

avr-libc

Generated by Doxygen 1.9.4

1 AVR Libc	2
1.1 Introduction	2
1.2 General information about this library	2
1.3 Supported Devices	2
1.4 avr-libc License	11
2 Toolchain Overview	13
2.1 Introduction	13
2.2 FSF and GNU	13
2.3 GCC	13
2.4 GNU Binutils	14
2.5 avr-libc	15
2.6 Building Software	15
2.7 AVRDUDE	16
2.8 GDB / Insight / DDD	16
2.9 AVaRICE	16
2.10 SimulAVR	16
2.11 Utilities	16
2.12 Toolchain Distributions (Distros)	16
2.13 Open Source	17
3 Memory Areas and Using malloc()	17
3.1 Introduction	17
3.2 Internal vs. external RAM	18
3.3 Tunables for malloc()	18
3.4 Implementation details	19
4 Memory Sections	20
4.1 The .text Section	21
4.2 The .data Section	21
4.3 The .bss Section	21
4.4 The .eeprom Section	21
4.5 The .noinit Section	21
4.6 The .initN Sections	22
4.7 The .finiN Sections	23
4.8 The .note.gnu.avr.deviceinfo Section	24
4.9 Using Sections in Assembler Code	24
4.10 Using Sections in C Code	24
5 Data in Program Space	25
5.1 Introduction	25
5.2 A Note On const	25
5.3 Storing and Retrieving Data in the Program Space	26
5.4 Storing and Retrieving Strings in the Program Space	27

5.5 Caveats	28
6 avr-libc and assembler programs	28
6.1 Introduction	28
6.2 Invoking the compiler	29
6.3 Example program	29
6.4 Pseudo-ops and operators	31
7 Inline Assembler Cookbook	32
7.1 GCC asm Statement	33
7.2 Assembler Code	34
7.3 Input and Output Operands	35
7.4 Clobbers	38
7.5 Assembler Macros	39
7.6 C Stub Functions	40
7.7 C Names Used in Assembler Code	40
7.8 Links	41
8 How to Build a Library	41
8.1 Introduction	41
8.2 How the Linker Works	41
8.3 How to Design a Library	41
8.4 Creating a Library	42
8.5 Using a Library	42
9 Benchmarks	43
9.1 A few of libc functions.	43
9.2 Math functions.	44
10 Porting From IAR to AVR GCC	45
10.1 Introduction	45
10.2 Registers	45
10.3 Interrupt Service Routines (ISRs)	46
10.4 Intrinsic Routines	46
10.5 Flash Variables	46
10.6 Non-Returning main()	47
10.7 Locking Registers	48
11 Frequently Asked Questions	48
11.1 FAQ Index	48
11.2 My program doesn't recognize a variable updated within an interrupt routine	49
11.3 I get "undefined reference to..." for functions like "sin()"	49
11.4 How to permanently bind a variable to a register?	50
11.5 How to modify MCUCR or WDTCR early?	50

11.6 What is all this <code>_BV()</code> stuff about?	51
11.7 Can I use C++ on the AVR?	51
11.8 Shouldn't I initialize all my variables?	52
11.9 Why do some 16-bit timer registers sometimes get trashed?	52
11.10 How do I use a <code>#define</code> 'd constant in an asm statement?	53
11.11 Why does the PC randomly jump around when single-stepping through my program in <code>avr-gdb</code> ?	53
11.12 How do I trace an assembler file in <code>avr-gdb</code> ?	54
11.13 How do I pass an IO port as a parameter to a function?	55
11.14 What registers are used by the C compiler?	56
11.15 How do I put an array of strings completely in ROM?	57
11.16 How to use external RAM?	59
11.17 Which <code>-O</code> flag to use?	59
11.18 How do I relocate code to a fixed address?	60
11.19 My UART is generating nonsense! My ATmega128 keeps crashing! Port F is completely broken!	60
11.20 Why do all my "foo...bar" strings eat up the SRAM?	61
11.21 Why does the compiler compile an 8-bit operation that uses bitwise operators into a 16-bit operation in assembly?	61
11.22 How to detect RAM memory and variable overlap problems?	62
11.23 Is it really impossible to program the ATtinyXX in C?	62
11.24 What is this "clock skew detected" message?	62
11.25 Why are (many) interrupt flags cleared by writing a logical 1?	63
11.26 Why have "programmed" fuses the bit value 0?	63
11.27 Which AVR-specific assembler operators are available?	63
11.28 Why are interrupts re-enabled in the middle of writing the stack pointer?	64
11.29 Why are there five different linker scripts?	64
11.30 How to add a raw binary image to linker output?	64
11.31 How do I perform a software reset of the AVR?	65
11.32 I am using floating point math. Why is the compiled code so big? Why does my code not work?	66
11.33 What pitfalls exist when writing reentrant code?	66
11.34 Why are some addresses of the EEPROM corrupted (usually address zero)?	68
11.35 Why is my baud rate wrong?	69
11.36 On a device with more than 128 KiB of flash, how to make function pointers work?	69
11.37 Why is assigning ports in a "chain" a bad idea?	69
12 Building and Installing the GNU Tool Chain	69
12.1 Building and Installing under Linux, FreeBSD, and Others	70
12.2 Required Tools	70
12.3 Optional Tools	71
12.4 GNU Binutils for the AVR target	71
12.5 GCC for the AVR target	72
12.6 AVR LibC	73
12.7 AVRDUDE	73
12.8 GDB for the AVR target	74

12.9 SimulAVR	74
12.10 AVaRICE	74
12.11 Building and Installing under Windows	75
12.12 Tools Required for Building the Toolchain for Windows	75
12.13 Building the Toolchain for Windows	77
13 Using the GNU tools	82
13.1 Options for the C compiler avr-gcc	82
13.1.1 Machine-specific options for the AVR	82
13.1.2 Selected general compiler options	90
13.2 Options for the assembler avr-as	92
13.2.1 Machine-specific assembler options	92
13.2.2 Examples for assembler options passed through the C compiler	93
13.3 Controlling the linker avr-ld	93
13.3.1 Selected linker options	93
13.3.2 Passing linker options from the C compiler	94
14 Compiler optimization	95
14.1 Problems with reordering code	95
15 Using the avrdude program	97
16 Release Numbering and Methodology	98
16.1 Release Version Numbering Scheme	98
16.2 Releasing AVR Libc	98
16.2.1 Creating an SVN branch	99
16.2.2 Making a release	99
17 Acknowledgments	101
18 Todo List	101
19 Deprecated List	101
20 Module Index	102
20.1 Modules	102
21 Data Structure Index	103
21.1 Data Structures	103
22 File Index	104
22.1 File List	104
23 Module Documentation	106
23.1 <alloca.h>: Allocate space in the stack	106
23.1.1 Detailed Description	106

23.1.2 Function Documentation	106
23.2 <assert.h>: Diagnostics	106
23.3 <ctype.h>: Character Operations	107
23.3.1 Detailed Description	107
23.3.2 Function Documentation	107
23.4 <errno.h>: System Errors	109
23.4.1 Detailed Description	109
23.4.2 Macro Definition Documentation	110
23.4.3 Variable Documentation	110
23.5 <inttypes.h>: Integer Type conversions	110
23.5.1 Detailed Description	113
23.5.2 Macro Definition Documentation	113
23.5.3 Typedef Documentation	123
23.6 <math.h>: Mathematics	123
23.6.1 Detailed Description	124
23.6.2 Macro Definition Documentation	124
23.7 <setjmp.h>: Non-local goto	124
23.7.1 Detailed Description	124
23.7.2 Function Documentation	125
23.8 <stdint.h>: Standard Integer Types	126
23.8.1 Detailed Description	128
23.8.2 Macro Definition Documentation	129
23.8.3 Typedef Documentation	134
23.9 <stdio.h>: Standard IO facilities	137
23.9.1 Detailed Description	137
23.9.2 Function Documentation	139
23.10 <stdlib.h>: General utilities	140
23.10.1 Detailed Description	140
23.10.2 Macro Definition Documentation	140
23.10.3 Function Documentation	141
23.11 <string.h>: Strings	145
23.11.1 Detailed Description	145
23.11.2 Macro Definition Documentation	146
23.11.3 Function Documentation	146
23.12 <time.h>: Time	158
23.12.1 Detailed Description	158
23.12.2 Typedef Documentation	159
23.13 <avr/boot.h>: Bootloader Support Utilities	160
23.13.1 Detailed Description	160
23.13.2 Macro Definition Documentation	161
23.14 <avr/cpufunc.h>: Special AVR CPU functions	165
23.14.1 Detailed Description	165

23.14.2 Macro Definition Documentation	165
23.14.3 Function Documentation	166
23.15 <avr/eeprom.h>: EEPROM handling	166
23.15.1 Detailed Description	167
23.15.2 Macro Definition Documentation	167
23.15.3 Function Documentation	168
23.16 <avr/fuse.h>: Fuse Support	170
23.17 <avr/interrupt.h>: Interrupts	173
23.17.1 Detailed Description	173
23.17.2 Macro Definition Documentation	191
23.18 <avr/io.h>: AVR device-specific IO definitions	194
23.18.1 Detailed Description	194
23.18.2 Macro Definition Documentation	195
23.19 <avr/lock.h>: Lockbit Support	195
23.20 <avr/pgmspace.h>: Program Space Utilities	197
23.20.1 Detailed Description	198
23.20.2 Macro Definition Documentation	199
23.20.3 Typedef Documentation	203
23.20.4 Function Documentation	206
23.21 <avr/power.h>: Power Reduction Management	221
23.21.1 Detailed Description	222
23.21.2 Function Documentation	225
23.22 Additional notes from <avr/sfr_defs.h>	225
23.23 <avr/sfr_defs.h>: Special function registers	226
23.23.1 Detailed Description	227
23.23.2 Macro Definition Documentation	227
23.24 <avr/signature.h>: Signature Support	228
23.25 <avr/sleep.h>: Power Management and Sleep Modes	229
23.25.1 Detailed Description	229
23.25.2 Function Documentation	230
23.26 <avr/version.h>: avr-libc version macros	230
23.26.1 Detailed Description	231
23.26.2 Macro Definition Documentation	231
23.27 <avr/wdt.h>: Watchdog timer handling	232
23.27.1 Detailed Description	232
23.27.2 Macro Definition Documentation	233
23.27.3 Function Documentation	234
23.28 <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks	235
23.28.1 Detailed Description	235
23.28.2 Macro Definition Documentation	236
23.29 <util/crc16.h>: CRC Computations	237
23.29.1 Detailed Description	237

23.29.2 Function Documentation	238
23.30 <util/delay.h>: Convenience functions for busy-wait delay loops	241
23.30.1 Detailed Description	241
23.30.2 Macro Definition Documentation	241
23.30.3 Function Documentation	242
23.31 <util/delay_basic.h>: Basic busy-wait delay loops	243
23.31.1 Detailed Description	243
23.31.2 Function Documentation	243
23.32 <util/parity.h>: Parity bit generation	243
23.32.1 Detailed Description	244
23.32.2 Macro Definition Documentation	244
23.33 <util/setbaud.h>: Helper macros for baud rate calculations	244
23.33.1 Detailed Description	244
23.33.2 Macro Definition Documentation	245
23.34 <util/twi.h>: TWI bit mask definitions	246
23.34.1 Detailed Description	247
23.34.2 Macro Definition Documentation	247
23.35 <compat/deprecated.h>: Deprecated items	250
23.35.1 Detailed Description	251
23.35.2 Macro Definition Documentation	251
23.35.3 Function Documentation	252
23.36 <compat/ina90.h>: Compatibility with IAR EWB 3.x	253
23.37 Demo projects	253
23.37.1 Detailed Description	253
23.38 Combining C and assembly source files	254
23.38.1 Hardware setup	254
23.38.2 A code walkthrough	255
23.38.3 The source code	256
23.39 A simple project	256
23.39.1 The Project	256
23.39.2 The Source Code	258
23.39.3 Compiling and Linking	259
23.39.4 Examining the Object File	260
23.39.5 Linker Map Files	263
23.39.6 Generating Intel Hex Files	264
23.39.7 Letting Make Build the Project	265
23.39.8 Reference to the source code	267
23.40 A more sophisticated project	267
23.40.1 Hardware setup	267
23.40.2 Functional overview	269
23.40.3 A code walkthrough	269
23.40.4 The source code	272

23.41 Using the standard IO facilities	272
23.41.1 Hardware setup	272
23.41.2 Functional overview	273
23.41.3 A code walkthrough	273
23.41.4 The source code	277
23.42 Example using the two-wire interface (TWI)	277
23.42.1 Introduction into TWI	278
23.42.2 The TWI example project	278
23.42.3 The Source Code	278
24 Data Structure Documentation	281
24.1 div_t Struct Reference	281
24.1.1 Detailed Description	281
24.1.2 Field Documentation	282
24.2 ldiv_t Struct Reference	282
24.2.1 Detailed Description	282
24.2.2 Field Documentation	282
24.3 tm Struct Reference	283
24.3.1 Detailed Description	283
24.3.2 Field Documentation	283
24.4 week_date Struct Reference	284
24.4.1 Detailed Description	284
24.4.2 Field Documentation	284
25 File Documentation	285
25.1 project.h	285
25.2 iocompat.h	285
25.3 defines.h	287
25.4 hd44780.h	288
25.5 lcd.h	289
25.6 uart.h	289
25.7 alloca.h	290
25.8 assert.h File Reference	291
25.8.1 Macro Definition Documentation	291
25.9 assert.h	291
25.10 boot.h File Reference	293
25.11 boot.h	293
25.12 cpufunc.h File Reference	301
25.13 cpufunc.h	301
25.14 eeprom.h	302
25.15 fuse.h File Reference	305
25.16 fuse.h	305
25.17 interrupt.h File Reference	309

25.17.1 Detailed Description	309
25.18 interrupt.h	310
25.19 io.h File Reference	314
25.20 io.h	314
25.21 lock.h File Reference	321
25.22 lock.h	321
25.23 pgmspace.h File Reference	324
25.24 pgmspace.h	326
25.25 portpins.h	345
25.26 power.h File Reference	352
25.26.1 Macro Definition Documentation	352
25.26.2 Function Documentation	352
25.27 power.h	353
25.28 sfr_defs.h	374
25.29 signal.h	378
25.30 signature.h File Reference	378
25.31 signature.h	378
25.32 sleep.h File Reference	379
25.33 sleep.h	379
25.34 version.h	384
25.35 wdt.h File Reference	385
25.36 wdt.h	385
25.37 xmega.h	392
25.38 deprecated.h	393
25.39 ina90.h	396
25.40 ctype.h File Reference	397
25.41 ctype.h	398
25.42 errno.h File Reference	400
25.43 errno.h	400
25.44 inttypes.h File Reference	402
25.45 inttypes.h	404
25.46 math.h File Reference	410
25.46.1 Macro Definition Documentation	412
25.46.2 Function Documentation	414
25.47 math.h	423
25.48 setjmp.h File Reference	429
25.49 setjmp.h	429
25.50 stdint.h File Reference	431
25.51 stdint.h	434
25.52 stdio.h File Reference	442
25.52.1 Macro Definition Documentation	443
25.52.2 Typedef Documentation	446

25.52.3 Function Documentation	446
25.53 stdio.h	456
25.54 stdlib.h File Reference	467
25.54.1 Macro Definition Documentation	469
25.54.2 Typedef Documentation	469
25.54.3 Function Documentation	469
25.54.4 Variable Documentation	473
25.55 stdlib.h	474
25.56 string.h File Reference	482
25.57 string.h	483
25.58 time.h File Reference	490
25.58.1 Macro Definition Documentation	491
25.58.2 Enumeration Type Documentation	492
25.58.3 Function Documentation	492
25.59 time.h	498
25.60 atomic.h File Reference	503
25.61 atomic.h	504
25.62 crc16.h File Reference	507
25.63 crc16.h	508
25.64 delay.h File Reference	512
25.65 delay.h	513
25.66 delay_basic.h File Reference	516
25.67 delay_basic.h	516
25.68 eu_dst.h	518
25.69 parity.h File Reference	519
25.70 parity.h	519
25.71 setbaud.h File Reference	520
25.72 setbaud.h	520
25.73 compat/twi.h	523
25.74 twi.h File Reference	524
25.75 util/twi.h	525
25.76 usa_dst.h	527
25.77 eedef.h	529
25.78 fdevopen.c File Reference	530
25.79 stdio_private.h	530
25.80 xtoa_fast.h	531
25.81 dtoa_conv.h	531
25.82 stdlib_private.h	532
25.83 ephamera_common.h	533
Index	535

1 AVR Libc

1.1 Introduction

The latest version of this document is always available from <http://savannah.nongnu.org/projects/avr-libc/>

The AVR Libc package provides a subset of the standard C library for [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 at <https://savannah.nongnu.org/bugs/?group=avr-libc> 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 IO 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/assembler 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
- atmega16a
- atmega2560
- atmega2561
- atmega32
- atmega32a
- atmega323
- atmega324a
- atmega324p
- atmega324pa
- atmega325
- atmega325a

- atmega325p
- atmega325pa
- atmega3250
- atmega3250a
- atmega3250p
- atmega3250pa
- atmega328
- atmega328p
- 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

tinyAVR Devices:

- attiny4
- attiny5
- attiny10
- attiny11 [\[1\]](#)
- attiny12 [\[1\]](#)
- attiny13
- attiny13a
- attiny15 [\[1\]](#)
- attiny20
- attiny22
- attiny24
- attiny24a
- attiny25
- attiny26
- attiny261
- attiny261a
- attiny28 [\[1\]](#)
- attiny2313
- attiny2313a
- attiny40
- attiny4313
- attiny43u
- attiny44
- attiny44a
- attiny441
- attiny45
- attiny461
- attiny461a
- attiny48
- attiny828
- attiny84
- attiny84a
- attiny841

- attiny85
- attiny861
- attiny861a
- attiny87
- attiny88
- attiny1634

Automotive AVR Devices:

- atmega16m1
- atmega32c1
- atmega32m1
- atmega64c1
- atmega64m1
- attiny167
- ata5505
- ata5272
- ata5702m322
- ata5782
- ata5790
- ata5790n
- ata5831
- ata5795
- ata6612c
- ata6613c
- ata6614q
- ata6616c
- ata6617c
- ata664251

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:

- at90usb82
- at90usb162
- at90usb646
- at90usb647
- at90usb1286
- at90usb1287
- atmega8u2
- atmega16u2
- atmega16u4
- atmega32u2
- atmega32u4
- atmega32u6

XMEGA Devices:

- atxmega8e5
- atxmega16a4
- atxmega16a4u
- atxmega16c4

- atxmega16d4
- atxmega32a4
- atxmega32a4u
- atxmega32c3
- atxmega32c4
- atxmega32d3
- atxmega32d4
- atxmega32e5
- atxmega64a1
- atxmega64a1u
- atxmega64a3
- atxmega64a3u
- atxmega64a4u
- atxmega64b1
- atxmega64b3
- atxmega64c3
- atxmega64d3
- atxmega64d4
- atxmega128a1
- atxmega128a1u
- atxmega128a3
- atxmega128a3u
- atxmega128a4u
- atxmega128b1
- atxmega128b3
- atxmega128c3
- atxmega128d3
- atxmega128d4
- atxmega192a3
- atxmega192a3u
- atxmega192c3
- atxmega192d3
- atxmega256a3
- atxmega256a3u
- atxmega256a3b
- atxmega256a3bu

- atxmega256c3
- atxmega256d3
- atxmega384c3
- atxmega384d3

Wireless AVR devices:

- atmega644rfr2
- atmega64rfr2
- atmega128rfa1
- atmega1284rfr2
- atmega128rfr2
- atmega2564rfr2
- atmega256rfr2

Miscellaneous Devices:

- at94K [\[2\]](#)
- at76c711 [\[3\]](#)
- at43usb320
- at43usb355
- at86rf401
- at90scr100
- ata6285
- ata6286
- ata6289
- 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.

Portions of avr-libc are Copyright (c) 1999-2016
Werner Boellmann,
Dean Camera,
Pieter Conradie,
Brian Dean,
Keith Gudger,
Wouter van Gulik,
Bjoern Haase,
Steinar Haugen,
Peter Jansen,
Reinhard Jessich,
Magnus Johansson,
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,
Atmel Corporation,
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 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 Atmel 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.

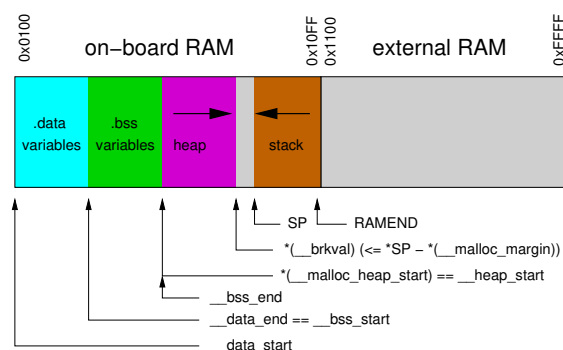


Figure 1 RAM map of a device with internal RAM

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.

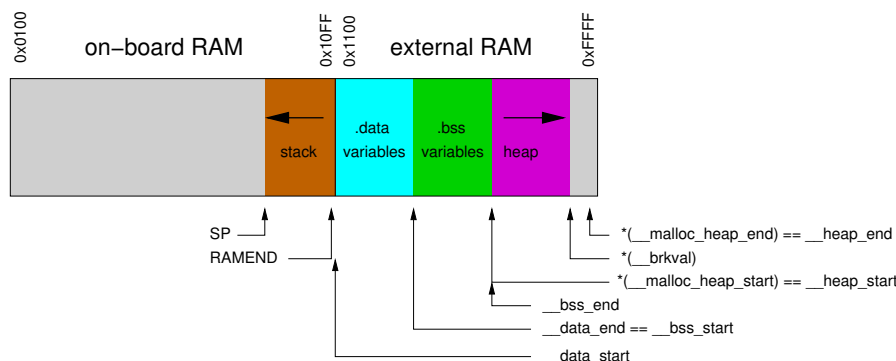


Figure 2 Internal RAM: stack only, external RAM: variables and heap

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 ...
```

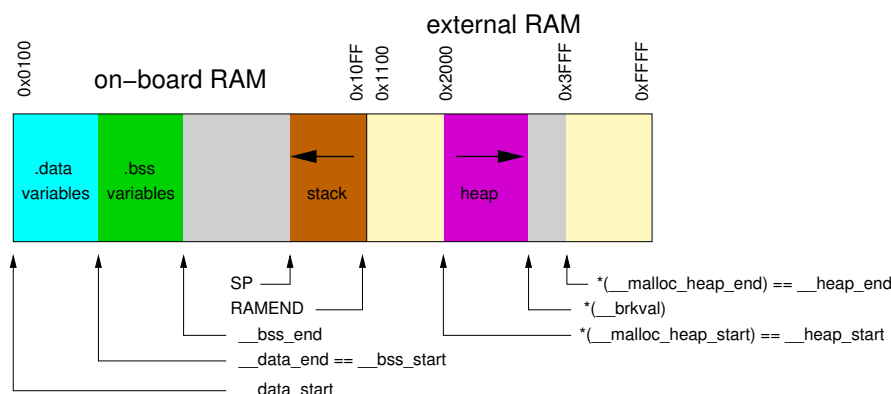


Figure 3 Internal RAM: variables and stack, external RAM: heap

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

Remarks

Need to list all the sections which are available to the avr.

Weak Bindings

FIXME: need to discuss the `.weak` directive.

The following describes the various sections available.

4.1 The .text Section

The .text section contains the actual machine instructions which make up your program. This section is further subdivided by the .initN and .finiN sections discussed below.

Note

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.

4.2 The .data Section

This section contains static data which was defined in your code. Things like the following would end up in .data:

```
char err_str[] = "Your program has died a horrible death!";
struct point pt = { 1, 1 };
```

It is possible to tell the linker the SRAM address of the beginning of the .data section. This is accomplished by adding `-Wl, -Tdata, addr` to the `avr-gcc` command used to link your program. Not 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 space. Thus, if you want the .data section to start at `0x1100`, pass `0x801100` at the address to the linker. [offset explained]

Note

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

4.3 The .bss Section

Uninitialized global or static variables end up in the .bss section.

4.4 The .eeprom Section

This is where eeprom variables are stored.

4.5 The .noinit Section

This section is a part of the .bss section. What makes the .noinit section special is that variables which are defined as such:

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

will not be initialized to zero during startup as would normal .bss data.

Only uninitialized variables can be placed in the .noinit section. Thus, the following code will cause `avr-gcc` to issue an error:

```
int bar __attribute__((section(".noinit"))) = 0xaa;
```

It is possible to tell the linker explicitly where to place the .noinit section by adding `-Wl, -section-start=.noinit=0x802000` to the `avr-gcc` command line at the linking stage. For example, suppose you wish to place the .noinit section at SRAM address `0x2000`:

```
$ avr-gcc ... -Wl,--section-start=.noinit=0x802000 ...
```

Note

Because of the Harvard architecture of the AVR devices, you must manually add `0x800000` to the address you pass to the linker as the start of the section. Otherwise, the linker thinks you want to put the .noinit section into the .text section instead of .data/.bss and will complain.

Alternatively, you can write your own linker script to automate this. [FIXME: need an example or ref to dox for writing linker scripts.]

4.6 The .initN Sections

These sections are used to define the startup code from reset up through the start of `main()`. These all are subparts of the [.text section](#).

The purpose of these sections is to allow for more specific placement of code within your program.

Note

Sometimes, it is convenient to think of the `.initN` and `.finiN` sections as functions, but in reality they are just symbolic names which tell the linker where to stick a chunk of code which is *not* a function. Notice that the examples for [asm](#) and [C](#) can not be called as functions and should not be jumped into.

The `.initN` sections are executed in order from 0 to 9.

`.init0:`

Weakly bound to `__init()`. If user defines `__init()`, it will be jumped into immediately after a reset.

`.init1:`

Unused. User definable.

`.init2:`

In C programs, weakly bound to initialize the stack, and to clear `__zero_reg__` (r1).

`.init3:`

Unused. User definable.

`.init4:`

For devices with > 64 KB of ROM, `.init4` defines the code which takes care of copying the contents of `.data` from the flash to SRAM. For all other devices, this code as well as the code to zero out the `.bss` section is loaded from `libgcc.a`.

`.init5:`

Unused. User definable.

`.init6:`

Unused for C programs, but used for constructors in C++ programs.

`.init7:`

Unused. User definable.

`.init8:`

Unused. User definable.

`.init9:`

Jumps into `main()`.

4.7 The .finiN Sections

These sections are used to define the exit code executed after return from `main()` or a call to `exit()`. These all are subparts of the [.text section](#).

The **.finiN** sections are executed in descending order from 9 to 0.

.fini9:

Unused. User definable. This is effectively where `_exit()` starts.

.fini8:

Unused. User definable.

.fini7:

Unused. User definable.

.fini6:

Unused for C programs, but used for destructors in C++ programs.

.fini5:

Unused. User definable.

.fini4:

Unused. User definable.

.fini3:

Unused. User definable.

.fini2:

Unused. User definable.

.fini1:

Unused. User definable.

.fini0:

Goes into an infinite loop after program termination and completion of any `_exit()` code (execution of code in the `.fini9` -> `.fini1` sections).

4.8 The .note.gnu.avr.deviceinfo Section

This section contains device specific information picked up from the device header file and compiler builtin macros. The layout conforms to the standard ELF note section layout (http://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-18048.html).

The section contents are laid out as below.

```
#define __NOTE_NAME_LEN 4
struct __note_gnu_avr_deviceinfo
{
    struct
    {
        uint32_t namesz;           /* = __NOTE_NAME_LEN */
        uint32_t descsize;        /* = size of avr_desc */
        uint32_t type;            /* = 1 - no other AVR note types exist */
        char note_name[__NOTE_NAME_LEN]; /* = "AVR\0" */
    }
    note_header;
    struct
    {
        uint32_t flash_start;
        uint32_t flash_size;
        uint32_t sram_start;
        uint32_t sram_size;
        uint32_t eeprom_start;
        uint32_t eeprom_size;
        uint32_t offset_table_size;
        uint32_t offset_table[1]; /* Offset table containing byte offsets into
string table that immediately follows it.
index 0: Device name byte offset
*/
        char str_table [2 +
            strlen(__AVR_DEVICE_NAME__)]; /* Standard ELF string table.
index 0 : NULL
index 1 : Device name
index 2 : NULL
*/
    }
    avr_desc;
};
```

4.9 Using Sections in Assembler Code

Example:

```
#include <avr/io.h>
.section .init1,"ax",@progbits
ldi r0, 0xff
out _SFR_IO_ADDR(PORTB), r0
out _SFR_IO_ADDR(DDRB), r0
```

Note

The `,"ax",@progbits` tells the assembler that the section is allocatable ("a"), executable ("x") and contains data ("@progbits"). For more detailed information on the `.section` directive, see the gas user manual.

4.10 Using Sections in C Code

Example:

```
#include <avr/io.h>
void my_init_portb (void) __attribute__(((naked)) \
    __attribute__((section (".init3")) \
    __attribute__((used));
void
my_init_portb (void)
{
    PORTB = 0xff;
    DDRB = 0xff;
}
```

Note

Section `.init3` is used in this example, as this ensures the internal `__zero_reg__` has already been set up. The code generated by the compiler might blindly rely on `__zero_reg__` being really 0. `__attribute__((used))` tells the compiler that code must be generated for this function even if it appears that the function is not referenced - this is necessary to prevent compiler optimizations (like LTO) from eliminating the function.

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:

```
char *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:

```
char *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:

```
PGM_P 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.

The `PGM_P` type above is also a macro that is defined as a pointer to a character in the Program Space.

