks = requested delay in 1/10 ms
00184         __ticks = (uint16_t) (__ms * 10.0);
00185         while(__ticks)
00186         {
00187             // wait 1/10 ms
00188             _delay_loop_2(((F_CPU) / 4e3) / 10);
00189             __ticks --;
00190         }
00191         return;
00192     }
00193     else
00194         __ticks = (uint16_t)__tmp;
00195     _delay_loop_2(__ticks);
00196 #endif
00197 }
00198
00199 /**
00200  \ingroup util_delay
00201
00202  Perform a delay of \c __us microseconds, using _delay_loop_1().
00203
00204  The macro #F_CPU is supposed to be defined to a
00205  constant defining the CPU clock frequency (in Hertz).
00206
00207  The maximal possible delay is 768 &mu;s / F_CPU in MHz.
00208
00209  If the user requests a delay greater than the maximal possible one,
00210  _delay_us() will automatically call _delay_ms() instead. The user
00211  will not be informed about this case.
00212
00213  If the avr-gcc toolchain has __builtin_avr_delay_cycles()
00214  support, maximal possible delay is 4294967.295 &mu;s/ F_CPU in MHz. For
00215  values greater than the maximal possible delay, overflow may result in
00216  no delay i.e., 0&nbsp;&mu;s.
00217
00218  Conversion of \c __us into clock cycles may not always result in
00219  integer. By default, the clock cycles are rounded up to next
00220  integer. This ensures that the user gets at least \c __us
00221  microseconds of delay.
00222
00223  Alternatively, by defining the macro \c __DELAY_ROUND_DOWN__, or
00224  \c __DELAY_ROUND_CLOSEST__, before including this header file, the
00225  algorithm can be made to round down, or round to closest integer,
00226  respectively.
00227
00228  \note The implementation of _delay_us() based on
00229  \c __builtin_avr_delay_cycles() is not backward compatible with older
00230  implementations. In order to get functionality backward compatible
00231  with previous versions, the macro \c __DELAY_BACKWARD_COMPATIBLE__
00232  must be defined before including this header file.
00233  */
00234 static __ATTR_ALWAYS_INLINE__ void _delay_us(double __us);
00235
00236 void
00237 _delay_us(double __us)
00238 {
00239     double __tmp ;
00240 #if __HAS_DELAY_CYCLES && defined(__OPTIMIZE__) \
00241     && !defined(__DELAY_BACKWARD_COMPATIBLE__)
00242     uint32_t __ticks_dc;
00243     extern void __builtin_avr_delay_cycles(uint32_t);
00244     __tmp = ((F_CPU) / 1e6) * __us;
00245
00246     #if defined(__DELAY_ROUND_DOWN__)
00247         __ticks_dc = (uint32_t)__builtin_fabs(__tmp);
00248
00249     #elif defined(__DELAY_ROUND_CLOSEST__)
00250         __ticks_dc = (uint32_t)(__builtin_fabs(__tmp)+0.5);
00251
00252     #else
00253         //round up by default
00254         __ticks_dc = (uint32_t)(__builtin_ceil(__builtin_fabs(__tmp)));
00255     #endif
00256
00257     __builtin_avr_delay_cycles(__ticks_dc);

```

```

00258
00259 #else
00260     uint8_t __ticks;
00261     double __tmp2 ;
00262     __tmp = ((F_CPU) / 3e6) * __us;
00263     __tmp2 = ((F_CPU) / 4e6) * __us;
00264     if (__tmp < 1.0)
00265         __ticks = 1;
00266     else if (__tmp2 > 65535)
00267     {
00268         _delay_ms(__us / 1000.0);
00269         return;
00270     }
00271     else if (__tmp > 255)
00272     {
00273         uint16_t __ticks=(uint16_t)__tmp2;
00274         _delay_loop_2(__ticks);
00275         return;
00276     }
00277     else
00278         __ticks = (uint8_t)__tmp;
00279     _delay_loop_1(__ticks);
00280 #endif
00281 }
00282
00283
00284 #endif /* _UTIL_DELAY_H_ */

```

## 23.68 delay\_basic.h File Reference

### Functions

- void `_delay_loop_1` (uint8\_t \_\_count)
- void `_delay_loop_2` (uint16\_t \_\_count)

## 23.69 delay\_basic.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, Marek Michalkiewicz
00002    Copyright (c) 2007 Joerg Wunsch
00003    All rights reserved.
00004
00005    Redistribution and use in source and binary forms, with or without
00006    modification, are permitted provided that the following conditions are met:
00007
00008    * Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010
00011    * Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in
00013    the documentation and/or other materials provided with the
00014    distribution.
00015
00016    * Neither the name of the copyright holders nor the names of
00017    contributors may be used to endorse or promote products derived
00018    from this software without specific prior written permission.
00019
00020    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00022    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030    POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033
00034 #ifndef _UTIL_DELAY_BASIC_H_
00035 #define _UTIL_DELAY_BASIC_H_ 1
00036
00037 #include <inttypes.h>
00038

```

```

00039 #if !defined(__DOXYGEN__)
00040 static __inline__ void _delay_loop_1(uint8_t __count) __attribute__((__always_inline__));
00041 static __inline__ void _delay_loop_2(uint16_t __count) __attribute__((__always_inline__));
00042 #endif
00043
00044 /** \file */
00045 /** \defgroup util_delay_basic <util/delay_basic.h>: Basic busy-wait delay loops
00046     \code
00047     #include <util/delay_basic.h>
00048     \endcode
00049
00050     The functions in this header file implement simple delay loops
00051     that perform a busy-waiting. They are typically used to
00052     facilitate short delays in the program execution. They are
00053     implemented as count-down loops with a well-known CPU cycle
00054     count per loop iteration. As such, no other processing can
00055     occur simultaneously. It should be kept in mind that the
00056     functions described here do not disable interrupts.
00057
00058     In general, for long delays, the use of hardware timers is
00059     much preferable, as they free the CPU, and allow for
00060     concurrent processing of other events while the timer is
00061     running. However, in particular for very short delays, the
00062     overhead of setting up a hardware timer is too much compared
00063     to the overall delay time.
00064
00065     Two inline functions are provided for the actual delay algorithms.
00066
00067 */
00068
00069 /** \ingroup util_delay_basic
00070
00071     Delay loop using an 8-bit counter \c __count, so up to 256
00072     iterations are possible. (The value 256 would have to be passed
00073     as 0.) The loop executes three CPU cycles per iteration, not
00074     including the overhead the compiler needs to setup the counter
00075     register.
00076
00077     Thus, at a CPU speed of 1 MHz, delays of up to 768 microseconds
00078     can be achieved.
00079 */
00080 void
00081 _delay_loop_1(uint8_t __count)
00082 {
00083     __asm__ volatile (
00084         "1: dec %0" "\n\t"
00085         "brne lb"
00086         : "=r" (__count)
00087         : "0" (__count)
00088     );
00089 }
00090
00091 /** \ingroup util_delay_basic
00092
00093     Delay loop using a 16-bit counter \c __count, so up to 65536
00094     iterations are possible. (The value 65536 would have to be
00095     passed as 0.) The loop executes four CPU cycles per iteration,
00096     not including the overhead the compiler requires to setup the
00097     counter register pair.
00098
00099     Thus, at a CPU speed of 1 MHz, delays of up to about 262.1
00100     milliseconds can be achieved.
00101 */
00102 void
00103 _delay_loop_2(uint16_t __count)
00104 {
00105     #if defined (__AVR_TINY__)
00106         __asm__ volatile (
00107             "1: subi %A0,1" "\n\t"
00108             "sbci %B0,0" "\n\t"
00109             "brne lb"
00110             : "+d" (__count)
00111         );
00112     #else
00113         __asm__ volatile (
00114             "1: sbiw %0,1" "\n\t"
00115             "brne lb"
00116             : "+w" (__count)
00117         );
00118     #endif /* AVR_TINY */
00119 }
00120
00121 #endif /* _UTIL_DELAY_BASIC_H_ */

```

## 23.70 eu\_dst.h File Reference

### Functions

- int `eu_dst` (const `time_t` \*timer, `int32_t` \*z)

### 23.71 eu\_dst.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * (c)2012 Michael Duane Rice All rights reserved.
00003  *
00004  * Redistribution and use in source and binary forms, with or without
00005  * modification, are permitted provided that the following conditions are
00006  * met:
00007  *
00008  * Redistributions of source code must retain the above copyright notice, this
00009  * list of conditions and the following disclaimer. Redistributions in binary
00010  * form must reproduce the above copyright notice, this list of conditions
00011  * and the following disclaimer in the documentation and/or other materials
00012  * provided with the distribution. Neither the name of the copyright holders
00013  * nor the names of contributors may be used to endorse or promote products
00014  * derived from this software without specific prior written permission.
00015  *
00016  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00017  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00018  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00019  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00020  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00021  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00022  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00023  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00024  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00025  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00026  * POSSIBILITY OF SUCH DAMAGE.
00027  */
00028
00029 /* $Id$ */
00030
00031 #ifndef EU_DST_H
00032 #define EU_DST_H
00033
00034 #ifdef __cplusplus
00035 extern "C" {
00036 #endif
00037
00038 #include <time.h>
00039 #include <stdint.h>
00040
00041 /** \file */
00042 /** \defgroup eu_dst <util/eu_dst.h>: Daylight Saving function for the European Union.
00043
00044     \code #include <util/eu_dst.h> \endcode
00045     Dayligh Saving Time for the European Union */
00046
00047 /** \ingroup eu_dst
00048     \fn int eu_dst (const time_t *timer, int32_t *z)
00049     To utilize this function, call \code set_dst(eu_dst); \endcode
00050
00051     Given the time stamp and time zone parameters provided, the Daylight
00052     Saving function must return a value appropriate for the tm structures'
00053     tm_isdst element. That is:
00054
00055     - \c 0 : If Daylight Saving is not in effect.
00056
00057     - \c -1 : If it cannot be determined if Daylight Saving is in effect.
00058
00059     - A positive integer: Represents the number of seconds a clock is advanced
00060     for Daylight Saving. This will typically be ONE_HOUR.
00061
00062     Daylight Saving 'rules' are subject to frequent change. For production
00063     applications it is recommended to write your own DST function, which
00064     uses 'rules' obtained from, and modifiable by, the end user (perhaps
00065     stored in EEPROM).
00066 */
00067 int eu_dst (const time_t *timer, int32_t *z);
00068
00069 #ifdef __cplusplus
00070 }
00071 #endif
00072
00073 #endif

```

## 23.72 parity.h File Reference

### Functions

- static `uint8_t parity_even_bit (uint8_t __val)`

## 23.73 parity.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2002, Marek Michalkiewicz
00002    Copyright (c) 2004,2005,2007 Joerg Wunsch
00003    All rights reserved.
00004
00005    Redistribution and use in source and binary forms, with or without
00006    modification, are permitted provided that the following conditions are met:
00007
00008    * Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010
00011    * Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in
00013    the documentation and/or other materials provided with the
00014    distribution.
00015
00016    * Neither the name of the copyright holders nor the names of
00017    contributors may be used to endorse or promote products derived
00018    from this software without specific prior written permission.
00019
00020    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00022    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030    POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033
00034 #ifndef _UTIL_PARITY_H_
00035 #define _UTIL_PARITY_H_
00036
00037 #include <stdint.h>
00038
00039 #ifndef __DOXYGEN__
00040 #ifndef __ATTR_ALWAYS_INLINE__
00041 #define __ATTR_ALWAYS_INLINE__ __inline__ __attribute__((__always_inline__))
00042 #endif
00043 #endif /* !DOXYGEN */
00044
00045 /** \file */
00046 /** \defgroup util_parity <util/parity.h>: Parity bit generation
00047     \code #include <util/parity.h> \endcode
00048
00049     This header file contains optimized assembler code to calculate
00050     the parity bit for a byte.
00051 */
00052 /** \fn uint8_t parity_even_bit (uint8_t val);
00053     \ingroup util_parity
00054     \returns 1 if \c val has an odd number of bits set, and 0 otherwise. */
00055
00056 static __ATTR_ALWAYS_INLINE__
00057 uint8_t parity_even_bit (uint8_t __val)
00058 {
00059     if (__builtin_constant_p (__builtin_parity (__val)))
00060         return (uint8_t) __builtin_parity (__val);
00061
00062     __asm (/* parity is in [0..7] */
00063           "mov  __tmp_reg__, %0"      "\n\t"
00064           "swap __tmp_reg__"         "\n\t"
00065           "eor  %0, __tmp_reg__"     "\n\t"
00066           /* parity is in [0..3] */
00067           "subi %0, -4"              "\n\t"
00068           "andi %0, -5"              "\n\t"
00069           "subi %0, -6"              "\n\t"
00070           /* parity is in [0,3] */

```

```

00071     "sbrc %0, 3"           "\n\t"
00072     "inc %0"
00073     /* parity is in [0] */
00074     : "+d" (__val) :: "r0");
00075
00076     return __val & 1;
00077 }
00078
00079 #endif /* _UTIL_PARITY_H_ */

```

## 23.74 setbaud.h File Reference

### Macros

- `#define BAUD_TOL 2`
- `#define UBRR_VALUE`
- `#define UBRRRL_VALUE`
- `#define UBRRRH_VALUE`
- `#define USE_2X 0`

## 23.75 setbaud.h

[Go to the documentation of this file.](#)

```

00001 /* Copyright (c) 2007 Cliff Lawson
00002 Copyright (c) 2007 Carlos Lamas
00003 All rights reserved.
00004
00005 Redistribution and use in source and binary forms, with or without
00006 modification, are permitted provided that the following conditions are met:
00007
00008 * Redistributions of source code must retain the above copyright
00009 notice, this list of conditions and the following disclaimer.
00010
00011 * Redistributions in binary form must reproduce the above copyright
00012 notice, this list of conditions and the following disclaimer in
00013 the documentation and/or other materials provided with the
00014 distribution.
00015
00016 * Neither the name of the copyright holders nor the names of
00017 contributors may be used to endorse or promote products derived
00018 from this software without specific prior written permission.
00019
00020 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00022 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030 POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033
00034 /**
00035  \file
00036 */
00037
00038 /**
00039  \defgroup util_setbaud <util/setbaud.h>: Helper macros for baud rate calculations
00040  \code
00041  #define F_CPU 11059200
00042  #define BAUD 38400
00043  #include <util/setbaud.h>
00044  \endcode
00045
00046  This header file requires that on entry values are already defined
00047  for F_CPU and BAUD. In addition, the macro BAUD_TOL will define
00048  the baud rate tolerance (in percent) that is acceptable during
00049  the calculations. The value of BAUD_TOL will default to 2 %.
00050
00051  This header file defines macros suitable to setup the UART baud

```



```

00052     rate prescaler registers of an AVR. All calculations are done
00053     using the C preprocessor. Including this header file causes no
00054     other side effects so it is possible to include this file more than
00055     once (supposedly, with different values for the BAUD parameter),
00056     possibly even within the same function.
00057
00058     Assuming that the requested BAUD is valid for the given F_CPU then
00059     the macro UBRR_VALUE is set to the required prescaler value. Two
00060     additional macros are provided for the low and high bytes of the
00061     prescaler, respectively: UBRRH_VALUE is set to the lower byte of
00062     the UBRR_VALUE and UBRRL_VALUE is set to the upper byte. An
00063     additional macro USE_2X will be defined. Its value is set to 1 if
00064     the desired BAUD rate within the given tolerance could only be
00065     achieved by setting the U2X bit in the UART configuration. It will
00066     be defined to 0 if U2X is not needed.
00067
00068     Example usage:
00069
00070     \code
00071     #include <avr/io.h>
00072
00073     #define F_CPU 4000000
00074
00075     static void
00076     uart_9600(void)
00077     {
00078         #define BAUD 9600
00079         #include <util/setbaud.h>
00080         UBRRH = UBRRH_VALUE;
00081         UBRRL = UBRRL_VALUE;
00082         #if USE_2X
00083             UCSRA |= (1 << U2X);
00084         #else
00085             UCSRA &= ~(1 << U2X);
00086         #endif
00087     }
00088
00089     static void
00090     uart_38400(void)
00091     {
00092         #undef BAUD // avoid compiler warning
00093         #define BAUD 38400
00094         #include <util/setbaud.h>
00095         UBRRH = UBRRH_VALUE;
00096         UBRRL = UBRRL_VALUE;
00097         #if USE_2X
00098             UCSRA |= (1 << U2X);
00099         #else
00100             UCSRA &= ~(1 << U2X);
00101         #endif
00102     }
00103     \endcode
00104
00105     In this example, two functions are defined to setup the UART
00106     to run at 9600 Bd, and 38400 Bd, respectively. Using a CPU
00107     clock of 4 MHz, 9600 Bd can be achieved with an acceptable
00108     tolerance without setting U2X (prescaler 25), while 38400 Bd
00109     require U2X to be set (prescaler 12).
00110 */
00111
00112 #ifndef F_CPU
00113 # error "setbaud.h requires F_CPU to be defined"
00114 #endif
00115
00116 #ifndef BAUD
00117 # error "setbaud.h requires BAUD to be defined"
00118 #endif
00119
00120 #if !(F_CPU)
00121 # error "F_CPU must be a constant value"
00122 #endif
00123
00124 #if !(BAUD)
00125 # error "BAUD must be a constant value"
00126 #endif
00127
00128 #if defined(__DOXYGEN__)
00129 /**
00130     \def BAUD_TOL
00131     \ingroup util_setbaud
00132
00133     Input and output macro for <util/setbaud.h>
00134
00135     Define the acceptable baud rate tolerance in percent. If not set
00136     on entry, it will be set to its default value of 2.
00137 */
00138 #define BAUD_TOL 2