Retrieving the strings is 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 (unsigned char i = 0; i < 5; i++)
    {
        strcpy_P(buffer, (PGM_P)pgm_read_word(&(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_word`. We use the `pgm_read_word` 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_word` macro will return a 16-bit unsigned integer. We then have to typecast it as a true pointer to program memory, `PGM_P`. 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

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 4.3.0). 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]
; Note [4]:
tmconst= 10700000 / 200000 ; 100 kHz => 200000 edges/s
fuzz= 8 ; # clocks in ISR until TCNT0 is set
.section .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 _SFR_IO_ADDR(TCNT0), inttmp ; Note [8]
    in intsav, _SFR_IO_ADDR(SREG) ; Note [9]
    sbic _SFR_IO_ADDR(PORTD), SQUARE
    rjmp 1f
    sbi _SFR_IO_ADDR(PORTD), SQUARE
    rjmp 2f
1: cbi _SFR_IO_ADDR(PORTD), SQUARE
2:
    out _SFR_IO_ADDR(SREG), intsav
    reti
ioinit:
    sbi _SFR_IO_ADDR(DDRD), SQUARE
    ldi work, _BV(TOIE0)
    out _SFR_IO_ADDR(TIMSK), work
    ldi work, _BV(CS00) ; tmr0: CK/1
    out _SFR_IO_ADDR(TCCR0), work
    ldi work, 256 - tmconst
    out _SFR_IO_ADDR(TCNT0), work
    sei
    ret
.global __vector_default ; Note [10]
__vector_default:
    reti
.end
```

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 Pseudo-ops and operators

The available pseudo-ops in the assembler are described in the GNU assembler (gas) manual. The manual can be found online as part of the current binutils release under <http://sources.redhat.com/binutils/>.

As gas comes from a Unix origin, its pseudo-op 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), expressions use a C-like syntax.

Some common pseudo-ops include:

- `.byte` allocates single byte constants
- `.ascii` allocates a non-terminated string of characters
- `.asciz` allocates a `\0`-terminated string of characters (C string)
- `.data` switches to the `.data` section (initialized RAM variables)
- `.text` switches to the `.text` section (code and ROM constants)
- `.set` declares a symbol as a constant expression (identical to `.equ`)
- `.global` (or `.globl`) declares a public symbol that is visible to the linker (e. g. function entry point, global variable)
- `.extern` declares a symbol to be externally defined; this is effectively a comment only, as gas treats all undefined symbols it encounters as globally undefined anyway

Note that `.org` is available in gas as well, but is a fairly pointless pseudo-op in an assembler environment that uses relocatable object files, as it is the linker that determines the final position of some object in ROM or RAM.

Along with the architecture-independent standard operators, there are some AVR-specific operators available which are unfortunately not yet described in the official documentation. The most notable operators are:

- `lo8` Takes the least significant 8 bits of a 16-bit integer
- `hi8` Takes the most significant 8 bits of a 16-bit integer
- `pm` Takes a program-memory (ROM) address, and converts it into a RAM address. This implies a division by 2 as the AVR handles ROM addresses as 16-bit words (e.g. in an `IJMP` or `ICALL` instruction), and can also handle relocatable symbols on the right-hand side.

Example:

```
ldi r24, lo8(pm(somefunc))
ldi r25, hi8(pm(somefunc))
call something
```

This passes the address of function `somefunc` as the first parameter to function `something`.

7 Inline Assembler Cookbook

AVR-GCC

Inline Assembler Cookbook

About this Document

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

Because of a lack of documentation, especially for the AVR version of the compiler, it may take some time to figure out the implementation details by studying the compiler and assembler source code. There are also a few sample programs available in the net. Hopefully this document will help to increase their number.

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 language tutorial either.

Note that this document does not cover file written completely in assembler 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 3.3 of the compiler. There may be some parts, which hadn't been completely understood by the author himself and not all samples had been tested so far. Because the author is German and not familiar with the English language, there are definitely some typos and syntax errors in the text. As a programmer the author knows, that a wrong documentation sometimes might be worse than none. Anyway, he decided to offer his little knowledge to the public, in the hope to get enough response to improve this document. Feel free to contact the author via e-mail. For the latest release check <http://www.ethernut.de/>.

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

Note

As of 26th of July 2002, this document has been merged into the documentation for avr-libc. The latest version is now available at <http://savannah.nongnu.org/projects/avr-libc/>.

7.1 GCC asm Statement

Let's start with a simple example of reading a value from port D:

```
asm("in %0, %1" : "=r" (value) : "I" (_SFR_IO_ADDR(PORTD)) );
```

Each `asm` statement is divided by colons into (up to) four parts:

1. The assembler instructions, defined as a single string constant:
"in %0, %1"
2. A list of output operands, separated by commas. Our example uses just one:
"=r" (value)
3. A comma separated list of input operands. Again our example uses one operand only:
"I" (_SFR_IO_ADDR(PORTD))
4. Clobbered registers, left empty in our example.

You can write assembler instructions in much the same way as you would write assembler programs. However, registers and constants are used in a different way if they refer to expressions of your C program. The connection between registers and C operands is specified in the second and third part of the `asm` instruction, the list of input and output operands, respectively. The general form is

```
asm(code : output operand list : input operand list [: clobber list]);
```

In the code section, operands are referenced by a percent sign followed by a single digit. %0 refers to the first %1 to the second operand and so forth. From the above example:

%0 refers to "=r" (value) and
%1 refers to "I" (_SFR_IO_ADDR(PORTD)).

This may still look a little odd now, but the syntax of an operand list will be explained soon. Let us first examine the part of a compiler listing which may have been generated from our example:

```
lds r24,value
/* #APP */
in r24, 12
```

```
/* #NOAPP */
    sts value,r24
```

The comments have been added by the compiler to inform the assembler that the included code was not generated by the compilation of C statements, but by inline assembler statements. The compiler selected register `r24` for storage of the value read from `PORTD`. The compiler could have selected any other register, though. It may not explicitly load or store the value and it may even decide not to include your assembler code at all. All these decisions are part of the compiler's optimization strategy. For example, if you never use the variable `value` in the remaining part of the C program, the compiler will most likely remove your code unless you switched off optimization. To avoid this, you can add the `volatile` attribute to the `asm` statement:

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

Alternatively, operands can be given names. The name is prepended in brackets to the constraints in the operand list, and references to the named operand use the bracketed name instead of a number after the `%` sign. Thus, the above example could also be written as

```
asm("in %[retval], %[port]" :
    [retval] "=r" (value) :
    [port] "I" (_SFR_IO_ADDR(PORTD)) );
```

The last part of the `asm` instruction, the clobber list, is mainly used to tell the compiler about modifications done by the assembler code. This part may be omitted, all other parts are required, but may be left empty. If your assembler routine won't use any input or output operand, two colons must still follow the assembler code string. A good example is a simple statement to disable interrupts:

```
asm volatile("cli");
```

7.2 Assembler Code

You can use the same assembler instruction mnemonics as you'd use with any other AVR assembler. And you can write as many assembler statements into one code string as you like and your flash memory is able to hold.

Note

The available assembler directives vary from one assembler to another.

To make it more readable, you should put each statement on a separate line:

```
asm volatile("nop\n\t"
    "nop\n\t"
    "nop\n\t"
    "nop\n\t"
    "::);
```

The linefeed and tab characters will make the assembler listing generated by the compiler more readable. It may look a bit odd for the first time, but that's the way the compiler creates its own assembler code.

You may also make use of some special registers.

Symbol	Register
<code>__SREG__</code>	Status register at address 0x3F
<code>__SP_H__</code>	Stack pointer high byte at address 0x3E
<code>__SP_L__</code>	Stack pointer low byte at address 0x3D
<code>__tmp_reg__</code> ↔	Register r0, used for temporary storage
<code>__zero_reg__</code> ↔	Register r1, always zero

Register `r0` may be freely used by your assembler code and need not be restored at the end of your code. It's a good idea to use `__tmp_reg__` and `__zero_reg__` instead of `r0` or `r1`, just in case a new compiler version

changes the register usage definitions.

7.3 Input and Output Operands

Each input and output operand is described by a constraint string followed by a C expression in parantheses. AVR-GCC 3.3 knows the following constraint characters:

Note

The most up-to-date and detailed information on constraints for the avr can be found in the gcc manual.

The `x` register is `r27:r26`, the `y` register is `r29:r28`, and the `z` register is `r31:r30`

Constraint	Used for	Range
a	Simple upper registers	r16 to r23
b	Base pointer registers pairs	y, z
d	Upper register	r16 to r31
e	Pointer register pairs	x, y, z
q	Stack pointer register	SPH:SPL
r	Any register	r0 to r31
t	Temporary register	r0
w	Special upper register pairs	r24, r26, r28, r30
x	Pointer register pair X	x (r27:r26)
y	Pointer register pair Y	y (r29:r28)
z	Pointer register pair Z	z (r31:r30)
G	Floating point constant	0.0
I	6-bit positive integer constant	0 to 63
J	6-bit negative integer constant	-63 to 0
K	Integer constant	2
L	Integer constant	0
l	Lower registers	r0 to r15
M	8-bit integer constant	0 to 255
N	Integer constant	-1
O	Integer constant	8, 16, 24
P	Integer constant	1
Q	(GCC >= 4.2.x) A memory address based on Y or Z pointer with displacement.	
R	(GCC >= 4.3.x) Integer constant.	-6 to 5

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. For example, if you specify the constraint `"r"` and you are using this register with an `"ori"` instruction in your assembler code, 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 value. Later on we will see how to pass multibyte expression results to the assembler code.

The following table shows all AVR assembler mnemonics which require operands, and the related constraints. Because of the improper constraint definitions in version 3.3, they aren't strict enough. There is, for example, no constraint, which restricts integer constants to the range 0 to 7 for bit set and bit clear operations.

Mnemonic	Constraints	Mnemonic	Constraints
adc	r,r	add	r,r
adiw	w,l	and	r,r
andi	d,M	asr	r
bclr	l	bld	r,l
brbc	l,label	brbs	l,label
bset	l	bst	r,l
cbi	l,l	cbr	d,l
com	r	cp	r,r
cpc	r,r	cpi	d,M
cpse	r,r	dec	r
elpm	t,z	eor	r,r
in	r,l	inc	r
ld	r,e	ldd	r,b
ldi	d,M	lds	r,label
lpm	t,z	lsl	r
lsr	r	mov	r,r
movw	r,r	mul	r,r
neg	r	or	r,r
ori	d,M	out	l,r
pop	r	push	r
rol	r	ror	r
sbc	r,r	sbc	d,M
sbi	l,l	sbic	l,l
sbiw	w,l	sbr	d,M
sbr	r,l	sbrs	r,l
ser	d	st	e,r
std	b,r	sts	label,r
sub	r,r	subi	d,M
swap	r		

Constraint characters may be prepended by a single constraint modifier. Constraints without a modifier specify read-only operands. Modifiers are:

Modifier	Specifies
=	Write-only operand, usually used for all output operands.
+	Read-write operand
&	Register should be used for output only

Output operands must be write-only and the C expression result must be an lvalue, which means that the operands must be valid on the left side of assignments. Note, that the compiler will not check if the operands are of reasonable type for the kind of operation used in the assembler instructions.

Input operands are, you guessed it, read-only. But what if you need the same operand for input and output? As stated above, read-write operands are not supported in inline assembler code. But there is another solution. For input operators it is possible to use a single digit in the constraint string. Using digit *n* tells the compiler to use the same register as for the *n*-th operand, starting with zero. Here is an example:

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

This statement will swap the nibbles of an 8-bit variable named `value`. Constraint `"0"` tells the compiler, to use the same input register as for the first operand. Note however, that this doesn't automatically imply the reverse case. The compiler may choose the same registers for input and output, even if not told to do so. This is not a problem in

most cases, but may be fatal if the output operator is modified by the assembler code before the input operator is used. In the situation where your code depends on different registers used for input and output operands, you must add the `&` constraint modifier to your output operand. The following example demonstrates this problem:

```
asm volatile("in %0,%1" "\n\t"
            "out %1, %2" "\n\t"
            : "=r" (input)
            : "I" (_SFR_IO_ADDR(port)), "r" (output)
            );
```

In this example an input value is read from a port and then an output value is written to the same port. If the compiler would have chosen the same register for input and output, then the output value would have been destroyed on the first assembler instruction. Fortunately, this example uses the `&` constraint modifier to instruct the compiler not to select any register for the output value, which is used for any of the input operands. Back to swapping. Here is the code to swap high and low byte of a 16-bit value:

```
asm volatile("mov __tmp_reg__, %A0" "\n\t"
            "mov %A0, %B0" "\n\t"
            "mov %B0, __tmp_reg__" "\n\t"
            : "=r" (value)
            : "0" (value)
            );
```

First you will notice the usage of register `__tmp_reg__`, which we listed among other special registers in the [Assembler Code](#) section. You can use this register without saving its contents. Completely new are those letters `A` and `B` in `%A0` and `%B0`. In fact they refer to two different 8-bit registers, both containing a part of value.

Another example to swap bytes of a 32-bit value:

```
asm volatile("mov __tmp_reg__, %A0" "\n\t"
            "mov %A0, %D0" "\n\t"
            "mov %D0, __tmp_reg__" "\n\t"
            "mov __tmp_reg__, %B0" "\n\t"
            "mov %B0, %C0" "\n\t"
            "mov %C0, __tmp_reg__" "\n\t"
            : "=r" (value)
            : "0" (value)
            );
```

Instead of listing the same operand as both, input and output operand, it can also be declared as a read-write operand. This must be applied to an output operand, and the respective input operand list remains empty:

```
asm volatile("mov __tmp_reg__, %A0" "\n\t"
            "mov %A0, %D0" "\n\t"
            "mov %D0, __tmp_reg__" "\n\t"
            "mov __tmp_reg__, %B0" "\n\t"
            "mov %B0, %C0" "\n\t"
            "mov %C0, __tmp_reg__" "\n\t"
            : "+r" (value));
```

If operands do not fit into a single register, the compiler will automatically assign enough registers to hold the entire operand. In the assembler code you use `%A0` to refer to the lowest byte of the first operand, `%A1` to the lowest byte of the second operand and so on. The next byte of the first operand will be `%B0`, the next byte `%C0` and so on.

This also implies, that it is often necessary to cast the type of an input operand to the desired size.

A final problem may arise while using pointer register pairs. If you define an input operand

```
"e" (ptr)
```

and the compiler selects register `Z` (`r30:r31`), then

`%A0` refers to `r30` and

`%B0` refers to `r31`.

But both versions will fail during the assembly stage of the compiler, if you explicitly need `Z`, like in

```
ld r24, Z
```

If you write

```
ld r24, %a0
```

with a lower case `a` following the percent sign, then the compiler will create the proper assembler line.

7.4 Clobbers

As stated previously, the last part of the `asm` statement, the list of clobbers, may be omitted, including the colon separator. However, if you are using registers, which had not been passed as operands, you need to inform the compiler about this. The following example will do an atomic increment. It increments an 8-bit value pointed to by a pointer variable in one go, without being interrupted by an interrupt routine or another thread in a multithreaded environment. Note, that we must use a pointer, because the incremented value needs to be stored before interrupts are enabled.

```
asm volatile(
    "cli"                "\n\t"
    "ld r24, %a0"        "\n\t"
    "inc r24"            "\n\t"
    "st %a0, r24"        "\n\t"
    "sei"                "\n\t"
    :
    : "e" (ptr)
    : "r24"
);
```

The compiler might produce the following code:

```
cli
ld r24, Z
inc r24
st Z, r24
sei
```

One easy solution to avoid clobbering register `r24` is, to make use of the special temporary register `__tmp_reg__` defined by the compiler.

```
asm volatile(
    "cli"                "\n\t"
    "ld __tmp_reg__, %a0" "\n\t"
    "inc __tmp_reg__"     "\n\t"
    "st %a0, __tmp_reg__" "\n\t"
    "sei"                "\n\t"
    :
    : "e" (ptr)
);
```

The compiler is prepared to reload this register next time it uses it. Another problem with the above code is, that it should not be called in code sections, where interrupts are disabled and should be kept disabled, because it will enable interrupts at the end. We may store the current status, but then we need another register. Again we can solve this without clobbering a fixed, but let the compiler select it. This could be done with the help of a local C variable.

```
{
    uint8_t s;
    asm volatile(
        "in %0, __SREG__"        "\n\t"
        "cli"                    "\n\t"
        "ld __tmp_reg__, %a1"    "\n\t"
        "inc __tmp_reg__"        "\n\t"
        "st %a1, __tmp_reg__"    "\n\t"
        "out __SREG__, %0"       "\n\t"
        : "=&r" (s)
        : "e" (ptr)
    );
}
```

Now every thing seems correct, but it isn't really. The assembler code modifies the variable, that `ptr` points to. The compiler will not recognize this and may keep its value in any of the other registers. Not only does the compiler work with the wrong value, but the assembler code does too. The C program may have modified the value too, but the compiler didn't update the memory location for optimization reasons. The worst thing you can do in this case is:

```
{
    uint8_t s;
    asm volatile(
        "in %0, __SREG__"        "\n\t"
        "cli"                    "\n\t"
        "ld __tmp_reg__, %a1"    "\n\t"
        "inc __tmp_reg__"        "\n\t"
        "st %a1, __tmp_reg__"    "\n\t"
        "out __SREG__, %0"       "\n\t"
        : "=&r" (s)
        : "e" (ptr)
        : "memory"
    );
}
```


The special clobber "memory" informs the compiler that the assembler code may modify any memory location. It forces the compiler to update all variables for which the contents are currently held in a register before executing the assembler code. And of course, everything has to be reloaded again after this code.

In most situations, a much better solution would be to declare the pointer destination itself volatile:

```
volatile uint8_t *ptr;
```

This way, the compiler expects the value pointed to by `ptr` to be changed and will load it whenever used and store it whenever modified.

Situations in which you need clobbers are very rare. In most cases there will be better ways. Clobbered registers will force the compiler to store their values before and reload them after your assembler code. Avoiding clobbers gives the compiler more freedom while optimizing your code.

7.5 Assembler Macros

In order to reuse your assembler language parts, it is useful to define them as macros and put them into include files. AVR Libc comes with a bunch of them, which could be found in the directory `avr/include`. Using such include files may produce compiler warnings, if they are used in modules, which are compiled in strict ANSI mode. To avoid that, you can write `__asm__` instead of `asm` and `__volatile__` instead of `volatile`. These are equivalent aliases.

Another problem with reused macros arises if you are using labels. In such cases you may make use of the special pattern `%=`, which is replaced by a unique number on each `asm` statement. The following code had been taken from `avr/include/iomacros.h`:

```
#define loop_until_bit_is_clear(port,bit) \
    __asm__ __volatile__ ( \
        "L_%=: " "sbic %0, %1" "\n\t" \
        "rjmp L_%= " \
        : /* no outputs */ \
        : "I" (_SFR_IO_ADDR(port)), \
        "I" (bit) \
    )
```

When used for the first time, `L_%=` may be translated to `L_1404`, the next usage might create `L_1405` or whatever. In any case, the labels became unique too.

Another option is to use Unix-assembler style numeric labels. They are explained in [How do I trace an assembler file in avr-gdb?](#). The above example would then look like:

```
#define loop_until_bit_is_clear(port,bit) \
    __asm__ __volatile__ ( \
        "1: " "sbic %0, %1" "\n\t" \
        "rjmp 1b" \
        : /* no outputs */ \
        : "I" (_SFR_IO_ADDR(port)), \
        "I" (bit) \
    )
```

7.6 C Stub Functions

Macro definitions will include the same assembler code whenever they are referenced. This may not be acceptable for larger routines. In this case you may define a C stub function, containing nothing other than your assembler code.

```
void delay(uint8_t ms)
{
    uint16_t cnt;
    asm volatile (
        "\n"
        "L_d11%=: " "\n\t"
        "mov %A0, %A2" "\n\t"
        "mov %B0, %B2" "\n"
        "L_d12%=: " "\n\t"
        "sbw %A0, 1" "\n\t"
        "brne L_d12%=" "\n\t"
        "dec %1" "\n\t"
        "brne L_d11%=" "\n\t"
        : "=w" (cnt)
        : "r" (ms), "r" (delay_count)
    );
}
```

The purpose of this function is to delay the program execution by a specified number of milliseconds using a counting loop. The global 16 bit variable `delay_count` must contain the CPU clock frequency in Hertz divided by 4000 and must have been set before calling this routine for the first time. As described in the [clobber](#) section, the routine uses a local variable to hold a temporary value.

Another use for a local variable is a return value. The following function returns a 16 bit value read from two successive port addresses.

```
uint16_t inw(uint8_t port)
{
    uint16_t result;
    asm volatile (
        "in %A0,%1" "\n\t"
        "in %B0, (%1) + 1"
        : "=r" (result)
        : "I" (_SFR_IO_ADDR(port))
    );
    return result;
}
```

Note

`inw()` is supplied by `avr-libc`.

7.7 C Names Used in Assembler Code

By default AVR-GCC uses the same symbolic names of functions or variables in C and assembler code. You can specify a different name for the assembler code by using a special form of the `asm` statement:

```
unsigned long value asm("clock") = 3686400;
```

This statement instructs the compiler to use the symbol name `clock` rather than `value`. This makes sense only for external or static variables, because local variables do not have symbolic names in the assembler code. However, local variables may be held in registers.

With AVR-GCC you can specify the use of a specific register:

```
void Count(void)
{
    register unsigned char counter asm("r3");
    ... some code...
    asm volatile("clr r3");
    ... more code...
}
```

The assembler instruction, `"clr r3"`, will clear the variable `counter`. AVR-GCC will not completely reserve the specified register. If the optimizer recognizes that the variable will not be referenced any longer, the register may be re-used. But the compiler is not able to check whether this register usage conflicts with any predefined register. If you reserve too many registers in this way, the compiler may even run out of registers during code generation.

In order to change the name of a function, you need a prototype declaration, because the compiler will not accept the `asm` keyword in the function definition:

```
extern long Calc(void) asm ("CALCULATE");
```

Calling the function `Calc()` will create assembler instructions to call the function `CALCULATE`.

7.8 Links

For a more thorough discussion of inline assembly usage, see the gcc user manual. The latest version of the gcc manual is always available here: <http://gcc.gnu.org/onlinedocs/>

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, with 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_flt
```

then the linker will look for a library called:

```
libprintf_flt.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_flt (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_flt ("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

1. [My program doesn't recognize a variable updated within an interrupt routine](#)
2. [I get "undefined reference to..." for functions like "sin\(\)"](#)
3. [How to permanently bind a variable to a register?](#)
4. [How to modify MCUCR or WDTCSR early?](#)
5. [What is all this _BV\(\) stuff about?](#)
6. [Can I use C++ on the AVR?](#)
7. [Shouldn't I initialize all my variables?](#)
8. [Why do some 16-bit timer registers sometimes get trashed?](#)
9. [How do I use a #define'd constant in an asm statement?](#)
10. [Why does the PC randomly jump around when single-stepping through my program in avr-gdb?](#)
11. [How do I trace an assembler file in avr-gdb?](#)
12. [How do I pass an IO port as a parameter to a function?](#)
13. [What registers are used by the C compiler?](#)
14. [How do I put an array of strings completely in ROM?](#)
15. [How to use external RAM?](#)
16. [Which -O flag to use?](#)
17. [How do I relocate code to a fixed address?](#)
18. [My UART is generating nonsense! My ATmega128 keeps crashing! Port F is completely broken!](#)

19. [Why do all my "foo...bar" strings eat up the SRAM?](#)
20. [Why does the compiler compile an 8-bit operation that uses bitwise operators into a 16-bit operation in assembly?](#)
21. [How to detect RAM memory and variable overlap problems?](#)
22. [Is it really impossible to program the ATtinyXX in C?](#)
23. [What is this "clock skew detected" message?](#)
24. [Why are \(many\) interrupt flags cleared by writing a logical 1?](#)
25. [Why have "programmed" fuses the bit value 0?](#)
26. [Which AVR-specific assembler operators are available?](#)
27. [Why are interrupts re-enabled in the middle of writing the stack pointer?](#)
28. [Why are there five different linker scripts?](#)
29. [How to add a raw binary image to linker output?](#)
30. [How do I perform a software reset of the AVR?](#)
31. [I am using floating point math. Why is the compiled code so big? Why does my code not work?](#)
32. [What pitfalls exist when writing reentrant code?](#)
33. [Why are some addresses of the EEPROM corrupted \(usually address zero\)?](#)
34. [Why is my baud rate wrong?](#)
35. [On a device with more than 128 KiB of flash, how to make function pointers work?](#)
36. [Why is assigning ports in a "chain" a bad idea?](#)

11.2 My program doesn't recognize a variable updated within 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 I get "undefined reference to..." for functions like "sin()"

In order to access the mathematical functions that are declared in `<math.h>`, the linker needs to be told to also link the mathematical library, `libm.a`.

Typically, system libraries like `libm.a` are given to the final C compiler command line that performs the linking step by adding a flag `-lm` at the end. (That is, the initial *lib* and the filename suffix from the library are written immediately after a `-l` flag. So for a `libfoo.a` library, `-lfoo` needs to be provided.) This will make the linker search the library in a path known to the system.

An alternative would be to specify the full path to the `libm.a` file at the same place on the command line, i. e. *after* all the object files (`*.o`). However, since this requires knowledge of where the build system will exactly find those library files, this is deprecated for system libraries.

Back to [FAQ Index](#).

11.4 How to permanently bind a variable to a register?

This can be done with

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

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

Registers r8 through r15 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.

See [C Names Used in Assembler Code](#) for more details.

Back to [FAQ Index](#).

11.5 How to modify MCUCR or WDTCR early?

The method of early initialization (MCUCR, WDTCR or anything else) is different (and more flexible) in the current version. Basically, write a small assembler file which looks like this:

```
;; begin xram.S
#include <avr/io.h>
.section .init1,"ax",@progbits
ldi r16,_BV(SRE) | _BV(SRW)
out _SFR_IO_ADDR(MCUCR),r16
;; end xram.S
```

Assemble it, link the resulting `xram.o` with other files in your program, and this piece of code will be inserted in initialization code, which is run right after reset. See the linker script for comments about the new `.initN` sections (which one to use, etc.).

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 [Using Sections in 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](#).

11.6 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](#).

11.7 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](#).

11.8 Shouldn't I initialize all my variables?

Global and static variables are guaranteed to be initialized to 0 by the C standard. `avr-gcc` does this by placing the appropriate code into section `.init4` (see [The .initN Sections](#)). 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 initialized (i. e. they don't have an equal sign and an initialization expression to the right within the definition of the variable), they go into the `.bss` section of the file. 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 an initializer go into the `.data` 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 since the flash ROM is the only way that the compiler can tell the target device the value this variable is going to be initialized to.

Now if some programmer "wants to make doubly sure" their variables really get a 0 at program startup, and adds an initializer just containing 0 on the right-hand side, they waste space. While this waste of space applies to virtually any platform C is implemented on, it's usually not noticeable on larger machines like PCs, while the waste of flash ROM storage can be very painful on a small microcontroller like the AVR.

So in general, variables should only be explicitly initialized if the initial value is non-zero.

Note

Recent versions of GCC are now smart enough to detect this situation, and revert variables that are explicitly initialized to 0 to the `.bss` section. Still, other compilers might not do that optimization, and as the C standard guarantees the initialization, it is safe to rely on it.

Back to [FAQ Index](#).

11.9 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 Atmel 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](#).

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

So you tried this:

```
asm volatile("sbi 0x18,0x07;");
```

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,0x07;");
```

you get a compilation error: "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, instead of 0x18, gets passed to the assembler. One way to avoid this problem is:

```
asm volatile("sbi %0, 0x07" : "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.

Back to [FAQ Index](#).

11.11 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. While it is not guaranteed, very often this code runs with the exact same optimizations as it would run without the `-g` switch.

This can have unwanted side effects. 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. 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](#).

11.12 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
        ...
        eor r16, 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   1f        ; return from myfunc prematurely
        ...
        inc r16
2:      cmp r16, r18
        brlo   1b        ; jump back to top of loop
1:      pop YH
        pop YL
        pop r18
        pop r17
        pop r16
        ret
```

Back to [FAQ Index](#).