```

```

00139
00140 /**
00141     \def UBRR_VALUE
00142     \ingroup util_setbaud
00143
00144     Output macro from <util/setbaud.h>
00145
00146     Contains the calculated baud rate prescaler value for the UBRR
00147     register.
00148 */
00149 #define UBRR_VALUE
00150
00151 /**
00152     \def UBRRL_VALUE
00153     \ingroup util_setbaud
00154
00155     Output macro from <util/setbaud.h>
00156
00157     Contains the lower byte of the calculated prescaler value
00158     (UBRR_VALUE).
00159 */
00160 #define UBRRL_VALUE
00161
00162 /**
00163     \def UBRRH_VALUE
00164     \ingroup util_setbaud
00165
00166     Output macro from <util/setbaud.h>
00167
00168     Contains the upper byte of the calculated prescaler value
00169     (UBRR_VALUE).
00170 */
00171 #define UBRRH_VALUE
00172
00173 /**
00174     \def USE_2X
00175     \ingroup util_setbaud
00176
00177     Output macro from <util/setbaud.h>
00178
00179     Contains the value 1 if the desired baud rate tolerance could only
00180     be achieved by setting the U2X bit in the UART configuration.
00181     Contains 0 otherwise.
00182 */
00183 #define USE_2X 0
00184
00185 #else /* !__DOXYGEN__ */
00186
00187 #undef USE_2X
00188
00189 /* Baud rate tolerance is 2 % unless previously defined */
00190 #ifndef BAUD_TOL
00191 #   define BAUD_TOL 2
00192 #endif
00193
00194 #ifdef __ASSEMBLER__
00195 #define UBRR_VALUE (((F_CPU) + 8 * (BAUD)) / (16 * (BAUD)) -1)
00196 #else
00197 #define UBRR_VALUE (((F_CPU) + 8UL * (BAUD)) / (16UL * (BAUD)) -1UL)
00198 #endif
00199
00200 #if 100 * (F_CPU) > \
00201     (16 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) + (BAUD) * (BAUD_TOL))
00202 #   define USE_2X 1
00203 #elif 100 * (F_CPU) < \
00204     (16 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) - (BAUD) * (BAUD_TOL))
00205 #   define USE_2X 1
00206 #else
00207 #   define USE_2X 0
00208 #endif
00209
00210 #if USE_2X
00211 /* U2X required, recalculate */
00212 #undef UBRR_VALUE
00213
00214 #ifdef __ASSEMBLER__
00215 #define UBRR_VALUE (((F_CPU) + 4 * (BAUD)) / (8 * (BAUD)) -1)
00216 #else
00217 #define UBRR_VALUE (((F_CPU) + 4UL * (BAUD)) / (8UL * (BAUD)) -1UL)
00218 #endif
00219
00220 #if 100 * (F_CPU) > \
00221     (8 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) + (BAUD) * (BAUD_TOL))
00222 #   warning "Baud rate achieved is higher than allowed"
00223 #endif
00224
00225 #if 100 * (F_CPU) < \

```

```

00226 (8 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) - (BAUD) * (BAUD_TOL))
00227 # warning "Baud rate achieved is lower than allowed"
00228 #endif
00229
00230 #endif /* USE_U2X */
00231
00232 #ifndef UBRR_VALUE
00233     /* Check for overflow */
00234 # if UBRR_VALUE >= (1 << 12)
00235 #     warning "UBRR value overflow"
00236 # endif
00237
00238 # define UBRRRL_VALUE (UBRR_VALUE & 0xff)
00239 # define UBRRRH_VALUE (UBRR_VALUE >> 8)
00240 #endif
00241
00242 #endif /* __DOXYGEN__ */
00243 /* end of util/setbaud.h */

```

## 23.76 compat/twi.h

```

00001 /* Copyright (c) 2005 Joerg Wunsch
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009
00010     * Redistributions in binary form must reproduce the above copyright
00011     notice, this list of conditions and the following disclaimer in
00012     the documentation and/or other materials provided with the
00013     distribution.
00014
00015     * Neither the name of the copyright holders nor the names of
00016     contributors may be used to endorse or promote products derived
00017     from this software without specific prior written permission.
00018
00019     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00022     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00023     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00024     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00025     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00026     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00028     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00029     POSSIBILITY OF SUCH DAMAGE. */
00030
00031 /* $Id$ */
00032
00033 #ifndef _COMPAT_TWI_H_
00034 #define _COMPAT_TWI_H_
00035
00036 #include <util/twi.h>
00037
00038 #endif /* _COMPAT_TWI_H_ */

```

## 23.77 twi.h File Reference

### Macros

#### TWSR values

*Mnemonics:*

*TW\_MT\_xxx - master transmitter*

*TW\_MR\_xxx - master receiver*

*TW\_ST\_xxx - slave transmitter*

*TW\_SR\_xxx - slave receiver*

- #define [TW\\_START](#) 0x08
- #define [TW\\_REP\\_START](#) 0x10
- #define [TW\\_MT\\_SLA\\_ACK](#) 0x18

- #define TW\_MT\_SLA\_NACK 0x20
- #define TW\_MT\_DATA\_ACK 0x28
- #define TW\_MT\_DATA\_NACK 0x30
- #define TW\_MT\_ARB\_LOST 0x38
- #define TW\_MR\_ARB\_LOST 0x38
- #define TW\_MR\_SLA\_ACK 0x40
- #define TW\_MR\_SLA\_NACK 0x48
- #define TW\_MR\_DATA\_ACK 0x50
- #define TW\_MR\_DATA\_NACK 0x58
- #define TW\_ST\_SLA\_ACK 0xA8
- #define TW\_ST\_ARB\_LOST\_SLA\_ACK 0xB0
- #define TW\_ST\_DATA\_ACK 0xB8
- #define TW\_ST\_DATA\_NACK 0xC0
- #define TW\_ST\_LAST\_DATA 0xC8
- #define TW\_SR\_SLA\_ACK 0x60
- #define TW\_SR\_ARB\_LOST\_SLA\_ACK 0x68
- #define TW\_SR\_GCALL\_ACK 0x70
- #define TW\_SR\_ARB\_LOST\_GCALL\_ACK 0x78
- #define TW\_SR\_DATA\_ACK 0x80
- #define TW\_SR\_DATA\_NACK 0x88
- #define TW\_SR\_GCALL\_DATA\_ACK 0x90
- #define TW\_SR\_GCALL\_DATA\_NACK 0x98
- #define TW\_SR\_STOP 0xA0
- #define TW\_NO\_INFO 0xF8
- #define TW\_BUS\_ERROR 0x00
- #define TW\_STATUS\_MASK
- #define TW\_STATUS (TWSR & TW\_STATUS\_MASK)

#### R/~W bit in SLA+R/W address field.

- #define TW\_READ 1
- #define TW\_WRITE 0

## 23.78 util/twi.h

### Go to the documentation of this file.

```

00001 /* Copyright (c) 2002, Marek Michalkiewicz
00002 Copyright (c) 2005, 2007 Joerg Wunsch
00003 All rights reserved.
00004
00005 Redistribution and use in source and binary forms, with or without
00006 modification, are permitted provided that the following conditions are met:
00007
00008 * Redistributions of source code must retain the above copyright
00009 notice, this list of conditions and the following disclaimer.
00010
00011 * Redistributions in binary form must reproduce the above copyright
00012 notice, this list of conditions and the following disclaimer in
00013 the documentation and/or other materials provided with the
00014 distribution.
00015
00016 * Neither the name of the copyright holders nor the names of
00017 contributors may be used to endorse or promote products derived
00018 from this software without specific prior written permission.
00019
00020 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00022 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00023 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00024 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00025 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00026 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00027 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00029 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00030 POSSIBILITY OF SUCH DAMAGE. */
00031
00032 /* $Id$ */
00033 /* copied from: Id: avr/twi.h,v 1.4 2004/11/01 21:19:54 arcanum Exp */
00034

```

```
00035 #ifndef _UTIL_TWI_H_
00036 #define _UTIL_TWI_H_ 1
00037
00038 #include <avr/io.h>
00039
00040 /** \file */
00041 /** \defgroup util_twi <util/twi.h>: TWI bit mask definitions
00042     \code #include <util/twi.h> \endcode
00043
00044     This header file contains bit mask definitions for use with
00045     the AVR TWI interface.
00046 */
00047 /** \name TWSR values
00048
00049     Mnemonics:
00050     <br>TW_MT_XXX - master transmitter
00051     <br>TW_MR_XXX - master receiver
00052     <br>TW_ST_XXX - slave transmitter
00053     <br>TW_SR_XXX - slave receiver
00054     */
00055
00056 /**@{*/
00057 /* Master */
00058 /** \ingroup util_twi
00059     \def TW_START
00060     start condition transmitted */
00061 #define TW_START      0x08
00062
00063 /** \ingroup util_twi
00064     \def TW_REP_START
00065     repeated start condition transmitted */
00066 #define TW_REP_START  0x10
00067
00068 /* Master Transmitter */
00069 /** \ingroup util_twi
00070     \def TW_MT_SLA_ACK
00071     SLA+W transmitted, ACK received */
00072 #define TW_MT_SLA_ACK 0x18
00073
00074 /** \ingroup util_twi
00075     \def TW_MT_SLA_NACK
00076     SLA+W transmitted, NACK received */
00077 #define TW_MT_SLA_NACK 0x20
00078
00079 /** \ingroup util_twi
00080     \def TW_MT_DATA_ACK
00081     data transmitted, ACK received */
00082 #define TW_MT_DATA_ACK 0x28
00083
00084 /** \ingroup util_twi
00085     \def TW_MT_DATA_NACK
00086     data transmitted, NACK received */
00087 #define TW_MT_DATA_NACK 0x30
00088
00089 /** \ingroup util_twi
00090     \def TW_MT_ARB_LOST
00091     arbitration lost in SLA+W or data */
00092 #define TW_MT_ARB_LOST 0x38
00093
00094 /* Master Receiver */
00095 /** \ingroup util_twi
00096     \def TW_MR_ARB_LOST
00097     arbitration lost in SLA+R or NACK */
00098 #define TW_MR_ARB_LOST 0x38
00099
00100 /** \ingroup util_twi
00101     \def TW_MR_SLA_ACK
00102     SLA+R transmitted, ACK received */
00103 #define TW_MR_SLA_ACK 0x40
00104
00105 /** \ingroup util_twi
00106     \def TW_MR_SLA_NACK
00107     SLA+R transmitted, NACK received */
00108 #define TW_MR_SLA_NACK 0x48
00109
00110 /** \ingroup util_twi
00111     \def TW_MR_DATA_ACK
00112     data received, ACK returned */
00113 #define TW_MR_DATA_ACK 0x50
00114
00115 /** \ingroup util_twi
00116     \def TW_MR_DATA_NACK
00117     data received, NACK returned */
00118 #define TW_MR_DATA_NACK 0x58
00119
00120 /* Slave Transmitter */
00121 /** \ingroup util_twi
```

```
00122     \def TW_ST_SLA_ACK
00123     SLA+R received, ACK returned */
00124 #define TW_ST_SLA_ACK      0xA8
00125
00126 /** \ingroup util_twi
00127     \def TW_ST_ARB_LOST_SLA_ACK
00128     arbitration lost in SLA+RW, SLA+R received, ACK returned */
00129 #define TW_ST_ARB_LOST_SLA_ACK  0xB0
00130
00131 /** \ingroup util_twi
00132     \def TW_ST_DATA_ACK
00133     data transmitted, ACK received */
00134 #define TW_ST_DATA_ACK      0xB8
00135
00136 /** \ingroup util_twi
00137     \def TW_ST_DATA_NACK
00138     data transmitted, NACK received */
00139 #define TW_ST_DATA_NACK     0xC0
00140
00141 /** \ingroup util_twi
00142     \def TW_ST_LAST_DATA
00143     last data byte transmitted, ACK received */
00144 #define TW_ST_LAST_DATA     0xC8
00145
00146 /* Slave Receiver */
00147 /** \ingroup util_twi
00148     \def TW_SR_SLA_ACK
00149     SLA+W received, ACK returned */
00150 #define TW_SR_SLA_ACK      0x60
00151
00152 /** \ingroup util_twi
00153     \def TW_SR_ARB_LOST_SLA_ACK
00154     arbitration lost in SLA+RW, SLA+W received, ACK returned */
00155 #define TW_SR_ARB_LOST_SLA_ACK  0x68
00156
00157 /** \ingroup util_twi
00158     \def TW_SR_GCALL_ACK
00159     general call received, ACK returned */
00160 #define TW_SR_GCALL_ACK     0x70
00161
00162 /** \ingroup util_twi
00163     \def TW_SR_ARB_LOST_GCALL_ACK
00164     arbitration lost in SLA+RW, general call received, ACK returned */
00165 #define TW_SR_ARB_LOST_GCALL_ACK 0x78
00166
00167 /** \ingroup util_twi
00168     \def TW_SR_DATA_ACK
00169     data received, ACK returned */
00170 #define TW_SR_DATA_ACK     0x80
00171
00172 /** \ingroup util_twi
00173     \def TW_SR_DATA_NACK
00174     data received, NACK returned */
00175 #define TW_SR_DATA_NACK    0x88
00176
00177 /** \ingroup util_twi
00178     \def TW_SR_GCALL_DATA_ACK
00179     general call data received, ACK returned */
00180 #define TW_SR_GCALL_DATA_ACK 0x90
00181
00182 /** \ingroup util_twi
00183     \def TW_SR_GCALL_DATA_NACK
00184     general call data received, NACK returned */
00185 #define TW_SR_GCALL_DATA_NACK 0x98
00186
00187 /** \ingroup util_twi
00188     \def TW_SR_STOP
00189     stop or repeated start condition received while selected */
00190 #define TW_SR_STOP        0xA0
00191
00192 /* Misc */
00193 /** \ingroup util_twi
00194     \def TW_NO_INFO
00195     no state information available */
00196 #define TW_NO_INFO        0xF8
00197
00198 /** \ingroup util_twi
00199     \def TW_BUS_ERROR
00200     illegal start or stop condition */
00201 #define TW_BUS_ERROR      0x00
00202
00203
00204 /**
00205  * \ingroup util_twi
00206  * \def TW_STATUS_MASK
00207  * The lower 3 bits of TWSR are reserved on the ATmega163.
00208  * The 2 LSB carry the prescaler bits on the newer ATmegas.
```

```

00209 */
00210 #define TW_STATUS_MASK      (_BV(TWS7) | _BV(TWS6) | _BV(TWS5) | _BV(TWS4) | \
00211                          _BV(TWS3))
00212 /**
00213  * \ingroup util_twi
00214  * \def TW_STATUS
00215  *
00216  * TWSR, masked by TW_STATUS_MASK
00217  */
00218 #define TW_STATUS          (TWSR & TW_STATUS_MASK)
00219 /**@{*/
00220
00221 /**
00222  * \name R/~W bit in SLA+R/W address field.
00223  */
00224
00225 /**@{*/
00226 /** \ingroup util_twi
00227     \def TW_READ
00228     SLA+R address */
00229 #define TW_READ          1
00230
00231 /** \ingroup util_twi
00232     \def TW_WRITE
00233     SLA+W address */
00234 #define TW_WRITE          0
00235 /**@}*/
00236
00237 #endif /* _UTIL_TWI_H_ */

```

## 23.79 usa\_dst.h File Reference

### Functions

- int `usa_dst` (const `time_t` \*timer, `int32_t` \*z)

## 23.80 usa\_dst.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * (c)2012 Michael Duane Rice All rights reserved.
00003  *
00004  * Redistribution and use in source and binary forms, with or without
00005  * modification, are permitted provided that the following conditions are
00006  * met:
00007  *
00008  * Redistributions of source code must retain the above copyright notice, this
00009  * list of conditions and the following disclaimer. Redistributions in binary
00010  * form must reproduce the above copyright notice, this list of conditions
00011  * and the following disclaimer in the documentation and/or other materials
00012  * provided with the distribution. Neither the name of the copyright holders
00013  * nor the names of contributors may be used to endorse or promote products
00014  * derived from this software without specific prior written permission.
00015  *
00016  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00017  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00018  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00019  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00020  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00021  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00022  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00023  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00024  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00025  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00026  * POSSIBILITY OF SUCH DAMAGE.
00027  */
00028
00029 /* $Id$ */
00030
00031 #ifndef USA_DST_H
00032 #define USA_DST_H
00033
00034 #ifdef __cplusplus
00035 extern "C" {
00036 #endif
00037

```

```

00038 #include <time.h>
00039 #include <stdint.h>
00040
00041 /** \file */
00042 /** \defgroup usa_dst <util/usa_dst.h>: Daylight Saving function for the USA.
00043     \code #include <util/usa_dst.h> \endcode
00044     Daylight Saving function for the USA. */
00045
00046 /** \ingroup usa_dst
00047     \fn int usa_dst (const time_t *timer, int32_t *z)
00048     To utilize this function, call
00049     \code set_dst(usa_dst); \endcode
00050
00051     Given the time stamp and time zone parameters provided, the Daylight
00052     Saving function must return a value appropriate for the tm structures'
00053     tm_isdst element. That is:
00054
00055     - \c 0 : If Daylight Saving is not in effect.
00056
00057     - \c -1 : If it cannot be determined if Daylight Saving is in effect.
00058
00059     - A positive integer : Represents the number of seconds a clock is
00060     advanced for Daylight Saving. This will typically be ONE_HOUR.
00061
00062     Daylight Saving 'rules' are subject to frequent change. For production
00063     applications it is recommended to write your own DST function, which
00064     uses 'rules' obtained from, and modifiable by, the end user
00065     (perhaps stored in EEPROM).
00066 */
00067 int usa_dst (const time_t *timer, int32_t *z);
00068
00069 #ifdef __cplusplus
00070 }
00071 #endif
00072
00073 #endif

```

## 23.81 eedef.h

```

00001 /* Copyright (c) 2009 Dmitry Xmelkov
00002     All rights reserved.
00003
00004     Redistribution and use in source and binary forms, with or without
00005     modification, are permitted provided that the following conditions are met:
00006
00007     * Redistributions of source code must retain the above copyright
00008     notice, this list of conditions and the following disclaimer.
00009     * Redistributions in binary form must reproduce the above copyright
00010     notice, this list of conditions and the following disclaimer in
00011     the documentation and/or other materials provided with the
00012     distribution.
00013     * Neither the name of the copyright holders nor the names of
00014     contributors may be used to endorse or promote products derived
00015     from this software without specific prior written permission.
00016
00017     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00018     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00019     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00020     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00021     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00022     CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00023     SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00024     INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00026     ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00027     POSSIBILITY OF SUCH DAMAGE. */
00028
00029 /* $Id$ */
00030
00031 #ifndef _EDEF_H_
00032 #define _EDEF_H_ 1
00033
00034 #ifndef __DOXYGEN__
00035
00036 /* EEPROM address arg for a set of byte/word/dword functions and for
00037     the internal eeprom_read_blraw(). */
00038 #define addr_lo r24
00039 #define addr_hi r25
00040
00041 /* Number of bytes arg for all block read/write functions, include
00042     internal. */
00043 #define n_lo r20
00044 #define n_hi r21
00045

```