11.13 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      ; 170
10c: 88 b3      in     r24, 0x18     ; 24
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      ; 85
114: 88 e3      ldi    r24, 0x38     ; 56
116: 90 e0      ldi    r25, 0x00     ; 0
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      ; 24
11e:  80 6f          ori      r24, 0xF0        ; 240
120:  88 bb          out     0x18, r24      ; 24
```

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.nongnu.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](#).

11.14 What registers are used by the C compiler?

- **Data types:**

`char` is 8 bits, `int` is 16 bits, `long` is 32 bits, `long long` is 64 bits, `float` and `double` are 32 bits (this is the only supported floating point format), 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` 8 bits, but that is not supported by `avr-libc` and violates C standards (`int` *must* be at least 16 bits). It may be removed in a future release.

- **Call-used registers (r18-r27, r30-r31):**

May be allocated by `gcc` for local data. You *may* use them freely in assembler subroutines. Calling C subroutines can clobber any of them - the caller is responsible for saving and restoring.

Note

For the `AVR_TINY` architecture (ATtiny10 and relatives), r18 and r19 are call-saved.

- **Call-saved registers (r2-r17, r28-r29):**

May be allocated by `gcc` for local data. Calling C subroutines leaves them unchanged. Assembler 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.

Note

For the AVR_TINY architecture (ATtiny10 and relatives), r18 and r19 are call-saved, and r0 through r16 do not exist.

- **Fixed registers (r0, r1):**

Never allocated by gcc for local data, but often used for fixed purposes:

r0 - temporary register, can be clobbered by any C code (except interrupt handlers which save it), *may* be used to remember something for a while within one piece of assembler code

r1 - 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 r1`). This includes any use of the `[f]mul[s[u]]` instructions, which return their result in r1:r0. Interrupt handlers save and clear r1 on entry, and restore r1 on exit (in case it was non-zero).

Note

For the AVR_TINY architecture (ATtiny10 and relatives), r16 is used as temporary register, and r17 as zero.

- **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.

Return values: 8-bit in r24 (not r25!), 16-bit in r25:r24, up to 32 bits in r22-r25, up to 64 bits in r18-r25. 8-bit return values are zero/sign-extended to 16 bits by the called function (`unsigned char` is more efficient than `signed char` - just `clr r25`). Arguments to functions with variable argument lists (`printf` etc.) are all passed on stack, and `char` is extended to `int`.

Warning

There was no such alignment before 2000-07-01, including the old patches for gcc-2.95.2. Check your old assembler subroutines, and adjust them accordingly.

Back to [FAQ Index](#).

11.15 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>
PGM_P 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 the `.data` section).

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

```
#include <avr/pgmspace.h>
const char foo[] PROGMEM = "Foo";
const char bar[] PROGMEM = "Bar";
PGM_P array[2] PROGMEM = {
    foo,
    bar
};
int main (void)
{
    char buf[32];
    PGM_P p;
    int i;
    memcpy_P(&p, &array[i], sizeof(PGM_P));
    strcpy_P(buf, p);
    return 0;
}
```

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

```
00000026 <array>:
26:      2e 00          .word   0x002e   ; ???
28:      2a 00          .word   0x002a   ; ???
0000002a <bar>:
2a:      42 61 72 00          Bar.
0000002e <foo>:
2e:      46 6f 6f 00          Foo.
```

foo is at addr 0x002e.

bar is at addr 0x002a.

array is at addr 0x0026.

Then in main we see this:

```
memcpy_P(&p, &array[i], sizeof(PGM_P));
70:      66 0f          add     r22, r22
72:      77 1f          adc     r23, r23
74:      6a 5d          subi   r22, 0xDA    ; 218
76:      7f 4f          sbci   r23, 0xFF    ; 255
78:      42 e0          ldi     r20, 0x02    ; 2
7a:      50 e0          ldi     r21, 0x00    ; 0
7c:      ce 01          movw   r24, r28
7e:      81 96          adiw   r24, 0x21    ; 33
80:      08 d0          rcall  .+16         ; 0x92
```

This code reads the pointer to the desired string from the ROM table `array` into a register pair.

The value of `i` (in `r22:r23`) is doubled to accommodate for the word offset required to access `array[]`, then the address of `array` (0x26) is added, by subtracting the negated address (0xffda). The address of variable `p` is computed by adding its offset within the stack frame (33) to the Y pointer register, and `memcpy_P` is called.

```
strcpy_P(buf, p);
82:      69 a1          ldd     r22, Y+33    ; 0x21
84:      7a a1          ldd     r23, Y+34    ; 0x22
86:      ce 01          movw   r24, r28
88:      01 96          adiw   r24, 0x01    ; 1
8a:      0c d0          rcall  .+24         ; 0xa4
```

This will finally copy the ROM string into the local buffer `buf`.

Variable `p` (located at `Y+33`) is read, and passed together with the address of `buf` (`Y+1`) to `strcpy_P`. This will copy the string from ROM to `buf`.

Note that when using a compile-time constant index, omitting the first step (reading the pointer from ROM via `memcpy_P`) usually remains unnoticed, since the compiler would then optimize the code for accessing `array` at compile-time.

Back to [FAQ Index](#).

11.16 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 `XMCR`A and `XMCR`B, 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](#).

11.17 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 `fvprintf()` 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
-O3	6898	903 μ s	19.7 ms
-O2	6666	972 μ s	20.1 ms
-Os	6618	955 μ s	20.1 ms
-Os -mcalls-prologues	6474	972 μ s	20.1 ms

(The difference between 955 μ s and 972 μ 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](#).

11.18 How do I relocate code to a fixed address?

First, the code should be put into a new [named section](#). This is done with a section attribute:

```
__attribute__((section (".bootloader")))
```

In this example, `.bootloader` is the name of the new section. This attribute needs to be placed after the prototype of any function to force the function into the new section.

```
void boot(void) __attribute__((section (".bootloader")));
```

To relocate the section to a fixed address the linker flag `-section-start` is used. This option can be passed to the linker using the [-Wl compiler option](#):

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

The name after `section-start` is the name of the section to be relocated. The number after the section name is the beginning address of the named section.

Back to [FAQ Index](#).

11.19 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](#).

11.20 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. The compiler can optimize multiple identical strings into a single one, but obviously only for one compilation unit (i. e., a single C source file).

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](#).

11.21 Why does the compiler compile an 8-bit operation that uses bitwise operators into a 16-bit operation in assembly?

Bitwise operations in Standard C will automatically promote their operands to an `int`, which is (by default) 16 bits in `avr-gcc`.

To work around this use typecasts on the operands, including literals, to declare that the values are to be 8 bit operands.

This may be especially important when clearing a bit:

```
var &= ~mask; /* wrong way! */
```

The bitwise "not" operator (`~`) will also promote the value in `mask` to an `int`. To keep it an 8-bit value, typecast before the "not" operator:

```
var &= (unsigned char)~mask;
```

Back to [FAQ Index](#).

11.22 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](#).

11.23 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](#).

11.24 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 a `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](#).

11.25 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](#).

11.26 Why have "programmed" fuses the bit value 0?

Basically, fuses are just a bit in a special EEPROM area. For technical reasons, erased E[EEPROM] 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](#).

11.27 Which AVR-specific assembler operators are available?

See [Pseudo-ops and operators](#).

Back to [FAQ Index](#).

11.28 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](#).

11.29 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 `.xbn` script is for linking with `-N` flag (mix text and data on same page).

Back to [FAQ Index](#).

11.30 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_${*_tmp}_start=${*_tmp}_start \
        --redefine-sym _binary_${*_tmp}_end=${*_tmp}_end \
        --redefine-sym _binary_${*_tmp}_size=${*_tmp}_size \
        $(*)_tmp $(@)
    @echo "extern const char" $(*)_tmp "[] PROGMEM;" > $(*)_h
    @echo "extern const char" $(*)_tmp_end "[] PROGMEM;" >> $(*)_h
    @echo "extern const char" $(*)_tmp_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_${*_bin}_start=${*_bin}_start \
        --redefine-sym _binary_${*_bin}_end=${*_bin}_end \
        --redefine-sym _binary_${*_bin}_size=${*_bin}_size \
        $(<) $(@)
    @echo "extern const char" $(*)_bin "[] PROGMEM;" > $(*)_h
    @echo "extern const char" $(*)_bin_end "[] PROGMEM;" >> $(*)_h
    @echo "extern const char" $(*)_bin_size_sym "[];" >> $(*)_h
    @echo "#define $(*)_size ((int)$(*)_size_sym)" >> $(*)_h
```

Back to [FAQ Index](#).

11.31 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>
...
#define soft_reset() \
do \
{ \
    wdt_enable(WDTO_15MS); \
    for(;;) \
    { \
        \
    } \
} while(0)
```

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
void wdt_init(void) __attribute__((naked)) __attribute__((section(".init3")));
...
// Function Implementation
void wdt_init(void)
{
    MCUSR = 0;
    wdt_disable();
    return;
}

```

Back to [FAQ Index](#).

11.32 I am using floating point math. Why is the compiled code so big? Why does my code not work?

You are not linking in the math library from AVR-LibC. GCC has a library that is used for floating point operations, but it is not optimized for the AVR, and so it generates big code, or it could be incorrect. This can happen even when you are not using any floating point math functions from the Standard C library, but you are just doing floating point math operations.

When you link in the math library from AVR-LibC, those routines get replaced by hand-optimized AVR assembly and it produces much smaller code.

See [I get "undefined reference to..." for functions like "sin\(\)"](#) for more details on how to link in the math library.

Back to [FAQ Index](#).

11.33 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
rand() , random()	Uses global variables to keep state information.	Use special reentrant versions↔: rand_r() , random_r() .
strtod() , strtol() , strtoul()	Uses the global variable <code>errno</code> to return success/failure.	Ignore <code>errno</code> , or protect calls with cli()/sei() or ATOMIC_BLOCK() if the application can tolerate it. Or use sccanf() or sccanf_P() 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 ISRs, 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>vprintf_P()</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 *_P 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 *_P versions.
<code>assert()</code>	Contains an embedded <code>fprintf()</code> . See above for <code>fprintf()</code> .	See above for <code>fprintf()</code> .
<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 ungetc 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](#).

11.34 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](#).

11.35 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](#).

11.36 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.)

Back to [FAQ Index](#).

11.37 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.

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.

12.1 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 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.

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
```

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.

12.2 Required Tools

- **GNU Binutils**
<http://sources.redhat.com/binutils/Installation>
- **GCC**
<http://gcc.gnu.org/Installation>
- **AVR LibC**
<http://savannah.gnu.org/projects/avr-libc/Installation>

12.3 Optional Tools

You can develop programs for AVR devices without the following tools. They may or may not be of use for you.

- **AVRDUDE**

<http://savannah.nongnu.org/projects/avrdude/>
[Installation](#)
[Usage Notes](#)

- **GDB**

<http://sources.redhat.com/gdb/>
[Installation](#)

- **SimulAVR**

<http://savannah.gnu.org/projects/simulavr/>
[Installation](#)

- **AVaRICE**

<http://avarice.sourceforge.net/>
[Installation](#)

12.4 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:

```
$ bunzip2 -c binutils-<version>.tar.bz2 | tar xf -  
$ cd binutils-<version>
```

Note

Replace `<version>` with the version of the package you downloaded.

If you obtained a gzip compressed file (`.gz`), use `gunzip` instead of `bunzip2`.

It is usually a good idea to configure and build **binutils** in a subdirectory so as not to pollute the source with the compiled files. This is recommended by the **binutils** developers.

```
$ mkdir obj-avr  
$ cd obj-avr
```

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.

```
$ ../configure --prefix=$PREFIX --target=avr --disable-nls
```

If you don't specify the `-prefix` option, the tools will get installed in the `/usr/local` hierarchy (i.e. the binaries will get installed in `/usr/local/bin`, the info pages get installed in `/usr/local/info`, etc.) Since these tools are changing frequently, it is preferable to put them in a location that is easily removed.

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. At this point, you can build the project.

```
$ 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`.

12.5 GCC for the AVR target

Warning

You **must** install [avr-binutils](#) and make sure your [path is set](#) properly before installing `avr-gcc`.

The steps to build `avr-gcc` are essentially same as for [binutils](#):

```
$ bunzip2 -c gcc-<version>.tar.bz2 | tar xf -
$ cd gcc-<version>
$ mkdir obj-avr
$ cd obj-avr
$ ../configure --prefix=$PREFIX --target=avr --enable-languages=c,c++ \
  --disable-nls --disable-libssp --with-dwarf2
$ make
$ make install
```

To save your self some download time, you can alternatively download only the `gcc-core-<version>.tar.bz2` and `gcc-c++-<version>.tar.bz2` parts of the gcc. Also, if you don't need C++ support, you only need the core part and should only enable the C language support. (Starting with GCC 4.7 releases, these split files are no longer available though.)

Note

Early versions of these tools did not support C++.

The `stdc++` libs are not included with C++ for AVR due to the size limitations of the devices.

12.6 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 cvs, you will have to run the `bootstrap` script before using either of the build methods described below.

To build and install avr-libc:

```
$ gunzip -c avr-libc-<version>.tar.gz | tar xf -
$ cd avr-libc-<version>
$ ./configure --prefix=$PREFIX --build='./config.guess' --host=avr
$ make
$ make install
```

Optionally, generation of debug information can be requested with:

```
$ gunzip -c avr-libc-<version>.tar.gz | tar xf -
$ cd avr-libc-<version>
$ ./configure --prefix=$PREFIX --build='./config.guess' --host=avr \
  --with-debug-info=DEBUG_INFO
$ make
$ make install
```

where `DEBUG_INFO` can be one of `stabs`, `dwarf-2`, or `dwarf-4`.

The default is to not generate any debug information, which is suitable for binary distributions of avr-libc, where the user does not have the source code installed the debug information would refer to.

12.7 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 `pp1(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
```

12.8 GDB for the AVR target

GDB also uses the `configure` system, so to build and install:

```
$ bunzip2 -c gdb-<version>.tar.bz2 | tar xf -
$ cd gdb-<version>
$ mkdir obj-avr
$ cd obj-avr
$ ../configure --prefix=$PREFIX --target=avr
$ make
$ make install
```

Note

If you are planning on using `avr-gdb`, you will probably want to install either [simulavr](#) or [avarice](#) since `avr-gdb` needs one of these to run as a remote target backend.

12.9 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.

12.10 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
```

12.11 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.

12.12 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, 20100909 (or later) <http://sourceforge.net/projects/mingw/files/Automated%20MinGW%20Installer/mingw-get-inst/mingw-get-inst.exe/download>

- Run `mingw-get-inst-20100909.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, and MPC are required to build GCC.

GMP is a prerequisite for building MPFR. Build GMP first.

MPFR is a prerequisite for building MPC. Build MPFR second.

- **Build GMP for MinGW**

- Latest Version

- <http://gmplib.org/>

- Build script:

```
./configure 2>&1 | tee gmp-configure.log
make        2>&1 | tee gmp-make.log
make check  2>&1 | tee gmp-make-check.log
make install 2>&1 | tee gmp-make-install.log
```

- GMP headers will be installed under `/usr/local/include` and library installed under `/usr/local/lib`.

- **Build MPFR for MinGW**

- Latest Version

- <http://www.mpfr.org/>

- Build script:

```
./configure --with-gmp=/usr/local --disable-shared 2>&1 | tee mpfr-configure.log
make        2>&1 | tee mpfr-make.log
make check  2>&1 | tee mpfr-make-check.log
make install 2>&1 | tee mpfr-make-install.log
```

- MPFR headers will be installed under `/usr/local/include` and library installed under `/usr/local/lib`.

- **Build MPC for MinGW**

- Latest Version

- <http://www.multiprecision.org/>

- Build script:

```
./configure --with-gmp=/usr/local --with-mpfr=/usr/local --disable-shared 2>&1 | tee mpfr-co
make        2>&1 | tee mpfr-make.log
make check  2>&1 | tee mpfr-make-check.log
make install 2>&1 | tee mpfr-make-install.log
```

- MPFR headers will be installed under `/usr/local/include` and library installed under `/usr/local/lib`.

Note

Doxygen is required to build AVR-LibC documentation.

- **Install Doxygen**

- Version 1.7.2

- <http://www.stack.nl/~dimitri/doxygen/>

- 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.netpbm.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://www.schmidt-web-berlin.de/winfig/>
- 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
- <http://miktex.org/>
- Download and install.

Ghostscript is required to build various documentation.

- **Install Ghostscript**

- Version 9.00
- <http://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.

12.13 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 \
--with-gmp=/usr/local \
--with-mpfr=/usr/local \
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

```
LDFLAGS='-L /usr/local/lib -R /usr/local/lib' \
CFLAGS='-D__USE_MINGW_ACCESS' \
../gcc-$version/configure \
--with-gmp=/usr/local \
--with-mpfr=/usr/local \
--with-mpc=/usr/local \
--prefix=$installdir \
--target=$target \
--enable-languages=c,c++ \
--with-dwarf2 \
--enable-doc \
--with-docdir=$installdir/doc/$project \
--disable-shared \
--disable-libada \
--disable-libssp \
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 \  
LDFLAGS='-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
```

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. In addition to the preprocessor macros indicated in the tables below, 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 when using the standard levels `gnu89` (default) and `gnu99` but not with `c89` and `c99`.

- `-mmcu=architecture`

Compile code for *architecture*. Currently known architectures are

Architecture	Macros	Description
avr1	<code>__AVR_ARCH__=1</code> <code>__AVR_ASM_ONLY__</code> <code>__AVR_2_BYTE_PC__ [2]</code>	Simple CPU core, only assembler support
avr2	<code>__AVR_ARCH__=2</code> <code>__AVR_2_BYTE_PC__ [2]</code>	"Classic" CPU core, up to 8 KB of ROM
avr25 [1]	<code>__AVR_ARCH__=25</code> <code>__AVR_HAVE_MO VW__ [1]</code> <code>__AVR_HAVE_LPMX__ [1]</code> <code>__AVR_2_BYTE_PC__ [2]</code>	"Classic" CPU core with 'MOVW' and 'LPM Rx, Z[+]' instruction, up to 8 KB of ROM
avr3	<code>__AVR_ARCH__=3</code> <code>__AVR_MEGA__ [5]</code> <code>__AVR_HAVE_JMP_CALL__ [4]</code> <code>__AVR_2_BYTE_PC__ [2]</code>	"Classic" CPU core, 16 KB to 64 KB of ROM
avr31	<code>__AVR_ARCH__=31</code> <code>__AVR_MEGA__ [5]</code> <code>__AVR_HAVE_JMP_CALL__ [4]</code> <code>__AVR_HAVE_RAMPZ__ [4]</code> <code>__AVR_HAVE_ELPM__ [4]</code> <code>__AVR_2_BYTE_PC__ [2]</code>	"Classic" CPU core, 128 KB of ROM
avr35 [3]	<code>__AVR_ARCH__=35</code> <code>__AVR_MEGA__ [5]</code> <code>__AVR_HAVE_JMP_CALL__ [4]</code> <code>__AVR_HAVE_MO VW__ [1]</code> <code>__AVR_HAVE_LPMX__ [1]</code> <code>__AVR_2_BYTE_PC__ [2]</code>	"Classic" CPU core with 'MOVW' and 'LPM Rx, Z[+]' instruction, 16 KB to 64 KB of ROM
avr4	<code>__AVR_ARCH__=4</code> <code>__AVR_ENHANCED__ [5]</code> <code>__AVR_HAVE_MO VW__ [1]</code> <code>__AVR_HAVE_LPMX__ [1]</code> <code>__AVR_HAVE_MUL__ [1]</code> <code>__AVR_2_BYTE_PC__ [2]</code>	"Enhanced" CPU core, up to 8 KB of ROM

avr5	<pre>__AVR_ARCH__=5 __AVR_MEGA__ [5] __AVR_ENHANCED__ [5] __AVR_HAVE_JMP_CALL__ [4] __AVR_HAVE_MO VW__ [1] __AVR_HAVE_LPMX__ [1] __AVR_HAVE_MUL__ [1] __AVR_2_BYTE_PC__ [2]</pre>	"Enhanced" CPU core, 16 KB to 64 KB of ROM
avr51	<pre>__AVR_ARCH__=51 __AVR_MEGA__ [5] __AVR_ENHANCED__ [5] __AVR_HAVE_JMP_CALL__ [4] __AVR_HAVE_MO VW__ [1] __AVR_HAVE_LPMX__ [1] __AVR_HAVE_MUL__ [1] __AVR_HAVE_RAMPZ__ [4] __AVR_HAVE_ELPM__ [4] __AVR_HAVE_ELPMX__ [4] __AVR_2_BYTE_PC__ [2]</pre>	"Enhanced" CPU core, 128 KB of ROM
avr6 [2]	<pre>__AVR_ARCH__=6 __AVR_MEGA__ [5] __AVR_ENHANCED__ [5] __AVR_HAVE_JMP_CALL__ [4] __AVR_HAVE_MO VW__ [1] __AVR_HAVE_LPMX__ [1] __AVR_HAVE_MUL__ [1] __AVR_HAVE_RAMPZ__ [4] __AVR_HAVE_ELPM__ [4] __AVR_HAVE_ELPMX__ [4] __AVR_3_BYTE_PC__ [2]</pre>	"Enhanced" CPU core, 256 KB of ROM

[1] New in GCC 4.2

[2] Unofficial patch for GCC 4.1

[3] New in GCC 4.2.3

[4] New in GCC 4.3

[5] Obsolete.

By default, code is generated for the `avr2` architecture.

Note that when only using `-mmcu=architecture` but no `-mmcu=MCU type`, including the file `<avr/io.h>` cannot work since it cannot decide which device's definitions to select.

- `-mmcu=MCU type`

The following MCU types are currently understood by `avr-gcc`. The table matches them against the corresponding `avr-gcc` architecture name, and shows the preprocessor symbol declared by the `-mmcu` option.

Architecture	MCU name	Macro
avr1	at90s1200	<code>__AVR_AT90S1200__</code>
avr1	attiny11	<code>__AVR_ATtiny11__</code>
avr1	attiny12	<code>__AVR_ATtiny12__</code>
avr1	attiny15	<code>__AVR_ATtiny15__</code>
avr1	attiny28	<code>__AVR_ATtiny28__</code>

avr2	at90s2313	__AVR_AT90S2313__
avr2	at90s2323	__AVR_AT90S2323__
avr2	at90s2333	__AVR_AT90S2333__
avr2	at90s2343	__AVR_AT90S2343__
avr2	attiny22	__AVR_ATtiny22__
avr2	attiny26	__AVR_ATtiny26__
avr2	at90s4414	__AVR_AT90S4414__
avr2	at90s4433	__AVR_AT90S4433__
avr2	at90s4434	__AVR_AT90S4434__
avr2	at90s8515	__AVR_AT90S8515__
avr2	at90c8534	__AVR_AT90C8534__
avr2	at90s8535	__AVR_AT90S8535__
avr2/avr25 [1]	at86rf401	__AVR_AT86RF401__
avr2/avr25 [1]	ata5272	__AVR_ATA5272__
avr2/avr25 [1]	ata6616c	__AVR_ATA6616C__
avr2/avr25 [1]	attiny13	__AVR_ATtiny13__
avr2/avr25 [1]	attiny13a	__AVR_ATtiny13A__
avr2/avr25 [1]	attiny2313	__AVR_ATtiny2313__
avr2/avr25 [1]	attiny2313a	__AVR_ATtiny2313A__
avr2/avr25 [1]	attiny24	__AVR_ATtiny24__
avr2/avr25 [1]	attiny24a	__AVR_ATtiny24A__
avr2/avr25 [1]	attiny25	__AVR_ATtiny25__
avr2/avr25 [1]	attiny261	__AVR_ATtiny261__
avr2/avr25 [1]	attiny261a	__AVR_ATtiny261A__
avr2/avr25 [1]	attiny4313	__AVR_ATtiny4313__
avr2/avr25 [1]	attiny43u	__AVR_ATtiny43U__
avr2/avr25 [1]	attiny44	__AVR_ATtiny44__
avr2/avr25 [1]	attiny44a	__AVR_ATtiny44A__
avr2/avr25 [1]	attiny441	__AVR_ATtiny441__
avr2/avr25 [1]	attiny45	__AVR_ATtiny45__
avr2/avr25 [1]	attiny461	__AVR_ATtiny461__
avr2/avr25 [1]	attiny461a	__AVR_ATtiny461A__
avr2/avr25 [1]	attiny48	__AVR_ATtiny48__
avr2/avr25 [1]	attiny828	__AVR_ATtiny828__
avr2/avr25 [1]	attiny84	__AVR_ATtiny84__
avr2/avr25 [1]	attiny84a	__AVR_ATtiny84A__
avr2/avr25 [1]	attiny841	__AVR_ATtiny841__
avr2/avr25 [1]	attiny85	__AVR_ATtiny85__
avr2/avr25 [1]	attiny861	__AVR_ATtiny861__
avr2/avr25 [1]	attiny861a	__AVR_ATtiny861A__
avr2/avr25 [1]	attiny87	__AVR_ATtiny87__
avr2/avr25 [1]	attiny88	__AVR_ATtiny88__
avr3	atmega603	__AVR_ATmega603__
avr3	at43usb355	__AVR_AT43USB355__
avr3/avr31 [3]	atmega103	__AVR_ATmega103__
avr3/avr31 [3]	at43usb320	__AVR_AT43USB320__
avr3/avr35 [2]	at90usb82	__AVR_AT90USB82__
avr3/avr35 [2]	at90usb162	__AVR_AT90USB162__

avr3/avr35 [2]	ata5505	__AVR_ATA5505__
avr3/avr35 [2]	ata6617c	__AVR_ATA6617C__
avr3/avr35 [2]	ata664251	__AVR_ATA664251__
avr3/avr35 [2]	atmega8u2	__AVR_ATmega8U2__
avr3/avr35 [2]	atmega16u2	__AVR_ATmega16U2__
avr3/avr35 [2]	atmega32u2	__AVR_ATmega32U2__
avr3/avr35 [2]	attiny167	__AVR_ATTiny167__
avr3/avr35 [2]	attiny1634	__AVR_ATTiny1634__
avr3	at76c711	__AVR_AT76C711__
avr4	ata6285	__AVR_ATA6285__
avr4	ata6286	__AVR_ATA6286__
avr4	ata6289	__AVR_ATA6289__
avr4	ata6612c	__AVR_ATA6612C__
avr4	atmega48	__AVR_ATmega48__
avr4	atmega48a	__AVR_ATmega48A__
avr4	atmega48pa	__AVR_ATmega48PA__
avr4	atmega48pb	__AVR_ATmega48PB__
avr4	atmega48p	__AVR_ATmega48P__
avr4	atmega8	__AVR_ATmega8__
avr4	atmega8a	__AVR_ATmega8A__
avr4	atmega8515	__AVR_ATmega8515__
avr4	atmega8535	__AVR_ATmega8535__
avr4	atmega88	__AVR_ATmega88__
avr4	atmega88a	__AVR_ATmega88A__
avr4	atmega88p	__AVR_ATmega88P__
avr4	atmega88pa	__AVR_ATmega88PA__
avr4	atmega88pb	__AVR_ATmega88PB__
avr4	atmega8hva	__AVR_ATmega8HVA__
avr4	at90pwm1	__AVR_AT90PWM1__
avr4	at90pwm2	__AVR_AT90PWM2__
avr4	at90pwm2b	__AVR_AT90PWM2B__
avr4	at90pwm3	__AVR_AT90PWM3__
avr4	at90pwm3b	__AVR_AT90PWM3B__
avr4	at90pwm81	__AVR_AT90PWM81__
avr5	at90can32	__AVR_AT90CAN32__
avr5	at90can64	__AVR_AT90CAN64__
avr5	at90pwm161	__AVR_AT90PWM161__
avr5	at90pwm216	__AVR_AT90PWM216__
avr5	at90pwm316	__AVR_AT90PWM316__
avr5	at90scr100	__AVR_AT90SCR100__
avr5	at90usb646	__AVR_AT90USB646__
avr5	at90usb647	__AVR_AT90USB647__
avr5	at94k	__AVR_AT94K__
avr5	atmega16	__AVR_ATmega16__
avr5	ata5702m322	__AVR_ATA5702M322__
avr5	ata5782	__AVR_ATA5782__
avr5	ata5790	__AVR_ATA5790__
avr5	ata5790n	__AVR_ATA5790N__
avr5	ata5795	__AVR_ATA5795__

avr5	ata5831	__AVR_ATA5831__
avr5	ata6613c	__AVR_ATA6613C__
avr5	ata6614q	__AVR_ATA6614Q__
avr5	atmega161	__AVR_ATmega161__
avr5	atmega162	__AVR_ATmega162__
avr5	atmega163	__AVR_ATmega163__
avr5	atmega164a	__AVR_ATmega164A__
avr5	atmega164p	__AVR_ATmega164P__
avr5	atmega164pa	__AVR_ATmega164PA__
avr5	atmega165	__AVR_ATmega165__
avr5	atmega165a	__AVR_ATmega165A__
avr5	atmega165p	__AVR_ATmega165P__
avr5	atmega165pa	__AVR_ATmega165PA__
avr5	atmega168	__AVR_ATmega168__
avr5	atmega168a	__AVR_ATmega168A__
avr5	atmega168p	__AVR_ATmega168P__
avr5	atmega168pa	__AVR_ATmega168PA__
avr5	atmega169	__AVR_ATmega169__
avr5	atmega169a	__AVR_ATmega169A__
avr5	atmega169p	__AVR_ATmega169P__
avr5	atmega169pa	__AVR_ATmega169PA__
avr5	atmega16a	__AVR_ATmega16A__
avr5	atmega16hva	__AVR_ATmega16HVA__
avr5	atmega16hva2	__AVR_ATmega16HVA2__
avr5	atmega16hvb	__AVR_ATmega16HVB__
avr5	atmega16hvbrevb	__AVR_ATmega16HVBREVB__↵
avr5	atmega16m1	__AVR_ATmega16M1__
avr5	atmega16u4	__AVR_ATmega16U4__
avr5	atmega32	__AVR_ATmega32__
avr5	atmega32a	__AVR_ATmega32A__
avr5	atmega323	__AVR_ATmega323__
avr5	atmega324a	__AVR_ATmega324A__
avr5	atmega324p	__AVR_ATmega324P__
avr5	atmega324pa	__AVR_ATmega324PA__
avr5	atmega325	__AVR_ATmega325__
avr5	atmega325a	__AVR_ATmega325A__
avr5	atmega325p	__AVR_ATmega325P__
avr5	atmega325pa	__AVR_ATmega325PA__
avr5	atmega3250	__AVR_ATmega3250__
avr5	atmega3250a	__AVR_ATmega3250A__
avr5	atmega3250p	__AVR_ATmega3250P__
avr5	atmega3250pa	__AVR_ATmega3250PA__
avr5	atmega328	__AVR_ATmega328__
avr5	atmega328p	__AVR_ATmega328P__
avr5	atmega329	__AVR_ATmega329__
avr5	atmega329a	__AVR_ATmega329A__
avr5	atmega329p	__AVR_ATmega329P__
avr5	atmega329pa	__AVR_ATmega329PA__
avr5	atmega3290	__AVR_ATmega3290__