```

00046 #if __AVR_XMEGA__
00047
00048 # define NVM_BASE    NVM_ADDR0
00049
00050 #if defined(NVMCTRL_CTRLA)
00051 # undef NVM_BASE
00052 # define NVM_BASE      NVMCTRL_CTRLA
00053
00054 # define NVM_ADDR0      NVMCTRL_ADDR0
00055 # define NVM_ADDR1      NVMCTRL_ADDR1
00056 # define NVM_DATA0      NVMCTRL_DATA0
00057 # define NVM_DATA1      NVMCTRL_DATA1
00058 # define NVM_NVMBUSY_bp NVMCTRL_EEBUSY_bp
00059 # define NVM_STATUS      NVMCTRL_STATUS
00060 # define NVM_CTRLA      NVMCTRL_CTRLA
00061 # define NVM_CTRLB      NVMCTRL_CTRLB
00062 # ifndef CCP_SPM_gc
00063 #   define CCP_SPM_gc    (0x9D)
00064 # endif
00065 # ifndef NVMCTRL_CMD_PAGEERASEWRITE_gc
00066 #   if NVMCTRL_CMD_gm == 0x7F
00067 #     if defined (__AVR_AVR16EA28__) || defined (__AVR_AVR16EA32__) || defined (__AVR_AVR16EA48__) ||
00068 #       defined (__AVR_AVR16EB14__) || defined (__AVR_AVR16EB20__) || defined (__AVR_AVR16EB28__) ||
00069 #       defined (__AVR_AVR16EB32__) || defined (__AVR_AVR32EA28__) || defined (__AVR_AVR32EA32__) ||
00070 #       defined (__AVR_AVR32EA48__) || defined (__AVR_AVR64EA28__) || defined (__AVR_AVR64EA32__) ||
00071 #       defined (__AVR_AVR64EA48__)
00072 #     /* AVR-Ex family
00073 #     * value of NVMCTRL_CMD_enum.NVMCTRL_CMD_EEPERW_gc */
00074 #     define NVMCTRL_CMD_PAGEERASEWRITE_gc (0x15<<0)
00075 #     elif defined (__AVR_AVR32DA28__) || defined (__AVR_AVR32DA32__) || defined (__AVR_AVR32DA48__)
00076 #     || \
00077 #       defined (__AVR_AVR64DA28__) || defined (__AVR_AVR64DA32__) || defined (__AVR_AVR64DA48__)
00078 #     || \
00079 #       defined (__AVR_AVR128DA32__) || \
00080 #       defined (__AVR_AVR128DA48__) || defined (__AVR_AVR128DA64__) || defined
00081 #       (__AVR_AVR32DB28__) || \
00082 #       defined (__AVR_AVR32DB32__) || defined (__AVR_AVR32DB48__) || defined (__AVR_AVR64DB28__)
00083 #     || \
00084 #       defined (__AVR_AVR64DB32__) || defined (__AVR_AVR64DB48__) || defined (__AVR_AVR64DB64__)
00085 #     || \
00086 #       defined (__AVR_AVR128DB28__) || defined (__AVR_AVR128DB32__) || defined
00087 #       (__AVR_AVR128DB48__) || \
00088 #       defined (__AVR_AVR128DB64__) || defined (__AVR_AVR16DD14__) || defined (__AVR_AVR16DD20__)
00089 #     || \
00090 #       defined (__AVR_AVR16DD28__) || defined (__AVR_AVR16DD32__) || defined (__AVR_AVR32DD14__)
00091 #     || \
00092 #       defined (__AVR_AVR32DD20__) || defined (__AVR_AVR32DD32__) || defined (__AVR_AVR32DD28__)
00093 #     || \
00094 #       defined (__AVR_AVR64DD14__) || defined (__AVR_AVR64DD20__) || defined (__AVR_AVR64DD28__)
00095 #     || \
00096 #       defined (__AVR_AVR64DD32__) \
00097 #       || defined (__AVR_AVR64DU28__) || defined (__AVR_AVR64DU32__)
00098 #     /* AVR-Dx family
00099 #     * value of NVMCTRL_CMD_enum.NVMCTRL_CMD_EEERWR_gc */
00100 #     define NVMCTRL_CMD_PAGEERASEWRITE_gc (0x13<<0)
00101 #     else
00102 #     /* To add support for a new device, define NVMCTRL_CMD_PAGEERASEWRITE_gc with the value
00103 #     * of "Erase and Write EEPROM Page" comand code for - Persistent Memory Controller
00104 #     (NVMCTRL).*/
00105 #     error "Not supported devices"
00106 #     endif
00107 #   else
00108 #     /* the rest of the AVR devices with NVMCTRL_CTRLA (0x07)
00109 #     * value of NVMCTRL_CMD_enum.NVMCTRL_CMD_PAGEERASEWRITE_gc */
00110 #     define NVMCTRL_CMD_PAGEERASEWRITE_gc 3
00111 #     endif
00112 #   endif /* NVMCTRL_CMD_PAGEERASEWRITE_gc */
00113 #endif /* defined(NVMCTRL_CTRLA) */
00114 #else
00115 # if !defined (EECR) && defined (DEECR) /* AT86RF401 */
00116 #   define EECR DEECR
00117 #   define EEARL DEEAR
00118 #   define EEDR DEEDR
00119 # endif
00120
00121 # if !defined (EERE) && defined (EER) /* AT86RF401 */
00122 #   define EERE EER
00123 # endif
00124
00125 # if !defined (EWE) && defined (EPE) /* A part of Mega and Tiny */
00126 #   define EWE EPE

```

```

00117 # endif
00118 # if !defined (EWE) && defined (EEL) /* AT86RF401 */
00119 # define EWE EEL
00120 # endif
00121
00122 # if !defined (EEMWE) && defined (EEMPE) /* A part of Mega and Tiny */
00123 # define EEMWE EEMPE
00124 # endif
00125 # if !defined (EEMWE) && defined (EEU) /* AT86RF401 */
00126 # define EEMWE EEU
00127 # endif
00128
00129 # if !_SFR_IO_REG_P (EECR) \
00130 || !_SFR_IO_REG_P (EEDR) \
00131 || !_SFR_IO_REG_P (EEARL) \
00132 || (defined (EEARH) && !_SFR_IO_REG_P (EEARH))
00133 # error
00134 # endif
00135
00136 #endif /* !__AVR_XMEGA__ */
00137 #endif /* !__DOXYGEN__ */
00138 #endif /* !_EEDF_H_ */

```

## 23.82 fdevopen.c File Reference

### Functions

- [FILE \\*](#) `fdevopen` (`int(*put)(char, FILE *)`, `int(*get)(FILE *)`)

## 23.83 stdio\_private.h

```

00001 /* Copyright (c) 2002,2005, Joerg Wunsch
00002 All rights reserved.
00003
00004 Redistribution and use in source and binary forms, with or without
00005 modification, are permitted provided that the following conditions are met:
00006
00007 * Redistributions of source code must retain the above copyright
00008 notice, this list of conditions and the following disclaimer.
00009 * Redistributions in binary form must reproduce the above copyright
00010 notice, this list of conditions and the following disclaimer in
00011 the documentation and/or other materials provided with the
00012 distribution.
00013 * Neither the name of the copyright holders nor the names of
00014 contributors may be used to endorse or promote products derived
00015 from this software without specific prior written permission.
00016
00017 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00018 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00019 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00020 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00021 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00022 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00023 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00024 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00026 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00027 POSSIBILITY OF SUCH DAMAGE.
00028 */
00029
00030 /* $Id$ */
00031
00032 #include <stdint.h>
00033 #include <stdio.h>
00034
00035 /* values for PRINTF_LEVEL */
00036 #define PRINTF_MIN 1
00037 #define PRINTF_STD 2
00038 #define PRINTF_FLT 3
00039
00040 /* values for SCANF_LEVEL */
00041 #define SCANF_MIN 1
00042 #define SCANF_STD 2
00043 #define SCANF_FLT 3

```

## 23.84 xtoa\_fast.h

```

00001 /* Copyright (c) 2005, Dmitry Xmelkov
00002    All rights reserved.
00003
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions are met:
00006
00007    * Redistributions of source code must retain the above copyright
00008    notice, this list of conditions and the following disclaimer.
00009    * Redistributions in binary form must reproduce the above copyright
00010    notice, this list of conditions and the following disclaimer in
00011    the documentation and/or other materials provided with the
00012    distribution.
00013    * Neither the name of the copyright holders nor the names of
00014    contributors may be used to endorse or promote products derived
00015    from this software without specific prior written permission.
00016
00017    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00018    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00019    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00020    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00021    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00022    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00023    SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00024    INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00026    ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00027    POSSIBILITY OF SUCH DAMAGE. */
00028
00029 /* $Id$ */
00030
00031 #ifndef _XTOA_FAST_H_
00032 #define _XTOA_FAST_H_
00033
00034 #ifndef __ASSEMBLER__
00035
00036 #include <stddef.h> /* for 'size_t' */
00037
00038 char * itoa_fast (int val, char *s, int base);
00039 char * utoa_fast (unsigned val, char *s, int base);
00040 char * ltoa_fast (long val, char *s, int base);
00041 char * ultoa_fast (unsigned long val, char *s, int base);
00042
00043 char * itoa_width (int val, char *s, int base, size_t width);
00044 char * utoa_width (unsigned val, char *s, int base, size_t width);
00045 char * ltoa_width (long val, char *s, int base, size_t width);
00046 char * ultoa_width (unsigned long val, char *s, int base, size_t width);
00047
00048 /* Internal function for use from 'printf'. */
00049 char * __ultoa_invert (unsigned long val, char *s, int base);
00050
00051 #endif /* ifndef __ASSEMBLER__ */
00052
00053 /* Next flags are to use with 'base'. Unused fields are reserved. */
00054 #define XTOA_PREFIX 0x0100 /* put prefix for octal or hex */
00055 #define XTOA_UPPER 0x0200 /* use upper case letters */
00056
00057 #endif /* _XTOA_FAST_H_ */

```

## 23.85 dtoa\_conv.h

```

00001 /* Copyright (c) 2005, Dmitry Xmelkov
00002    All rights reserved.
00003
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions are met:
00006
00007    * Redistributions of source code must retain the above copyright
00008    notice, this list of conditions and the following disclaimer.
00009    * Redistributions in binary form must reproduce the above copyright
00010    notice, this list of conditions and the following disclaimer in
00011    the documentation and/or other materials provided with the
00012    distribution.
00013    * Neither the name of the copyright holders nor the names of
00014    contributors may be used to endorse or promote products derived
00015    from this software without specific prior written permission.
00016
00017    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00018    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00019    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00020    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00021    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00022    CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF

```

```

00023  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00024  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00026  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00027  POSSIBILITY OF SUCH DAMAGE. */
00028
00029  /* $Id$ */
00030
00031  #ifndef _DTOA_CONV_H
00032  #define _DTOA_CONV_H
00033
00034  #include <stdio.h>
00035
00036  int ftoa_prf (float val, char *s, unsigned char width, unsigned char prec,
00037              unsigned char flags);
00038
00039  #define DTOA_SPACE 0x01 /* put space for positives */
00040  #define DTOA_PLUS 0x02 /* put '+' for positives */
00041  #define DTOA_UPPER 0x04 /* use uppercase letters */
00042  #define DTOA_ZFILL 0x08 /* fill zeroes */
00043  #define DTOA_LEFT 0x10 /* adjust to left */
00044  #define DTOA_NOFILL 0x20 /* do not fill to width */
00045  #define DTOA_EXP 0x40 /* d2stream: 'e(E)' format */
00046  #define DTOA_FIX 0x80 /* d2stream: 'f(F)' format */
00047
00048  #define DTOA_EWIDTH (-1) /* Width too small */
00049  #define DTOA_NONFINITE (-2) /* Value is NaN or Inf */
00050
00051  #endif /* !_DTOA_CONV_H */

```

## 23.86 stdlib\_private.h

```

00001  /* Copyright (c) 2004, Joerg Wunsch
00002  All rights reserved.
00003
00004  Redistribution and use in source and binary forms, with or without
00005  modification, are permitted provided that the following conditions are met:
00006
00007  * Redistributions of source code must retain the above copyright
00008  notice, this list of conditions and the following disclaimer.
00009  * Redistributions in binary form must reproduce the above copyright
00010  notice, this list of conditions and the following disclaimer in
00011  the documentation and/or other materials provided with the
00012  distribution.
00013  * Neither the name of the copyright holders nor the names of
00014  contributors may be used to endorse or promote products derived
00015  from this software without specific prior written permission.
00016
00017  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00018  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00019  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00020  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00021  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00022  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00023  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00024  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00026  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00027  POSSIBILITY OF SUCH DAMAGE.
00028  */
00029
00030  /* $Id$ */
00031
00032  #include <inttypes.h>
00033  #include <stdlib.h>
00034  #include <avr/io.h>
00035
00036  #if !defined(__DOXYGEN__)
00037
00038  struct __freelist {
00039      size_t sz;
00040      struct __freelist *nx;
00041  };
00042
00043  #endif
00044
00045  extern char *__brkval; /* first location not yet allocated */
00046  extern struct __freelist *__flp; /* freelist pointer (head of freelist) */
00047  extern size_t __malloc_margin; /* user-changeable before the first malloc() */
00048  extern char *__malloc_heap_start;
00049  extern char *__malloc_heap_end;
00050
00051  #ifndef __AVR__
00052

```

```

00053 /*
00054  * When compiling malloc.c/realloc.c natively on a host machine, it will
00055  * include a main() that performs a regression test. This is meant as
00056  * a debugging aid, where a normal source-level debugger will help to
00057  * verify that the various allocator structures have the desired
00058  * appearance at each stage.
00059  *
00060  * When cross-compiling with avr-gcc, it will compile into just the
00061  * library functions malloc() and free().
00062  */
00063 #define MALLOC_TEST
00064
00065 #endif /* !__AVR__ */
00066
00067 #ifndef MALLOC_TEST
00068
00069 extern void *mymalloc(size_t);
00070 extern void myfree(void *);
00071 extern void *myrealloc(void *, size_t);
00072
00073 #define malloc mymalloc
00074 #define free myfree
00075 #define realloc myrealloc
00076
00077 #define __heap_start mymem[0]
00078 #define __heap_end mymem[256]
00079 extern char mymem[];
00080 #define STACK_POINTER() (mymem + 256)
00081
00082 #else /* !MALLOC_TEST */
00083
00084 extern char __heap_start;
00085 extern char __heap_end;
00086
00087 /* Needed for definition of AVR_STACK_POINTER_REG. */
00088 #include <avr/io.h>
00089
00090 #define STACK_POINTER() ((char *)AVR_STACK_POINTER_REG)
00091
00092 #endif /* MALLOC_TEST */

```

## 23.87 ephemer\_common.h

```

00001 /*
00002  * (C)2012 Michael Duane Rice All rights reserved.
00003  *
00004  * Redistribution and use in source and binary forms, with or without
00005  * modification, are permitted provided that the following conditions are
00006  * met:
00007  *
00008  * Redistributions of source code must retain the above copyright notice, this
00009  * list of conditions and the following disclaimer. Redistributions in binary
00010  * form must reproduce the above copyright notice, this list of conditions
00011  * and the following disclaimer in the documentation and/or other materials
00012  * provided with the distribution. Neither the name of the copyright holders
00013  * nor the names of contributors may be used to endorse or promote products
00014  * derived from this software without specific prior written permission.
00015  *
00016  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00017  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00018  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00019  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
00020  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00021  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00022  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00023  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00024  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00025  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00026  * POSSIBILITY OF SUCH DAMAGE.
00027  */
00028
00029 /* $Id$ */
00030
00031 #ifndef EPHEMERA_PRIVATE_H
00032 #define EPHEMERA_PRIVATE_H
00033
00034 #define TROP_YEAR 31556925
00035 #define ANOM_YEAR 31558433
00036 #define INCLINATION 0.409105176667471 /* Earths axial tilt at the epoch */
00037 #define PERIHELION 31316400 /* perihelion of 1999, 03 jan 13:00 UTC */
00038 #define SOLSTICE 836160 /* winter solstice of 1999, 22 Dec 07:44 UTC */
00039 #define TWO_PI 6.283185307179586
00040 #define TROP_CYCLE 5022440.6025
00041 #define ANOM_CYCLE 5022680.6082

```

```
00042 #define DELTA_V 0.03342044 /* 2x orbital eccentricity */
00043
00044 #endif
```



## Index

- <alloca.h>: Allocate space in the stack, [103](#)
  - [alloca](#), [103](#)
- <assert.h>: Diagnostics, [104](#)
  - [assert](#), [104](#)
- <avr/boot.h>: Bootloader Support Utilities, [207](#)
  - [boot\\_is\\_spm\\_interrupt](#), [208](#)
  - [boot\\_lock\\_bits\\_set](#), [208](#)
  - [boot\\_lock\\_bits\\_set\\_safe](#), [209](#)
  - [boot\\_lock\\_fuse\\_bits\\_get](#), [209](#)
  - [boot\\_page\\_erase](#), [209](#)
  - [boot\\_page\\_erase\\_safe](#), [209](#)
  - [boot\\_page\\_fill](#), [210](#)
  - [boot\\_page\\_fill\\_safe](#), [210](#)
  - [boot\\_page\\_write](#), [210](#)
  - [boot\\_page\\_write\\_safe](#), [210](#)
  - [boot\\_rww\\_busy](#), [211](#)
  - [boot\\_rww\\_enable](#), [211](#)
  - [boot\\_rww\\_enable\\_safe](#), [211](#)
  - [boot\\_signature\\_byte\\_get](#), [211](#)
  - [boot\\_spm\\_busy](#), [211](#)
  - [boot\\_spm\\_busy\\_wait](#), [212](#)
  - [boot\\_spm\\_interrupt\\_disable](#), [212](#)
  - [boot\\_spm\\_interrupt\\_enable](#), [212](#)
  - [BOOTLOADER\\_SECTION](#), [212](#)
  - [GET\\_EXTENDED\\_FUSE\\_BITS](#), [212](#)
  - [GET\\_HIGH\\_FUSE\\_BITS](#), [212](#)
  - [GET\\_LOCK\\_BITS](#), [212](#)
  - [GET\\_LOW\\_FUSE\\_BITS](#), [212](#)
- <avr/builtins.h>: avr-gcc builtins documentation, [269](#)
  - [\\_\\_builtin\\_avr\\_cli](#), [269](#)
  - [\\_\\_builtin\\_avr\\_fmul](#), [270](#)
  - [\\_\\_builtin\\_avr\\_fmuls](#), [270](#)
  - [\\_\\_builtin\\_avr\\_fmulsu](#), [270](#)
  - [\\_\\_builtin\\_avr\\_sei](#), [270](#)
  - [\\_\\_builtin\\_avr\\_sleep](#), [270](#)
  - [\\_\\_builtin\\_avr\\_swap](#), [270](#)
  - [\\_\\_builtin\\_avr\\_wdr](#), [270](#)
- <avr/cpufunc.h>: Special AVR CPU functions, [212](#)
  - [\\_MemoryBarrier](#), [213](#)
  - [\\_NOP](#), [213](#)
  - [ccp\\_write\\_io](#), [213](#)
  - [ccp\\_write\\_spm](#), [213](#)
- <avr/eeprom.h>: EEPROM handling, [214](#)
  - [\\_EETGET](#), [215](#)
  - [\\_EETPUT](#), [215](#)
  - [\\_\\_EETGET](#), [215](#)
  - [\\_\\_EETPUT](#), [215](#)
  - [EEMEM](#), [216](#)
  - [eeprom\\_busy\\_wait](#), [216](#)
  - [eeprom\\_is\\_ready](#), [216](#)
  - [eeprom\\_read\\_block](#), [216](#)
  - [eeprom\\_read\\_byte](#), [216](#)
  - [eeprom\\_read\\_double](#), [216](#)
  - [eeprom\\_read\\_dword](#), [216](#)
  - [eeprom\\_read\\_float](#), [217](#)
  - [eeprom\\_read\\_long\\_double](#), [217](#)
  - [eeprom\\_read\\_qword](#), [217](#)
  - [eeprom\\_read\\_word](#), [217](#)
  - [eeprom\\_update\\_block](#), [217](#)
  - [eeprom\\_update\\_byte](#), [217](#)
  - [eeprom\\_update\\_double](#), [217](#)
  - [eeprom\\_update\\_dword](#), [217](#)
  - [eeprom\\_update\\_float](#), [218](#)
  - [eeprom\\_update\\_long\\_double](#), [218](#)
  - [eeprom\\_update\\_qword](#), [218](#)
  - [eeprom\\_update\\_word](#), [218](#)
  - [eeprom\\_write\\_block](#), [218](#)
  - [eeprom\\_write\\_byte](#), [218](#)
  - [eeprom\\_write\\_double](#), [218](#)
  - [eeprom\\_write\\_dword](#), [219](#)
  - [eeprom\\_write\\_float](#), [219](#)
  - [eeprom\\_write\\_long\\_double](#), [219](#)
  - [eeprom\\_write\\_qword](#), [219](#)
  - [eeprom\\_write\\_word](#), [219](#)
- <avr/fuse.h>: Fuse Support, [219](#)
- <avr/interrupt.h>: Interrupts, [222](#)
  - [BADISR\\_vect](#), [225](#)
  - [cli](#), [225](#)
  - [EMPTY\\_INTERRUPT](#), [226](#)
  - [ISR](#), [226](#)
  - [ISR\\_ALIAS](#), [226](#)
  - [ISR\\_ALIASOF](#), [226](#)
  - [ISR\\_BLOCK](#), [227](#)
  - [ISR\\_FLATTEN](#), [227](#)
  - [ISR\\_NAKED](#), [227](#)
  - [ISR\\_NOBLOCK](#), [227](#)
  - [ISR\\_NOGCCISR](#), [227](#)
  - [ISR\\_NOICF](#), [228](#)
  - [reti](#), [228](#)
  - [sei](#), [228](#)
  - [SIGNAL](#), [228](#)
- <avr/io.h>: AVR device-specific IO definitions, [228](#)
  - [\\_PROTECTED\\_WRITE](#), [229](#)
  - [\\_PROTECTED\\_WRITE\\_SPM](#), [229](#)
- <avr/lock.h>: Lockbit Support, [230](#)
- <avr/pgmspace.h>: Program Space Utilities, [232](#)
  - [memccpy\\_P](#), [237](#)
  - [memchr\\_P](#), [237](#)
  - [memcmp\\_P](#), [238](#)
  - [memcmp\\_PF](#), [238](#)
  - [memcpy\\_P](#), [238](#)
  - [memcpy\\_PF](#), [239](#)
  - [memmem\\_P](#), [239](#)
  - [memrchr\\_P](#), [239](#)
  - [pgm\\_get\\_far\\_address](#), [234](#)
  - [pgm\\_read< T >](#), [240](#)
  - [pgm\\_read\\_byte](#), [235](#)
  - [pgm\\_read\\_byte\\_far](#), [235](#)
  - [pgm\\_read\\_byte\\_near](#), [235](#)
  - [pgm\\_read\\_char](#), [240](#)



- [pgm\\_read\\_char\\_far](#), 240
- [pgm\\_read\\_double](#), 240
- [pgm\\_read\\_double\\_far](#), 240
- [pgm\\_read\\_dword](#), 235
- [pgm\\_read\\_dword\\_far](#), 236
- [pgm\\_read\\_dword\\_near](#), 236
- [pgm\\_read\\_far< T >](#), 240
- [pgm\\_read\\_float](#), 240
- [pgm\\_read\\_float\\_far](#), 240
- [pgm\\_read\\_float\\_near](#), 236
- [pgm\\_read\\_i16](#), 241
- [pgm\\_read\\_i16\\_far](#), 241
- [pgm\\_read\\_i24](#), 241
- [pgm\\_read\\_i24\\_far](#), 241
- [pgm\\_read\\_i32](#), 241
- [pgm\\_read\\_i32\\_far](#), 241
- [pgm\\_read\\_i64](#), 241
- [pgm\\_read\\_i64\\_far](#), 241
- [pgm\\_read\\_i8](#), 241
- [pgm\\_read\\_i8\\_far](#), 242
- [pgm\\_read\\_int](#), 242
- [pgm\\_read\\_int\\_far](#), 242
- [pgm\\_read\\_long](#), 242
- [pgm\\_read\\_long\\_double](#), 242
- [pgm\\_read\\_long\\_double\\_far](#), 242
- [pgm\\_read\\_long\\_far](#), 242
- [pgm\\_read\\_long\\_long](#), 242
- [pgm\\_read\\_long\\_long\\_far](#), 242
- [pgm\\_read\\_ptr](#), 236
- [pgm\\_read\\_ptr\\_far](#), 236
- [pgm\\_read\\_ptr\\_near](#), 236
- [pgm\\_read\\_qword](#), 236
- [pgm\\_read\\_qword\\_far](#), 236
- [pgm\\_read\\_qword\\_near](#), 236
- [pgm\\_read\\_short](#), 243
- [pgm\\_read\\_short\\_far](#), 243
- [pgm\\_read\\_signed](#), 243
- [pgm\\_read\\_signed\\_char](#), 243
- [pgm\\_read\\_signed\\_char\\_far](#), 243
- [pgm\\_read\\_signed\\_far](#), 243
- [pgm\\_read\\_signed\\_int](#), 243
- [pgm\\_read\\_signed\\_int\\_far](#), 243
- [pgm\\_read\\_u16](#), 243
- [pgm\\_read\\_u16\\_far](#), 244
- [pgm\\_read\\_u24](#), 244
- [pgm\\_read\\_u24\\_far](#), 244
- [pgm\\_read\\_u32](#), 244
- [pgm\\_read\\_u32\\_far](#), 244
- [pgm\\_read\\_u64](#), 244
- [pgm\\_read\\_u64\\_far](#), 244
- [pgm\\_read\\_u8](#), 244
- [pgm\\_read\\_u8\\_far](#), 244
- [pgm\\_read\\_unsigned](#), 245
- [pgm\\_read\\_unsigned\\_char](#), 245
- [pgm\\_read\\_unsigned\\_char\\_far](#), 245
- [pgm\\_read\\_unsigned\\_far](#), 245
- [pgm\\_read\\_unsigned\\_int](#), 245
- [pgm\\_read\\_unsigned\\_int\\_far](#), 245
- [pgm\\_read\\_unsigned\\_long](#), 245
- [pgm\\_read\\_unsigned\\_long\\_far](#), 245
- [pgm\\_read\\_unsigned\\_long\\_long](#), 245
- [pgm\\_read\\_unsigned\\_long\\_long\\_far](#), 246
- [pgm\\_read\\_unsigned\\_short](#), 246
- [pgm\\_read\\_unsigned\\_short\\_far](#), 246
- [pgm\\_read\\_word](#), 236
- [pgm\\_read\\_word\\_far](#), 237
- [pgm\\_read\\_word\\_near](#), 237
- [PROGMEM](#), 237
- [PROGMEM\\_FAR](#), 237
- [PSTR](#), 237
- [PSTR\\_FAR](#), 237
- [strcasecmp\\_P](#), 246
- [strcasecmp\\_PF](#), 246
- [strcasestr\\_P](#), 247
- [strcat\\_P](#), 247
- [strcat\\_PF](#), 247
- [strchr\\_P](#), 248
- [strchr\\_PF](#), 248
- [strchrnul\\_P](#), 248
- [strcmp\\_P](#), 248
- [strcmp\\_PF](#), 249
- [strcpy\\_P](#), 249
- [strcpy\\_PF](#), 249
- [strcspn\\_P](#), 250
- [strlcat\\_P](#), 250
- [strlcat\\_PF](#), 250
- [strncpy\\_P](#), 251
- [strncpy\\_PF](#), 251
- [strlen\\_P](#), 251
- [strlen\\_PF](#), 252
- [strncasecmp\\_P](#), 252
- [strncasecmp\\_PF](#), 253
- [strncat\\_P](#), 253
- [strncat\\_PF](#), 253
- [strncmp\\_P](#), 254
- [strncmp\\_PF](#), 254
- [strncpy\\_P](#), 254
- [strncpy\\_PF](#), 255
- [strnlen\\_P](#), 255
- [strnlen\\_PF](#), 255
- [strpbrk\\_P](#), 256
- [strrchr\\_P](#), 256
- [strsep\\_P](#), 256
- [strspn\\_P](#), 257
- [strstr\\_P](#), 257
- [strstr\\_PF](#), 257
- [strtok\\_P](#), 258
- [strtok\\_rP](#), 258
- [<avr/power.h>: Power Reduction Management](#), 259
  - [clock\\_prescale\\_get](#), 261
  - [clock\\_prescale\\_set](#), 262
  - [power\\_all\\_disable](#), 262
  - [power\\_all\\_enable](#), 262
- [<avr/sfr\\_defs.h>: Special function registers](#), 263
  - [\\_BV](#), 264
  - [bit\\_is\\_clear](#), 265

- bit\_is\_set, [265](#)
- loop\_until\_bit\_is\_clear, [265](#)
- loop\_until\_bit\_is\_set, [265](#)
- <avr/signature.h>: Signature Support, [265](#)
- <avr/sleep.h>: Power Management and Sleep Modes, [266](#)
  - sleep\_bod\_disable, [266](#)
  - sleep\_cpu, [266](#)
  - sleep\_disable, [266](#)
  - sleep\_enable, [266](#)
  - sleep\_mode, [267](#)
- <avr/version.h>: avr-libc version macros, [268](#)
  - \_\_AVR\_LIBC\_DATE\_\_, [268](#)
  - \_\_AVR\_LIBC\_DATE\_STRING\_\_, [268](#)
  - \_\_AVR\_LIBC\_MAJOR\_\_, [268](#)
  - \_\_AVR\_LIBC\_MINOR\_\_, [268](#)
  - \_\_AVR\_LIBC\_REVISION\_\_, [269](#)
  - \_\_AVR\_LIBC\_VERSION\_STRING\_\_, [269](#)
  - \_\_AVR\_LIBC\_VERSION\_\_, [269](#)
- <avr/wdt.h>: Watchdog timer handling, [271](#)
  - wdt\_enable, [271](#)
  - wdt\_reset, [272](#)
  - WDTO\_120MS, [272](#)
  - WDTO\_15MS, [272](#)
  - WDTO\_1S, [272](#)
  - WDTO\_250MS, [272](#)
  - WDTO\_2S, [272](#)
  - WDTO\_30MS, [272](#)
  - WDTO\_4S, [272](#)
  - WDTO\_500MS, [273](#)
  - WDTO\_60MS, [273](#)
  - WDTO\_8S, [273](#)
- <compat/deprecated.h>: Deprecated items, [291](#)
  - cbi, [292](#)
  - enable\_external\_int, [292](#)
  - inb, [292](#)
  - inp, [292](#)
  - INTERRUPT, [292](#)
  - outb, [292](#)
  - outp, [293](#)
  - sbi, [293](#)
  - timer\_enable\_int, [293](#)
- <compat/ina90.h>: Compatibility with IAR EWB 3.x, [293](#)
- <ctype.h>: Character Operations, [105](#)
  - isalnum, [105](#)
  - isalpha, [106](#)
  - isascii, [106](#)
  - isblank, [106](#)
  - iscntrl, [106](#)
  - isdigit, [106](#)
  - isgraph, [106](#)
  - islower, [106](#)
  - isprint, [106](#)
  - ispunct, [106](#)
  - isspace, [106](#)
  - isupper, [107](#)
  - isxdigit, [107](#)
  - toascii, [107](#)
  - tolower, [107](#)
  - toupper, [107](#)
- <errno.h>: System Errors, [107](#)
  - EDOM, [108](#)
  - ERANGE, [108](#)
  - errno, [108](#)
- <inttypes.h>: Integer Type conversions, [108](#)
  - int\_farptr\_t, [121](#)
  - PRId16, [111](#)
  - PRId32, [111](#)
  - PRId8, [111](#)
  - PRIdFAST16, [111](#)
  - PRIdFAST32, [111](#)
  - PRIdFAST8, [111](#)
  - PRIdLEAST16, [112](#)
  - PRIdLEAST32, [112](#)
  - PRIdLEAST8, [112](#)
  - PRIdPTR, [112](#)
  - PRId16, [112](#)
  - PRId32, [112](#)
  - PRId8, [112](#)
  - PRIdFAST16, [112](#)
  - PRIdFAST32, [112](#)
  - PRIdFAST8, [112](#)
  - PRIdLEAST16, [112](#)
  - PRIdLEAST32, [113](#)
  - PRIdLEAST8, [113](#)
  - PRIdPTR, [113](#)
  - PRId16, [113](#)
  - PRId32, [113](#)
  - PRId8, [113](#)
  - PRIdFAST16, [113](#)
  - PRIdFAST32, [113](#)
  - PRIdFAST8, [113](#)
  - PRIdLEAST16, [113](#)
  - PRIdLEAST32, [113](#)
  - PRIdLEAST8, [114](#)
  - PRIdPTR, [114](#)
  - PRId16, [114](#)
  - PRId32, [114](#)
  - PRId8, [114](#)
  - PRIdFAST16, [114](#)
  - PRIdFAST32, [114](#)
  - PRIdFAST8, [114](#)
  - PRIdLEAST16, [114](#)
  - PRIdLEAST32, [114](#)
  - PRIdLEAST8, [114](#)
  - PRIdPTR, [115](#)
  - PRId16, [115](#)
  - PRId32, [115](#)
  - PRId8, [115](#)
  - PRIdFAST16, [115](#)
  - PRIdFAST16, [115](#)
  - PRIdFAST32, [115](#)

PRiXFAST32, 115  
PRiXFAST8, 116  
PRiXFAST8, 116  
PRiXLEAST16, 116  
PRiXLEAST16, 116  
PRiXLEAST32, 116  
PRiXLEAST32, 116  
PRiXLEAST8, 116  
PRiXLEAST8, 116  
PRiXPTR, 116  
PRiXPTR, 116  
SCNd16, 116  
SCNd32, 117  
SCNd8, 117  
SCNdFAST16, 117  
SCNdFAST32, 117  
SCNdFAST8, 117  
SCNdLEAST16, 117  
SCNdLEAST32, 117  
SCNdLEAST8, 117  
SCNdPTR, 117  
SCNi16, 117  
SCNi32, 117  
SCNi8, 118  
SCNiFAST16, 118  
SCNiFAST32, 118  
SCNiFAST8, 118  
SCNiLEAST16, 118  
SCNiLEAST32, 118  
SCNiLEAST8, 118  
SCNiPTR, 118  
SCNo16, 118  
SCNo32, 118  
SCNo8, 118  
SCNoFAST16, 119  
SCNoFAST32, 119  
SCNoFAST8, 119  
SCNoLEAST16, 119  
SCNoLEAST32, 119  
SCNoLEAST8, 119  
SCNoPTR, 119  
SCNu16, 119  
SCNu32, 119  
SCNu8, 119  
SCNuFAST16, 119  
SCNuFAST32, 120  
SCNuFAST8, 120  
SCNuLEAST16, 120  
SCNuLEAST32, 120  
SCNuLEAST8, 120  
SCNuPTR, 120  
SCNx16, 120  
SCNx32, 120  
SCNx8, 120  
SCNxFAST16, 120  
SCNxFAST32, 120  
SCNxFAST8, 121  
SCNxLEAST16, 121  
SCNxLEAST32, 121  
SCNxLEAST8, 121  
SCNxPTR, 121  
uint\_farptr\_t, 121  
<math.h>: Mathematics, 122  
acos, 127  
acosf, 127  
acosl, 127  
asin, 127  
asinf, 127  
asinl, 127  
atan, 127  
atan2, 127  
atan2f, 128  
atan2l, 128  
atanf, 128  
atanl, 128  
cbrt, 128  
cbrtf, 128  
cbrtl, 128  
ceil, 128  
ceilf, 129  
ceill, 129  
copysign, 129  
copysignf, 129  
copysignl, 129  
cos, 129  
cosf, 129  
cosh, 129  
coshf, 129  
coshl, 130  
cosl, 130  
exp, 130  
expf, 130  
expl, 130  
fabs, 130  
fabsf, 130  
fabsl, 130  
fdim, 130  
fdimf, 131  
fdiml, 131  
floor, 131  
floorf, 131  
floorl, 131  
fma, 131  
fmaf, 131  
fmal, 131  
fmax, 132  
fmaxf, 132  
fmaxl, 132  
fmin, 132  
fminf, 132  
fminl, 132  
fmod, 132  
fmodf, 133  
fmodl, 133  
frexp, 133  
frexpf, 133

- frexpl, [133](#)
- HUGE\_VAL, [125](#)
- HUGE\_VALF, [125](#)
- HUGE\_VALL, [125](#)
- hypot, [133](#)
- hypotf, [134](#)
- hypotl, [134](#)
- INFINITY, [125](#)
- isfinite, [134](#)
- isfinitef, [134](#)
- isfinitel, [134](#)
- isinf, [134](#)
- isinff, [134](#)
- isinfl, [134](#)
- isnan, [135](#)
- isnanf, [135](#)
- isnanl, [135](#)
- ldexp, [135](#)
- ldexpf, [135](#)
- ldexpl, [135](#)
- log, [135](#)
- log10, [135](#)
- log10f, [135](#)
- log10l, [136](#)
- logf, [136](#)
- logl, [136](#)
- lrint, [136](#)
- lrintf, [136](#)
- lrintl, [136](#)
- lround, [137](#)
- lroundf, [137](#)
- lroundl, [137](#)
- M\_1\_PI, [125](#)
- M\_2\_PI, [125](#)
- M\_2\_SQRTPI, [125](#)
- M\_E, [125](#)
- M\_LN10, [125](#)
- M\_LN2, [125](#)
- M\_LOG10E, [125](#)
- M\_LOG2E, [126](#)
- M\_PI, [126](#)
- M\_PI\_2, [126](#)
- M\_PI\_4, [126](#)
- M\_SQRT1\_2, [126](#)
- M\_SQRT2, [126](#)
- modf, [137](#)
- modff, [138](#)
- modfl, [138](#)
- NAN, [126](#)
- nan, [126](#)
- nanf, [126](#)
- nanl, [126](#)
- pow, [138](#)
- powf, [138](#)
- powl, [138](#)
- round, [139](#)
- roundf, [139](#)
- roundl, [139](#)
- signbit, [139](#)
- signbitf, [139](#)
- signbitl, [140](#)
- sin, [140](#)
- sinf, [140](#)
- sinh, [140](#)
- sinhf, [140](#)
- sinhl, [140](#)
- sinl, [140](#)
- sqrt, [140](#)
- sqrtf, [140](#)
- sqrtl, [141](#)
- square, [141](#)
- squaref, [141](#)
- squarel, [141](#)
- tan, [141](#)
- tanf, [141](#)
- tanh, [142](#)
- tanhf, [142](#)
- tanhl, [142](#)
- tanl, [142](#)
- trunc, [142](#)
- truncf, [142](#)
- truncl, [142](#)
- <setjmp.h>: Non-local goto, [142](#)
  - longjmp, [143](#)
  - setjmp, [144](#)
- <stdint.h>: Standard Integer Types, [144](#)
  - INT16\_C, [147](#)
  - INT16\_MAX, [147](#)
  - INT16\_MIN, [147](#)
  - int16\_t, [152](#)
  - INT32\_C, [147](#)
  - INT32\_MAX, [147](#)
  - INT32\_MIN, [148](#)
  - int32\_t, [152](#)
  - INT64\_C, [148](#)
  - INT64\_MAX, [148](#)
  - INT64\_MIN, [148](#)
  - int64\_t, [153](#)