avr5	atmega3290a	__AVR_ATmega3290A__
avr5	atmega3290p	__AVR_ATmega3290P__
avr5	atmega3290pa	__AVR_ATmega3290PA__
avr5	atmega32c1	__AVR_ATmega32C1__
avr5	atmega32hvb	__AVR_ATmega32HVB__
avr5	atmega32hvbrevb	__AVR_ATmega32HVBREVB__
avr5	atmega32m1	__AVR_ATmega32M1__
avr5	atmega32u4	__AVR_ATmega32U4__
avr5	atmega32u6	__AVR_ATmega32U6__
avr5	atmega406	__AVR_ATmega406__
avr5	atmega644rfr2	__AVR_ATmega644RFR2__
avr5	atmega64rfr2	__AVR_ATmega64RFR2__
avr5	atmega64	__AVR_ATmega64__
avr5	atmega64a	__AVR_ATmega64A__
avr5	atmega640	__AVR_ATmega640__
avr5	atmega644	__AVR_ATmega644__
avr5	atmega644a	__AVR_ATmega644A__
avr5	atmega644p	__AVR_ATmega644P__
avr5	atmega644pa	__AVR_ATmega644PA__
avr5	atmega645	__AVR_ATmega645__
avr5	atmega645a	__AVR_ATmega645A__
avr5	atmega645p	__AVR_ATmega645P__
avr5	atmega6450	__AVR_ATmega6450__
avr5	atmega6450a	__AVR_ATmega6450A__
avr5	atmega6450p	__AVR_ATmega6450P__
avr5	atmega649	__AVR_ATmega649__
avr5	atmega649a	__AVR_ATmega649A__
avr5	atmega6490	__AVR_ATmega6490__
avr5	atmega6490a	__AVR_ATmega6490A__
avr5	atmega6490p	__AVR_ATmega6490P__
avr5	atmega649p	__AVR_ATmega649P__
avr5	atmega64c1	__AVR_ATmega64C1__
avr5	atmega64hve	__AVR_ATmega64HVE__
avr5	atmega64hve2	__AVR_ATmega64HVE2__
avr5	atmega64m1	__AVR_ATmega64M1__
avr5	m3000	__AVR_M3000__
avr5/avr51 [3]	at90can128	__AVR_AT90CAN128__
avr5/avr51 [3]	at90usb1286	__AVR_AT90USB1286__
avr5/avr51 [3]	at90usb1287	__AVR_AT90USB1287__
avr5/avr51 [3]	atmega128	__AVR_ATmega128__
avr5/avr51 [3]	atmega128a	__AVR_ATmega128A__
avr5/avr51 [3]	atmega1280	__AVR_ATmega1280__
avr5/avr51 [3]	atmega1281	__AVR_ATmega1281__
avr5/avr51 [3]	atmega1284	__AVR_ATmega1284__
avr5/avr51 [3]	atmega1284p	__AVR_ATmega1284P__
avr5/avr51 [3]	atmega1284rfr2	__AVR_ATmega1284RFR2__
avr5/avr51 [3]	atmega128rfr2	__AVR_ATmega128RFR2__
avr6	atmega2560	__AVR_ATmega2560__

avr6	atmega2561	__AVR_ATmega2561__
avr6	atmega2564rfr2	__AVR_ATmega2564RFR2__
avr6	atmega256rfr2	__AVR_ATmega256RFR2__
avrxmega2	atxmega8e5	__AVR_ATxmega8E5__
avrxmega2	atxmega16a4	__AVR_ATxmega16A4__
avrxmega2	atxmega16a4u	__AVR_ATxmega16A4U__
avrxmega2	atxmega16c4	__AVR_ATxmega16C4__
avrxmega2	atxmega16d4	__AVR_ATxmega16D4__
avrxmega2	atxmega32a4	__AVR_ATxmega32A4__
avrxmega2	atxmega32a4u	__AVR_ATxmega32A4U__
avrxmega2	atxmega32c3	__AVR_ATxmega32C3__
avrxmega2	atxmega32c4	__AVR_ATxmega32C4__
avrxmega2	atxmega32d3	__AVR_ATxmega32D3__
avrxmega2	atxmega32d4	__AVR_ATxmega32D4__
avrxmega2	atxmega32e5	__AVR_ATxmega32E5__
avrxmega4	atxmega64a3	__AVR_ATxmega64A3__
avrxmega4	atxmega64a3u	__AVR_ATxmega64A3U__
avrxmega4	atxmega64a4u	__AVR_ATxmega64A4U__
avrxmega4	atxmega64b1	__AVR_ATxmega64B1__
avrxmega4	atxmega64b3	__AVR_ATxmega64B3__
avrxmega4	atxmega64c3	__AVR_ATxmega64C3__
avrxmega4	atxmega64d3	__AVR_ATxmega64D3__
avrxmega4	atxmega64d4	__AVR_ATxmega64D4__
avrxmega5	atxmega64a1	__AVR_ATxmega64A1__
avrxmega5	atxmega64a1u	__AVR_ATxmega64A1U__
avrxmega6	atxmega128a3	__AVR_ATxmega128A3__
avrxmega6	atxmega128a3u	__AVR_ATxmega128A3U__
avrxmega6	atxmega128b1	__AVR_ATxmega128B1__
avrxmega6	atxmega128b3	__AVR_ATxmega128B3__
avrxmega6	atxmega128c3	__AVR_ATxmega128C3__
avrxmega6	atxmega128d3	__AVR_ATxmega128D3__
avrxmega6	atxmega128d4	__AVR_ATxmega128D4__
avrxmega6	atxmega192a3	__AVR_ATxmega192A3__
avrxmega6	atxmega192a3u	__AVR_ATxmega192A3U__
avrxmega6	atxmega192c3	__AVR_ATxmega192C3__
avrxmega6	atxmega192d3	__AVR_ATxmega192D3__
avrxmega6	atxmega256a3	__AVR_ATxmega256A3__
avrxmega6	atxmega256a3u	__AVR_ATxmega256A3U__
avrxmega6	atxmega256a3b	__AVR_ATxmega256A3B__
avrxmega6	atxmega256a3bu	__AVR_ATxmega256A3BU__
avrxmega6	atxmega256c3	__AVR_ATxmega256C3__
avrxmega6	atxmega256d3	__AVR_ATxmega256D3__
avrxmega6	atxmega384c3	__AVR_ATxmega384C3__
avrxmega6	atxmega384d3	__AVR_ATxmega384D3__
avrxmega7	atxmega128a1	__AVR_ATxmega128A1__
avrxmega7	atxmega128a1u	__AVR_ATxmega128A1U__

<code>avrxmega7</code>	<code>atxmega128a4u</code>	<code>__AVR_ATxmega128A4U__</code>
<code>avrtiny10</code>	<code>attiny4</code>	<code>__AVR_ATtiny4__</code>
<code>avrtiny10</code>	<code>attiny5</code>	<code>__AVR_ATtiny5__</code>
<code>avrtiny10</code>	<code>attiny9</code>	<code>__AVR_ATtiny9__</code>
<code>avrtiny10</code>	<code>attiny10</code>	<code>__AVR_ATtiny10__</code>
<code>avrtiny10</code>	<code>attiny20</code>	<code>__AVR_ATtiny20__</code>
<code>avrtiny10</code>	<code>attiny40</code>	<code>__AVR_ATtiny40__</code>

[1] 'avr25' architecture is new in GCC 4.2

[2] 'avr35' architecture is new in GCC 4.2.3

[3] 'avr31' and 'avr51' architectures is new in GCC 4.3

- `-morder1`
- `-morder2`

Change the order of register assignment. The default is

`r24, r25, r18, r19, r20, r21, r22, r23, r30, r31, r26, r27, r28, r29, r17, r16, r15, r14, r13, r12, r11, r10, r9, r8, r7, r6, r5, r4, r3, r2, r0, r1`

Order 1 uses

`r18, r19, r20, r21, r22, r23, r24, r25, r30, r31, r26, r27, r28, r29, r17, r16, r15, r14, r13, r12, r11, r10, r9, r8, r7, r6, r5, r4, r3, r2, r0, r1`

Order 2 uses

`r25, r24, r23, r22, r21, r20, r19, r18, r30, r31, r26, r27, r28, r29, r17, r16, r15, r14, r13, r12, r11, r10, r9, r8, r7, r6, r5, r4, r3, r2, r1, r0`

- `-mint8`

Assume `int` to be an 8-bit integer. Note that this is not really supported by `avr-libc`, so it should normally not be used. The default is to use 16-bit integers.

- `-mno-interrupts`

Generates code that changes the stack pointer without disabling interrupts. Normally, the state of the status register `SREG` is saved in a temporary register, interrupts are disabled while changing the stack pointer, and `SREG` is restored.

Specifying this option will define the preprocessor macro `__NO_INTERRUPTS__` to the value 1.

- `-mcall-prologues`

Use subroutines for function prologue/epilogue. For complex functions that use many registers (that needs to be saved/restored on function entry/exit), this saves some space at the cost of a slightly increased execution time.

- `-mtiny-stack`

Change only the low 8 bits of the stack pointer.

- `-mno-tablejump`

Deprecated, use `-fno-jump-tables` instead.

- `-mshort-calls`

Use `rjmp/rcall` (limited range) on >8K devices. On `avr2` and `avr4` architectures (less than 8 KB of flash memory), this is always the case. On `avr3` and `avr5` architectures, calls and jumps to targets outside the current function will by default use `jmp/call` instructions that can cover the entire address range, but that require more flash ROM and execution time.

- `-mrtl`

Dump the internal compilation result called "RTL" into comments in the generated assembler code. Used for debugging `avr-gcc`.

- `-msize`

Dump the address, size, and relative cost of each statement into comments in the generated assembler code. Used for debugging `avr-gcc`.

- `-mdeb`

Generate lots of debugging information to `stderr`.

13.1.2 Selected general compiler options

The following general gcc options might be of some interest to AVR users.

- `-O n`

Optimization level n . Increasing n is meant to optimize more, an optimization level of 0 means no optimization at all, which is the default if no `-O` option is present. The special option `-Os` is meant to turn on all `-O2` optimizations that are not expected to increase code size.

Note that at `-O3`, gcc attempts to inline all "simple" functions. For the AVR target, this will normally constitute a large pessimization due to the code increasement. The only other optimization turned on with `-O3` is `-frename-registers`, which could rather be enabled manually instead.

A simple `-O` option is equivalent to `-O1`.

Note also that 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).

See also the [appropriate FAQ entry](#) for issues regarding debugging optimized code.

- `-Wa, assembler-options`
- `-Wl, linker-options`

Pass the listed options to the assembler, or linker, respectively.

- `-g`

Generate debugging information that can be used by `avr-gdb`.

- `-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 which makes some sense in a microcontroller environment where the application cannot meaningfully provide a return value to its environment (in most cases, `main()` won't even return anyway). However, this 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. E. g., 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.

- `-funsigned-char`

Make any unqualified `char` type an unsigned char. Without this option, they default to a signed char.

- `-funsigned-bitfields`

Make any unqualified bitfield type unsigned. By default, they are signed.

- `-fshort-enums`

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`

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: The `tablejump` instructions use the LPM assembler instruction for access to jump tables. Always use `-fno-jump-tables` switch, if compiling a bootloader for devices with more than 64 KB of code memory.

13.2 Options for the assembler `avr-as`

13.2.1 Machine-specific assembler options

- `-mmcu=architecture`
- `-mmcu=MCU name`

`avr-as` understands the same `-mmcu=` options as [avr-gcc](#). By default, `avr2` is assumed, but this can be altered by using the appropriate `.arch` pseudo-instruction inside the assembler source file.

- `-mall-opcodes`

Turns off opcode checking for the actual MCU type, 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 KB 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 [above](#)), so in order to include the C source code into the assembler listing in file `foo.lst`, when compiling `foo.c`, the following compiler command-line can be used:

```
$ avr-gcc -c -O foo.c -o foo.o -Wa,-ahls=foo.lst
```

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.S -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`

13.3.1 Selected linker options

While there are no machine-specific options for `avr-ld`, a number of the standard options might be of interest to AVR users.

- `-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 compile 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 location to look for archive libraries requested by `-l` options.

- `-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`.

- `-section-start sectionname=org`

Start section *sectionname* at absolute address *org*.

- `-Tbss org`
- `-Tdata org`
- `-Ttext org`

Start the `bss`, `data`, or `text` section at *org*, respectively.

- `-T scriptfile`

Use *scriptfile* as the linker script, replacing 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 `avr5`) with the suffix `.x` appended. They describe how the various [memory sections](#) will be linked together.

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 two system libraries, `libc.a`, and `libm.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` needs to be explicitly requested using `-lm`. See also the [entry in the FAQ](#) explaining this.

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 [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 -O -o foo.out -Wl,-Map=foo.map -Wl,--cref foo.c
```


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 -o foo.out -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. These don't constitute a variable access at all, not even volatile, so the compiler is free to move them around. Although there is a "volatile" qualifier which can be attached to the `__asm__()` statement, its effect on (re)ordering is not clear from the documentation (and is more likely only to prevent complete removal by the optimiser), as it (among other) states:

Note that even a volatile asm instruction can be moved relative to other code, including across jump instructions. [...] Similarly, you can't expect a sequence of volatile asm instructions to remain perfectly consecutive.

See also

<http://gcc.gnu.org/onlinedocs/gcc-4.3.4/gcc/Extended-Asm.html>

There is another mechanism which can be used to achieve something similar: *memory barriers*. This is accomplished through adding a special "memory" clobber to the inline `asm` statement, and ensures that all variables are flushed from registers to memory before the statement, and then re-read after the statement. The purpose of memory barriers is slightly different than to enforce code ordering: it is supposed to ensure that there are no variables "cached" in registers, so that it is safe to change the content of registers e.g. when switching context in a multitasking OS (on "big" processors with out-of-order execution they also imply usage of special instructions which force the processor into "in-order" state (this is not the case of AVR)).

However, memory barrier works well in ensuring that all volatile accesses before and after the barrier occur in the given order with respect to the barrier. However, it does not ensure the compiler moving non-volatile-related statements across the barrier. Peter Dannegger 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();
}
```

compiles 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.

To sum it up:

- memory barriers ensure proper ordering of volatile accesses
- 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].

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 Release Numbering and Methodology

16.1 Release Version Numbering Scheme

Release numbers consist of three parts, a major number, a minor number, and a revision number, each separated by a dot.

The major number is currently 2, to indicate the multilib layout has been adapted to the fairly different one used starting with AVR-GCC version 5. Nevertheless, it is still believed to be generally API-compatible with release versions 1.x.

In the past (up to 1.6.x), even minor numbers have been used to indicate "stable" releases, and odd minor numbers have been reserved for development branches/versions. As the latter has never really been used, and maintaining a stable branch that eventually became effectively the same as the development version has proven to be just a cumbersome and tedious job, this scheme has given up in early 2010, so starting with 1.7.0, every minor number will be used. Minor numbers will be bumped upon judgement of the development team, whenever it seems appropriate, but at least in cases where some API was changed.

Starting with version 1.4.0, a file [<avr/version.h>](#) indicates the library version of an installed library tree.

16.2 Releasing AVR Libc

The information in this section is only relevant to AVR Libc developers and can be ignored by end users.

Note

In what follows, I assume you know how to use SVN and how to checkout multiple source trees in a single directory without having them clobber each other. If you don't know how to do this, you probably shouldn't be making releases or cutting branches.

16.2.1 Creating an SVN branch

The following steps should be taken to cut a branch in SVN (assuming \$username is set to your savannah username):

1. Check out a fresh source tree from SVN trunk.
2. Update the NEWS file with pending release number and commit to SVN trunk:
Change *Changes since avr-libc-<last_release>*: to *Changes in avr-libc-<this_release>*.
3. Set the branch-point tag (setting <major> and <minor> accordingly):

```
svn copy svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/trunk
svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/tags/avr-libc-<major>↵
_<minor>-branchpoint
```
4. Create the branch:

```
svn copy svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/trunk
svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/branches/avr-libc-<major>↵
_<minor>-branch
```
5. Update the package version in configure.ac and commit configure.ac to SVN trunk:
Change minor number to next odd value.
6. Update the NEWS file and commit to SVN trunk:
Add *Changes since avr-libc-<this_release>*:
7. Check out a new tree for the branch:

```
svn co svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/branches/avr-libc-<major>↵
_<minor>-branch
```
8. Update the package version in configure.ac and commit configure.ac to SVN branch:
Change the patch number to 90 to denote that this now a branch leading up to a release. Be sure to leave the <date> part of the version.
9. Bring the build system up to date by running bootstrap and configure.
10. Perform a 'make distcheck' and make sure it succeeds. This will create the snapshot source tarball. This should be considered the first release candidate.
11. Upload the snapshot tarball to savannah.
12. Update the bug tracker interface on Savannah: Bugs —> Edit field values —> Release / Fixed Release
13. Announce the branch and the branch tag to the avr-libc-dev list so other developers can checkout the branch.

16.2.2 Making a release

A stable release will only be done on a branch, not from the SVN trunk.

The following steps should be taken when making a release:

1. Make sure the source tree you are working from is on the correct branch:

```
svn switch svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/branches/avr-libc-<major>↵
_<minor>-branch
```
2. Update the package version in configure.ac and commit it to SVN.
3. Update the gnu tool chain version requirements in the README and commit to SVN.

4. Update the ChangeLog file to note the release and commit to SVN on the branch:
Add *Released avr-libc-<this_release>*.
5. Update the NEWS file with pending release number and commit to SVN:
Change *Changes since avr-libc-<last_release>*: to *Changes in avr-libc-<this_release>*..
6. Bring the build system up to date by running bootstrap and configure.
7. Perform a 'make distcheck' and make sure it succeeds. This will create the source tarball.
8. Tag the release:

```
svn copy . svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/tags/avr-libc-<major>_<minor>_<patch>-release
```

or

```
svn copy svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/branches/avr-libc-<major>_<minor>-branch svn+ssh://$username@svn.savannah.nongnu.org/avr-libc/tags/avr-libc-<major>_<minor>_<patch>-release
```
9. Upload the tarball to savannah.
10. Update the NEWS file, and commit to SVN:
Add *Changes since avr-libc-<major>_<minor>_<patch>*:
11. Update the bug tracker interface on Savannah: Bugs —> Edit field values —> Release / Fixed Release
12. Generate the latest documentation and upload to savannah.
13. Announce the release.

The following hypothetical diagram should help clarify version and branch relationships.

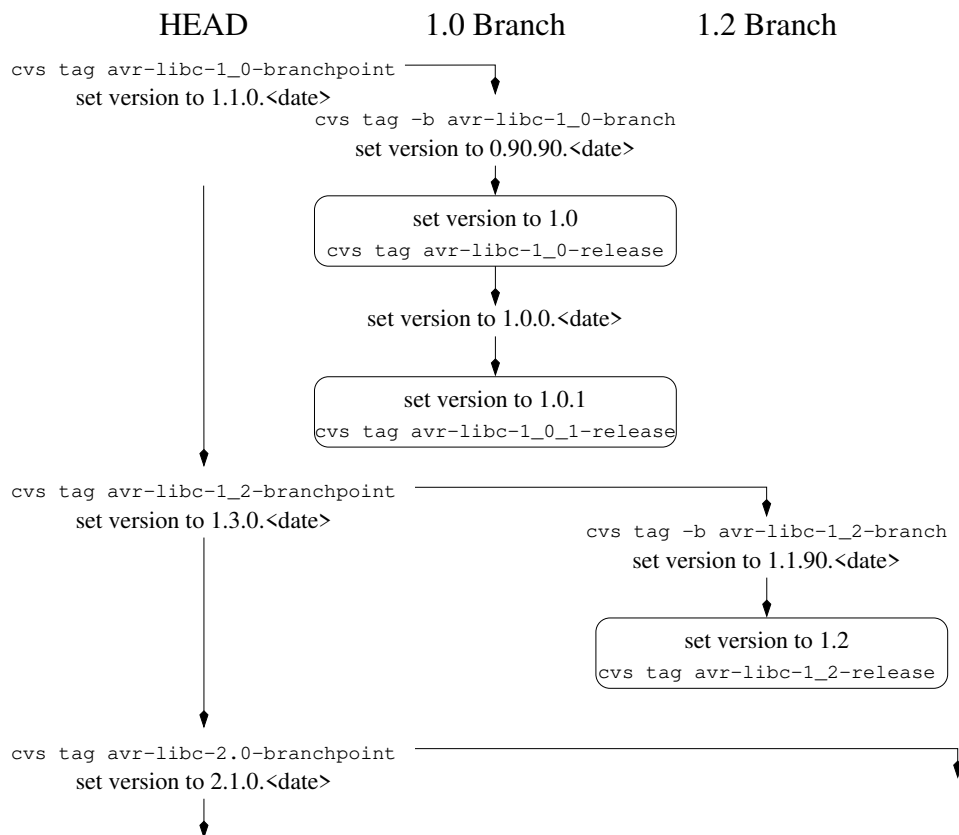


Figure 4 Release tree

17 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] for making the AVR-specific changes to the GNU tools.
- Denis Chertykov and Marek Michalkiewicz [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] 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] 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] 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. ;-)

18 Todo List

Module [avr_boot](#)

From email with Marek: On smaller devices (all except ATmega64/128), `__SPM_REG` is in the I/O space, accessible with the shorter "in" and "out" instructions - since the boot loader has a limited size, this could be an important optimization.

19 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)

20 Module Index

20.1 Modules

Here is a list of all modules:

<alloca.h>: Allocate space in the stack	106
<assert.h>: Diagnostics	106
<ctype.h>: Character Operations	107
<errno.h>: System Errors	109
<inttypes.h>: Integer Type conversions	110
<math.h>: Mathematics	123
<setjmp.h>: Non-local goto	124
<stdint.h>: Standard Integer Types	126
<stdio.h>: Standard IO facilities	137
<stdlib.h>: General utilities	140
<string.h>: Strings	145
<time.h>: Time	158
<avr/boot.h>: Bootloader Support Utilities	160
<avr/cpufunc.h>: Special AVR CPU functions	165
<avr/eeprom.h>: EEPROM handling	166
<avr/fuse.h>: Fuse Support	170
<avr/interrupt.h>: Interrupts	173

<code><avr/io.h></code> : AVR device-specific IO definitions	194
<code><avr/lock.h></code> : Lockbit Support	195
<code><avr/pgmspace.h></code> : Program Space Utilities	197
<code><avr/power.h></code> : Power Reduction Management	221
<code><avr/sfr_defs.h></code> : Special function registers	226
Additional notes from <code><avr/sfr_defs.h></code>	225
<code><avr/signature.h></code> : Signature Support	228
<code><avr/sleep.h></code> : Power Management and Sleep Modes	229
<code><avr/version.h></code> : avr-libc version macros	230
<code><avr/wdt.h></code> : Watchdog timer handling	232
<code><util/atomic.h></code> Atomically and Non-Atomically Executed Code Blocks	235
<code><util/crc16.h></code> : CRC Computations	237
<code><util/delay.h></code> : Convenience functions for busy-wait delay loops	241
<code><util/delay_basic.h></code> : Basic busy-wait delay loops	243
<code><util/parity.h></code> : Parity bit generation	243
<code><util/setbaud.h></code> : Helper macros for baud rate calculations	244
<code><util/twi.h></code> : TWI bit mask definitions	246
<code><compat/deprecated.h></code> : Deprecated items	250
<code><compat/ina90.h></code> : Compatibility with IAR EWB 3.x	253
Demo projects	253
Combining C and assembly source files	254
A simple project	256
A more sophisticated project	267
Using the standard IO facilities	272
Example using the two-wire interface (TWI)	277

21 Data Structure Index

21.1 Data Structures

Here are the data structures with brief descriptions:

<code>div_t</code>	281
<code>ldiv_t</code>	282

tm	283
week_date	284

22 File Index

22.1 File List

Here is a list of all documented files with brief descriptions:

project.h	285
iocompat.h	285
defines.h	287
hd44780.h	288
lcd.h	289
uart.h	289
alloca.h	290
assert.h	291
boot.h	293
cpufunc.h	301
eeprom.h	302
fuse.h	305
interrupt.h	309
io.h	314
lock.h	321
pgmspace.h	324
portpins.h	345
power.h	352
sfr_defs.h	374
signal.h	378
signature.h	378
sleep.h	379
version.h	384
wdt.h	385
xmega.h	392

deprecated.h	393
ina90.h	396
ctype.h	397
errno.h	400
inttypes.h	402
math.h	410
setjmp.h	429
stdint.h	431
stdio.h	442
stdlib.h	467
string.h	482
time.h	490
atomic.h	503
crc16.h	507
delay.h	512
delay_basic.h	516
eu_dst.h	518
parity.h	519
setbaud.h	520
compat/twi.h	523
util/twi.h	524
usa_dst.h	527
eedef.h	529
fdevopen.c	530
stdio_private.h	530
xtoa_fast.h	531
dtoa_conv.h	531
stdlib_private.h	532
ephemera_common.h	533

23 Module Documentation

23.1 <alloca.h>: Allocate space in the stack

Functions

- void * [alloca](#) (size_t __size)

23.1.1 Detailed Description

23.1.2 Function Documentation

23.1.2.1 [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.

23.2 <assert.h>: Diagnostics

```
#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.

23.3 <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 these functions 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)

23.3.1 Detailed Description

These functions perform various operations on characters.