  - INT8\_C, [148](#)
  - INT8\_MAX, [148](#)
  - INT8\_MIN, [148](#)
  - int8\_t, [153](#)
  - INT\_FAST16\_MAX, [148](#)
  - INT\_FAST16\_MIN, [148](#)
  - int\_fast16\_t, [153](#)
  - INT\_FAST32\_MAX, [148](#)
  - INT\_FAST32\_MIN, [148](#)
  - int\_fast32\_t, [153](#)
  - INT\_FAST64\_MAX, [149](#)
  - INT\_FAST64\_MIN, [149](#)
  - int\_fast64\_t, [153](#)
  - INT\_FAST8\_MAX, [149](#)
  - INT\_FAST8\_MIN, [149](#)
  - int\_fast8\_t, [153](#)
  - INT\_LEAST16\_MAX, [149](#)
  - INT\_LEAST16\_MIN, [149](#)

int\_least16\_t, 153  
 INT\_LEAST32\_MAX, 149  
 INT\_LEAST32\_MIN, 149  
 int\_least32\_t, 153  
 INT\_LEAST64\_MAX, 149  
 INT\_LEAST64\_MIN, 149  
 int\_least64\_t, 153  
 INT\_LEAST8\_MAX, 149  
 INT\_LEAST8\_MIN, 150  
 int\_least8\_t, 154  
 INTMAX\_C, 150  
 INTMAX\_MAX, 150  
 INTMAX\_MIN, 150  
 intmax\_t, 154  
 INTPTR\_MAX, 150  
 INTPTR\_MIN, 150  
 intptr\_t, 154  
 PTRDIFF\_MAX, 150  
 PTRDIFF\_MIN, 150  
 SIG\_ATOMIC\_MAX, 150  
 SIG\_ATOMIC\_MIN, 150  
 SIZE\_MAX, 150  
 UINT16\_C, 151  
 UINT16\_MAX, 151  
 uint16\_t, 154  
 UINT32\_C, 151  
 UINT32\_MAX, 151  
 uint32\_t, 154  
 UINT64\_C, 151  
 UINT64\_MAX, 151  
 uint64\_t, 154  
 UINT8\_C, 151  
 UINT8\_MAX, 151  
 uint8\_t, 154  
 UINT\_FAST16\_MAX, 151  
 uint\_fast16\_t, 154  
 UINT\_FAST32\_MAX, 151  
 uint\_fast32\_t, 154  
 UINT\_FAST64\_MAX, 151  
 uint\_fast64\_t, 155  
 UINT\_FAST8\_MAX, 152  
 uint\_fast8\_t, 155  
 UINT\_LEAST16\_MAX, 152  
 uint\_least16\_t, 155  
 UINT\_LEAST32\_MAX, 152  
 uint\_least32\_t, 155  
 UINT\_LEAST64\_MAX, 152  
 uint\_least64\_t, 155  
 UINT\_LEAST8\_MAX, 152  
 uint\_least8\_t, 155  
 UINTMAX\_C, 152  
 UINTMAX\_MAX, 152  
 uintmax\_t, 155  
 UINTPTR\_MAX, 152  
 intptr\_t, 155  
 <stdio.h>: Standard IO facilities, 156  
   \_FDEV\_EOF, 159  
   \_FDEV\_ERR, 159  
   \_FDEV\_SETUP\_READ, 159  
   \_FDEV\_SETUP\_RW, 159  
   \_FDEV\_SETUP\_WRITE, 159  
 clearerr, 162  
 EOF, 159  
 fclose, 162  
 fdev\_close, 160  
 fdev\_get\_udata, 160  
 fdev\_set\_udata, 160  
 FDEV\_SETUP\_STREAM, 160  
 fdev\_setup\_stream, 160  
 fdevopen, 162  
 feof, 162  
 ferror, 163  
 fflush, 163  
 fgetc, 163  
 fgets, 163  
 FILE, 161  
 fprintf, 163  
 fprintf\_P, 163  
 fputc, 163  
 fputs, 164  
 fputs\_P, 164  
 fread, 164  
 fscanf, 164  
 fscanf\_P, 164  
 fwrite, 164  
 getc, 160  
 getchar, 161  
 gets, 165  
 printf, 165  
 printf\_P, 165  
 putc, 161  
 putchar, 161  
 puts, 165  
 puts\_P, 165  
 scanf, 165  
 scanf\_P, 165  
 snprintf, 165  
 snprintf\_P, 166  
 sprintf, 166  
 sprintf\_P, 166  
 sscanf, 166  
 sscanf\_P, 166  
 stderr, 161  
 stdin, 161  
 stdout, 161  
 ungetc, 166  
 vfprintf, 167  
 vfprintf\_P, 169  
 vfscanf, 169  
 vfscanf\_P, 170  
 vprintf, 171  
 vscanf, 171  
 vsnprintf, 171  
 vsnprintf\_P, 171  
 vsprintf, 171  
 vsprintf\_P, 171

- <stdlib.h>: General utilities, 172
  - \_\_compar\_fn\_t, 174
  - \_\_malloc\_heap\_end, 183
  - \_\_malloc\_heap\_start, 184
  - \_\_malloc\_margin, 184
  - abort, 174
  - abs, 174
  - atexit, 174
  - atof, 175
  - atoff, 175
  - atofl, 175
  - atoi, 175
  - atol, 175
  - bsearch, 175
  - calloc, 176
  - div, 176
  - DTOSTR\_ALWAYS\_SIGN, 173
  - DTOSTR\_PLUS\_SIGN, 173
  - DTOSTR\_UPPERCASE, 173
  - dtostre, 176
  - dtostrf, 176
  - exit, 176
  - EXIT\_FAILURE, 174
  - EXIT\_SUCCESS, 174
  - free, 177
  - fstre, 177
  - fstorf, 177
  - itoa, 177
  - labs, 178
  - ldiv, 178
  - ldtostre, 178
  - ldtostrf, 178
  - ltoa, 179
  - malloc, 179
  - qsort, 179
  - rand, 180
  - RAND\_MAX, 174
  - rand\_r, 180
  - random, 180
  - RANDOM\_MAX, 174
  - random\_r, 180
  - realloc, 180
  - srand, 181
  - srandom, 181
  - strtod, 181
  - strtof, 181
  - strtol, 181
  - strtold, 182
  - strtoul, 182
  - ultoa, 182
  - utoa, 183
- <string.h>: Strings, 184
  - \_FFS, 185
  - ffs, 185
  - ffsl, 185
  - ffsll, 186
  - memccpy, 186
  - memchr, 186
  - memcmp, 186
  - memcpy, 187
  - memmem, 187
  - memmove, 187
  - memrchr, 188
  - memset, 188
  - strcasecmp, 188
  - strcasestr, 189
  - strcat, 189
  - strchr, 189
  - strchrnul, 189
  - strcmp, 190
  - strcpy, 190
  - strcspn, 190
  - strdup, 191
  - strlcat, 191
  - strncpy, 191
  - strlen, 192
  - strlwr, 192
  - strncasecmp, 192
  - strncat, 193
  - strncmp, 193
  - strncpy, 193
  - strndup, 194
  - strnlen, 194
  - strpbrk, 194
  - strrchr, 195
  - strrev, 195
  - strsep, 195
  - strspn, 195
  - strstr, 196
  - strtok, 196
  - strtok\_r, 196
  - strupr, 197
- <time.h>: Time, 197
  - \_MONTHS\_, 200
  - \_WEEK\_DAYS\_, 200
  - asctime, 201
  - asctime\_r, 201
  - ctime, 201
  - ctime\_r, 201
  - daylight\_seconds, 201
  - difftime, 201
  - equation\_of\_time, 201
  - fatfs\_time, 202
  - gm\_sidereal, 202
  - gmtime, 202
  - gmtime\_r, 202
  - is\_leap\_year, 202
  - iso\_week\_date, 202
  - iso\_week\_date\_r, 202
  - isotime, 202
  - isotime\_r, 203
  - lm\_sidereal, 203
  - localtime, 203
  - localtime\_r, 203
  - mk\_gmtime, 203
  - mktime, 203

- month\_length, 203
- moon\_phase, 204
- NTP\_OFFSET, 199
- ONE\_DAY, 200
- ONE\_DEGREE, 200
- ONE\_HOUR, 200
- set\_dst, 204
- set\_position, 204
- set\_system\_time, 204
- set\_zone, 204
- solar\_declination, 205
- solar\_declinationf, 205
- solar\_declinationl, 205
- solar\_noon, 205
- strftime, 205
- sun\_rise, 205
- sun\_set, 205
- system\_tick, 206
- time, 206
- time\_t, 200
- UNIX\_OFFSET, 200
- week\_of\_month, 206
- week\_of\_year, 206
- <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks, 276
  - ATOMIC\_BLOCK, 277
  - ATOMIC\_FORCEON, 277
  - ATOMIC\_RESTORESTATE, 277
  - NONATOMIC\_BLOCK, 278
  - NONATOMIC\_FORCEOFF, 278
  - NONATOMIC\_RESTORESTATE, 278
- <util/crc16.h>: CRC Computations, 278
  - \_crc16\_update, 279
  - \_crc8\_ccitt\_update, 279
  - \_crc\_ccitt\_update, 280
  - \_crc\_ibutton\_update, 281
  - \_crc\_xmodem\_update, 281
- <util/delay.h>: Convenience functions for busy-wait delay loops, 273
  - \_delay\_ms, 274
  - \_delay\_us, 275
  - F\_CPU, 274
- <util/delay\_basic.h>: Basic busy-wait delay loops, 282
  - \_delay\_loop\_1, 282
  - \_delay\_loop\_2, 282
- <util/eu\_dst.h>: Daylight Saving function for the European Union., 282
  - eu\_dst, 283
- <util/parity.h>: Parity bit generation, 283
  - parity\_even\_bit, 283
- <util/setbaud.h>: Helper macros for baud rate calculations, 284
  - BAUD\_TOL, 285
  - UBRR\_VALUE, 285
  - UBRRH\_VALUE, 285
  - UBRRL\_VALUE, 285
  - USE\_2X, 285
- <util/twi.h>: TWI bit mask definitions, 286
  - TW\_BUS\_ERROR, 287
  - TW\_MR\_ARB\_LOST, 287
  - TW\_MR\_DATA\_ACK, 287
  - TW\_MR\_DATA\_NACK, 287
  - TW\_MR\_SLA\_ACK, 287
  - TW\_MR\_SLA\_NACK, 287
  - TW\_MT\_ARB\_LOST, 287
  - TW\_MT\_DATA\_ACK, 287
  - TW\_MT\_DATA\_NACK, 287
  - TW\_MT\_SLA\_ACK, 287
  - TW\_MT\_SLA\_NACK, 287
  - TW\_NO\_INFO, 288
  - TW\_READ, 288
  - TW\_REP\_START, 288
  - TW\_SR\_ARB\_LOST\_GCALL\_ACK, 288
  - TW\_SR\_ARB\_LOST\_SLA\_ACK, 288
  - TW\_SR\_DATA\_ACK, 288
  - TW\_SR\_DATA\_NACK, 288
  - TW\_SR\_GCALL\_ACK, 288
  - TW\_SR\_GCALL\_DATA\_ACK, 288
  - TW\_SR\_GCALL\_DATA\_NACK, 288
  - TW\_SR\_SLA\_ACK, 288
  - TW\_SR\_STOP, 289
  - TW\_ST\_ARB\_LOST\_SLA\_ACK, 289
  - TW\_ST\_DATA\_ACK, 289
  - TW\_ST\_DATA\_NACK, 289
  - TW\_ST\_LAST\_DATA, 289
  - TW\_ST\_SLA\_ACK, 289
  - TW\_START, 289
  - TW\_STATUS, 289
  - TW\_STATUS\_MASK, 289
  - TW\_WRITE, 289
- <util/usa\_dst.h>: Daylight Saving function for the USA., 290
  - usa\_dst, 290
- prefix, 77
- \$PATH, 77
- \$PREFIX, 77
- \_BV
  - <avr/sfr\_defs.h>: Special function registers, 264
- \_EETGET
  - <avr/EEPROM.h>: EEPROM handling, 215
- \_EEPWRITE
  - <avr/EEPROM.h>: EEPROM handling, 215
- \_FDEV\_EOF
  - <stdio.h>: Standard IO facilities, 159
- \_FDEV\_ERR
  - <stdio.h>: Standard IO facilities, 159
- \_FDEV\_SETUP\_READ
  - <stdio.h>: Standard IO facilities, 159
- \_FDEV\_SETUP\_RW
  - <stdio.h>: Standard IO facilities, 159
- \_FDEV\_SETUP\_WRITE
  - <stdio.h>: Standard IO facilities, 159
- \_FFS
  - <string.h>: Strings, 185
- \_MONTHS\_
- <time.h>: Time, 200

- `__MemoryBarrier`
  - <avr/cpufunc.h>: Special AVR CPU functions, [213](#)
- `__NOP`
  - <avr/cpufunc.h>: Special AVR CPU functions, [213](#)
- `__PROTECTED_WRITE`
  - <avr/io.h>: AVR device-specific IO definitions, [229](#)
- `__PROTECTED_WRITE_SPM`
  - <avr/io.h>: AVR device-specific IO definitions, [229](#)
- `__WEEK_DAYS__`
  - <time.h>: Time, [200](#)
- `__AVR_LIBC_DATE__`
  - <avr/version.h>: avr-libc version macros, [268](#)
- `__AVR_LIBC_DATE_STRING__`
  - <avr/version.h>: avr-libc version macros, [268](#)
- `__AVR_LIBC_MAJOR__`
  - <avr/version.h>: avr-libc version macros, [268](#)
- `__AVR_LIBC_MINOR__`
  - <avr/version.h>: avr-libc version macros, [268](#)
- `__AVR_LIBC_REVISION__`
  - <avr/version.h>: avr-libc version macros, [269](#)
- `__AVR_LIBC_VERSION_STRING__`
  - <avr/version.h>: avr-libc version macros, [269](#)
- `__AVR_LIBC_VERSION__`
  - <avr/version.h>: avr-libc version macros, [269](#)
- `__EGET`
  - <avr/eeprom.h>: EEPROM handling, [215](#)
- `__EPUT`
  - <avr/eeprom.h>: EEPROM handling, [215](#)
- `__builtin_avr_cli`
  - <avr/builtins.h>: avr-gcc builtins documentation, [269](#)
- `__builtin_avr_fmul`
  - <avr/builtins.h>: avr-gcc builtins documentation, [270](#)
- `__builtin_avr_fmuls`
  - <avr/builtins.h>: avr-gcc builtins documentation, [270](#)
- `__builtin_avr_fmulsu`
  - <avr/builtins.h>: avr-gcc builtins documentation, [270](#)
- `__builtin_avr_sei`
  - <avr/builtins.h>: avr-gcc builtins documentation, [270](#)
- `__builtin_avr_sleep`
  - <avr/builtins.h>: avr-gcc builtins documentation, [270](#)
- `__builtin_avr_swap`
  - <avr/builtins.h>: avr-gcc builtins documentation, [270](#)
- `__builtin_avr_wdr`
  - <avr/builtins.h>: avr-gcc builtins documentation, [270](#)
- `__compar_fn_t`
  - <stdlib.h>: General utilities, [174](#)
- `__malloc_heap_end`
  - <stdlib.h>: General utilities, [183](#)
- `__malloc_heap_start`
  - <stdlib.h>: General utilities, [184](#)
- `__malloc_margin`
  - <stdlib.h>: General utilities, [184](#)
- `__crc16_update`
  - <util/crc16.h>: CRC Computations, [279](#)
- `__crc8_ccitt_update`
  - <util/crc16.h>: CRC Computations, [279](#)
- `__crc_ccitt_update`
  - <util/crc16.h>: CRC Computations, [280](#)
- `__crc_ibutton_update`
  - <util/crc16.h>: CRC Computations, [281](#)
- `__crc_xmodem_update`
  - <util/crc16.h>: CRC Computations, [281](#)
- `__delay_loop_1`
  - <util/delay\_basic.h>: Basic busy-wait delay loops, [282](#)
- `__delay_loop_2`
  - <util/delay\_basic.h>: Basic busy-wait delay loops, [282](#)
- `__delay_ms`
  - <util/delay.h>: Convenience functions for busy-wait delay loops, [274](#)
- `__delay_us`
  - <util/delay.h>: Convenience functions for busy-wait delay loops, [275](#)
- A more sophisticated project, [308](#)
- A simple project, [297](#)
- `abort`
  - <stdlib.h>: General utilities, [174](#)
- `abs`
  - <stdlib.h>: General utilities, [174](#)
- `acos`
  - <math.h>: Mathematics, [127](#)
- `acosf`
  - <math.h>: Mathematics, [127](#)
- `acosl`
  - <math.h>: Mathematics, [127](#)
- Additional notes from <avr/sfr\_defs.h>, [263](#)
- `alloca`
  - <alloca.h>: Allocate space in the stack, [103](#)
- `alloca.h`, [330](#)
- `asctime`
  - <time.h>: Time, [201](#)
- `asctime_r`
  - <time.h>: Time, [201](#)
- `asin`
  - <math.h>: Mathematics, [127](#)
- `asinf`
  - <math.h>: Mathematics, [127](#)
- `asinl`
  - <math.h>: Mathematics, [127](#)
- `assert`
  - <assert.h>: Diagnostics, [104](#)
- `assert.h`, [331](#)
- `atan`
  - <math.h>: Mathematics, [127](#)
- `atan2`
  - <math.h>: Mathematics, [127](#)
- `atan2f`



- `<math.h>`: Mathematics, [128](#)
- `atan2l`
  - `<math.h>`: Mathematics, [128](#)
- `atanf`
  - `<math.h>`: Mathematics, [128](#)
- `atanl`
  - `<math.h>`: Mathematics, [128](#)
- `atexit`
  - `<stdlib.h>`: General utilities, [174](#)
- `atof`
  - `<stdlib.h>`: General utilities, [175](#)
- `atoff`
  - `<stdlib.h>`: General utilities, [175](#)
- `atofl`
  - `<stdlib.h>`: General utilities, [175](#)
- `atoi`
  - `<stdlib.h>`: General utilities, [175](#)
- `atol`
  - `<stdlib.h>`: General utilities, [175](#)
- `atomic.h`, [525](#)
- `ATOMIC_BLOCK`
  - `<util/atomic.h>` Atomically and Non-Atomically Executed Code Blocks, [277](#)
- `ATOMIC_FORCEON`
  - `<util/atomic.h>` Atomically and Non-Atomically Executed Code Blocks, [277](#)
- `ATOMIC_RESTORESTATE`
  - `<util/atomic.h>` Atomically and Non-Atomically Executed Code Blocks, [277](#)
- `avrdude`, usage, [97](#)
- `avrprog`, usage, [97](#)
- `BADISR_vect`
  - `<avr/interrupt.h>`: Interrupts, [225](#)
- `BAUD_TOL`
  - `<util/setbaud.h>`: Helper macros for baud rate calculations, [285](#)
- `bit_is_clear`
  - `<avr/sfr_defs.h>`: Special function registers, [265](#)
- `bit_is_set`
  - `<avr/sfr_defs.h>`: Special function registers, [265](#)
- `boot.h`, [332](#), [333](#)
- `boot_is_spm_interrupt`
  - `<avr/boot.h>`: Bootloader Support Utilities, [208](#)
- `boot_lock_bits_set`
  - `<avr/boot.h>`: Bootloader Support Utilities, [208](#)
- `boot_lock_bits_set_safe`
  - `<avr/boot.h>`: Bootloader Support Utilities, [209](#)
- `boot_lock_fuse_bits_get`
  - `<avr/boot.h>`: Bootloader Support Utilities, [209](#)
- `boot_page_erase`
  - `<avr/boot.h>`: Bootloader Support Utilities, [209](#)
- `boot_page_erase_safe`
  - `<avr/boot.h>`: Bootloader Support Utilities, [209](#)
- `boot_page_fill`
  - `<avr/boot.h>`: Bootloader Support Utilities, [210](#)
- `boot_page_fill_safe`
  - `<avr/boot.h>`: Bootloader Support Utilities, [210](#)
- `boot_page_write`
  - `<avr/boot.h>`: Bootloader Support Utilities, [210](#)
- `boot_page_write_safe`
  - `<avr/boot.h>`: Bootloader Support Utilities, [210](#)
- `boot_rww_busy`
  - `<avr/boot.h>`: Bootloader Support Utilities, [211](#)
- `boot_rww_enable`
  - `<avr/boot.h>`: Bootloader Support Utilities, [211](#)
- `boot_rww_enable_safe`
  - `<avr/boot.h>`: Bootloader Support Utilities, [211](#)
- `boot_signature_byte_get`
  - `<avr/boot.h>`: Bootloader Support Utilities, [211](#)
- `boot_spm_busy`
  - `<avr/boot.h>`: Bootloader Support Utilities, [211](#)
- `boot_spm_busy_wait`
  - `<avr/boot.h>`: Bootloader Support Utilities, [212](#)
- `boot_spm_interrupt_disable`
  - `<avr/boot.h>`: Bootloader Support Utilities, [212](#)
- `boot_spm_interrupt_enable`
  - `<avr/boot.h>`: Bootloader Support Utilities, [212](#)
- `BOOTLOADER_SECTION`
  - `<avr/boot.h>`: Bootloader Support Utilities, [212](#)
- `bsearch`
  - `<stdlib.h>`: General utilities, [175](#)
- `builtins.h`, [341](#)
- `calloc`
  - `<stdlib.h>`: General utilities, [176](#)
- `cbi`
  - `<compat/deprecated.h>`: Deprecated items, [292](#)
- `cbt`
  - `<math.h>`: Mathematics, [128](#)
- `cbtrf`
  - `<math.h>`: Mathematics, [128](#)
- `cbtrl`
  - `<math.h>`: Mathematics, [128](#)
- `ccp_write_io`
  - `<avr/cpufunc.h>`: Special AVR CPU functions, [213](#)
- `ccp_write_spm`
  - `<avr/cpufunc.h>`: Special AVR CPU functions, [213](#)
- `ceil`
  - `<math.h>`: Mathematics, [128](#)
- `ceilf`
  - `<math.h>`: Mathematics, [129](#)
- `ceilll`
  - `<math.h>`: Mathematics, [129](#)
- `clearerr`
  - `<stdio.h>`: Standard IO facilities, [162](#)
- `cli`
  - `<avr/interrupt.h>`: Interrupts, [225](#)
- `clock_prescale_get`
  - `<avr/power.h>`: Power Reduction Management, [261](#)
- `clock_prescale_set`
  - `<avr/power.h>`: Power Reduction Management, [262](#)
- Combining C and assembly source files, [295](#)
- `copysign`
  - `<math.h>`: Mathematics, [129](#)
- `copysignf`

- <math.h>: Mathematics, [129](#)
- copysignl
  - <math.h>: Mathematics, [129](#)
- cos
  - <math.h>: Mathematics, [129](#)
- cosf
  - <math.h>: Mathematics, [129](#)
- cosh
  - <math.h>: Mathematics, [129](#)
- coshf
  - <math.h>: Mathematics, [129](#)
- coshl
  - <math.h>: Mathematics, [130](#)
- cosl
  - <math.h>: Mathematics, [130](#)
- cpufunc.h, [343](#)
- crc16.h, [529](#)
- ctime
  - <time.h>: Time, [201](#)
- ctime\_r
  - <time.h>: Time, [201](#)
- ctype.h, [447](#), [448](#)
- day
  - week\_date, [324](#)
- daylight\_seconds
  - <time.h>: Time, [201](#)
- defines.h, [327](#)
- delay.h, [533](#), [534](#)
- delay\_basic.h, [537](#)
- Demo projects, [294](#)
- deprecated.h, [443](#)
- difftime
  - <time.h>: Time, [201](#)
- disassembling, [300](#)
- div
  - <stdlib.h>: General utilities, [176](#)
- div\_t, [321](#)
  - quot, [322](#)
  - rem, [322](#)
- dtoa\_conv.h, [552](#)
- DTOSTR\_ALWAYS\_SIGN
  - <stdlib.h>: General utilities, [173](#)
- DTOSTR\_PLUS\_SIGN
  - <stdlib.h>: General utilities, [173](#)
- DTOSTR\_UPPERCASE
  - <stdlib.h>: General utilities, [173](#)
- dtostre
  - <stdlib.h>: General utilities, [176](#)
- dtostrf
  - <stdlib.h>: General utilities, [176](#)
- EDOM
  - <errno.h>: System Errors, [108](#)
- eedef.h, [549](#)
- EEMEM
  - <avr/eeprom.h>: EEPROM handling, [216](#)
- eeprom.h, [344](#)
- eeprom\_busy\_wait
  - <avr/eeprom.h>: EEPROM handling, [216](#)
- eeprom\_is\_ready
  - <avr/eeprom.h>: EEPROM handling, [216](#)
- eeprom\_read\_block
  - <avr/eeprom.h>: EEPROM handling, [216](#)
- eeprom\_read\_byte
  - <avr/eeprom.h>: EEPROM handling, [216](#)
- eeprom\_read\_double
  - <avr/eeprom.h>: EEPROM handling, [216](#)
- eeprom\_read\_dword
  - <avr/eeprom.h>: EEPROM handling, [216](#)
- eeprom\_read\_float
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_read\_long\_double
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_read\_qword
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_read\_word
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_update\_block
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_update\_byte
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_update\_double
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_update\_dword
  - <avr/eeprom.h>: EEPROM handling, [217](#)
- eeprom\_update\_float
  - <avr/eeprom.h>: EEPROM handling, [218](#)
- eeprom\_update\_long\_double
  - <avr/eeprom.h>: EEPROM handling, [218](#)
- eeprom\_update\_qword
  - <avr/eeprom.h>: EEPROM handling, [218](#)
- eeprom\_update\_word
  - <avr/eeprom.h>: EEPROM handling, [218](#)
- eeprom\_write\_block
  - <avr/eeprom.h>: EEPROM handling, [218](#)
- eeprom\_write\_byte
  - <avr/eeprom.h>: EEPROM handling, [218](#)
- eeprom\_write\_double
  - <avr/eeprom.h>: EEPROM handling, [218](#)
- eeprom\_write\_dword
  - <avr/eeprom.h>: EEPROM handling, [219](#)
- eeprom\_write\_float
  - <avr/eeprom.h>: EEPROM handling, [219](#)
- eeprom\_write\_long\_double
  - <avr/eeprom.h>: EEPROM handling, [219](#)
- eeprom\_write\_qword
  - <avr/eeprom.h>: EEPROM handling, [219](#)
- eeprom\_write\_word
  - <avr/eeprom.h>: EEPROM handling, [219](#)
- EMPTY\_INTERRUPT
  - <avr/interrupt.h>: Interrupts, [226](#)
- enable\_external\_int
  - <compat/deprecated.h>: Deprecated items, [292](#)
- EOF
  - <stdio.h>: Standard IO facilities, [159](#)
- ephemera\_common.h, [554](#)

- equation\_of\_time
  - <time.h>: Time, [201](#)
- ERANGE
  - <errno.h>: System Errors, [108](#)
- errno
  - <errno.h>: System Errors, [108](#)
- errno.h, [450](#)
- eu\_dst
  - <util/eu\_dst.h>: Daylight Saving function for the European Union., [283](#)
- eu\_dst.h, [539](#)
- Example using the two-wire interface (TWI), [317](#)
- exit
  - <stdlib.h>: General utilities, [176](#)
- EXIT\_FAILURE
  - <stdlib.h>: General utilities, [174](#)
- EXIT\_SUCCESS
  - <stdlib.h>: General utilities, [174](#)
- exp
  - <math.h>: Mathematics, [130](#)
- expf
  - <math.h>: Mathematics, [130](#)
- expl
  - <math.h>: Mathematics, [130](#)
- F\_CPU
  - <util/delay.h>: Convenience functions for busy-wait delay loops, [274](#)
- fabs
  - <math.h>: Mathematics, [130](#)
- fabsf
  - <math.h>: Mathematics, [130](#)
- fabsl
  - <math.h>: Mathematics, [130](#)
- FAQ, [53](#)
- fatfs\_time
  - <time.h>: Time, [202](#)
- fclose
  - <stdio.h>: Standard IO facilities, [162](#)
- fdev\_close
  - <stdio.h>: Standard IO facilities, [160](#)
- fdev\_get\_udata
  - <stdio.h>: Standard IO facilities, [160](#)
- fdev\_set\_udata
  - <stdio.h>: Standard IO facilities, [160](#)
- FDEV\_SETUP\_STREAM
  - <stdio.h>: Standard IO facilities, [160](#)
- fdev\_setup\_stream
  - <stdio.h>: Standard IO facilities, [160](#)
- fdevopen
  - <stdio.h>: Standard IO facilities, [162](#)
- fdevopen.c, [551](#)
- fdim
  - <math.h>: Mathematics, [130](#)
- fdimf
  - <math.h>: Mathematics, [131](#)
- fdiml
  - <math.h>: Mathematics, [131](#)
- feof
  - <stdio.h>: Standard IO facilities, [162](#)
- ferror
  - <stdio.h>: Standard IO facilities, [163](#)
- fflush
  - <stdio.h>: Standard IO facilities, [163](#)
- ffs
  - <string.h>: Strings, [185](#)
- ffsl
  - <string.h>: Strings, [185](#)
- ffsll
  - <string.h>: Strings, [186](#)
- fgetc
  - <stdio.h>: Standard IO facilities, [163](#)
- fgets
  - <stdio.h>: Standard IO facilities, [163](#)
- FILE
  - <stdio.h>: Standard IO facilities, [161](#)
- floor
  - <math.h>: Mathematics, [131](#)
- floorf
  - <math.h>: Mathematics, [131](#)
- floorl
  - <math.h>: Mathematics, [131](#)
- fma
  - <math.h>: Mathematics, [131](#)
- fmaf
  - <math.h>: Mathematics, [131](#)
- fmal
  - <math.h>: Mathematics, [131](#)
- fmax
  - <math.h>: Mathematics, [132](#)
- fmaxf
  - <math.h>: Mathematics, [132](#)
- fmaxl
  - <math.h>: Mathematics, [132](#)
- fmin
  - <math.h>: Mathematics, [132](#)
- fminf
  - <math.h>: Mathematics, [132](#)
- fminl
  - <math.h>: Mathematics, [132](#)
- fmod
  - <math.h>: Mathematics, [132](#)
- fmodf
  - <math.h>: Mathematics, [133](#)
- fmodl
  - <math.h>: Mathematics, [133](#)
- fprintf
  - <stdio.h>: Standard IO facilities, [163](#)
- fprintf\_P
  - <stdio.h>: Standard IO facilities, [163](#)
- fputc
  - <stdio.h>: Standard IO facilities, [163](#)
- fputs
  - <stdio.h>: Standard IO facilities, [164](#)
- fputs\_P
  - <stdio.h>: Standard IO facilities, [164](#)
- fread

- <stdio.h>: Standard IO facilities, [164](#)
- free
  - <stdlib.h>: General utilities, [177](#)
- frexp
  - <math.h>: Mathematics, [133](#)
- frexpf
  - <math.h>: Mathematics, [133](#)
- frexpl
  - <math.h>: Mathematics, [133](#)
- fscanf
  - <stdio.h>: Standard IO facilities, [164](#)
- fscanf\_P
  - <stdio.h>: Standard IO facilities, [164](#)
- fstre
  - <stdlib.h>: General utilities, [177](#)
- fstorf
  - <stdlib.h>: General utilities, [177](#)
- fuse.h, [348](#)
- fwrite
  - <stdio.h>: Standard IO facilities, [164](#)
- GET\_EXTENDED\_FUSE\_BITS
  - <avr/boot.h>: Bootloader Support Utilities, [212](#)
- GET\_HIGH\_FUSE\_BITS
  - <avr/boot.h>: Bootloader Support Utilities, [212](#)
- GET\_LOCK\_BITS
  - <avr/boot.h>: Bootloader Support Utilities, [212](#)
- GET\_LOW\_FUSE\_BITS
  - <avr/boot.h>: Bootloader Support Utilities, [212](#)
- getc
  - <stdio.h>: Standard IO facilities, [160](#)
- getchar
  - <stdio.h>: Standard IO facilities, [161](#)
- gets
  - <stdio.h>: Standard IO facilities, [165](#)
- gm\_sideral
  - <time.h>: Time, [202](#)
- gmtime
  - <time.h>: Time, [202](#)
- gmtime\_r
  - <time.h>: Time, [202](#)
- hd44780.h, [328](#)
- HUGE\_VAL
  - <math.h>: Mathematics, [125](#)
- HUGE\_VALF
  - <math.h>: Mathematics, [125](#)
- HUGE\_VALL
  - <math.h>: Mathematics, [125](#)
- hypot
  - <math.h>: Mathematics, [133](#)
- hypotf
  - <math.h>: Mathematics, [134](#)
- hypotl
  - <math.h>: Mathematics, [134](#)
- ina90.h, [446](#)
- inp
  - <compat/deprecated.h>: Deprecated items, [292](#)
- installation, [76](#)
- installation, avarice, [81](#)
- installation, avr-libc, [80](#)
- installation, avrdude, [80](#)
- installation, avrprog, [80](#)
- installation, binutils, [78](#)
- installation, gcc, [79](#)
- installation, simulavr, [81](#)
- INT16\_C
  - <stdint.h>: Standard Integer Types, [147](#)
- INT16\_MAX
  - <stdint.h>: Standard Integer Types, [147](#)
- INT16\_MIN
  - <stdint.h>: Standard Integer Types, [147](#)
- int16\_t
  - <stdint.h>: Standard Integer Types, [152](#)
- INT32\_C
  - <stdint.h>: Standard Integer Types, [147](#)
- INT32\_MAX
  - <stdint.h>: Standard Integer Types, [147](#)
- INT32\_MIN
  - <stdint.h>: Standard Integer Types, [148](#)
- int32\_t
  - <stdint.h>: Standard Integer Types, [152](#)
- INT64\_C
  - <stdint.h>: Standard Integer Types, [148](#)
- INT64\_MAX
  - <stdint.h>: Standard Integer Types, [148](#)
- INT64\_MIN
  - <stdint.h>: Standard Integer Types, [148](#)
- int64\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- INT8\_C
  - <stdint.h>: Standard Integer Types, [148](#)
- INT8\_MAX
  - <stdint.h>: Standard Integer Types, [148](#)
- INT8\_MIN
  - <stdint.h>: Standard Integer Types, [148](#)
- int8\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- int\_farptr\_t
  - <inttypes.h>: Integer Type conversions, [121](#)
- INT\_FAST16\_MAX
  - <stdint.h>: Standard Integer Types, [148](#)
- INT\_FAST16\_MIN
  - <stdint.h>: Standard Integer Types, [148](#)
- int\_fast16\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- INT\_FAST32\_MAX
  - <stdint.h>: Standard Integer Types, [148](#)
- INT\_FAST32\_MIN
  - <stdint.h>: Standard Integer Types, [148](#)
- int\_fast32\_t
  - <stdint.h>: Standard Integer Types, [153](#)

- INT\_FAST64\_MAX
  - <stdint.h>: Standard Integer Types, [149](#)
- INT\_FAST64\_MIN
  - <stdint.h>: Standard Integer Types, [149](#)
- int\_fast64\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- INT\_FAST8\_MAX
  - <stdint.h>: Standard Integer Types, [149](#)
- INT\_FAST8\_MIN
  - <stdint.h>: Standard Integer Types, [149](#)
- int\_fast8\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- INT\_LEAST16\_MAX
  - <stdint.h>: Standard Integer Types, [149](#)
- INT\_LEAST16\_MIN
  - <stdint.h>: Standard Integer Types, [149](#)
- int\_least16\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- INT\_LEAST32\_MAX
  - <stdint.h>: Standard Integer Types, [149](#)
- INT\_LEAST32\_MIN
  - <stdint.h>: Standard Integer Types, [149](#)
- int\_least32\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- INT\_LEAST64\_MAX
  - <stdint.h>: Standard Integer Types, [149](#)
- INT\_LEAST64\_MIN
  - <stdint.h>: Standard Integer Types, [149](#)
- int\_least64\_t
  - <stdint.h>: Standard Integer Types, [153](#)
- INT\_LEAST8\_MAX
  - <stdint.h>: Standard Integer Types, [149](#)
- INT\_LEAST8\_MIN
  - <stdint.h>: Standard Integer Types, [150](#)
- int\_least8\_t
  - <stdint.h>: Standard Integer Types, [154](#)
- INTERRUPT
  - <compat/deprecated.h>: Deprecated items, [292](#)
- interrupt.h, [352](#)
- INTMAX\_C
  - <stdint.h>: Standard Integer Types, [150](#)
- INTMAX\_MAX
  - <stdint.h>: Standard Integer Types, [150](#)
- INTMAX\_MIN
  - <stdint.h>: Standard Integer Types, [150](#)
- intmax\_t
  - <stdint.h>: Standard Integer Types, [154](#)
- INTPTR\_MAX
  - <stdint.h>: Standard Integer Types, [150](#)
- INTPTR\_MIN
  - <stdint.h>: Standard Integer Types, [150](#)
- intptr\_t
  - <stdint.h>: Standard Integer Types, [154](#)
- inttypes.h, [452](#), [454](#)
- io.h, [357](#)
- iocompat.h, [325](#)
- is\_leap\_year
  - <time.h>: Time, [202](#)
- isalnum
  - <ctype.h>: Character Operations, [105](#)
- isalpha
  - <ctype.h>: Character Operations, [106](#)
- isascii
  - <ctype.h>: Character Operations, [106](#)
- isblank
  - <ctype.h>: Character Operations, [106](#)
- iscntrl
  - <ctype.h>: Character Operations, [106](#)
- isdigit
  - <ctype.h>: Character Operations, [106](#)
- isfinite
  - <math.h>: Mathematics, [134](#)
- isfinitf
  - <math.h>: Mathematics, [134](#)
- isfinitel
  - <math.h>: Mathematics, [134](#)
- isgraph
  - <ctype.h>: Character Operations, [106](#)
- isinf
  - <math.h>: Mathematics, [134](#)
- isnff
  - <math.h>: Mathematics, [134](#)
- isnfl
  - <math.h>: Mathematics, [134](#)
- islower
  - <ctype.h>: Character Operations, [106](#)
- isnan
  - <math.h>: Mathematics, [135](#)
- isnanf
  - <math.h>: Mathematics, [135](#)
- isnanl
  - <math.h>: Mathematics, [135](#)
- iso\_week\_date
  - <time.h>: Time, [202](#)
- iso\_week\_date\_r
  - <time.h>: Time, [202](#)
- isotime
  - <time.h>: Time, [202](#)
- isotime\_r
  - <time.h>: Time, [203](#)
- isprint
  - <ctype.h>: Character Operations, [106](#)
- ispunct
  - <ctype.h>: Character Operations, [106](#)
- ISR
  - <avr/interrupt.h>: Interrupts, [226](#)
- ISR\_ALIAS
  - <avr/interrupt.h>: Interrupts, [226](#)
- ISR\_ALIASOF
  - <avr/interrupt.h>: Interrupts, [226](#)
- ISR\_BLOCK
  - <avr/interrupt.h>: Interrupts, [227](#)
- ISR\_FLATTEN
  - <avr/interrupt.h>: Interrupts, [227](#)
- ISR\_NAKED
  - <avr/interrupt.h>: Interrupts, [227](#)

- ISR\_NOBLOCK
  - <avr/interrupt.h>: Interrupts, 227
- ISR\_NOGCCISR
  - <avr/interrupt.h>: Interrupts, 227
- ISR\_NOICF
  - <avr/interrupt.h>: Interrupts, 228
- isspace
  - <ctype.h>: Character Operations, 106
- isupper
  - <ctype.h>: Character Operations, 107
- isxdigit
  - <ctype.h>: Character Operations, 107
- itoa
  - <stdlib.h>: General utilities, 177
- labs
  - <stdlib.h>: General utilities, 178
- lcd.h, 329
- ldexp
  - <math.h>: Mathematics, 135
- ldexpf
  - <math.h>: Mathematics, 135
- ldexpl
  - <math.h>: Mathematics, 135
- ldiv
  - <stdlib.h>: General utilities, 178
- ldiv\_t, 322
  - quot, 322
  - rem, 322
- ldtostre
  - <stdlib.h>: General utilities, 178
- ldtostrf
  - <stdlib.h>: General utilities, 178
- lm\_sidereal
  - <time.h>: Time, 203
- localtime
  - <time.h>: Time, 203
- localtime\_r
  - <time.h>: Time, 203
- lock.h, 366
- log
  - <math.h>: Mathematics, 135
- log10
  - <math.h>: Mathematics, 135
- log10f
  - <math.h>: Mathematics, 135
- log10l
  - <math.h>: Mathematics, 136
- logf
  - <math.h>: Mathematics, 136
- logl
  - <math.h>: Mathematics, 136
- longjmp
  - <setjmp.h>: Non-local goto, 143
- loop\_until\_bit\_is\_clear
  - <avr/sfr\_defs.h>: Special function registers, 265
- loop\_until\_bit\_is\_set
  - <avr/sfr\_defs.h>: Special function registers, 265
- lrint
  - <math.h>: Mathematics, 136
- lrintf
  - <math.h>: Mathematics, 136
- lrintl
  - <math.h>: Mathematics, 136
- lround
  - <math.h>: Mathematics, 137
- lroundf
  - <math.h>: Mathematics, 137
- lroundl
  - <math.h>: Mathematics, 137
- ltoa
  - <stdlib.h>: General utilities, 179
- M\_1\_PI
  - <math.h>: Mathematics, 125