```
#include <ctype.h>
```

23.3.2 Function Documentation

23.3.2.1 `isalnum()` int `isalnum` (
int __c)

Checks for an alphanumeric character. It is equivalent to `(isalpha(c) || isdigit(c))`.

23.3.2.2 `isalpha()` int `isalpha` (
int __c)

Checks for an alphabetic character. It is equivalent to `(isupper(c) || islower(c))`.

23.3.2.3 isascii() `int isascii (`
 `int __c)`

Checks whether `c` is a 7-bit unsigned char value that fits into the ASCII character set.

23.3.2.4 isblank() `int isblank (`
 `int __c)`

Checks for a blank character, that is, a space or a tab.

23.3.2.5 iscntrl() `int iscntrl (`
 `int __c)`

Checks for a control character.

23.3.2.6 isdigit() `int isdigit (`
 `int __c)`

Checks for a digit (0 through 9).

23.3.2.7 isgraph() `int isgraph (`
 `int __c)`

Checks for any printable character except space.

23.3.2.8 islower() `int islower (`
 `int __c)`

Checks for a lower-case character.

23.3.2.9 isprint() `int isprint (`
 `int __c)`

Checks for any printable character including space.

23.3.2.10 ispunct() `int ispunct (`
 `int __c)`

Checks for any printable character which is not a space or an alphanumeric character.

23.3.2.11 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'`).

23.3.2.12 isupper() `int isupper (`
`int __c)`

Checks for an uppercase letter.

23.3.2.13 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.

23.3.2.14 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.

23.3.2.15 tolower() `int tolower (`
`int __c)`

Converts the letter `c` to lower case, if possible.

23.3.2.16 toupper() `int toupper (`
`int __c)`

Converts the letter `c` to upper case, if possible.

23.4 <errno.h>: System Errors

Macros

- `#define` `EDOM` 33
- `#define` `ERANGE` 34

Variables

- `int` `errno`

23.4.1 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.

23.4.2 Macro Definition Documentation

23.4.2.1 EDOM `#define EDOM 33`

Domain error.

23.4.2.2 ERANGE `#define ERANGE 34`

Range error.

23.4.3 Variable Documentation

23.4.3.1 `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.

23.5 `<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 PRIu8 "u"`
- `#define PRIdLEAST8 "d"`
- `#define PRIuLEAST8 "u"`
- `#define PRIdFAST8 "d"`
- `#define PRIuFAST8 "u"`
- `#define PRId16 "d"`
- `#define PRIu16 "u"`
- `#define PRIdLEAST16 "d"`
- `#define PRIuLEAST16 "u"`
- `#define PRIdFAST16 "d"`
- `#define PRIuFAST16 "u"`
- `#define PRId32 "d"`
- `#define PRIu32 "u"`
- `#define PRIdLEAST32 "d"`
- `#define PRIuLEAST32 "u"`
- `#define PRIdFAST32 "d"`
- `#define PRIuFAST32 "u"`
- `#define PRIdPTR PRId16`
- `#define PRIuPTR PRIu16`
- `#define PRId8 "d"`
- `#define PRIu8 "u"`
- `#define PRIdLEAST8 "d"`
- `#define PRIuLEAST8 "u"`
- `#define PRIdFAST8 "d"`
- `#define PRIuFAST8 "u"`
- `#define PRId16 "d"`
- `#define PRIu16 "u"`
- `#define PRIdLEAST16 "d"`
- `#define PRIuLEAST16 "u"`
- `#define PRIdFAST16 "d"`
- `#define PRIuFAST16 "u"`
- `#define PRId32 "d"`
- `#define PRIu32 "u"`
- `#define PRIdLEAST32 "d"`
- `#define PRIuLEAST32 "u"`
- `#define PRIdFAST32 "d"`
- `#define PRIuFAST32 "u"`
- `#define PRIdPTR PRId16`
- `#define PRIuPTR PRIu16`
- `#define PRId8 "d"`
- `#define PRIu8 "u"`
- `#define PRIdLEAST8 "d"`
- `#define PRIuLEAST8 "u"`
- `#define PRIdFAST8 "d"`
- `#define PRIuFAST8 "u"`
- `#define PRId16 "d"`
- `#define PRIu16 "u"`
- `#define PRIdLEAST16 "d"`
- `#define PRIuLEAST16 "u"`
- `#define PRIdFAST16 "d"`
- `#define PRIuFAST16 "u"`
- `#define PRId32 "d"`
- `#define PRIu32 "u"`
- `#define PRIdLEAST32 "d"`
- `#define PRIuLEAST32 "u"`
- `#define PRIdFAST32 "d"`
- `#define PRIuFAST32 "u"`
- `#define PRIdPTR PRId16`
- `#define PRIuPTR PRIu16`

- #define `PRIx32` "lx"
- #define `PRIxLEAST32` "lx"
- #define `PRIxFAST32` "lx"
- #define `PRIX32` "IX"
- #define `PRIXLEAST32` "IX"
- #define `PRIXFAST32` "IX"
- #define `PRIoPTR` `PRIo16`
- #define `PRIoPTR` `PRIo16`
- #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`

23.5.1 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);
```

23.5.2 Macro Definition Documentation

23.5.2.1 PRId16 `#define PRId16 "d"`

decimal printf format for `int16_t`

23.5.2.2 PRId32 `#define PRId32 "ld"`

decimal printf format for `int32_t`

23.5.2.3 PRId8 `#define PRId8 "d"`

decimal printf format for `int8_t`

23.5.2.4 PRIdFAST16 `#define PRIdFAST16 "d"`

decimal printf format for `int_fast16_t`

23.5.2.5 PRIdFAST32 `#define PRIdFAST32 "ld"`

decimal printf format for `int_fast32_t`

23.5.2.6 PRIdFAST8 `#define PRIdFAST8 "d"`

decimal printf format for `int_fast8_t`

23.5.2.7 PRIdLEAST16 `#define PRIdLEAST16 "d"`

decimal printf format for `int_least16_t`

23.5.2.8 PRIdLEAST32 `#define PRIdLEAST32 "ld"`

decimal printf format for `int_least32_t`

23.5.2.9 PRIdLEAST8 `#define PRIdLEAST8 "d"`

decimal printf format for `int_least8_t`

23.5.2.10 PRIdPTR `#define PRIdPTR PRId16`

decimal printf format for `intptr_t`

23.5.2.11 PRIi16 `#define PRIi16 "i"`

integer printf format for `int16_t`

23.5.2.12 PRIi32 `#define PRIi32 "li"`

integer printf format for `int32_t`

23.5.2.13 PRIi8 `#define PRIi8 "i"`

integer printf format for `int8_t`

23.5.2.14 PRIiFAST16 `#define PRIiFAST16 "i"`

integer printf format for `int_fast16_t`

23.5.2.15 PRIiFAST32 `#define PRIiFAST32 "li"`

integer printf format for `int_fast32_t`

23.5.2.16 PRIiFAST8 `#define PRIiFAST8 "i"`

integer printf format for `int_fast8_t`

23.5.2.17 PRIiLEAST16 `#define PRIiLEAST16 "i"`

integer printf format for `int_least16_t`

23.5.2.18 PRIiLEAST32 `#define PRIiLEAST32 "li"`

integer printf format for `int_least32_t`

23.5.2.19 PRIiLEAST8 `#define PRIiLEAST8 "i"`

integer printf format for `int_least8_t`

23.5.2.20 PRIiPTR `#define PRIiPTR PRIi16`

integer printf format for `intptr_t`

23.5.2.21 PRIo16 `#define PRIo16 "o"`

octal printf format for `uint16_t`

23.5.2.22 PRIo32 `#define PRIo32 "lo"`

octal printf format for `uint32_t`

23.5.2.23 PRIo8 `#define PRIo8 "o"`

octal printf format for `uint8_t`

23.5.2.24 PRIoFAST16 `#define PRIoFAST16 "o"`

octal printf format for `uint_fast16_t`

23.5.2.25 PRIoFAST32 `#define PRIoFAST32 "lo"`

octal printf format for `uint_fast32_t`

23.5.2.26 PRIoFAST8 `#define PRIoFAST8 "o"`

octal printf format for `uint_fast8_t`

23.5.2.27 PRIoLEAST16 `#define PRIoLEAST16 "o"`

octal printf format for `uint_least16_t`

23.5.2.28 PRIoLEAST32 `#define PRIoLEAST32 "lo"`

octal printf format for `uint_least32_t`

23.5.2.29 PRIoLEAST8 `#define PRIoLEAST8 "o"`

octal printf format for `uint_least8_t`

23.5.2.30 PRIoPTR `#define PRIoPTR PRIo16`

octal printf format for `uintptr_t`

23.5.2.31 PRIu16 `#define PRIu16 "u"`

decimal printf format for `uint16_t`

23.5.2.32 PRIu32 `#define PRIu32 "lu"`

decimal printf format for `uint32_t`

23.5.2.33 PRIu8 `#define PRIu8 "u"`

decimal printf format for `uint8_t`

23.5.2.34 PRIuFAST16 `#define PRIuFAST16 "u"`

decimal printf format for `uint_fast16_t`

23.5.2.35 PRIuFAST32 `#define PRIuFAST32 "lu"`

decimal printf format for `uint_fast32_t`

23.5.2.36 PRIuFAST8 `#define PRIuFAST8 "u"`

decimal printf format for `uint_fast8_t`

23.5.2.37 PRIuLEAST16 `#define PRIuLEAST16 "u"`

decimal printf format for `uint_least16_t`

23.5.2.38 PRIuLEAST32 `#define PRIuLEAST32 "lu"`

decimal printf format for `uint_least32_t`

23.5.2.39 PRIuLEAST8 `#define PRIuLEAST8 "u"`

decimal printf format for `uint_least8_t`

23.5.2.40 PRIuPTR `#define PRIuPTR PRIu16`

decimal printf format for `uintptr_t`

23.5.2.41 PRIx16 `#define PRIx16 "x"`

hexadecimal printf format for `uint16_t`

23.5.2.42 PRIX16 `#define PRIX16 "X"`

uppercase hexadecimal printf format for `uint16_t`

23.5.2.43 PRIx32 `#define PRIx32 "lx"`

hexadecimal printf format for `uint32_t`

23.5.2.44 PRIX32 `#define PRIX32 "lX"`

uppercase hexadecimal printf format for `uint32_t`

23.5.2.45 PRIx8 `#define PRIx8 "x"`

hexadecimal printf format for `uint8_t`

23.5.2.46 PRIX8 `#define PRIX8 "X"`

uppercase hexadecimal printf format for `uint8_t`

23.5.2.47 PRIxFAST16 `#define PRIxFAST16 "x"`

hexadecimal printf format for `uint_fast16_t`

23.5.2.48 PRIXFAST16 `#define PRIXFAST16 "X"`

uppercase hexadecimal printf format for `uint_fast16_t`

23.5.2.49 PRIxFAST32 `#define PRIxFAST32 "lx"`

hexadecimal printf format for `uint_fast32_t`

23.5.2.50 PRIXFAST32 `#define PRIXFAST32 "lX"`

uppercase hexadecimal printf format for `uint_fast32_t`

23.5.2.51 PRIxFAST8 `#define PRIxFAST8 "x"`

hexadecimal printf format for `uint_fast8_t`

23.5.2.52 PRIXFAST8 `#define PRIXFAST8 "X"`

uppercase hexadecimal printf format for `uint_fast8_t`

23.5.2.53 PRIxLEAST16 `#define PRIxLEAST16 "x"`

hexadecimal printf format for `uint_least16_t`

23.5.2.54 PRIXLEAST16 `#define PRIXLEAST16 "X"`

uppercase hexadecimal printf format for `uint_least16_t`

23.5.2.55 PRIxLEAST32 `#define PRIxLEAST32 "lx"`

hexadecimal printf format for `uint_least32_t`

23.5.2.56 PRIXLEAST32 `#define PRIXLEAST32 "lX"`

uppercase hexadecimal printf format for `uint_least32_t`

23.5.2.57 PRIxLEAST8 `#define PRIxLEAST8 "x"`

hexadecimal printf format for `uint_least8_t`

23.5.2.58 PRIXLEAST8 `#define PRIXLEAST8 "X"`

uppercase hexadecimal printf format for `uint_least8_t`

23.5.2.59 PRIxPTR `#define PRIxPTR PRIx16`

hexadecimal printf format for `uintptr_t`

23.5.2.60 PRIXPTR `#define PRIXPTR PRIX16`

uppercase hexadecimal printf format for `uintptr_t`

23.5.2.61 SCNd16 `#define SCNd16 "d"`

decimal scanf format for `int16_t`

23.5.2.62 SCNd32 `#define SCNd32 "ld"`

decimal scanf format for `int32_t`

23.5.2.63 SCNd8 `#define SCNd8 "hhd"`

decimal scanf format for `int8_t`

23.5.2.64 SCNdFAST16 `#define SCNdFAST16 "d"`

decimal scanf format for `int_fast16_t`

23.5.2.65 SCNdFAST32 `#define SCNdFAST32 "ld"`

decimal scanf format for `int_fast32_t`

23.5.2.66 SCNdFAST8 `#define SCNdFAST8 "hhd"`

decimal scanf format for `int_fast8_t`

23.5.2.67 SCNdLEAST16 `#define SCNdLEAST16 "d"`

decimal scanf format for `int_least16_t`

23.5.2.68 SCNdLEAST32 `#define SCNdLEAST32 "ld"`

decimal scanf format for `int_least32_t`

23.5.2.69 SCNdLEAST8 `#define SCNdLEAST8 "hhd"`

decimal scanf format for `int_least8_t`

23.5.2.70 SCNdPTR `#define SCNdPTR SCNd16`

decimal scanf format for `intptr_t`

23.5.2.71 SCNi16 `#define SCNi16 "i"`

generic-integer scanf format for `int16_t`

23.5.2.72 SCNi32 `#define SCNi32 "li"`

generic-integer scanf format for `int32_t`

23.5.2.73 SCNi8 `#define SCNi8 "hhi"`

generic-integer scanf format for `int8_t`

23.5.2.74 SCNiFAST16 `#define SCNiFAST16 "i"`

generic-integer scanf format for `int_fast16_t`

23.5.2.75 SCNiFAST32 `#define SCNiFAST32 "li"`

generic-integer scanf format for `int_fast32_t`

23.5.2.76 SCNiFAST8 `#define SCNiFAST8 "hhi"`

generic-integer scanf format for `int_fast8_t`

23.5.2.77 SCNiLEAST16 `#define SCNiLEAST16 "i"`

generic-integer scanf format for `int_least16_t`

23.5.2.78 SCNiLEAST32 `#define SCNiLEAST32 "li"`

generic-integer scanf format for `int_least32_t`

23.5.2.79 SCNiLEAST8 `#define SCNiLEAST8 "hhi"`

generic-integer scanf format for `int_least8_t`

23.5.2.80 SCNiPTR `#define SCNiPTR SCNi16`

generic-integer scanf format for `intptr_t`

23.5.2.81 SCNo16 `#define SCNo16 "o"`

octal scanf format for `uint16_t`

23.5.2.82 SCNo32 `#define SCNo32 "lo"`

octal scanf format for `uint32_t`

23.5.2.83 SCNo8 `#define SCNo8 "hho"`

octal scanf format for `uint8_t`

23.5.2.84 SCNoFAST16 `#define SCNoFAST16 "o"`

octal scanf format for `uint_fast16_t`

23.5.2.85 SCNoFAST32 `#define SCNoFAST32 "lo"`

octal scanf format for `uint_fast32_t`

23.5.2.86 SCNoFAST8 `#define SCNoFAST8 "hho"`

octal scanf format for `uint_fast8_t`

23.5.2.87 SCNoLEAST16 `#define SCNoLEAST16 "o"`

octal scanf format for `uint_least16_t`

23.5.2.88 SCNoLEAST32 `#define SCNoLEAST32 "lo"`

octal scanf format for `uint_least32_t`

23.5.2.89 SCNoLEAST8 `#define SCNoLEAST8 "hho"`

octal scanf format for `uint_least8_t`

23.5.2.90 SCNoPTR `#define SCNoPTR SCNo16`

octal scanf format for `uintptr_t`

23.5.2.91 SCNu16 `#define SCNu16 "u"`

decimal scanf format for `uint16_t`

23.5.2.92 SCNu32 `#define SCNu32 "lu"`

decimal scanf format for `uint32_t`

23.5.2.93 SCNu8 `#define SCNu8 "hhu"`

decimal scanf format for `uint8_t`

23.5.2.94 SCNuFAST16 `#define SCNuFAST16 "u"`

decimal scanf format for `uint_fast16_t`

23.5.2.95 SCNuFAST32 `#define SCNuFAST32 "lu"`

decimal scanf format for `uint_fast32_t`

23.5.2.96 SCNuFAST8 `#define SCNuFAST8 "hhu"`

decimal scanf format for `uint_fast8_t`

23.5.2.97 SCNuLEAST16 `#define SCNuLEAST16 "u"`

decimal scanf format for `uint_least16_t`

23.5.2.98 SCNuLEAST32 `#define SCNuLEAST32 "lu"`

decimal scanf format for `uint_least32_t`

23.5.2.99 SCNuLEAST8 `#define SCNuLEAST8 "hhu"`

decimal scanf format for `uint_least8_t`

23.5.2.100 SCNuPTR `#define SCNuPTR SCNu16`

decimal scanf format for `uintptr_t`

23.5.2.101 SCNx16 `#define SCNx16 "x"`

hexadecimal scanf format for `uint16_t`

23.5.2.102 SCNx32 `#define SCNx32 "lx"`

hexadecimal scanf format for `uint32_t`

23.5.2.103 SCNx8 `#define SCNx8 "hxx"`

hexadecimal scanf format for `uint8_t`

23.5.2.104 SCNxFAST16 `#define SCNxFAST16 "x"`

hexadecimal scanf format for `uint_fast16_t`

23.5.2.105 SCNxFAST32 `#define SCNxFAST32 "lx"`

hexadecimal scanf format for `uint_fast32_t`

23.5.2.106 SCNxFAST8 `#define SCNxFAST8 "hhx"`

hexadecimal scanf format for `uint_fast8_t`

23.5.2.107 SCNxLEAST16 `#define SCNxLEAST16 "x"`

hexadecimal scanf format for `uint_least16_t`

23.5.2.108 SCNxLEAST32 `#define SCNxLEAST32 "lx"`

hexadecimal scanf format for `uint_least32_t`

23.5.2.109 SCNxLEAST8 `#define SCNxLEAST8 "hhx"`

hexadecimal scanf format for `uint_least8_t`

23.5.2.110 SCNxPTR `#define SCNxPTR SCNx16`

hexadecimal scanf format for `uintptr_t`

23.5.3 Typedef Documentation

23.5.3.1 int_farptr_t `typedef int32_t int_farptr_t`

signed integer type that can hold a pointer > 64 KB

23.5.3.2 uint_farptr_t `typedef uint32_t uint_farptr_t`

unsigned integer type that can hold a pointer > 64 KB

23.6 <math.h>: Mathematics

Macros

- `#define M_E 2.7182818284590452354`

23.6.1 Detailed Description

```
#include <math.h>
```

This header file declares basic mathematics constants and functions.

Notes:

- In order to access the functions declared herein, it is usually also required to additionally link against the library `libm.a`. See also the related [FAQ entry](#).
- 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.

23.6.2 Macro Definition Documentation

23.6.2.1 M_E `#define M_E 2.7182818284590452354`

The constant `e`.

23.7 <setjmp.h>: Non-local goto

Functions

- int [setjmp](#) (jmp_buf __jmpb)
- void [longjmp](#) (jmp_buf __jmpb, int __ret) __ATTR_NORETURN__

23.7.1 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 sub-routine 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() some where ...
    }
}
...
void foo (void)
{
    ... blah, blah, blah ...
    if (err)
    {
        longjmp (env, 1);
    }
}
```

23.7.2 Function Documentation

23.7.2.1 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.

23.7.2.2 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.

23.8 <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)`

23.8.1 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.

23.8.2 Macro Definition Documentation

23.8.2.1 INT16_C `#define INT16_C(
value) value`

define a constant of type `int16_t`

23.8.2.2 INT16_MAX `#define INT16_MAX 0x7fff`

largest positive value an `int16_t` can hold.

23.8.2.3 INT16_MIN `#define INT16_MIN (-INT16_MAX - 1)`

smallest negative value an `int16_t` can hold.

23.8.2.4 INT32_C `#define INT32_C(
value) __CONCAT(value, L)`

define a constant of type `int32_t`

23.8.2.5 INT32_MAX `#define INT32_MAX 0x7fffffffL`

largest positive value an `int32_t` can hold.

23.8.2.6 INT32_MIN `#define INT32_MIN (-INT32_MAX - 1L)`

smallest negative value an `int32_t` can hold.

23.8.2.7 INT64_C `#define INT64_C(
value) __CONCAT(value, LL)`

define a constant of type `int64_t`

23.8.2.8 INT64_MAX `#define INT64_MAX 0xffffffffffffffffLL`

largest positive value an `int64_t` can hold.

23.8.2.9 INT64_MIN `#define INT64_MIN (-INT64_MAX - 1LL)`

smallest negative value an `int64_t` can hold.

23.8.2.10 INT8_C `#define INT8_C(
value) ((int8_t) value)`

define a constant of type `int8_t`

23.8.2.11 INT8_MAX `#define INT8_MAX 0x7f`

largest positive value an `int8_t` can hold.

23.8.2.12 INT8_MIN `#define INT8_MIN (-INT8_MAX - 1)`

smallest negative value an `int8_t` can hold.

23.8.2.13 INT_FAST16_MAX `#define INT_FAST16_MAX INT16_MAX`

largest positive value an `int_fast16_t` can hold.

23.8.2.14 INT_FAST16_MIN `#define INT_FAST16_MIN INT16_MIN`

smallest negative value an `int_fast16_t` can hold.

23.8.2.15 INT_FAST32_MAX `#define INT_FAST32_MAX INT32_MAX`

largest positive value an `int_fast32_t` can hold.

23.8.2.16 INT_FAST32_MIN `#define INT_FAST32_MIN INT32_MIN`

smallest negative value an `int_fast32_t` can hold.

23.8.2.17 INT_FAST64_MAX `#define INT_FAST64_MAX INT64_MAX`

largest positive value an `int_fast64_t` can hold.

23.8.2.18 INT_FAST64_MIN `#define INT_FAST64_MIN INT64_MIN`

smallest negative value an `int_fast64_t` can hold.

23.8.2.19 INT_FAST8_MAX `#define INT_FAST8_MAX INT8_MAX`

largest positive value an `int_fast8_t` can hold.

23.8.2.20 INT_FAST8_MIN `#define INT_FAST8_MIN INT8_MIN`

smallest negative value an `int_fast8_t` can hold.

23.8.2.21 INT_LEAST16_MAX `#define INT_LEAST16_MAX INT16_MAX`

largest positive value an `int_least16_t` can hold.

23.8.2.22 INT_LEAST16_MIN `#define INT_LEAST16_MIN INT16_MIN`

smallest negative value an `int_least16_t` can hold.

23.8.2.23 INT_LEAST32_MAX `#define INT_LEAST32_MAX INT32_MAX`

largest positive value an `int_least32_t` can hold.

23.8.2.24 INT_LEAST32_MIN `#define INT_LEAST32_MIN INT32_MIN`

smallest negative value an `int_least32_t` can hold.

23.8.2.25 INT_LEAST64_MAX `#define INT_LEAST64_MAX INT64_MAX`

largest positive value an `int_least64_t` can hold.

23.8.2.26 INT_LEAST64_MIN `#define INT_LEAST64_MIN INT64_MIN`

smallest negative value an `int_least64_t` can hold.

23.8.2.27 INT_LEAST8_MAX `#define INT_LEAST8_MAX INT8_MAX`

largest positive value an `int_least8_t` can hold.

23.8.2.28 INT_LEAST8_MIN `#define INT_LEAST8_MIN INT8_MIN`

smallest negative value an `int_least8_t` can hold.

23.8.2.29 INTMAX_C `#define INTMAX_C(
value) __CONCAT(value, LL)`

define a constant of type `intmax_t`

23.8.2.30 INTMAX_MAX `#define INTMAX_MAX INT64_MAX`

largest positive value an `intmax_t` can hold.

23.8.2.31 INTMAX_MIN `#define INTMAX_MIN INT64_MIN`

smallest negative value an `intmax_t` can hold.

23.8.2.32 INTPTR_MAX `#define INTPTR_MAX INT16_MAX`

largest positive value an `intptr_t` can hold.

23.8.2.33 INTPTR_MIN `#define INTPTR_MIN INT16_MIN`

smallest negative value an `intptr_t` can hold.

23.8.2.34 PTRDIFF_MAX `#define PTRDIFF_MAX INT16_MAX`

largest positive value a `ptrdiff_t` can hold.

23.8.2.35 PTRDIFF_MIN `#define PTRDIFF_MIN INT16_MIN`

smallest negative value a `ptrdiff_t` can hold.

23.8.2.36 SIG_ATOMIC_MAX `#define SIG_ATOMIC_MAX INT8_MAX`

largest positive value a `sig_atomic_t` can hold.

23.8.2.37 SIG_ATOMIC_MIN `#define SIG_ATOMIC_MIN INT8_MIN`

smallest negative value a `sig_atomic_t` can hold.

23.8.2.38 SIZE_MAX `#define SIZE_MAX UINT16_MAX`

largest value a `size_t` can hold.

23.8.2.39 UINT16_C `#define UINT16_C(
value) __CONCAT(value, U)`

define a constant of type `uint16_t`

23.8.2.40 UINT16_MAX `#define UINT16_MAX (__CONCAT(INT16_MAX, U) * 2U + 1U)`

largest value an `uint16_t` can hold.

23.8.2.41 UINT32_C `#define UINT32_C(
value) __CONCAT(value, UL)`

define a constant of type `uint32_t`

23.8.2.42 UINT32_MAX `#define UINT32_MAX (__CONCAT(INT32_MAX, U) * 2UL + 1UL)`

largest value an `uint32_t` can hold.

23.8.2.43 UINT64_C `#define UINT64_C(
value) __CONCAT(value, ULL)`

define a constant of type `uint64_t`

23.8.2.44 **UINT64_MAX** `#define UINT64_MAX (__CONCAT(INT64_MAX, U) * 2ULL + 1ULL)`

largest value an uint64_t can hold.

23.8.2.45 **UINT8_C** `#define UINT8_C(
value) ((uint8_t) __CONCAT(value, U))`

define a constant of type uint8_t

23.8.2.46 **UINT8_MAX** `#define UINT8_MAX (INT8_MAX * 2 + 1)`

largest value an uint8_t can hold.

23.8.2.47 **UINT_FAST16_MAX** `#define UINT_FAST16_MAX UINT16_MAX`

largest value an uint_fast16_t can hold.

23.8.2.48 **UINT_FAST32_MAX** `#define UINT_FAST32_MAX UINT32_MAX`

largest value an uint_fast32_t can hold.

23.8.2.49 **UINT_FAST64_MAX** `#define UINT_FAST64_MAX UINT64_MAX`

largest value an uint_fast64_t can hold.

23.8.2.50 **UINT_FAST8_MAX** `#define UINT_FAST8_MAX UINT8_MAX`

largest value an uint_fast8_t can hold.

23.8.2.51 **UINT_LEAST16_MAX** `#define UINT_LEAST16_MAX UINT16_MAX`

largest value an uint_least16_t can hold.

23.8.2.52 **UINT_LEAST32_MAX** `#define UINT_LEAST32_MAX UINT32_MAX`

largest value an uint_least32_t can hold.

23.8.2.53 **UINT_LEAST64_MAX** `#define UINT_LEAST64_MAX UINT64_MAX`

largest value an uint_least64_t can hold.

23.8.2.54 **UINT_LEAST8_MAX** `#define UINT_LEAST8_MAX UINT8_MAX`

largest value an uint_least8_t can hold.

23.8.2.55 UINTMAX_C `#define UINTMAX_C(
value) __CONCAT(value, ULL)`

define a constant of type `uintmax_t`

23.8.2.56 UINTMAX_MAX `#define UINTMAX_MAX UINT64_MAX`

largest value an `uintmax_t` can hold.

23.8.2.57 UINTPTR_MAX `#define UINTPTR_MAX UINT16_MAX`

largest value an `uintptr_t` can hold.

23.8.3 Typedef Documentation

23.8.3.1 int16_t `typedef signed int int16_t`

16-bit signed type.

23.8.3.2 int32_t `typedef signed long int int32_t`

32-bit signed type.

23.8.3.3 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.

23.8.3.4 int8_t `typedef signed char int8_t`

8-bit signed type.

23.8.3.5 int_fast16_t `typedef int16_t int_fast16_t`

fastest signed int with at least 16 bits.

23.8.3.6 int_fast32_t `typedef int32_t int_fast32_t`

fastest signed int with at least 32 bits.

23.8.3.7 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.

23.8.3.8 int_fast8_t typedef int8_t int_fast8_t

fastest signed int with at least 8 bits.

23.8.3.9 int_least16_t typedef int16_t int_least16_t

signed int with at least 16 bits.

23.8.3.10 int_least32_t typedef int32_t int_least32_t

signed int with at least 32 bits.

23.8.3.11 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.

23.8.3.12 int_least8_t typedef int8_t int_least8_t

signed int with at least 8 bits.

23.8.3.13 intmax_t typedef int64_t intmax_t

largest signed int available.

23.8.3.14 intptr_t typedef int16_t intptr_t

Signed pointer compatible type.

23.8.3.15 uint16_t typedef unsigned int uint16_t

16-bit unsigned type.

23.8.3.16 uint32_t typedef unsigned long int uint32_t

32-bit unsigned type.

23.8.3.17 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.

23.8.3.18 uint8_t typedef unsigned char uint8_t

8-bit unsigned type.

23.8.3.19 uint_fast16_t typedef uint16_t uint_fast16_t

fastest unsigned int with at least 16 bits.

23.8.3.20 uint_fast32_t typedef uint32_t uint_fast32_t

fastest unsigned int with at least 32 bits.

23.8.3.21 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.

23.8.3.22 uint_fast8_t typedef uint8_t uint_fast8_t

fastest unsigned int with at least 8 bits.

23.8.3.23 uint_least16_t typedef uint16_t uint_least16_t

unsigned int with at least 16 bits.

23.8.3.24 uint_least32_t typedef uint32_t uint_least32_t

unsigned int with at least 32 bits.

23.8.3.25 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.

23.8.3.26 uint_least8_t typedef uint8_t uint_least8_t

unsigned int with at least 8 bits.

23.8.3.27 uintmax_t typedef uint64_t uintmax_t

largest unsigned int available.

23.8.3.28 uintptr_t typedef uint16_t uintptr_t

Unsigned pointer compatible type.

23.9 <stdio.h>: Standard IO facilities

Functions

- **FILE * fdevopen** (int(*put)(char, FILE *), int(*get)(FILE *))

23.9.1 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 `vprintf()` which is the heart of the `printf` family can be selected in different flavours using linker options. See the documentation of `vprintf()` 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.

23.9.2 Function Documentation

23.9.2.1 `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.

23.10 `<stdlib.h>`: General utilities

Functions

- double `atof` (const char * __nptr)

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.

Note that these functions are not located in the default library, `libc.a`, but in the mathematical library, `libm.a`. So when linking the application, the `-lm` option needs to be specified.

- char * `dtostr` (double __val, char * __s, unsigned char __prec, unsigned char __flags)
- char * `dtostrf` (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 than 'e' */
- #define `EXIT_SUCCESS` 0
- #define `EXIT_FAILURE` 1

23.10.1 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.

23.10.2 Macro Definition Documentation

23.10.2.1 DTOSTR_ALWAYS_SIGN `#define DTOSTR_ALWAYS_SIGN 0x01 /* put '+' or '-' for positives */`

Bit value that can be passed in flags to [dtostre\(\)](#).

23.10.2.2 DTOSTR_PLUS_SIGN `#define DTOSTR_PLUS_SIGN 0x02 /* put '+' rather than '-' */`

Bit value that can be passed in flags to [dtostre\(\)](#).

23.10.2.3 DTOSTR_UPPERCASE `#define DTOSTR_UPPERCASE 0x04 /* put 'E' rather 'e' */`

Bit value that can be passed in flags to [dtostre\(\)](#).

23.10.2.4 EXIT_FAILURE `#define EXIT_FAILURE 1`

Unsuccessful termination for [exit\(\)](#); evaluates to a non-zero value.

23.10.2.5 EXIT_SUCCESS `#define EXIT_SUCCESS 0`

Successful termination for [exit\(\)](#); evaluates to 0.

23.10.2.6 RANDOM_MAX `#define RANDOM_MAX 0x7FFFFFFF`

Highest number that can be generated by [random\(\)](#).

23.10.3 Function Documentation

23.10.3.1 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);`

```
23.10.3.2 dtostre() char * dtostre (
    double __val,
    char * __s,
    unsigned char __prec,
    unsigned char __flags )
```

The `dtostre()` function converts the double 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 `dtostre()` function returns the pointer to the converted string `s`.

```
23.10.3.3 dtostrf() char * dtostrf (
    double __val,
    signed char __width,
    unsigned char __prec,
    char * __s )
```

The `dtostrf()` function converts the double 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.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 `dtostrf()` function returns the pointer to the converted string `s`.

```
23.10.3.4 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`.

23.10.3.5 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 * sizeof (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`.

23.10.3.6 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.

23.10.3.7 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.

23.10.3.8 srandom() `void srandom (`
 `unsigned long __seed)`

Pseudo-random number generator seeding; see `random()`.

23.10.3.9 `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 * sizeof (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`.

23.10.3.10 `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 * sizeof (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`.

23.11 <string.h>: Strings

Macros

- `#define _FFS(x)`

Functions

- `int ffs (int __val)`
- `int ffsi (long __val)`
- `__extension__ int ffsll (long long __val)`
- `void * memccpy (void *, const void *, int, size_t)`
- `void * memchr (const void *, int, size_t) __ATTR_PURE__`
- `int memcmp (const void *, const void *, size_t) __ATTR_PURE__`
- `void * memcpy (void *, const void *, size_t)`
- `void * memmem (const void *, size_t, const void *, size_t) __ATTR_PURE__`
- `void * memmove (void *, const void *, size_t)`
- `void * memrchr (const void *, int, size_t) __ATTR_PURE__`
- `void * memset (void *, int, size_t)`
- `char * strcat (char *, const char *)`
- `char * strchr (const char *, int) __ATTR_PURE__`
- `char * strchrnul (const char *, int) __ATTR_PURE__`
- `int strcmp (const char *, const char *) __ATTR_PURE__`
- `char * strcpy (char *, const char *)`
- `int strcasecmp (const char *, const char *) __ATTR_PURE__`
- `char * strcasestr (const char *, const char *) __ATTR_PURE__`
- `size_t strcspn (const char *__s, const char *__reject) __ATTR_PURE__`
- `char * strdup (const char *s1)`
- `size_t strlcat (char *, const char *, size_t)`
- `size_t strlcpy (char *, const char *, size_t)`
- `size_t strlen (const char *) __ATTR_PURE__`
- `char * strlwr (char *)`
- `char * strncat (char *, const char *, size_t)`
- `int strncmp (const char *, const char *, size_t) __ATTR_PURE__`
- `char * strncpy (char *, const char *, size_t)`
- `int strncasecmp (const char *, const char *, size_t) __ATTR_PURE__`
- `size_t strnlen (const char *, size_t) __ATTR_PURE__`
- `char * strpbrk (const char *__s, const char *__accept) __ATTR_PURE__`
- `char * strrchr (const char *, int) __ATTR_PURE__`
- `char * strrev (char *)`
- `char * strsep (char **, const char *)`
- `size_t strspn (const char *__s, const char *__accept) __ATTR_PURE__`
- `char * strstr (const char *, const char *) __ATTR_PURE__`
- `char * strtok (char *, const char *)`
- `char * strtok_r (char *, const char *, char **)`
- `char * strupr (char *)`

23.11.1 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](#).

23.11.2 Macro Definition Documentation

23.11.2.1 `_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.

23.11.3 Function Documentation

23.11.3.1 `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.

23.11.3.2 `ffsl()` `int ffsl (` `long __val)`

Same as `ffs()`, for an argument of type `long`.

23.11.3.3 `ffsll()` `int ffsll (` `long long __val)`

Same as `ffs()`, for an argument of type `long long`.

23.11.3.4 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`.

23.11.3.5 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.

23.11.3.6 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.

23.11.3.7 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`.

23.11.3.8 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`.

23.11.3.9 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`.

23.11.3.10 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.

23.11.3.11 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`.

23.11.3.12 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`.

23.11.3.13 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`.

23.11.3.14 `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`.

23.11.3.15 `strchr()` `char * strchr (`
 `const char * src,`
 `int val)`

Locate character in string.

The `strchr()` function returns a pointer to the first 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 `strchr()` function returns a pointer to the matched character or `NULL` if the character is not found.

23.11.3.16 `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.

23.11.3.17 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`.

23.11.3.18 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.

23.11.3.19 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.

23.11.3.20 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!

23.11.3.21 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 `NULL` terminates (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.

23.11.3.22 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 NULL terminates (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.

23.11.3.23 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`.

23.11.3.24 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.

23.11.3.25 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`.

23.11.3.26 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 `n` characters of `src` are appended to `dest`.

Returns

The [strncat\(\)](#) function returns a pointer to the resulting string `dest`.

23.11.3.27 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) `n` 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 `n` bytes thereof) is found, respectively, to be less than, to match, or be greater than `s2`.

23.11.3.28 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 `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()` function returns a pointer to the destination string `dest`.

23.11.3.29 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`.

23.11.3.30 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`.

23.11.3.31 strchr() `char * strchr (`
 `const char * src,`
 `int val)`

Locate character in string.

The `strchr()` 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 `strchr()` function returns a pointer to the matched character or `NULL` if the character is not found.

23.11.3.32 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.

23.11.3.33 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`.

23.11.3.34 `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.

23.11.3.35 `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`.

23.11.3.36 `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()`.

23.11.3.37 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.

23.11.3.38 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.

23.12 <time.h>: Time

Typedefs

- typedef `uint32_t time_t`

23.12.1 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, `struct 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.

23.12.2 Typedef Documentation

23.12.2.1 `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.

23.13 `<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)`

23.13.1 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.

Todo From email with Marek: On smaller devices (all except ATmega64/128), `__SPM_REG` is in the I/O space, accessible with the shorter "in" and "out" instructions - since the boot loader has a limited size, this could be an important optimization.

API Usage Example

The following code shows typical usage of the boot API.

```
#include <inttypes.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
void boot_program_page (uint32_t page, uint8_t *buf)
{
    uint16_t i;
    uint8_t sreg;
    // Disable interrupts.
    sreg = SREG;
    cli();
    eeprom_busy_wait ();
    boot_page_erase (page);
    boot_spm_busy_wait (); // Wait until the memory is erased.
    for (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;
}
```

23.13.2 Macro Definition Documentation

23.13.2.1 boot_is_spm_interrupt #define boot_is_spm_interrupt() (__SPM_REG & (uint8_t)_BV(SPMIE))

Check if the SPM interrupt is enabled.

23.13.2.2 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.

23.13.2.3 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.

23.13.2.4 boot_lock_fuse_bits_get #define boot_lock_fuse_bits_get(
 address)

Value:

```
(__extension__({
    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.

23.13.2.5 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.

23.13.2.6 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.

23.13.2.7 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.

23.13.2.8 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.

23.13.2.9 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.

23.13.2.10 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.

23.13.2.11 boot_rww_busy #define boot_rww_busy() (__SPM_REG & (uint8_t)_BV(__COMMON_ASB))

Check if the RWW section is busy.

23.13.2.12 boot_rww_enable #define boot_rww_enable() __boot_rww_enable()

Enable the Read-While-Write memory section.

23.13.2.13 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.

23.13.2.14 boot_signature_byte_get #define boot_signature_byte_get(
 addr)

Value:

```
(__extension__({ \
    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_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.

23.13.2.15 boot_spm_busy #define boot_spm_busy() (__SPM_REG & (uint8_t)_BV(__SPM_ENABLE))

Check if the SPM instruction is busy.

23.13.2.16 boot_spm_busy_wait #define boot_spm_busy_wait() do{}while(boot_spm_busy())

Wait while the SPM instruction is busy.

23.13.2.17 boot_spm_interrupt_disable #define boot_spm_interrupt_disable() (__SPM_REG &= (uint8_t)~_BV(SPMIE))

Disable the SPM interrupt.

23.13.2.18 boot_spm_interrupt_enable `#define boot_spm_interrupt_enable() (__SPM_REG |= (uint8_t)_BV(SPMIE))`

Enable the SPM interrupt.

23.13.2.19 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.

23.13.2.20 GET_EXTENDED_FUSE_BITS `#define GET_EXTENDED_FUSE_BITS (0x0002)`

address to read the extended fuse bits, using `boot_lock_fuse_bits_get`

23.13.2.21 GET_HIGH_FUSE_BITS `#define GET_HIGH_FUSE_BITS (0x0003)`

address to read the high fuse bits, using `boot_lock_fuse_bits_get`

23.13.2.22 GET_LOCK_BITS `#define GET_LOCK_BITS (0x0001)`

address to read the lock bits, using `boot_lock_fuse_bits_get`

23.13.2.23 GET_LOW_FUSE_BITS `#define GET_LOW_FUSE_BITS (0x0000)`

address to read the low fuse bits, using `boot_lock_fuse_bits_get`

23.14 <avr/cpufunc.h>: Special AVR CPU functions

Macros

- `#define _NOP()`
- `#define _MemoryBarrier()`

Functions

- `void ccp_write_io (uint8_t *__ioaddr, uint8_t __value)`

23.14.1 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.

23.14.2 Macro Definition Documentation

23.14.2.1 `_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.

23.14.2.2 `_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.

23.14.3 Function Documentation

23.14.3.1 `ccp_write_io()` `void ccp_write_io (` `uint8_t * __ioaddr,` `uint8_t __value)`

Write `__value` to Configuration Change Protected (CCP) IO register at `__ioaddr`.

23.15 `<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) __ATTR_PURE__`
- `uint16_t eeprom_read_word (const uint16_t * __p) __ATTR_PURE__`
- `uint32_t eeprom_read_dword (const uint32_t * __p) __ATTR_PURE__`
- `float eeprom_read_float (const float * __p) __ATTR_PURE__`
- `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_float (float * __p, float __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_float (float * __p, float __value)`
- `void eeprom_update_block (const void * __src, void * __dst, size_t __n)`

IAR C compatibility defines

- `#define __EEPUT(addr, val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))`
- `#define __EEPUT(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))`

23.15.1 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 are not wait until `SELFPRGEN` in `SPMCSR` becomes zero. Do this manually, if your software 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.

23.15.2 Macro Definition Documentation

23.15.2.1 `__EEGET` `#define __EEGET(
var,
addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))`

Read a byte from EEPROM. Compatibility define for IAR C.

23.15.2.2 `__EEPUT` `#define __EEPUT(
addr,
val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))`

Write a byte to EEPROM. Compatibility define for IAR C.

23.15.2.3 _EEGET `#define _EEGET(
 var,
 addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))`

Read a byte from EEPROM. Compatibility define for IAR C.

23.15.2.4 _EEPUT `#define _EEPUT(
 addr,
 val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))`

Write a byte to EEPROM. Compatibility define for IAR C.

23.15.2.5 EEMEM `#define EEMEM __attribute__((section(".eeprom")))`

Attribute expression causing a variable to be allocated within the .eeprom section.

23.15.2.6 eeprom_busy_wait `#define eeprom_busy_wait() do {} while (!eeprom_is_ready())`

Loops until the eeprom is no longer busy.

Returns

Nothing.

23.15.2.7 eeprom_is_ready `#define eeprom_is_ready()`

Returns

1 if EEPROM is ready for a new read/write operation, 0 if not.

23.15.3 Function Documentation

23.15.3.1 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`.

23.15.3.2 eeprom_read_byte() `uint8_t eeprom_read_byte (
 const uint8_t * __p)`

Read one byte from EEPROM address `__p`.

23.15.3.3 eeprom_read_dword() `uint32_t eeprom_read_dword (`
`const uint32_t * __p)`

Read one 32-bit double word (little endian) from EEPROM address `__p`.

23.15.3.4 eeprom_read_float() `float eeprom_read_float (`
`const float * __p)`

Read one float value (little endian) from EEPROM address `__p`.

23.15.3.5 eeprom_read_word() `uint16_t eeprom_read_word (`
`const uint16_t * __p)`

Read one 16-bit word (little endian) from EEPROM address `__p`.

23.15.3.6 eeprom_update_block() `void eeprom_update_block (`
`const void * __src,`
`void * __dst,`
`size_t __n)`

Update a block of `__n` bytes to EEPROM address `__dst` from `__src`.

Note

The argument order is mismatch with common functions like `strcpy()`.

23.15.3.7 eeprom_update_byte() `void eeprom_update_byte (`
`uint8_t * __p,`
`uint8_t __value)`

Update a byte `__value` to EEPROM address `__p`.

23.15.3.8 eeprom_update_dword() `void eeprom_update_dword (`
`uint32_t * __p,`
`uint32_t __value)`

Update a 32-bit double word `__value` to EEPROM address `__p`.

23.15.3.9 eeprom_update_float() `void eeprom_update_float (`
`float * __p,`
`float __value)`

Update a float `__value` to EEPROM address `__p`.

23.15.3.10 eeprom_update_word() `void eeprom_update_word (`
`uint16_t * __p,`
`uint16_t __value)`

Update a word `__value` to EEPROM address `__p`.

23.15.3.11 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\(\)](#).

23.15.3.12 eeprom_write_byte() `void eeprom_write_byte (`
 `uint8_t * __p,`
 `uint8_t __value)`

Write a byte `__value` to EEPROM address `__p`.

23.15.3.13 eeprom_write_dword() `void eeprom_write_dword (`
 `uint32_t * __p,`
 `uint32_t __value)`

Write a 32-bit double word `__value` to EEPROM address `__p`.

23.15.3.14 eeprom_write_float() `void eeprom_write_float (`
 `float * __p,`
 `float __value)`

Write a float `__value` to EEPROM address `__p`.

23.15.3.15 eeprom_write_word() `void eeprom_write_word (`
 `uint16_t * __p,`
 `uint16_t __value)`

Write a word `__value` to EEPROM address `__p`.

23.16 <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.

23.17 <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()`
- `#define cli()`

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()`
- `#define BADISR_vect`

ISR attributes

- `#define ISR_BLOCK`
- `#define ISR_NOBLOCK`
- `#define ISR_NAKED`
- `#define ISR_FLATTEN`
- `#define ISR_NOICF`
- `#define ISR_ALIASOF(target_vector)`

23.17.1 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. The extra code needed to do this is enabled by tagging the interrupt function with `__attribute__((signal))`.

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 assembler language.

Catch-all interrupt vector If an unexpected interrupt occurs (interrupt is enabled and 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` before entering an interrupt vector. 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 interrupt handler) 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 an `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).

```
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 symbols in following table.

There are currently two different styles present for naming the vectors. One form 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.

Starting with `avr-libc` version 1.4.0, a second style of interrupt vector names has been added, where a short phrase for the vector description is followed by `_vect`. The short phrase matches the vector name as described in the datasheet of the respective device (and in Atmel's XML 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.

The historical naming style might become deprecated in a future release, so it is not recommended for new projects.

Note

The `ISR()` macro cannot really spell-check the argument passed to them. Thus, by misspelling one of the names below in a call to `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 a `ISR()` function (i.e. one that after macro replacement does not start with `"__vector_"`).

Vector name	Old vector name	Description	Applicable for device
-------------	-----------------	-------------	-----------------------

ADC_vect	SIG_ADC	ADC Conversion Complete	AT90S2333, AT90S4433, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega163, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny13, ATtiny15, ATtiny26, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
ANALOG_COMP_0_vect	SIG_COMPARATOR0	Analog Comparator 0	AT90PWM3, AT90PWM1, AT90PWM2
ANALOG_COMP_1_vect	SIG_COMPARATOR1	Analog Comparator 1	AT90PWM3, AT90PWM1, AT90PWM2
ANALOG_COMP_2_vect	SIG_COMPARATOR2	Analog Comparator 2	AT90PWM3, AT90PWM1, AT90PWM2
ANALOG_COMP_vect	SIG_COMPARATOR	Analog Comparator	AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646

ANA_COMP_vect	SIG_COMPARATOR	Analog Comparator	AT90S1200, AT90S2313, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, ATmega16, ATmega161, ATmega162, ATmega163, ATmega32, ATmega323, ATmega8, ATmega8515, AT- mega8535, ATtiny11, ATtiny12, ATtiny13, ATtiny15, ATtiny2313, ATtiny26, ATtiny28, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861
CANIT_vect	SIG_CAN_INTERRUPT1	CAN Transfer Complete or Error	AT90CAN128, AT90CAN32, AT90CAN64
EEPROM_READY_vect	SIG_EEPROM_READY, SIG↔ _EE_READY		ATtiny2313
EE_RDY_vect	SIG_EEPROM_READY	EEPROM Ready	AT90S2333, AT90S4433, AT90S4434, AT90S8535, ATmega16, ATmega161, ATmega162, ATmega163, ATmega32, ATmega323, ATmega8, ATmega8515, AT- mega8535, ATtiny12, ATtiny13, ATtiny15, ATtiny26, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861
EE_READY_vect	SIG_EEPROM_READY	EEPROM Ready	AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32↔ HVB, ATmega406, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, AT- mega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
EXT_INT0_vect	SIG_INTERRUPT0	External Interrupt Request 0	ATtiny24, ATtiny44, ATtiny84

INT0_vect	SIG_INTERRUPT0	External Interrupt 0	AT90S1200, AT90S2313, AT90S2323, AT90S2333, AT90S2343, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32↔ HVB, ATmega406, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, AT- mega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, AT- mega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny11, ATtiny12, ATtiny13, ATtiny15, ATtiny22, ATtiny2313, ATtiny26, ATtiny28, ATtiny43U, ATtiny48, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
INT1_vect	SIG_INTERRUPT1	External Interrupt Request 1	AT90S2313, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega168P, ATmega32, ATmega323, ATmega328P, ATmega32HVB, ATmega406, ATmega48P, ATmega64, AT- mega8, ATmega8515, AT- mega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, AT- mega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny2313, ATtiny28, ATtiny48, ATtiny261, ATtiny461, ATtiny861, AT90↔ USB162, AT90USB82, AT90↔ USB1287, AT90USB1286, AT90USB647, AT90USB646

INT2_vect	SIG_INTERRUPT2	External Interrupt Request 2	AT90PWM3, AT90PWM1, AT90CAN32, ATmega103, ATmega1284P, ATmega161, ATmega32, ATmega32HVB, ATmega64, ATmega8535, ATmega1280, ATmega2560, ATmega324P, ATmega644P, ATmega16HVA, AT90USB162, AT90USB82, AT90USB1286, AT90USB647, AT90USB646	AT90PWM2, AT90CAN128, AT90CAN64, ATmega128, ATmega16, ATmega162, ATmega323, ATmega406, ATmega8515, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644, AT90USB1287, AT90USB647, AT90USB646
INT3_vect	SIG_INTERRUPT3	External Interrupt Request 3	AT90PWM3, AT90PWM1, AT90CAN32, ATmega103, ATmega32HVB, ATmega64, ATmega8535, ATmega1280, ATmega2560, AT90USB162, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90PWM2, AT90CAN128, AT90CAN64, ATmega128, ATmega406, ATmega640, ATmega1281, ATmega2561, AT90USB82, AT90USB1286, AT90USB647, AT90USB646
INT4_vect	SIG_INTERRUPT4	External Interrupt Request 4	AT90CAN128, AT90CAN64, ATmega128, ATmega640, ATmega1281, ATmega2561, AT90USB82, AT90USB1286, AT90USB646	AT90CAN32, ATmega103, ATmega64, ATmega1280, ATmega2560, AT90USB162, AT90USB1287, AT90USB647, AT90USB646
INT5_vect	SIG_INTERRUPT5	External Interrupt Request 5	AT90CAN128, AT90CAN64, ATmega128, ATmega640, ATmega1281, ATmega2561, AT90USB82, AT90USB1286, AT90USB646	AT90CAN32, ATmega103, ATmega64, ATmega1280, ATmega2560, AT90USB162, AT90USB1287, AT90USB647, AT90USB646
INT6_vect	SIG_INTERRUPT6	External Interrupt Request 6	AT90CAN128, AT90CAN64, ATmega128, ATmega640, ATmega1281, ATmega2561, AT90USB82, AT90USB1286, AT90USB646	AT90CAN32, ATmega103, ATmega64, ATmega1280, ATmega2560, AT90USB162, AT90USB1287, AT90USB647, AT90USB646
INT7_vect	SIG_INTERRUPT7	External Interrupt Request 7	AT90CAN128, AT90CAN64, ATmega128, ATmega640, ATmega1281, ATmega2561, AT90USB82, AT90USB1286, AT90USB646	AT90CAN32, ATmega103, ATmega64, ATmega1280, ATmega2560, AT90USB162, AT90USB1287, AT90USB647, AT90USB646
IO_PINS_vect	SIG_PIN, SIG_PIN_CHANGE	External Interrupt Request 0	ATtiny11, ATtiny12, ATtiny15, ATtiny26	
LCD_vect	SIG_LCD	LCD Start of Frame	ATmega169, ATmega169P, ATmega329, ATmega3290, ATmega3290P, ATmega649, ATmega6490	
LOWLEVEL_IO_PINS_vect	SIG_PIN	Low-level Input on Port B	ATtiny28	

OVRIT_vect	SIG_CAN_OVERFLOW1	CAN Timer Overrun	AT90CAN128, AT90CAN64	AT90CAN32,
PCINT0_vect	SIG_PIN_CHANGE0	Pin Change Interrupt Request 0	ATmega162, ATmega165P, ATmega169, ATmega325, ATmega3250P, ATmega329, ATmega3290P, HVB, ATmega406, ATmega48P, ATmega645, ATmega649, ATmega88P, ATmega48, ATmega640, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny13, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	ATmega165, ATmega168P, ATmega169P, ATmega3250, ATmega328P, ATmega3290, ATmega32↔, ATmega48P, ATmega6450, ATmega6490, ATmega168, ATmega88, ATmega1280, ATmega2560, ATmega324P, ATmega644P, ATtiny13, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny85, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
PCINT1_vect	SIG_PIN_CHANGE1	Pin Change Interrupt Request 1	ATmega162, ATmega165P, ATmega169, ATmega325, ATmega3250P, ATmega329, ATmega3290P, HVB, ATmega406, ATmega48P, ATmega645, ATmega649, ATmega88P, ATmega48, ATmega640, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, AT90USB162, AT90USB82	ATmega165, ATmega168P, ATmega169P, ATmega3250, ATmega328P, ATmega3290, ATmega32↔, ATmega48P, ATmega6450, ATmega6490, ATmega168, ATmega88, ATmega1280, ATmega2560, ATmega324P, ATmega644P, ATmega644, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, AT90USB162, AT90USB82
PCINT2_vect	SIG_PIN_CHANGE2	Pin Change Interrupt Request 2	ATmega3250, ATmega328P, ATmega3290P, ATmega6450, ATmega88P, ATmega48, ATmega640, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny48	ATmega3250P, ATmega3290, ATmega48P, ATmega6490, ATmega168, ATmega88, ATmega1280, ATmega2560, ATmega324P, ATmega644P, ATmega644, ATtiny48
PCINT3_vect	SIG_PIN_CHANGE3	Pin Change Interrupt Request 3	ATmega3250, ATmega3290, ATmega6450, ATmega324P, ATmega644P, ATtiny48	ATmega3250P, ATmega3290P, ATmega6490, ATmega164P, ATmega644, ATtiny48
PCINT_vect	SIG_PIN_CHANGE, SIG_↔, PCINT		ATtiny2313, ATtiny261, ATtiny461, ATtiny861	
PSC0_CAPT_vect	SIG_PSC0_CAPTURE	PSC0 Capture Event	AT90PWM3, AT90PWM1	AT90PWM2,
PSC0_EC_vect	SIG_PSC0_END_CYCLE	PSC0 End Cycle	AT90PWM3, AT90PWM1	AT90PWM2,
PSC1_CAPT_vect	SIG_PSC1_CAPTURE	PSC1 Capture Event	AT90PWM3, AT90PWM1	AT90PWM2,
PSC1_EC_vect	SIG_PSC1_END_CYCLE	PSC1 End Cycle	AT90PWM3, AT90PWM1	AT90PWM2,

PSC2_CAPT_vect	SIG_PSC2_CAPTURE	PSC2 Capture Event	AT90PWM3, AT90PWM1	AT90PWM2,
PSC2_EC_vect	SIG_PSC2_END_CYCLE	PSC2 End Cycle	AT90PWM3, AT90PWM1	AT90PWM2,
SPI_STC_vect	SIG_SPI	Serial Transfer Complete	AT90S2333, AT90S4433, AT90S8515, AT90PWM216, AT90PWM316, AT90PWM3, AT90PWM1, AT90CAN32, ATmega103, ATmega1284P, ATmega161, ATmega163, ATmega165P, ATmega169, ATmega32, ATmega325, ATmega3250P, ATmega329, ATmega3290P, HVB, ATmega48P, ATmega64, ATmega645, ATmega649, ATmega8, ATmega8535, ATmega168, ATmega88, ATmega640, AT- mega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny48, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90S4414, AT90S4434, AT90S8535, AT90PWM2B, AT90PWM3B, AT90PWM2, AT90CAN128, AT90CAN64, ATmega128, ATmega16, ATmega162, ATmega165, ATmega168P, ATmega169P, ATmega323, ATmega3250, ATmega328P, ATmega3290, ATmega32↔ ATmega32↔
SPM_RDY_vect	SIG_SPM_READY	Store Program Memory Ready	ATmega16, ATmega32, ATmega8, ATmega8515, AT- mega8535	ATmega162, ATmega323,
SPM_READY_vect	SIG_SPM_READY	Store Program Memory Read	AT90PWM3, AT90PWM1, AT90CAN32, ATmega128, ATmega165, ATmega168P, ATmega169P, ATmega3250, ATmega328P, ATmega3290, ATmega406, ATmega64, ATmega6450, ATmega6490, ATmega168, ATmega88, ATmega640, AT- mega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90PWM2, AT90CAN128, AT90CAN64, ATmega1284P, ATmega165P, ATmega169, ATmega325, ATmega3250P, ATmega329, ATmega3290P, ATmega48P, ATmega645, ATmega649, ATmega88P, ATmega48, ATmega640, AT- mega1281, ATmega2561, ATmega164P, ATmega644, AT90USB82, AT90USB1286, AT90USB646
TIM0_COMPA_vect	SIG_OUTPUT_COMPARE0A	Timer/Counter Compare Match A	ATtiny13, ATtiny43U, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85	
TIM0_COMPB_vect	SIG_OUTPUT_COMPARE0B	Timer/Counter Compare Match B	ATtiny13, ATtiny43U, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85	

TIM0_OVF_vect	SIG_OVERFLOW0	Timer/Counter0 Overflow	ATtiny13, ATtiny43U, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85
TIM1_CAPT_vect	SIG_INPUT_CAPTURE1	Timer/Counter1 Capture Event	ATtiny24, ATtiny44, ATtiny84
TIM1_COMPA_vect	SIG_OUTPUT_COMPARE1A	Timer/Counter1 Compare Match A	ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85
TIM1_COMPB_vect	SIG_OUTPUT_COMPARE1B	Timer/Counter1 Compare Match B	ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85
TIM1_OVF_vect	SIG_OVERFLOW1	Timer/Counter1 Overflow	ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85
TIMER0_CAPT_vect	SIG_INPUT_CAPTURE0	ADC Conversion Complete	ATtiny261, ATtiny461, ATtiny861
TIMER0_COMPA_vect	SIG_OUTPUT_COMPARE0A	TimerCounter0 Compare Match A	ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny2313, ATtiny48, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TIMER0_COMPB_vect	SIG_OUTPUT_COMPARE0B, SIG_OUTPUT_COMPARE0↔_B	Timer Counter 0 Compare Match B	AT90PWM3, AT90PWM2, AT90PWM1, ATmega1284P, ATmega168P, ATmega328P, ATmega32HVB, ATmega48P, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny2313, ATtiny48, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TIMER0_COMP_A_vect	SIG_OUTPUT_COMPARE0A, SIG_OUTPUT_COMPARE0↔_A	Timer/Counter0 Compare Match A	AT90PWM3, AT90PWM2, AT90PWM1
TIMER0_COMP_vect	SIG_OUTPUT_COMPARE0	Timer/Counter0 Compare Match	AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega16, ATmega161, ATmega162, ATmega165, ATmega165P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega329, ATmega3290, ATmega3290P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8515, ATmega8535
TIMER0_OVF0_vect	SIG_OVERFLOW0	Timer/Counter0 Overflow	AT90S2313, AT90S2323, AT90S2343, ATtiny22, ATtiny26

TIMER0_OVF_vect	SIG_OVERFLOW0	Timer/Counter0 Overflow	AT90S1200, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32HVB, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, AT- mega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, AT- mega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny11, ATtiny12, ATtiny15, ATtiny2313, ATtiny28, AT- tiny48, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TIMER1_CAPT1_vect	SIG_INPUT_CAPTURE1	Timer/Counter1 Capture Event	AT90S2313
TIMER1_CAPT_vect	SIG_INPUT_CAPTURE1	Timer/Counter Capture Event	AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, AT- mega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, AT- mega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny2313, ATtiny48, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646

TIMER1_CMPA_vect	SIG_OUTPUT_COMPARE1A	Timer/Counter1 Match 1A	Compare	ATtiny26
TIMER1_CMPB_vect	SIG_OUTPUT_COMPARE1B	Timer/Counter1 Match 1B	Compare	ATtiny26
TIMER1_COMP1_vect	SIG_OUTPUT_COMPARE1A	Timer/Counter1 Match	Compare	AT90S2313
TIMER1_CMPA_vect	SIG_OUTPUT_COMPARE1A	Timer/Counter1 Match A	Compare	AT90S4414, AT90S8515, AT90PWM216, AT90PWM316, AT90PWM3, AT90PWM1, AT90CAN32, ATmega103, ATmega1284P, ATmega161, ATmega163, ATmega165P, ATmega169, ATmega32, ATmega325, ATmega3250P, ATmega329, ATmega3290P, HV, ATmega48P, ATmega64, ATmega645, ATmega649, ATmega8, ATmega8535, ATmega168, ATmega88, ATmega640, ATmega1280, ATmega2560, ATmega324P, ATmega644P, ATmega16HVA, ATtiny48, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646

TIMER1_COMPB_vect	SIG_OUTPUT_COMPARE1B	Timer/Counter1 MatchB	Compare	AT90S4414, AT90S8515, AT90PWM216, AT90PWM316, AT90PWM3, AT90PWM1, AT90CAN32, ATmega103, ATmega1284P, ATmega161, ATmega163, ATmega165P, ATmega169, ATmega32, ATmega325, ATmega3250P, ATmega329, ATmega3290P, ATmega3290P, ATmega48P, HVB, ATmega645, ATmega649, ATmega8, ATmega8535, ATmega168, ATmega88, ATmega1280, ATmega2560, ATmega324P, ATmega644P, ATmega16HVA, ATtiny48, ATtiny261, ATtiny461, ATtiny861, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90S4434, AT90S8535, AT90PWM2B, AT90PWM3B, AT90PWM2, AT90CAN128, AT90CAN64, ATmega128, ATmega16, ATmega162, ATmega165, ATmega168P, ATmega169P, ATmega323, ATmega3250, ATmega328P, ATmega3290, ATmega32↔, ATmega64, ATmega6450, ATmega6490, ATmega8515, ATmega88P, ATmega48, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644, ATtiny2313, ATtiny461, AT90USB162, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TIMER1_COMPC_vect	SIG_OUTPUT_COMPARE1C	Timer/Counter1 Match C	Compare	AT90CAN128, AT90CAN64, ATmega128, ATmega64, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90CAN32, ATmega128, ATmega640, ATmega1281, ATmega2561, AT90USB82, AT90USB1287, AT90USB1286, AT90USB646
TIMER1_COMPD_vect	SIG_OUTPUT_COMPARE0D	Timer/Counter1 Match D	Compare	ATtiny261, ATtiny861	ATtiny461, ATtiny861
TIMER1_COMP_vect	SIG_OUTPUT_COMPARE1A	Timer/Counter1 Match	Compare	AT90S2333, ATtiny15	AT90S4433, ATtiny15
TIMER1_OVF1_vect	SIG_OVERFLOW1	Timer/Counter1 Overflow		AT90S2313, ATtiny26	

TIMER1_OVF_vect	SIG_OVERFLOW1	Timer/Counter1 Overflow	AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32↔ HVB, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, ATmega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, AT- mega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny15, ATtiny2313, ATtiny48, AT- tiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TIMER2_COMPA_vect	SIG_OUTPUT_COMPARE2A	Timer/Counter2 Compare Match A	ATmega168, ATmega48, ATmega88, ATmega640, AT- mega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TIMER2_COMPB_vect	SIG_OUTPUT_COMPARE2B	Timer/Counter2 Compare Match A	ATmega168, ATmega48, ATmega88, ATmega640, AT- mega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TIMER2_COMP_vect	SIG_OUTPUT_COMPARE2	Timer/Counter2 Compare Match	AT90S4434, AT90S8535, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega16, ATmega161, ATmega162, ATmega163, ATmega165, ATmega165P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega329, ATmega3290, ATmega3290P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, ATmega8535

TIMER2_OVF_vect	SIG_OVERFLOW2	Timer/Counter2 Overflow	AT90S4434, AT90CAN128, AT90CAN64, ATmega128, ATmega16, ATmega162, ATmega165, ATmega168P, ATmega169P, ATmega323, ATmega3250, ATmega328P, ATmega3290, ATmega48P, ATmega645, ATmega649, ATmega8, ATmega88P, ATmega48, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644, AT90USB1286, AT90USB646	AT90S8535, AT90CAN32, ATmega103, ATmega1284P, ATmega161, ATmega163, ATmega165P, ATmega169, ATmega32, ATmega325, ATmega3250P, ATmega329, ATmega3290P, ATmega64, ATmega6450, ATmega6490, ATmega8535, ATmega168, ATmega88, ATmega1280, ATmega2560, ATmega324P, ATmega644P, AT90USB1287, AT90USB647
TIMER3_CAPT_vect	SIG_INPUT_CAPTURE3	Timer/Counter3 Capture Event	AT90CAN128, AT90CAN64, ATmega1284P, ATmega64, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90CAN32, ATmega128, ATmega162, ATmega162, ATmega640, ATmega1281, ATmega2561, AT90USB1286, AT90USB646
TIMER3_COMPA_vect	SIG_OUTPUT_COMPARE3A	Timer/Counter3 Compare Match A	AT90CAN128, AT90CAN64, ATmega1284P, ATmega64, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90CAN32, ATmega128, ATmega162, ATmega162, ATmega640, ATmega1281, ATmega2561, AT90USB1286, AT90USB646
TIMER3_COMPB_vect	SIG_OUTPUT_COMPARE3B	Timer/Counter3 Compare Match B	AT90CAN128, AT90CAN64, ATmega1284P, ATmega64, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90CAN32, ATmega128, ATmega162, ATmega162, ATmega640, ATmega1281, ATmega2561, AT90USB1286, AT90USB646
TIMER3_COMPC_vect	SIG_OUTPUT_COMPARE3C	Timer/Counter3 Compare Match C	AT90CAN128, AT90CAN64, ATmega1284P, ATmega64, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90CAN32, ATmega128, ATmega162, ATmega162, ATmega640, ATmega1281, ATmega2561, AT90USB1286, AT90USB646
TIMER3_OVF_vect	SIG_OVERFLOW3	Timer/Counter3 Overflow	AT90CAN128, AT90CAN64, ATmega1284P, ATmega64, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646	AT90CAN32, ATmega128, ATmega162, ATmega162, ATmega640, ATmega1281, ATmega2561, AT90USB1286, AT90USB646
TIMER4_CAPT_vect	SIG_INPUT_CAPTURE4	Timer/Counter4 Capture Event	ATmega640, ATmega1280, ATmega2560, ATmega2561	ATmega1280, ATmega2560, ATmega2561
TIMER4_COMPA_vect	SIG_OUTPUT_COMPARE4A	Timer/Counter4 Compare Match A	ATmega640, ATmega1280, ATmega2560, ATmega2561	ATmega1280, ATmega2560, ATmega2561

TIMER4_COMPB_vect	SIG_OUTPUT_COMPARE4B	Timer/Counter4 Compare Match B	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TIMER4_COMPC_vect	SIG_OUTPUT_COMPARE4C	Timer/Counter4 Compare Match C	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TIMER4_OVF_vect	SIG_OVERFLOW4	Timer/Counter4 Overflow	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TIMER5_CAPT_vect	SIG_INPUT_CAPTURE5	Timer/Counter5 Capture Event	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TIMER5_COMPA_vect	SIG_OUTPUT_COMPARE5A	Timer/Counter5 Compare Match A	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TIMER5_COMPB_vect	SIG_OUTPUT_COMPARE5B	Timer/Counter5 Compare Match B	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TIMER5_COMPC_vect	SIG_OUTPUT_COMPARE5C	Timer/Counter5 Compare Match C	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TIMER5_OVF_vect	SIG_OVERFLOW5	Timer/Counter5 Overflow	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
TWI_vect	SIG_2WIRE_SERIAL	2-wire Serial Interface	AT90CAN128, AT90CAN32, AT90CAN64, ATmega128, ATmega1284P, ATmega16, ATmega163, ATmega168P, ATmega32, ATmega323, ATmega328P, ATmega32HVB, ATmega406, ATmega48P, ATmega64, ATmega8, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny48, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
TXDONE_vect	SIG_TXDONE	Transmission Done, Bit Timer Flag 2 Interrupt	AT86RF401
TXEMPTY_vect	SIG_TXBE	Transmit Buffer Empty, Bit Timer Flag 0 Interrupt	AT86RF401
UART0_RX_vect	SIG_UART0_RECV	UART0, Rx Complete	ATmega161
UART0_TX_vect	SIG_UART0_TRANS	UART0, Tx Complete	ATmega161
UART0_UDRE_vect	SIG_UART0_DATA	UART0 Data Register Empty	ATmega161
UART1_RX_vect	SIG_UART1_RECV	UART1, Rx Complete	ATmega161
UART1_TX_vect	SIG_UART1_TRANS	UART1, Tx Complete	ATmega161
UART1_UDRE_vect	SIG_UART1_DATA	UART1 Data Register Empty	ATmega161
UART_RX_vect	SIG_UART_RECV	UART, Rx Complete	AT90S2313, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, ATmega103, ATmega163, ATmega8515
UART_TX_vect	SIG_UART_TRANS	UART, Tx Complete	AT90S2313, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, ATmega103, ATmega163, ATmega8515
UART_UDRE_vect	SIG_UART_DATA	UART Data Register Empty	AT90S2313, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, ATmega103, ATmega163, ATmega8515

USART0_RXC_vect	SIG_USART0_RECV	USART0, Rx Complete	ATmega162
USART0_RX_vect	SIG_USART0_RECV	USART0, Rx Complete	AT90CAN128, AT90CAN64, ATmega1284P, ATmega165P, ATmega169P, ATmega329, ATmega645, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644, AT90CAN32, ATmega128, ATmega165, ATmega169, ATmega325, ATmega64, ATmega649, ATmega1280, ATmega2560, ATmega324P, ATmega644P,
USART0_TXC_vect	SIG_USART0_TRANS	USART0, Tx Complete	ATmega162
USART0_TX_vect	SIG_USART0_TRANS	USART0, Tx Complete	AT90CAN128, AT90CAN64, ATmega1284P, ATmega165P, ATmega169P, ATmega3250, ATmega329, ATmega3290P, ATmega645, ATmega649, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644, AT90CAN32, ATmega128, ATmega165, ATmega169, ATmega325, ATmega3250P, ATmega3290, ATmega64, ATmega6450, ATmega6490, ATmega1280, ATmega2560, ATmega324P, ATmega644P,
USART0_UDRE_vect	SIG_USART0_DATA	USART0 Data Register Empty	AT90CAN128, AT90CAN64, ATmega1284P, ATmega165, ATmega169, ATmega325, ATmega64, ATmega649, ATmega1280, ATmega2560, ATmega324P, ATmega644P, ATmega644, AT90CAN32, ATmega128, ATmega162, ATmega165P, ATmega169P, ATmega329, ATmega645, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644
USART1_RXC_vect	SIG_USART1_RECV	USART1, Rx Complete	ATmega162
USART1_RX_vect	SIG_USART1_RECV	USART1, Rx Complete	AT90CAN128, AT90CAN64, ATmega1284P, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644, AT90CAN32, ATmega128, ATmega64, ATmega1280, ATmega2560, ATmega324P, ATmega644P, AT90USB162, AT90USB82, AT90USB1286, AT90USB646, AT90USB1287, AT90USB647,
USART1_TXC_vect	SIG_USART1_TRANS	USART1, Tx Complete	ATmega162
USART1_TX_vect	SIG_USART1_TRANS	USART1, Tx Complete	AT90CAN128, AT90CAN64, ATmega1284P, ATmega640, ATmega1281, ATmega2561, ATmega164P, ATmega644, AT90CAN32, ATmega128, ATmega64, ATmega1280, ATmega2560, ATmega324P, ATmega644P, AT90USB162, AT90USB82, AT90USB1286, AT90USB646, AT90USB1287, AT90USB647,

USART1_UDRE_vect	SIG_USART1_DATA	USART1, Data Register Empty	AT90CAN128, AT90CAN64, ATmega1284P, ATmega64, ATmega1280, ATmega2560, ATmega324P, ATmega644P, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
USART2_RX_vect	SIG_USART2_RECV	USART2, Rx Complete	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
USART2_TX_vect	SIG_USART2_TRANS	USART2, Tx Complete	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
USART2_UDRE_vect	SIG_USART2_DATA	USART2 Data register Empty	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
USART3_RX_vect	SIG_USART3_RECV	USART3, Rx Complete	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
USART3_TX_vect	SIG_USART3_TRANS	USART3, Tx Complete	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
USART3_UDRE_vect	SIG_USART3_DATA	USART3 Data register Empty	ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561
USART_RXC_vect	SIG_USART_RECV, UART_RECV	USART, Rx Complete	ATmega16, ATmega32, ATmega323, ATmega8
USART_RX_vect	SIG_USART_RECV, UART_RECV	USART, Rx Complete	AT90PWM3, AT90PWM1, ATmega3250, ATmega328P, ATmega3290P, ATmega48P, ATmega6450, ATmega8535, ATmega168, ATmega48, ATmega88, ATtiny2313
USART_TXC_vect	SIG_USART_TRANS, UART_TRANS	USART, Tx Complete	ATmega16, ATmega32, ATmega323, ATmega8
USART_TX_vect	SIG_USART_TRANS, UART_TRANS	USART, Tx Complete	AT90PWM3, AT90PWM1, ATmega328P, ATmega8535, ATmega168, ATmega48, ATmega88, ATtiny2313
USART_UDRE_vect	SIG_USART_DATA, UART_DATA	USART Data Register Empty	AT90PWM3, AT90PWM1, ATmega168P, ATmega323, ATmega3250P, ATmega328P, ATmega3290, ATmega48P, ATmega6450, ATmega8535, ATmega168, ATmega48, ATmega88, ATtiny2313
USI_OVERFLOW_vect	SIG_USI_OVERFLOW	USI Overflow	ATmega165, ATmega169, ATmega325, ATmega3250P, ATmega3290, ATmega645, ATmega649, ATtiny2313

USI_OVF_vect	SIG_USI_OVERFLOW	USI Overflow	ATtiny26, ATtiny43U, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861
USI_START_vect	SIG_USI_START	USI Start Condition	ATmega165, ATmega165P, ATmega169, ATmega169P, ATmega325, ATmega3250, ATmega3250P, ATmega329, ATmega3290, ATmega3290P, ATmega645, ATmega6450, ATmega649, ATmega6490, ATtiny2313, ATtiny43U, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861
USI_STRT_vect	SIG_USI_START	USI Start	ATtiny26
USI_STR_vect	SIG_USI_START	USI START	ATtiny24, ATtiny44, ATtiny84
WATCHDOG_vect	SIG_WATCHDOG_TIMEOUT	Watchdog Time-out	ATtiny24, ATtiny44, ATtiny84
WDT_OVERFLOW_vect	SIG_WATCHDOG_TIMEOUT, SIG_WDT_OVERFLOW	Watchdog Timer Overflow	ATtiny2313
WDT_vect	SIG_WDT, SIG_WDT_OVERFLOW, SIG_WATCHDOG_TIMEOUT	Watchdog Timeout Interrupt	AT90PWM3, AT90PWM2, AT90PWM1, ATmega1284P, ATmega168P, ATmega328P, ATmega32HVB, ATmega406, ATmega48P, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny13, ATtiny43U, ATtiny48, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646

23.17.2 Macro Definition Documentation

23.17.2.1 BADISR_vect `#define BADISR_vect` `#include <avr/interrupt.h>`

This is a vector which is aliased to `__vector_default`, the vector executed when an ISR 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.

23.17.2.2 cli `#define cli()`

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()`.

23.17.2.3 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);
```