- M\_2\_PI
  - <math.h>: Mathematics, 125
- M\_2\_SQRTPI
  - <math.h>: Mathematics, 125
- M\_E
  - <math.h>: Mathematics, 125
- M\_LN10
  - <math.h>: Mathematics, 125
- M\_LN2
  - <math.h>: Mathematics, 125
- M\_LOG10E
  - <math.h>: Mathematics, 125
- M\_LOG2E
  - <math.h>: Mathematics, 126
- M\_PI
  - <math.h>: Mathematics, 126
- M\_PI\_2
  - <math.h>: Mathematics, 126
- M\_PI\_4
  - <math.h>: Mathematics, 126
- M\_SQRT1\_2
  - <math.h>: Mathematics, 126
- M\_SQRT2
  - <math.h>: Mathematics, 126
- malloc
  - <stdlib.h>: General utilities, 179
- math.h, 461, 464
- memcpy
  - <string.h>: Strings, 186
- memcpy\_P
  - <avr/pgmspace.h>: Program Space Utilities, 237
- memchr
  - <string.h>: Strings, 186
- memchr\_P
  - <avr/pgmspace.h>: Program Space Utilities, 237
- memcmp
  - <string.h>: Strings, 186
- memcmp\_P
  - <avr/pgmspace.h>: Program Space Utilities, 238
- memcmp\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 238
- memcpy
  - <string.h>: Strings, 187

- memcpy\_P
  - <avr/pgmspace.h>: Program Space Utilities, 238
- memcpy\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 239
- memmem
  - <string.h>: Strings, 187
- memmem\_P
  - <avr/pgmspace.h>: Program Space Utilities, 239
- memmove
  - <string.h>: Strings, 187
- memrchr
  - <string.h>: Strings, 188
- memrchr\_P
  - <avr/pgmspace.h>: Program Space Utilities, 239
- memset
  - <string.h>: Strings, 188
- mk\_gmtime
  - <time.h>: Time, 203
- mktime
  - <time.h>: Time, 203
- modf
  - <math.h>: Mathematics, 137
- modff
  - <math.h>: Mathematics, 138
- modfl
  - <math.h>: Mathematics, 138
- month\_length
  - <time.h>: Time, 203
- moon\_phase
  - <time.h>: Time, 204
- NAN
  - <math.h>: Mathematics, 126
- nan
  - <math.h>: Mathematics, 126
- nanf
  - <math.h>: Mathematics, 126
- nanl
  - <math.h>: Mathematics, 126
- NONATOMIC\_BLOCK
  - <util/atomic.h>: Atomically and Non-Atomically Executed Code Blocks, 278
- NONATOMIC\_FORCEOFF
  - <util/atomic.h>: Atomically and Non-Atomically Executed Code Blocks, 278
- NONATOMIC\_RESTORESTATE
  - <util/atomic.h>: Atomically and Non-Atomically Executed Code Blocks, 278
- NTP\_OFFSET
  - <time.h>: Time, 199
- ONE\_DAY
  - <time.h>: Time, 200
- ONE\_DEGREE
  - <time.h>: Time, 200
- ONE\_HOUR
  - <time.h>: Time, 200
- outb
  - <compat/deprecated.h>: Deprecated items, 292
- outp
  - <compat/deprecated.h>: Deprecated items, 293
- parity.h, 540
- parity\_even\_bit
  - <util/parity.h>: Parity bit generation, 283
- pgm\_get\_far\_address
  - <avr/pgmspace.h>: Program Space Utilities, 234
- pgm\_read< T >
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_byte
  - <avr/pgmspace.h>: Program Space Utilities, 235
- pgm\_read\_byte\_far
  - <avr/pgmspace.h>: Program Space Utilities, 235
- pgm\_read\_byte\_near
  - <avr/pgmspace.h>: Program Space Utilities, 235
- pgm\_read\_char
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_char\_far
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_double
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_double\_far
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_dword
  - <avr/pgmspace.h>: Program Space Utilities, 235
- pgm\_read\_dword\_far
  - <avr/pgmspace.h>: Program Space Utilities, 236
- pgm\_read\_dword\_near
  - <avr/pgmspace.h>: Program Space Utilities, 236
- pgm\_read\_far< T >
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_float
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_float\_far
  - <avr/pgmspace.h>: Program Space Utilities, 240
- pgm\_read\_float\_near
  - <avr/pgmspace.h>: Program Space Utilities, 236
- pgm\_read\_i16
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i16\_far
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i24
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i24\_far
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i32
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i32\_far
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i64
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i64\_far
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i8
  - <avr/pgmspace.h>: Program Space Utilities, 241
- pgm\_read\_i8\_far
  - <avr/pgmspace.h>: Program Space Utilities, 242
- pgm\_read\_int

- `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_int_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_long`
  - `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_long_double`
  - `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_long_double_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_long_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_long_long`
  - `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_long_long_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [242](#)
- `pgm_read_ptr`
  - `<avr/pgmspace.h>`: Program Space Utilities, [236](#)
- `pgm_read_ptr_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [236](#)
- `pgm_read_ptr_near`
  - `<avr/pgmspace.h>`: Program Space Utilities, [236](#)
- `pgm_read_qword`
  - `<avr/pgmspace.h>`: Program Space Utilities, [236](#)
- `pgm_read_qword_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [236](#)
- `pgm_read_qword_near`
  - `<avr/pgmspace.h>`: Program Space Utilities, [236](#)
- `pgm_read_short`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_short_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_signed`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_signed_char`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_signed_char_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_signed_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_signed_int`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_signed_int_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_u16`
  - `<avr/pgmspace.h>`: Program Space Utilities, [243](#)
- `pgm_read_u16_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u24`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u24_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u32`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u32_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u64`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u64_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u8`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_u8_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [244](#)
- `pgm_read_unsigned`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_char`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_char_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_int`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_int_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_long`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_long_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_long_long`
  - `<avr/pgmspace.h>`: Program Space Utilities, [245](#)
- `pgm_read_unsigned_long_long_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [246](#)
- `pgm_read_unsigned_short`
  - `<avr/pgmspace.h>`: Program Space Utilities, [246](#)
- `pgm_read_unsigned_short_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [246](#)
- `pgm_read_word`
  - `<avr/pgmspace.h>`: Program Space Utilities, [236](#)
- `pgm_read_word_far`
  - `<avr/pgmspace.h>`: Program Space Utilities, [237](#)
- `pgm_read_word_near`
  - `<avr/pgmspace.h>`: Program Space Utilities, [237](#)
- `pgmspace.h`, [369](#), [372](#)
- `portpins.h`, [396](#)
- `pow`
  - `<math.h>`: Mathematics, [138](#)
- `power.h`, [402](#), [403](#)
- `power_all_disable`
  - `<avr/power.h>`: Power Reduction Management, [262](#)
- `power_all_enable`
  - `<avr/power.h>`: Power Reduction Management, [262](#)
- `powf`
  - `<math.h>`: Mathematics, [138](#)
- `powl`
  - `<math.h>`: Mathematics, [138](#)
- `PRId16`
  - `<inttypes.h>`: Integer Type conversions, [111](#)
- `PRId32`
  - `<inttypes.h>`: Integer Type conversions, [111](#)
- `PRId8`
  - `<inttypes.h>`: Integer Type conversions, [111](#)
- `PRIdFAST16`
  - `<inttypes.h>`: Integer Type conversions, [111](#)



- PRIdFAST32
  - <inttypes.h>: Integer Type conversions, [111](#)
- PRIdFAST8
  - <inttypes.h>: Integer Type conversions, [111](#)
- PRIdLEAST16
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRIdLEAST32
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRIdLEAST8
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRIdPTR
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRli16
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRli32
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRli8
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRliFAST16
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRliFAST32
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRliFAST8
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRliLEAST16
  - <inttypes.h>: Integer Type conversions, [112](#)
- PRliLEAST32
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRliLEAST8
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRliPTR
  - <inttypes.h>: Integer Type conversions, [113](#)
- printf
  - <stdio.h>: Standard IO facilities, [165](#)
- printf\_P
  - <stdio.h>: Standard IO facilities, [165](#)
- PRlo16
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRlo32
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRlo8
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRloFAST16
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRloFAST32
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRloFAST8
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRloLEAST16
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRloLEAST32
  - <inttypes.h>: Integer Type conversions, [113](#)
- PRloLEAST8
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRloPTR
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRlu16
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRlu32
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRlu8
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRluFAST16
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRluFAST32
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRluFAST8
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRluLEAST16
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRluLEAST32
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRluLEAST8
  - <inttypes.h>: Integer Type conversions, [114](#)
- PRluPTR
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIX16
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIx16
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIX32
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIx32
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIX8
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIx8
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIXFAST16
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIxFAST16
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIXFAST32
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIxFAST32
  - <inttypes.h>: Integer Type conversions, [115](#)
- PRIXFAST8
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIxFAST8
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIXLEAST16
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIxLEAST16
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIXLEAST32
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIxLEAST32
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIXLEAST8
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIxLEAST8
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIXPTR
  - <inttypes.h>: Integer Type conversions, [116](#)
- PRIxPTR
  - <inttypes.h>: Integer Type conversions, [116](#)

- PROGMEM
  - <avr/pgmspace.h>: Program Space Utilities, [237](#)
- PROGMEM\_FAR
  - <avr/pgmspace.h>: Program Space Utilities, [237](#)
- project.h, [325](#)
- PSTR
  - <avr/pgmspace.h>: Program Space Utilities, [237](#)
- PSTR\_FAR
  - <avr/pgmspace.h>: Program Space Utilities, [237](#)
- PTRDIFF\_MAX
  - <stdint.h>: Standard Integer Types, [150](#)
- PTRDIFF\_MIN
  - <stdint.h>: Standard Integer Types, [150](#)
- putc
  - <stdio.h>: Standard IO facilities, [161](#)
- putchar
  - <stdio.h>: Standard IO facilities, [161](#)
- puts
  - <stdio.h>: Standard IO facilities, [165](#)
- puts\_P
  - <stdio.h>: Standard IO facilities, [165](#)
- qsort
  - <stdlib.h>: General utilities, [179](#)
- quot
  - div\_t, [322](#)
  - ldiv\_t, [322](#)
- rand
  - <stdlib.h>: General utilities, [180](#)
- RAND\_MAX
  - <stdlib.h>: General utilities, [174](#)
- rand\_r
  - <stdlib.h>: General utilities, [180](#)
- random
  - <stdlib.h>: General utilities, [180](#)
- RANDOM\_MAX
  - <stdlib.h>: General utilities, [174](#)
- random\_r
  - <stdlib.h>: General utilities, [180](#)
- realloc
  - <stdlib.h>: General utilities, [180](#)
- rem
  - div\_t, [322](#)
  - ldiv\_t, [322](#)
- reti
  - <avr/interrupt.h>: Interrupts, [228](#)
- round
  - <math.h>: Mathematics, [139](#)
- roundf
  - <math.h>: Mathematics, [139](#)
- roundl
  - <math.h>: Mathematics, [139](#)
- sbi
  - <compat/deprecated.h>: Deprecated items, [293](#)
- scanf
  - <stdio.h>: Standard IO facilities, [165](#)
- scanf\_P
  - <stdio.h>: Standard IO facilities, [165](#)
- SCNd16
  - <inttypes.h>: Integer Type conversions, [116](#)
- SCNd32
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNd8
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNdFAST16
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNdFAST32
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNdFAST8
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNdLEAST16
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNdLEAST32
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNdLEAST8
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNdPTR
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNi16
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNi32
  - <inttypes.h>: Integer Type conversions, [117](#)
- SCNi8
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNiFAST16
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNiFAST32
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNiFAST8
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNiLEAST16
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNiLEAST32
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNiLEAST8
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNiPTR
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNo16
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNo32
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNo8
  - <inttypes.h>: Integer Type conversions, [118](#)
- SCNoFAST16
  - <inttypes.h>: Integer Type conversions, [119](#)
- SCNoFAST32
  - <inttypes.h>: Integer Type conversions, [119](#)
- SCNoFAST8
  - <inttypes.h>: Integer Type conversions, [119](#)
- SCNoLEAST16
  - <inttypes.h>: Integer Type conversions, [119](#)
- SCNoLEAST32
  - <inttypes.h>: Integer Type conversions, [119](#)
- SCNoLEAST8

- <inttypes.h>: Integer Type conversions, 119
- SCNoPTR
  - <inttypes.h>: Integer Type conversions, 119
- SCNu16
  - <inttypes.h>: Integer Type conversions, 119