23.17.2.4 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` and `ISR_ALIASOF(vect)`.

`vector` must be one of the interrupt vector names that are valid for the particular MCU type.

23.17.2.5 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);
```

23.17.2.6 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));
```

23.17.2.7 **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.

23.17.2.8 **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.

23.17.2.9 **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.

23.17.2.10 **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.

23.17.2.11 **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.

23.17.2.12 **reti** `#define reti()`

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.

23.17.2.13 **sei** `#define sei()`

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\(\)](#).

23.17.2.14 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.

23.18 `<avr/io.h>`: AVR device-specific IO definitions

Macros

- `#define _PROTECTED_WRITE(reg, value)`

23.18.1 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.

23.18.2 Macro Definition Documentation

23.18.2.1 `_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);
```

23.19 <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>
```

23.20 <avr/pgmspace.h>: Program Space Utilities

Macros

- #define `PROGMEM __ATTR_PROGMEM__`
- #define `PGM_P` const char *
- #define `PGM_VOID_P` const void *
- #define `PSTR(s)` ((const `PROGMEM` char *) (s))
- #define `pgm_read_byte_near`(address_short) `__LPM__((uint16_t)(address_short))`
- #define `pgm_read_word_near`(address_short) `__LPM_word__((uint16_t)(address_short))`
- #define `pgm_read_dword_near`(address_short) `__LPM_dword__((uint16_t)(address_short))`
- #define `pgm_read_float_near`(address_short) `__LPM_float__((uint16_t)(address_short))`
- #define `pgm_read_ptr_near`(address_short) (void*) `__LPM_word__((uint16_t)(address_short))`
- #define `pgm_read_byte_far`(address_long) `__ELPM__((uint32_t)(address_long))`
- #define `pgm_read_word_far`(address_long) `__ELPM_word__((uint32_t)(address_long))`
- #define `pgm_read_dword_far`(address_long) `__ELPM_dword__((uint32_t)(address_long))`
- #define `pgm_read_float_far`(address_long) `__ELPM_float__((uint32_t)(address_long))`
- #define `pgm_read_ptr_far`(address_long) (void*) `__ELPM_word__((uint32_t)(address_long))`
- #define `pgm_read_byte`(address_short) `pgm_read_byte_near`(address_short)
- #define `pgm_read_word`(address_short) `pgm_read_word_near`(address_short)
- #define `pgm_read_dword`(address_short) `pgm_read_dword_near`(address_short)
- #define `pgm_read_float`(address_short) `pgm_read_float_near`(address_short)
- #define `pgm_read_ptr`(address_short) `pgm_read_ptr_near`(address_short)
- #define `pgm_get_far_address`(var)

Typedefs

- typedef void `PROGMEM prog_void`
- typedef char `PROGMEM prog_char`
- typedef unsigned char `PROGMEM prog_uchar`
- typedef `int8_t` `PROGMEM prog_int8_t`
- typedef `uint8_t` `PROGMEM prog_uint8_t`
- typedef `int16_t` `PROGMEM prog_int16_t`
- typedef `uint16_t` `PROGMEM prog_uint16_t`
- typedef `int32_t` `PROGMEM prog_int32_t`
- typedef `uint32_t` `PROGMEM prog_uint32_t`
- typedef `int64_t` `PROGMEM prog_int64_t`
- typedef `uint64_t` `PROGMEM prog_uint64_t`

Functions

- const void * `memchr_P` (const void *, int __val, size_t __len)
- int `memcmp_P` (const void *, const void *, size_t) `__ATTR_PURE__`
- 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) `__ATTR_PURE__`
- 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 *) `__ATTR_PURE__`
- char * `strcpy_P` (char *, const char *)
- int `strcasecmp_P` (const char *, const char *) `__ATTR_PURE__`

- `char * strcasestr_P` (const char *, const char *) `__ATTR_PURE__`
- `size_t strcspn_P` (const char * __s, const char * __reject) `__ATTR_PURE__`
- `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) `__ATTR_PURE__`
- `int strncasecmp_P` (const char *, const char *, size_t) `__ATTR_PURE__`
- `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) `__ATTR_PURE__`
- `const char * strchr_P` (const char *, int __val)
- `char * strsep_P` (char ** __sp, const char * __delim)
- `size_t strspn_P` (const char * __s, const char * __accept) `__ATTR_PURE__`
- `char * strstr_P` (const char *, const char *) `__ATTR_PURE__`
- `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) `__ATTR_PURE__`
- `int strncmp_PF` (const char *s1, uint_farptr_t s2, size_t n) `__ATTR_PURE__`
- `int strcasecmp_PF` (const char *s1, uint_farptr_t s2) `__ATTR_PURE__`
- `int strncasecmp_PF` (const char *s1, uint_farptr_t s2, size_t n) `__ATTR_PURE__`
- `char * strstr_PF` (const char *s1, uint_farptr_t s2)
- `size_t strlcpy_PF` (char *dst, uint_farptr_t src, size_t siz)
- `int memcpy_PF` (const void *, uint_farptr_t, size_t) `__ATTR_PURE__`
- `static size_t strlen_P` (const char *s)

23.20.1 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. In order to use these functions, the target device must support either the LPM or ELPM instructions.

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.

23.20.2 Macro Definition Documentation

23.20.2.1 `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 " " \n\t"
        :
        : "=d" (tmp)
        : "p" (&(var))
    );
    tmp;
}
```

This macro facilitates the obtention of a 32 bit "far" pointer (only 24 bits used) to data even passed the 64KB limit for the 16 bit ordinary pointer. It is similar to the '&' operator, with some limitations.

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.
- `hh8()` is an undocumented feature but seems to give the third significant byte of a 32 bit data and accepts symbols, complementing the functionality of `hi8()` and `lo8()`. There is not an equivalent assembler function to get the high significant byte.
- 'var' has to be resolved at linking time as an existing symbol, i.e, a simple type variable name, an array name (not an indexed element of the array, if the index is a constant the compiler does not complain but fails to get the address if optimization is enabled), a struct name or a struct field name, a function identifier, a linker defined identifier,...
- The returned value is the identifier'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 address spaces in the processor: Flash ROM, RAM, and EEPROM. Typical offset for this are 0x00000000, 0x00800xx0, and 0x00810000 respectively, derived from the linker script used and linker options. The value returned can be seen then as a universal pointer.

23.20.2.2 `PGM_P` `#define PGM_P const char *`

Used to declare a variable that is a pointer to a string in program space.

23.20.2.3 `pgm_read_byte` `#define pgm_read_byte(address_short) pgm_read_byte_near(address_short)`

Read a byte from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.4 pgm_read_byte_far `#define pgm_read_byte_far(
 address_long) __ELPM__(uint32_t)(address_long)`

Read a byte from the program space with a 32-bit (far) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.5 pgm_read_byte_near `#define pgm_read_byte_near(
 address_short) __LPM__(uint16_t)(address_short)`

Read a byte from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.6 pgm_read_dword `#define pgm_read_dword(
 address_short) pgm_read_dword_near(address_short)`

Read a double word from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.7 pgm_read_dword_far `#define pgm_read_dword_far(
 address_long) __ELPM_dword__(uint32_t)(address_long)`

Read a double word from the program space with a 32-bit (far) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.8 pgm_read_dword_near `#define pgm_read_dword_near(
 address_short) __LPM_dword__(uint16_t)(address_short)`

Read a double word from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.9 pgm_read_float #define pgm_read_float(
 address_short) pgm_read_float_near(address_short)

Read a float from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.10 pgm_read_float_far #define pgm_read_float_far(
 address_long) __ELPM_float((uint32_t)(address_long))

Read a float from the program space with a 32-bit (far) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.11 pgm_read_float_near #define pgm_read_float_near(
 address_short) __LPM_float((uint16_t)(address_short))

Read a float from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.12 pgm_read_ptr #define pgm_read_ptr(
 address_short) pgm_read_ptr_near(address_short)

Read a pointer from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.13 pgm_read_ptr_far #define pgm_read_ptr_far(
 address_long) (void*)__ELPM_word((uint32_t)(address_long))

Read a pointer from the program space with a 32-bit (far) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.14 pgm_read_ptr_near `#define pgm_read_ptr_near(
 address_short) (void*)__LPM_word((uint16_t) (address_short))`

Read a pointer from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.15 pgm_read_word `#define pgm_read_word(
 address_short) pgm_read_word_near(address_short)`

Read a word from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.16 pgm_read_word_far `#define pgm_read_word_far(
 address_long) __ELPM_word((uint32_t) (address_long))`

Read a word from the program space with a 32-bit (far) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.17 pgm_read_word_near `#define pgm_read_word_near(
 address_short) __LPM_word((uint16_t) (address_short))`

Read a word from the program space with a 16-bit (near) address.

Note

The address is a byte address. The address is in the program space.

23.20.2.18 PGM_VOID_P `#define PGM_VOID_P const void *`

Used to declare a generic pointer to an object in program space.

23.20.2.19 PROGMEM `#define PROGMEM __ATTR_PROGMEM__`

Attribute to use in order to declare an object being located in flash ROM.

23.20.2.20 PSTR `#define PSTR(
s) ((const PROGMEM char *) (s))`

Used to declare a static pointer to a string in program space.

23.20.3 Typedef Documentation**23.20.3.1 prog_char** `prog_char`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of a "char" object located in flash ROM.

23.20.3.2 prog_int16_t `prog_int16_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "int16_t" object located in flash ROM.

23.20.3.3 prog_int32_t `prog_int32_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "int32_t" object located in flash ROM.

23.20.3.4 `prog_int64_t` `prog_int64_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "int64_t" object located in flash ROM.

Note

This type is not available when the compiler option `-mint8` is in effect.

23.20.3.5 `prog_int8_t` `prog_int8_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "int8_t" object located in flash ROM.

23.20.3.6 `prog_uchar` `prog_uchar`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "unsigned char" object located in flash ROM.

23.20.3.7 `prog_uint16_t` `prog_uint16_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "uint16_t" object located in flash ROM.

23.20.3.8 `prog_uint32_t` `prog_uint32_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "uint32_t" object located in flash ROM.

23.20.3.9 `prog_uint64_t` `prog_uint64_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "uint64_t" object located in flash ROM.

Note

This type is not available when the compiler option `-mint8` is in effect.

23.20.3.10 `prog_uint8_t` `prog_uint8_t`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of an "uint8_t" object located in flash ROM.

23.20.3.11 `prog_void` `prog_void`

Note

DEPRECATED

This typedef is now deprecated because the usage of the `__progmem__` attribute on a type is not supported in GCC. However, the use of the `__progmem__` attribute on a variable declaration is supported, and this is now the recommended usage.

The typedef is only visible if the macro `__PROG_TYPES_COMPAT__` has been defined before including `<avr/pgmspace.h>` (either by a `#define` directive, or by a `-D` compiler option.)

Type of a "void" object located in flash ROM. Does not make much sense by itself, but can be used to declare a "void *" object in flash ROM.

23.20.4 Function Documentation

23.20.4.1 `memcpy_P()` `void * memcpy_P (`
 `void * dest,`
 `const void * src,`
 `int val,`
 `size_t len)`

This function is similar to `memcpy()` except that `src` is pointer to a string in program space.

23.20.4.2 `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.

23.20.4.3 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`.

23.20.4.4 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`.

23.20.4.5 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`.

23.20.4.6 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.

23.20.4.7 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.

23.20.4.8 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 [memrchr_P\(\)](#) function returns a pointer to the matching byte or `NULL` if the character does not occur in the given memory area.

23.20.4.9 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

<i>s1</i>	A pointer to a string in the devices SRAM.
<i>s2</i>	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*.

23.20.4.10 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.

23.20.4.11 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.

23.20.4.12 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*.

23.20.4.13 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

23.20.4.14 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.

23.20.4.15 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.

23.20.4.16 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*.

```
23.20.4.17  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

<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 [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.

23.20.4.18 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*.

23.20.4.19 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.

23.20.4.20 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.

23.20.4.21 `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.

23.20.4.22 `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.

23.20.4.23 `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.

23.20.4.24 `strlcpy_PF()` `size_t strlcpy_PF (`
 `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_PF\(\)`](#) function returns `strlen(src)`. If `retval >= siz`, truncation occurred. The contents of RAMPZ SFR are undefined when the function returns.

23.20.4.25 `strlen_P()` `size_t strlen_P (`
 `const char * src) [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.

23.20.4.26 `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

<i>s</i>	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.

23.20.4.27 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

<i>s1</i>	A pointer to a string in the devices SRAM.
<i>s2</i>	A pointer to a string in the devices Flash.
<i>n</i>	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*.

23.20.4.28 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

<i>s1</i>	A pointer to a string in SRAM
<i>s2</i>	A far pointer to a string in Flash
<i>n</i>	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.

23.20.4.29 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*.

23.20.4.30 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.

23.20.4.31 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*.

23.20.4.32 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.

23.20.4.33 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*.

23.20.4.34 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.

23.20.4.35 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*.

23.20.4.36 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.

23.20.4.37 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`.

23.20.4.38 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.

23.20.4.39 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`.

23.20.4.40 `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.

23.20.4.41 `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`.

23.20.4.42 `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.

23.20.4.43 `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()`.

23.20.4.44 `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.

23.21 <avr/power.h>: Power Reduction Management

Functions

- void `clock_prescale_set` (`clock_div_t __x`)

23.21.1 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.

Power Macro	Description
power_aca_disable()	Disable the Analog Comparator on PortA.
power_aca_enable()	Enable the Analog Comparator on PortA.
power_adc_enable()	Enable the Analog to Digital Converter module.
power_adc_disable()	Disable the Analog to Digital Converter module.
power_adca_disable()	Disable the Analog to Digital Converter module on PortA
power_adca_enable()	Enable the Analog to Digital Converter module on PortA
power_evsys_disable()	Disable the EVSYS module
power_evsys_enable()	Enable the EVSYS module
power_hiresc_disable()	Disable the HIRES module on PortC
power_hiresc_enable()	Enable the HIRES module on PortC
power_lcd_enable()	Enable the LCD module.
power_lcd_disable().	Disable the LCD module.
power_pga_enable()	Enable the Programmable Gain Amplifier module.
power_pga_disable()	Disable the Programmable Gain Amplifier module.
power_pscr_enable()	Enable the Reduced Power Stage Controller module.
power_pscr_disable()	Disable the Reduced Power Stage Controller module.
power_psc0_enable()	Enable the Power Stage Controller 0 module.
power_psc0_disable()	Disable the Power Stage Controller 0 module.
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_tc1c_disable()	Disable the TC1 module on PortC
power_tc1c_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.

<code>power_timer5_enable()</code>	Enable the Timer 5 module.
<code>power_timer5_disable()</code>	Disable the Timer 5 module.
<code>power_tw_i_enable()</code>	Enable the Two Wire Interface module.
<code>power_tw_i_disable()</code>	Disable the Two Wire Interface module.
<code>power_usart_enable()</code>	Enable the USART module.
<code>power_usart_disable()</code>	Disable the USART module.
<code>power_usart0_enable()</code>	Enable the USART 0 module.
<code>power_usart0_disable()</code>	Disable the USART 0 module.
<code>power_usart1_enable()</code>	Enable the USART 1 module.
<code>power_usart1_disable()</code>	Disable the USART 1 module.
<code>power_usart2_enable()</code>	Enable the USART 2 module.
<code>power_usart2_disable()</code>	Disable the USART 2 module.
<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.

23.21.2 Function Documentation

23.21.2.1 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.

23.22 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_IO_ADDR(SPMCR)` macro can be used to get the address of the `SPMCR` register (0x57 or 0x68 depending on device).