- SCNu32
  - <inttypes.h>: Integer Type conversions, 119
- SCNu8
  - <inttypes.h>: Integer Type conversions, 119
- SCNuFAST16
  - <inttypes.h>: Integer Type conversions, 119
- SCNuFAST32
  - <inttypes.h>: Integer Type conversions, 120
- SCNuFAST8
  - <inttypes.h>: Integer Type conversions, 120
- SCNuLEAST16
  - <inttypes.h>: Integer Type conversions, 120
- SCNuLEAST32
  - <inttypes.h>: Integer Type conversions, 120
- SCNuLEAST8
  - <inttypes.h>: Integer Type conversions, 120
- SCNuPTR
  - <inttypes.h>: Integer Type conversions, 120
- SCNx16
  - <inttypes.h>: Integer Type conversions, 120
- SCNx32
  - <inttypes.h>: Integer Type conversions, 120
- SCNx8
  - <inttypes.h>: Integer Type conversions, 120
- SCNxFAST16
  - <inttypes.h>: Integer Type conversions, 120
- SCNxFAST32
  - <inttypes.h>: Integer Type conversions, 120
- SCNxFAST8
  - <inttypes.h>: Integer Type conversions, 121
- SCNxLEAST16
  - <inttypes.h>: Integer Type conversions, 121
- SCNxLEAST32
  - <inttypes.h>: Integer Type conversions, 121
- SCNxLEAST8
  - <inttypes.h>: Integer Type conversions, 121
- SCNxPTR
  - <inttypes.h>: Integer Type conversions, 121
- sei
  - <avr/interrupt.h>: Interrupts, 228
- set\_dst
  - <time.h>: Time, 204
- set\_position
  - <time.h>: Time, 204
- set\_system\_time
  - <time.h>: Time, 204
- set\_zone
  - <time.h>: Time, 204
- setbaud.h, 541
- setjmp
  - <setjmp.h>: Non-local goto, 144
- setjmp.h, 472
- sfr\_defs.h, 424
- SIG\_ATOMIC\_MAX
  - <stdint.h>: Standard Integer Types, 150
- SIG\_ATOMIC\_MIN
  - <stdint.h>: Standard Integer Types, 150
- SIGNAL
  - <avr/interrupt.h>: Interrupts, 228
- signal.h, 427
- signature.h, 428
- signbit
  - <math.h>: Mathematics, 139
- signbitf
  - <math.h>: Mathematics, 139
- signbitl
  - <math.h>: Mathematics, 140
- sin
  - <math.h>: Mathematics, 140
- sinf
  - <math.h>: Mathematics, 140
- sinh
  - <math.h>: Mathematics, 140
- sinhf
  - <math.h>: Mathematics, 140
- sinhl
  - <math.h>: Mathematics, 140
- sinl
  - <math.h>: Mathematics, 140
- SIZE\_MAX
  - <stdint.h>: Standard Integer Types, 150
- sleep.h, 429
- sleep\_bod\_disable
  - <avr/sleep.h>: Power Management and Sleep Modes, 266
- sleep\_cpu
  - <avr/sleep.h>: Power Management and Sleep Modes, 266
- sleep\_disable
  - <avr/sleep.h>: Power Management and Sleep Modes, 266
- sleep\_enable
  - <avr/sleep.h>: Power Management and Sleep Modes, 266
- sleep\_mode
  - <avr/sleep.h>: Power Management and Sleep Modes, 267
- snprintf
  - <stdio.h>: Standard IO facilities, 165
- snprintf\_P
  - <stdio.h>: Standard IO facilities, 166
- solar\_declination
  - <time.h>: Time, 205
- solar\_declinationf
  - <time.h>: Time, 205
- solar\_declinationl
  - <time.h>: Time, 205
- solar\_noon
  - <time.h>: Time, 205
- sprintf
  - <stdio.h>: Standard IO facilities, 166

- sprintf\_P
  - <stdio.h>: Standard IO facilities, 166
- sqrt
  - <math.h>: Mathematics, 140
- sqrtf
  - <math.h>: Mathematics, 140
- sqrtl
  - <math.h>: Mathematics, 141
- square
  - <math.h>: Mathematics, 141
- squaref
  - <math.h>: Mathematics, 141
- squarel
  - <math.h>: Mathematics, 141
- srand
  - <stdlib.h>: General utilities, 181
- random
  - <stdlib.h>: General utilities, 181
- sscanf
  - <stdio.h>: Standard IO facilities, 166
- sscanf\_P
  - <stdio.h>: Standard IO facilities, 166
- stderr
  - <stdio.h>: Standard IO facilities, 161
- stdin
  - <stdio.h>: Standard IO facilities, 161
- stdint.h, 474, 477
- stdio.h, 485, 486
- stdio\_private.h, 551
- stdlib.h, 498, 500
- stdlib\_private.h, 553
- stdout
  - <stdio.h>: Standard IO facilities, 161
- strcasemp
  - <string.h>: Strings, 188
- strcasemp\_P
  - <avr/pgmspace.h>: Program Space Utilities, 246
- strcasemp\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 246
- strcasestr
  - <string.h>: Strings, 189
- strcasestr\_P
  - <avr/pgmspace.h>: Program Space Utilities, 247
- strcat
  - <string.h>: Strings, 189
- strcat\_P
  - <avr/pgmspace.h>: Program Space Utilities, 247
- strcat\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 247
- strchr
  - <string.h>: Strings, 189
- strchr\_P
  - <avr/pgmspace.h>: Program Space Utilities, 248
- strchr\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 248
- strchrnul
  - <string.h>: Strings, 189
- strchrnul\_P
  - <avr/pgmspace.h>: Program Space Utilities, 248
- strdup
  - <string.h>: Strings, 191
- strftime
  - <time.h>: Time, 205
- string.h, 509, 510
- strlcat
  - <string.h>: Strings, 191
- strlcat\_P
  - <avr/pgmspace.h>: Program Space Utilities, 250
- strlcat\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 250
- strlcpy
  - <string.h>: Strings, 191
- strlcpy\_P
  - <avr/pgmspace.h>: Program Space Utilities, 251
- strlcpy\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 251
- strlen
  - <string.h>: Strings, 192
- strlen\_P
  - <avr/pgmspace.h>: Program Space Utilities, 251
- strlen\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 252
- strlwr
  - <string.h>: Strings, 192
- strncasemp
  - <string.h>: Strings, 192
- strncasemp\_P
  - <avr/pgmspace.h>: Program Space Utilities, 252
- strncasemp\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 253
- strncat
  - <string.h>: Strings, 193
- strncat\_P
  - <avr/pgmspace.h>: Program Space Utilities, 253
- strncat\_PF
  - <avr/pgmspace.h>: Program Space Utilities, 253
- strncmp
  - <string.h>: Strings, 193
- strncmp\_P
  - <avr/pgmspace.h>: Program Space Utilities, 254

- strncmp\_PF
  - <avr/pgmspace.h>: Program Space Utilities, [254](#)
- strncpy
  - <string.h>: Strings, [193](#)
- strncpy\_P
  - <avr/pgmspace.h>: Program Space Utilities, [254](#)
- strncpy\_PF
  - <avr/pgmspace.h>: Program Space Utilities, [255](#)
- strndup
  - <string.h>: Strings, [194](#)
- strlen
  - <string.h>: Strings, [194](#)
- strlen\_P
  - <avr/pgmspace.h>: Program Space Utilities, [255](#)
- strlen\_PF
  - <avr/pgmspace.h>: Program Space Utilities, [255](#)
- strpbrk
  - <string.h>: Strings, [194](#)
- strpbrk\_P
  - <avr/pgmspace.h>: Program Space Utilities, [256](#)
- strrchr
  - <string.h>: Strings, [195](#)
- strrchr\_P
  - <avr/pgmspace.h>: Program Space Utilities, [256](#)
- strrev
  - <string.h>: Strings, [195](#)
- strsep
  - <string.h>: Strings, [195](#)
- strsep\_P
  - <avr/pgmspace.h>: Program Space Utilities, [256](#)
- strspn
  - <string.h>: Strings, [195](#)
- strspn\_P
  - <avr/pgmspace.h>: Program Space Utilities, [257](#)
- strstr
  - <string.h>: Strings, [196](#)
- strstr\_P
  - <avr/pgmspace.h>: Program Space Utilities, [257](#)
- strstr\_PF
  - <avr/pgmspace.h>: Program Space Utilities, [257](#)
- strtod
  - <stdlib.h>: General utilities, [181](#)
- strtof
  - <stdlib.h>: General utilities, [181](#)
- strtok
  - <string.h>: Strings, [196](#)
- strtok\_P
  - <avr/pgmspace.h>: Program Space Utilities, [258](#)
- strtok\_r
  - <string.h>: Strings, [196](#)
- strtok\_rP
  - <avr/pgmspace.h>: Program Space Utilities, [258](#)
- strtol
  - <stdlib.h>: General utilities, [181](#)
- strtold
  - <stdlib.h>: General utilities, [182](#)
- strtoul
  - <stdlib.h>: General utilities, [182](#)
- strupr
  - <string.h>: Strings, [197](#)
- sun\_rise
  - <time.h>: Time, [205](#)
- sun\_set
  - <time.h>: Time, [205](#)
- supported devices, [2](#)
- system\_tick
  - <time.h>: Time, [206](#)
- tan
  - <math.h>: Mathematics, [141](#)
- tanf
  - <math.h>: Mathematics, [141](#)
- tanh
  - <math.h>: Mathematics, [142](#)
- tanhf
  - <math.h>: Mathematics, [142](#)
- tanhl
  - <math.h>: Mathematics, [142](#)
- tanl
  - <math.h>: Mathematics, [142](#)
- time
  - <time.h>: Time, [206](#)
- time.h, [517](#), [518](#)
- time\_t
  - <time.h>: Time, [200](#)
- timer\_enable\_int
  - <compat/deprecated.h>: Deprecated items, [293](#)
- tm, [323](#)
  - tm\_hour, [323](#)
  - tm\_isdst, [323](#)
  - tm\_mday, [323](#)
  - tm\_min, [323](#)
  - tm\_mon, [323](#)
  - tm\_sec, [324](#)
  - tm\_wday, [324](#)
  - tm\_yday, [324](#)
  - tm\_year, [324](#)
- tm\_hour
  - tm, [323](#)
- tm\_isdst
  - tm, [323](#)
- tm\_mday
  - tm, [323](#)
- tm\_min
  - tm, [323](#)
- tm\_mon
  - tm, [323](#)
- tm\_sec
  - tm, [324](#)
- tm\_wday
  - tm, [324](#)
- tm\_yday
  - tm, [324](#)
- tm\_year
  - tm, [324](#)
- toascii
  - <ctype.h>: Character Operations, [107](#)

- tolower
  - <ctype.h>: Character Operations, [107](#)
- tools, optional, [77](#)
- tools, required, [76](#)
- toupper
  - <ctype.h>: Character Operations, [107](#)
- trunc
  - <math.h>: Mathematics, [142](#)
- truncf
  - <math.h>: Mathematics, [142](#)
- trunc1
  - <math.h>: Mathematics, [142](#)
- TW\_BUS\_ERROR
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MR\_ARB\_LOST
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MR\_DATA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MR\_DATA\_NACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MR\_SLA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MR\_SLA\_NACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MT\_ARB\_LOST
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MT\_DATA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MT\_DATA\_NACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MT\_SLA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_MT\_SLA\_NACK
  - <util/twi.h>: TWI bit mask definitions, [287](#)
- TW\_NO\_INFO
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_READ
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_REP\_START
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_ARB\_LOST\_GCALL\_ACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_ARB\_LOST\_SLA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_DATA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_DATA\_NACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_GCALL\_ACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_GCALL\_DATA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_GCALL\_DATA\_NACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_SLA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [288](#)
- TW\_SR\_STOP
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_ST\_ARB\_LOST\_SLA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_ST\_DATA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_ST\_DATA\_NACK
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_ST\_LAST\_DATA
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_ST\_SLA\_ACK
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_START
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_STATUS
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_STATUS\_MASK
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- TW\_WRITE
  - <util/twi.h>: TWI bit mask definitions, [289](#)
- twi.h, [544](#), [545](#)
- uart.h, [329](#)
- UBRR\_VALUE
  - <util/setbaud.h>: Helper macros for baud rate calculations, [285](#)
- UBRRH\_VALUE
  - <util/setbaud.h>: Helper macros for baud rate calculations, [285](#)
- UBRRL\_VALUE
  - <util/setbaud.h>: Helper macros for baud rate calculations, [285](#)
- UINT16\_C
  - <stdint.h>: Standard Integer Types, [151](#)
- UINT16\_MAX
  - <stdint.h>: Standard Integer Types, [151](#)
- uint16\_t
  - <stdint.h>: Standard Integer Types, [154](#)
- UINT32\_C
  - <stdint.h>: Standard Integer Types, [151](#)
- UINT32\_MAX
  - <stdint.h>: Standard Integer Types, [151](#)
- uint32\_t
  - <stdint.h>: Standard Integer Types, [154](#)
- UINT64\_C
  - <stdint.h>: Standard Integer Types, [151](#)
- UINT64\_MAX
  - <stdint.h>: Standard Integer Types, [151](#)
- uint64\_t
  - <stdint.h>: Standard Integer Types, [154](#)
- UINT8\_C
  - <stdint.h>: Standard Integer Types, [151](#)
- UINT8\_MAX
  - <stdint.h>: Standard Integer Types, [151](#)
- uint8\_t
  - <stdint.h>: Standard Integer Types, [154](#)
- uint\_farptr\_t
  - <inttypes.h>: Integer Type conversions, [121](#)
- UINT\_FAST16\_MAX
  - <stdint.h>: Standard Integer Types, [151](#)
- uint\_fast16\_t

- `<stdint.h>`: Standard Integer Types, 154
- UINT\_FAST32\_MAX
  - `<stdint.h>`: Standard Integer Types, 151
- uint\_fast32\_t
  - `<stdint.h>`: Standard Integer Types, 154
- UINT\_FAST64\_MAX
  - `<stdint.h>`: Standard Integer Types, 151
- uint\_fast64\_t
  - `<stdint.h>`: Standard Integer Types, 155
- UINT\_FAST8\_MAX
  - `<stdint.h>`: Standard Integer Types, 152
- uint\_fast8\_t
  - `<stdint.h>`: Standard Integer Types, 155
- UINT\_LEAST16\_MAX
  - `<stdint.h>`: Standard Integer Types, 152
- uint\_least16\_t
  - `<stdint.h>`: Standard Integer Types, 155
- UINT\_LEAST32\_MAX
  - `<stdint.h>`: Standard Integer Types, 152
- uint\_least32\_t
  - `<stdint.h>`: Standard Integer Types, 155
- UINT\_LEAST64\_MAX
  - `<stdint.h>`: Standard Integer Types, 152
- uint\_least64\_t
  - `<stdint.h>`: Standard Integer Types, 155
- UINT\_LEAST8\_MAX
  - `<stdint.h>`: Standard Integer Types, 152
- uint\_least8\_t
  - `<stdint.h>`: Standard Integer Types, 155
- UINTMAX\_C
  - `<stdint.h>`: Standard Integer Types, 152
- UINTMAX\_MAX
  - `<stdint.h>`: Standard Integer Types, 152
- uintmax\_t
  - `<stdint.h>`: Standard Integer Types, 155
- UINTPTR\_MAX
  - `<stdint.h>`: Standard Integer Types, 152
- uintptr\_t
  - `<stdint.h>`: Standard Integer Types, 155
- ultoa
  - `<stdlib.h>`: General utilities, 182
- ungetc
  - `<stdio.h>`: Standard IO facilities, 166
- UNIX\_OFFSET
  - `<time.h>`: Time, 200
- usa\_dst
  - `<util/usa_dst.h>`: Daylight Saving function for the USA., 290
- usa\_dst.h, 548
- USE\_2X
  - `<util/setbaud.h>`: Helper macros for baud rate calculations, 285
- Using the standard IO facilities, 312
- utoa
  - `<stdlib.h>`: General utilities, 183
- version.h, 433
- vfprintf
  - `<stdio.h>`: Standard IO facilities, 167
- vfprintf\_P
  - `<stdio.h>`: Standard IO facilities, 169
- vfscanf
  - `<stdio.h>`: Standard IO facilities, 169
- vfscanf\_P
  - `<stdio.h>`: Standard IO facilities, 170
- vprintf
  - `<stdio.h>`: Standard IO facilities, 171
- vscanf
  - `<stdio.h>`: Standard IO facilities, 171
- vsprintf
  - `<stdio.h>`: Standard IO facilities, 171
- vsprintf\_P
  - `<stdio.h>`: Standard IO facilities, 171
- vsprintf\_P
  - `<stdio.h>`: Standard IO facilities, 171
- wdt.h, 434, 435
- wdt\_enable
  - `<avr/wdt.h>`: Watchdog timer handling, 271
- wdt\_reset
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_120MS
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_15MS
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_1S
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_250MS
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_2S
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_30MS
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_4S
  - `<avr/wdt.h>`: Watchdog timer handling, 272
- WDTO\_500MS
  - `<avr/wdt.h>`: Watchdog timer handling, 273
- WDTO\_60MS
  - `<avr/wdt.h>`: Watchdog timer handling, 273
- WDTO\_8S
  - `<avr/wdt.h>`: Watchdog timer handling, 273
- week
  - week\_date, 325
- week\_date, 324
  - day, 324
  - week, 325
  - year, 325
- week\_of\_month
  - `<time.h>`: Time, 206
- week\_of\_year
  - `<time.h>`: Time, 206
- xmega.h, 442
- xtoa\_fast.h, 552
- year

`week_date`, [325](#)