23.23 <avr/sfr_defs.h>: Special function registers

Modules

- [Additional notes from <avr/sfr_defs.h>](#)

Bit manipulation

- `#define _BV(bit) (1 << (bit))`
- `#define _VECTOR(N) __vector_ ## N`

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))`

23.23.1 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\(bit\)](#));

23.23.2 Macro Definition Documentation

23.23.2.1 [_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\(\)](#).

```
23.23.2.2 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.

```
23.23.2.3 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.

```
23.23.2.4 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.

```
23.23.2.5 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.

23.24 <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.

23.25 <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)

23.25.1 Detailed Description

```
#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();
```

23.25.2 Function Documentation

23.25.2.1 sleep_bod_disable() void sleep_bod_disable (void)

Disable BOD before going to sleep. Not available on all devices.

23.25.2.2 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.

23.25.2.3 sleep_disable() void sleep_disable (void)

Clear the SE (sleep enable) bit.

23.25.2.4 sleep_enable() void sleep_enable (void)

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.

23.25.2.5 sleep_mode() void sleep_mode (void)

Put the device into sleep mode, taking care of setting the SE bit before, and clearing it afterwards.

23.26 <avr/version.h>: avr-libc version macros

Macros

- `#define __AVR_LIBC_VERSION_STRING__ "2.1.0"`
- `#define __AVR_LIBC_VERSION__ 20100UL`
- `#define __AVR_LIBC_DATE_STRING__ "20220128"`
- `#define __AVR_LIBC_DATE__ 20220128UL`
- `#define __AVR_LIBC_MAJOR__ 2`
- `#define __AVR_LIBC_MINOR__ 1`
- `#define __AVR_LIBC_REVISION__ 0`

23.26.1 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.

23.26.2 Macro Definition Documentation

23.26.2.1 `__AVR_LIBC_DATE__` `#define __AVR_LIBC_DATE__ 20220128UL`

Numerical representation of the release date.

23.26.2.2 `__AVR_LIBC_DATE_STRING__` `#define __AVR_LIBC_DATE_STRING__ "20220128"`

String literal representation of the release date.

23.26.2.3 `__AVR_LIBC_MAJOR__` `#define __AVR_LIBC_MAJOR__ 2`

Library major version number.

23.26.2.4 `__AVR_LIBC_MINOR__` `#define __AVR_LIBC_MINOR__ 1`

Library minor version number.

23.26.2.5 `__AVR_LIBC_REVISION__` `#define __AVR_LIBC_REVISION__ 0`

Library revision number.

23.26.2.6 `__AVR_LIBC_VERSION__` `#define __AVR_LIBC_VERSION__ 20100UL`

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.

23.26.2.7 `__AVR_LIBC_VERSION_STRING__` `#define __AVR_LIBC_VERSION_STRING__ "2.1.0"`

String literal representation of the current library version.

23.27 `<avr/wdt.h>`: Watchdog timer handling

Macros

- `#define wdt_reset() __asm__ __volatile__ ("wdr")`
- `#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 __inline__ __attribute__((__always_inline__)) void wdt_enable(const uint8_t value)`

23.27.1 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")));
void get_mcusr(void) \
__attribute__((naked)) \
__attribute__((section(".init3")));
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.

23.27.2 Macro Definition Documentation

23.27.2.1 `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.

23.27.2.2 `WDTO_120MS` `#define WDTO_120MS 3`

See `WDTO_15MS`

23.27.2.3 `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);
```

23.27.2.4 `WDTO_1S` `#define WDTO_1S 6`

See `WDTO_15MS`

23.27.2.5 `WDTO_250MS` `#define WDTO_250MS 4`

See `WDTO_15MS`

23.27.2.6 `WDTO_2S` `#define WDTO_2S 7`

See `WDTO_15MS`

23.27.2.7 `WDTO_30MS` `#define WDTO_30MS 1`

See `WDTO_15MS`

23.27.2.8 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, ATmega48P, ATmega88P, ATmega168P, ATmega328P, ATmega164P, ATmega324P, ATmega324PA, 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()`.

23.27.2.9 WDTO_500MS `#define WDTO_500MS 5`

See `WDTO_15MS`

23.27.2.10 WDTO_60MS `#define WDTO_60MS 2`

See `WDTO_15MS`

23.27.2.11 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, ATmega48A, ATmega48PA, ATmega88, ATmega168, ATmega48P, ATmega88P, ATmega168P, ATmega328P, ATmega164P, ATmega324P, ATmega324PA, 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()`.

23.27.3 Function Documentation

23.27.3.1 `__attribute__()` `static __inline__ __attribute__ (` `(__always_inline__)) const [static]`

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.

23.28 <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`

23.28.1 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` or `gnu99`.

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).

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 <inttypes.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 <inttypes.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.

23.28.2 Macro Definition Documentation

23.28.2.1 `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`.

23.28.2.2 `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.

23.28.2.3 `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.

23.28.2.4 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.

23.28.2.5 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.

23.28.2.6 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.

23.29 <util/crc16.h>: CRC Computations

Functions

- static `__inline__ uint16_t _crc16_update (uint16_t __crc, uint8_t __data)`
- static `__inline__ uint16_t _crc_xmodem_update (uint16_t __crc, uint8_t __data)`
- static `__inline__ uint16_t _crc_ccitt_update (uint16_t __crc, uint8_t __data)`
- static `__inline__ uint8_t _crc_ibutton_update (uint8_t __crc, uint8_t __data)`
- static `__inline__ uint8_t _crc8_ccitt_update (uint8_t __crc, uint8_t __data)`

23.29.1 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
}
```

23.29.2 Function Documentation

23.29.2.1 `_crc16_update()` `static __inline__ uint16_t _crc16_update (`
 `uint16_t __crc,`
 `uint8_t __data) [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)
{
    int i;
    crc ^= a;
    for (i = 0; i < 8; ++i)
    {
        if (crc & 1)
            crc = (crc >> 1) ^ 0xA001;
        else
            crc = (crc >> 1);
    }
    return crc;
}
```


23.29.2.2 `_crc8_ccitt_update()` `static __inline__ uint8_t _crc8_ccitt_update (`
`uint8_t __crc,`
`uint8_t __data) [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 i;
    uint8_t data;
    data = inCrc ^ inData;
    for ( i = 0; i < 8; i++ )
    {
        if (( data & 0x80 ) != 0 )
        {
            data <<= 1;
            data ^= 0x07;
        }
        else
        {
            data <<= 1;
        }
    }
    return data;
}
```

23.29.2.3 `_crc_ccitt_update()` `static __inline__ uint16_t _crc_ccitt_update (`
`uint16_t __crc,`
`uint8_t __data) [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));
}
```

23.29.2.4 _crc_ibutton_update() static __inline__ uint8_t _crc_ibutton_update (
 uint8_t __crc,
 uint8_t __data) [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

The following is the equivalent functionality written in C.

```
uint8_t
_crc_ibutton_update(uint8_t crc, uint8_t data)
{
    uint8_t i;
    crc = crc ^ data;
    for (i = 0; i < 8; i++)
    {
        if (crc & 0x01)
            crc = (crc >> 1) ^ 0x8C;
        else
            crc >>= 1;
    }
    return crc;
}
```

23.29.2.5 _crc_xmodem_update() static __inline__ uint16_t _crc_xmodem_update (
 uint16_t __crc,
 uint8_t __data) [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)
{
    int i;
    crc = crc ^ ((uint16_t)data << 8);
    for (i=0; i<8; i++)
    {
        if (crc & 0x8000)
            crc = (crc << 1) ^ 0x1021;
        else
            crc <<= 1;
    }
    return crc;
}
```

23.30 <util/delay.h>: Convenience functions for busy-wait delay loops

Macros

- `#define F_CPU 1000000UL`

Functions

- `void _delay_ms (double __ms)`
- `void _delay_us (double __us)`

23.30.1 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).

23.30.2 Macro Definition Documentation

23.30.2.1 `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.

23.30.3 Function Documentation

23.30.3.1 `_delay_ms()` `void _delay_ms (`
`double __ms)`

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 $262.14 \text{ ms} / F_{\text{CPU}}$ 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 $4294967.295 \text{ ms} / F_{\text{CPU}}$ in MHz. For values greater than the maximal possible delay, overflows results in no delay i.e., 0ms.

Conversion of `__ms` into clock cycles may not always result in integer. By default, the clock cycles rounded up to 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. Also, the backward compatible algorithm will be chosen if the code is compiled in a *freestanding environment* (GCC option `-ffreestanding`), as the math functions required for rounding are not available to the compiler then.

23.30.3.2 `_delay_us()` `void _delay_us (`
`double __us)`

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 $768 \text{ us} / F_{\text{CPU}}$ 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 $4294967.295 \text{ us} / F_{\text{CPU}}$ in MHz. For values greater than the maximal possible delay, overflow results in no delay i.e., 0us.

Conversion of `__us` into clock cycles may not always result in integer. By default, the clock cycles 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_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. Also, the backward compatible algorithm will be chosen if the code is compiled in a *freestanding environment* (GCC option `-ffreestanding`), as the math functions required for rounding are not available to the compiler then.

23.31 <util/delay_basic.h>: Basic busy-wait delay loops

Functions

- void `_delay_loop_1` (uint8_t __count)
- void `_delay_loop_2` (uint16_t __count)

23.31.1 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.

23.31.2 Function Documentation

23.31.2.1 `_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.

23.31.2.2 `_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.

23.32 <util/parity.h>: Parity bit generation

Macros

- #define `parity_even_bit`(val)

23.32.1 Detailed Description

```
#include <util/parity.h>
```

This header file contains optimized assembler code to calculate the parity bit for a byte.

23.32.2 Macro Definition Documentation

23.32.2.1 parity_even_bit #define parity_even_bit(
 val)

Value:

```
(__extension__({
    unsigned char __t;
    __asm__ (
        "mov __tmp_reg__, %0" "\n\t"
        "swap %0" "\n\t"
        "eor %0, __tmp_reg__" "\n\t"
        "mov __tmp_reg__, %0" "\n\t"
        "lsr %0" "\n\t"
        "lsr %0" "\n\t"
        "eor %0, __tmp_reg__"
        : "=r" (__t)
        : "0" ((unsigned char) (val))
        : "r0"
    );
    (((__t + 1) >> 1) & 1);
}))
```

Returns

1 if `val` has an odd number of bits set.

23.33 <util/setbaud.h>: Helper macros for baud rate calculations

Macros

- #define BAUD_TOL 2
- #define UBRR_VALUE
- #define UBRRRL_VALUE
- #define UBRRH_VALUE
- #define USE_2X 0

23.33.1 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↔: UBRRH_VALUE is set to the lower byte of the UBRR_VALUE and UBRRL_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).

23.33.2 Macro Definition Documentation

23.33.2.1 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.

23.33.2.2 UBRR_VALUE #define UBRR_VALUE

Output macro from <util/setbaud.h>

Contains the calculated baud rate prescaler value for the UBRR register.

23.33.2.3 UBRRH_VALUE #define UBRRH_VALUE

Output macro from <util/setbaud.h>

Contains the upper byte of the calculated prescaler value (UBRR_VALUE).

23.33.2.4 UBRRL_VALUE `#define UBRRL_VALUE`

Output macro from `<util/setbaud.h>`

Contains the lower byte of the calculated prescaler value (UBRR_VALUE).

23.33.2.5 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.

23.34 `<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`

23.34.1 Detailed Description

```
#include <util/twi.h>
```

This header file contains bit mask definitions for use with the AVR TWI interface.

23.34.2 Macro Definition Documentation

23.34.2.1 TW_BUS_ERROR `#define TW_BUS_ERROR 0x00`

illegal start or stop condition

23.34.2.2 TW_MR_ARB_LOST `#define TW_MR_ARB_LOST 0x38`

arbitration lost in SLA+R or NACK

23.34.2.3 TW_MR_DATA_ACK `#define TW_MR_DATA_ACK 0x50`

data received, ACK returned

23.34.2.4 TW_MR_DATA_NACK `#define TW_MR_DATA_NACK 0x58`

data received, NACK returned

23.34.2.5 TW_MR_SLA_ACK `#define TW_MR_SLA_ACK 0x40`

SLA+R transmitted, ACK received

23.34.2.6 TW_MR_SLA_NACK `#define TW_MR_SLA_NACK 0x48`

SLA+R transmitted, NACK received

23.34.2.7 TW_MT_ARB_LOST `#define TW_MT_ARB_LOST 0x38`

arbitration lost in SLA+W or data

23.34.2.8 TW_MT_DATA_ACK `#define TW_MT_DATA_ACK 0x28`

data transmitted, ACK received

23.34.2.9 TW_MT_DATA_NACK `#define TW_MT_DATA_NACK 0x30`

data transmitted, NACK received

23.34.2.10 TW_MT_SLA_ACK `#define TW_MT_SLA_ACK 0x18`

SLA+W transmitted, ACK received

23.34.2.11 TW_MT_SLA_NACK `#define TW_MT_SLA_NACK 0x20`

SLA+W transmitted, NACK received

23.34.2.12 TW_NO_INFO `#define TW_NO_INFO 0xF8`

no state information available

23.34.2.13 TW_READ `#define TW_READ 1`

SLA+R address

23.34.2.14 TW_REP_START `#define TW_REP_START 0x10`

repeated start condition transmitted

23.34.2.15 TW_SR_ARB_LOST_GCALL_ACK `#define TW_SR_ARB_LOST_GCALL_ACK 0x78`

arbitration lost in SLA+RW, general call received, ACK returned

23.34.2.16 TW_SR_ARB_LOST_SLA_ACK `#define TW_SR_ARB_LOST_SLA_ACK 0x68`

arbitration lost in SLA+RW, SLA+W received, ACK returned

23.34.2.17 TW_SR_DATA_ACK `#define TW_SR_DATA_ACK 0x80`

data received, ACK returned

23.34.2.18 TW_SR_DATA_NACK `#define TW_SR_DATA_NACK 0x88`

data received, NACK returned

23.34.2.19 TW_SR_GCALL_ACK `#define TW_SR_GCALL_ACK 0x70`

general call received, ACK returned

23.34.2.20 TW_SR_GCALL_DATA_ACK `#define TW_SR_GCALL_DATA_ACK 0x90`

general call data received, ACK returned

23.34.2.21 TW_SR_GCALL_DATA_NACK `#define TW_SR_GCALL_DATA_NACK 0x98`

general call data received, NACK returned

23.34.2.22 TW_SR_SLA_ACK `#define TW_SR_SLA_ACK 0x60`

SLA+W received, ACK returned

23.34.2.23 TW_SR_STOP `#define TW_SR_STOP 0xA0`

stop or repeated start condition received while selected

23.34.2.24 TW_ST_ARB_LOST_SLA_ACK `#define TW_ST_ARB_LOST_SLA_ACK 0xB0`

arbitration lost in SLA+RW, SLA+R received, ACK returned

23.34.2.25 TW_ST_DATA_ACK `#define TW_ST_DATA_ACK 0xB8`

data transmitted, ACK received

23.34.2.26 TW_ST_DATA_NACK `#define TW_ST_DATA_NACK 0xC0`

data transmitted, NACK received

23.34.2.27 TW_ST_LAST_DATA `#define TW_ST_LAST_DATA 0xC8`

last data byte transmitted, ACK received

23.34.2.28 TW_ST_SLA_ACK `#define TW_ST_SLA_ACK 0xA8`

SLA+R received, ACK returned

23.34.2.29 TW_START `#define TW_START 0x08`

start condition transmitted

23.34.2.30 TW_STATUS `#define TW_STATUS (TWSR & TW_STATUS_MASK)`

TWSR, masked by TW_STATUS_MASK

23.34.2.31 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 ATmegs.

23.34.2.32 TW_WRITE `#define TW_WRITE 0`

SLA+W address

23.35 <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 `__inline__ void timer_enable_int` (unsigned char ints)
- `#define enable_external_int`(mask) (`__EICR` = mask)
- `#define INTERRUPT`(signature)
- `#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))

23.35.1 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.

23.35.2 Macro Definition Documentation

23.35.2.1 cbi

```
#define cbi(  
    port,  
    bit ) (port) &= ~(1 << (bit))
```

Deprecated

Clear `bit` in IO port `port`.

23.35.2.2 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.

23.35.2.3 inb

```
#define inb(  
    port ) (port)
```

Deprecated

Read a value from an IO port `port`.

23.35.2.4 inp

```
#define inp(  
    port ) (port)
```

Deprecated

Read a value from an IO port `port`.

23.35.2.5 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](#).

23.35.2.6 outb `#define outb(
 port,
 val) (port) = (val)`

Deprecated

Write *val* to IO port *port*.

23.35.2.7 outp `#define outp(
 val,
 port) (port) = (val)`

Deprecated

Write *val* to IO port *port*.

23.35.2.8 sbi `#define sbi(
 port,
 bit) (port) |= (1 << (bit))`

Deprecated

Set *bit* in IO port *port*.

23.35.3 Function Documentation

```
23.35.3.1 timer_enable_int()  static __inline__ void timer_enable_int (
    unsigned char ints )  [static]
```

Deprecated

This function modifies the `timsk` register. The value you pass via `ints` is device specific.

23.36 <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.

23.37 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 \(TWI\)](#)

23.37.1 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.

23.38 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.

23.38.1 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.

23.38.2 A code walkthrough

23.38.2.1 `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.

23.38.2.2 `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.

23.38.2.3 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.

23.38.3 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`.

23.39 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.

23.39.1 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.

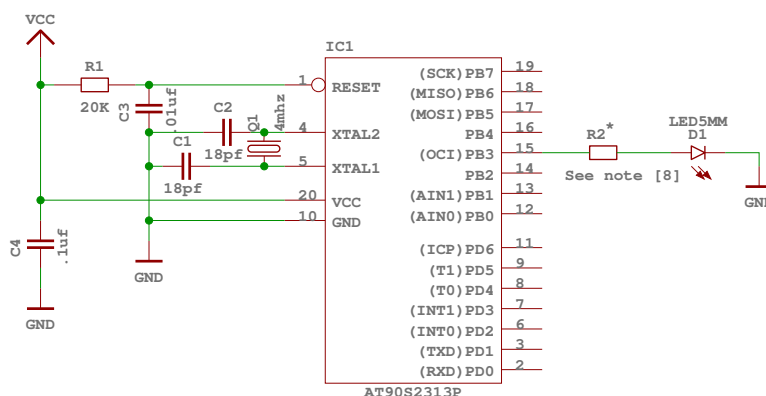


Figure 5 Schematic of circuit for demo project

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.

23.39.2 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: demo.c 1637 2008-03-17 21:49:41Z joerg_wunsch $
 */
#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
iointit (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.
    */
    #if defined(TIMER1_SETUP_HOOK)
        TIMER1_SETUP_HOOK();
    #endif
    /* Set PWM value to 0. */
    OCR = 0;
    /* Enable OC1 as output. */
    DDROC = _BV (OC1);
    /* Enable timer 1 overflow interrupt. */
    TIMSK = _BV (TOIE1);
    sei ();
}
int
main (void)
{
    iointit ();
    /* loop forever, the interrupts are doing the rest */
    for (;;)                /* Note [7] */
        sleep_mode();
    return (0);
}

```

23.39.3 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.

23.39.4 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          000000a4  00000000  00000000  00000094  2**1
    CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data          00000000  00800060  000000a4  00000138  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000003  00800060  00800060  00000138  2**0
    ALLOC
  3 .stab          00000354  00000000  00000000  00000138  2**2
    CONTENTS, READONLY, DEBUGGING
  4 .stabstr       000001ee  00000000  00000000  0000048c  2**0
    CONTENTS, READONLY, DEBUGGING
  5 .comment       00000012  00000000  00000000  0000067a  2**0
    CONTENTS, READONLY

Disassembly of section .text:

00000000 <__do_clear_bss>:
  0: 20 e0          ldi r18, 0x00 ; 0

00000002 <L0^AL2>:
  2: a0 e6          ldi r26, 0x60 ; 96

00000004 <L0^AL3>:
  4: b0 e0          ldi r27, 0x00 ; 0

00000006 <L0^AL4>:
  6: 01 c0          rjmp .+2          ; 0xa <L0^AL6>

00000008 <L0^AL5>:
  8: 1d 92          st X+, r1

0000000a <L0^AL6>:
 a: a3 36          cpi r26, 0x63 ; 99

0000000c <L0^AL7>:
 c: b2 07          cpc r27, r18

0000000e <L0^AL8>:
 e: e1 f7          brne .-8          ; 0x8 <L0^AL5>

00000010 <__vector_8>:
#include "iocompat.h" /* Note [1] */
```

```

enum { UP, DOWN };

ISR (TIMER1_OVF_vect) /* Note [2] */
{
    10: 1f 92      push r1
    12: 1f b6      in r1, 0x3f ; 63
    14: 1f 92      push r1
    16: 11 24      eor r1, r1
    18: 2f 93      push r18
    1a: 8f 93      push r24
    1c: 9f 93      push r25

0000001e <.LM1>:
    static uint16_t pwm; /* Note [3] */
    static uint8_t direction;

    switch (direction) /* Note [4] */
    1e: 20 91 62 00 lds r18, 0x0062 ; 0x800062 <direction.1>

00000022 <.LM2>:
    {
        case UP:
            if (++pwm == TIMER1_TOP)
    22: 80 91 60 00 lds r24, 0x0060 ; 0x800060 <pwm.0>
    26: 90 91 61 00 lds r25, 0x0061 ; 0x800061 <pwm.0+0x1>

0000002a <.LM3>:
        switch (direction) /* Note [4] */
    2a: 22 23      and r18, r18
    2c: a1 f0      breq .+40 ; 0x56 <.L2>
    2e: 21 30      cpi r18, 0x01 ; 1
    30: 49 f4      brne .+18 ; 0x44 <.L4>

00000032 <.LM4>:
        direction = DOWN;
        break;

        case DOWN:
            if (--pwm == 0)
    32: 01 97      sbiw r24, 0x01 ; 1

00000034 <.LM5>:
    34: 90 93 61 00 sts 0x0061, r25 ; 0x800061 <pwm.0+0x1>
    38: 80 93 60 00 sts 0x0060, r24 ; 0x800060 <pwm.0>
    3c: 00 97      sbiw r24, 0x00 ; 0
    3e: 11 f4      brne .+4 ; 0x44 <.L4>

00000040 <.LM6>:
        direction = UP;
    40: 10 92 62 00 sts 0x0062, r1 ; 0x800062 <direction.1>

00000044 <.L4>:
        break;
    }

    OCR = pwm; /* Note [5] */
    44: 9b bd      out 0x2b, r25 ; 43
    46: 8a bd      out 0x2a, r24 ; 42

00000048 <.LM8>:
}
    48: 9f 91      pop r25
    4a: 8f 91      pop r24
    4c: 2f 91      pop r18
    4e: 1f 90      pop r1
    50: 1f be      out 0x3f, r1 ; 63
    52: 1f 90      pop r1
    54: 18 95      reti

00000056 <.L2>:
    if (++pwm == TIMER1_TOP)
    56: 01 96      adiw r24, 0x01 ; 1

```

```

00000058 <.LM10>:
58: 90 93 61 00    sts 0x0061, r25 ; 0x800061 <pwm.0+0x1>
5c: 80 93 60 00    sts 0x0060, r24 ; 0x800060 <pwm.0>
60: 8f 3f          cpi r24, 0xFF ; 255
62: 23 e0          ldi r18, 0x03 ; 3
64: 92 07          cpc r25, r18
66: 71 f7          brne .-36 ; 0x44 <.L4>

00000068 <.LM11>:
        direction = DOWN;
68: 21 e0          ldi r18, 0x01 ; 1
6a: 20 93 62 00    sts 0x0062, r18 ; 0x800062 <direction.1>
6e: ea cf          rjmp .-44 ; 0x44 <.L4>

00000070 <ioint>:

void
ioint (void) /* Note [6] */
{
    /* Timer 1 is 10-bit PWM (8-bit PWM on some ATTinys). */
    TCCR1A = TIMER1_PWM_INIT;
70: 83 e8          ldi r24, 0x83 ; 131
72: 8f bd          out 0x2f, r24 ; 47

00000074 <.LM14>:
    * Start timer 1.
    *
    * NB: TCCR1A and TCCR1B could actually be the same register, so
    * take care to not clobber it.
    */
    TCCR1B |= TIMER1_CLOCKSOURCE;
74: 8e b5          in r24, 0x2e ; 46
76: 81 60          ori r24, 0x01 ; 1
78: 8e bd          out 0x2e, r24 ; 46

0000007a <.LM15>:
#ifdef(TIMER1_SETUP_HOOK)
    TIMER1_SETUP_HOOK();
#endif

    /* Set PWM value to 0. */
    OCR = 0;
7a: 1b bc          out 0x2b, r1 ; 43
7c: 1a bc          out 0x2a, r1 ; 42

0000007e <.LM16>:

    /* Enable OC1 as output. */
    DDROC = _BV (OC1);
7e: 82 e0          ldi r24, 0x02 ; 2
80: 87 bb          out 0x17, r24 ; 23

00000082 <.LM17>:

    /* Enable timer 1 overflow interrupt. */
    TIMSK = _BV (TOIE1);
82: 84 e0          ldi r24, 0x04 ; 4
84: 89 bf          out 0x39, r24 ; 57

00000086 <.LM18>:
    sei ();
86: 78 94          sei

00000088 <.LM19>:
}
88: 08 95          ret

0000008a <main>:

int
main (void)
{

```



```

    ioinit ();
8a: f2 df          rcall .-28          ; 0x70 <ioinit>

0000008c <.L8>:

    /* loop forever, the interrupts are doing the rest */

    for (;;) /* Note [7] */
        sleep_mode();
8c: 85 b7          in r24, 0x35 ; 53
8e: 80 68          ori r24, 0x80 ; 128
90: 85 bf          out 0x35, r24 ; 53
92: 88 95          sleep
94: 85 b7          in r24, 0x35 ; 53
96: 8f 77          andi r24, 0x7f ; 127
98: 85 bf          out 0x35, r24 ; 53
9a: f8 cf          rjmp .-16          ; 0x8c <.L8>

0000009c <_exit>:
9c: f8 94          cli
9e: 00 c0          rjmp .+0          ; 0xa0 <_exit>

000000a0 <_exit>:
a0: f8 94          cli

000000a2 <L0^AL3>:
a2: ff cf          rjmp .-2          ; 0xa2 <L0^AL3>

```

23.39.5 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          0x00000000          0xa4
*(.vectors)
*(.vectors)
*(.progmem.gcc*)
0x00000000          . = ALIGN (0x2)
0x00000000          __trampolines_start = .
*(.trampolines)
.trampolines  0x00000000          0x0 linker stubs
*(.trampolines*)
0x00000000          __trampolines_end = .
*libprintf_flt.a:*(.progmem.data)
*libc.a:*(.progmem.data)
*(.progmem.*)
0x00000000          . = ALIGN (0x2)
*(.lowtext)
*(.lowtext*)
0x00000000          __ctors_start = .

```

The `.text` segment (where program instructions are stored) starts at location 0x0.

```

*(.fini2)
*(.fini2)
*(.fini1)
*(.fini1)
*(.fini0)
.fini0          0x000000a0          0x4 /usr/pkg/lib/gcc/avr/10.3.0/avr4/libgcc.a(_exit.o)
*(.fini0)
*(.hightext)
*(.hightext*)
*(.progmemx.*)

```

```

0x000000a4          . = ALIGN (0x2)
*(.jumptables)
*(.jumptables*)
0x000000a4          _etext = .
.data              0x00800060      0x0 load address 0x000000a4
                  [!provide]      PROVIDE (__data_start = .)
*(.data)
.data              0x00800060      0x0 demo.o
.data              0x00800060      0x0 /tmp/work/cross/avr-libc/work/avr-libc-2.1.0/avr/lib/avr4/exit.o
.data              0x00800060      0x0 /usr/pkg/lib/gcc/avr/10.3.0/avr4/libgcc.a(_exit.o)
.data              0x00800060      0x0 /usr/pkg/lib/gcc/avr/10.3.0/avr4/libgcc.a(_clear_bss.o)
*(.data*)
*(.gnu.linkonce.d*)
*(.rodata)
*(.rodata*)
*(.gnu.linkonce.r*)
0x00800060          . = ALIGN (0x2)
0x00800060          _edata = .
                  [!provide]      PROVIDE (__data_end = .)
.bss               0x00800060      0x3
                  0x00800060      PROVIDE (__bss_start = .)
*(.bss)
.bss               0x00800060      0x3 demo.o
.bss               0x00800063      0x0 /tmp/work/cross/avr-libc/work/avr-libc-2.1.0/avr/lib/avr4/exit.o
.bss               0x00800063      0x0 /usr/pkg/lib/gcc/avr/10.3.0/avr4/libgcc.a(_exit.o)
.bss               0x00800063      0x0 /usr/pkg/lib/gcc/avr/10.3.0/avr4/libgcc.a(_clear_bss.o)
*(.bss*)
*(COMMON)
0x00800063          PROVIDE (__bss_end = .)
0x000000a4          __data_load_start = LOADADDR (.data)
0x000000a4          __data_load_end = (__data_load_start + SIZEOF (.data))
.noinit            0x00800063      0x0
                  [!provide]      PROVIDE (__noinit_start = .)
*(.noinit .noinit.* .gnu.linkonce.n.*)
                  [!provide]      PROVIDE (__noinit_end = .)
0x00800063          _end = .
                  [!provide]      PROVIDE (__heap_start = .)
.eeprom            0x00810000      0x0
*(.eeprom*)
0x00810000          __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.

23.39.6 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:

```
:1000000020E0A0E6B0E001C01D92A336B207E1F700
:100010001F921FB61F9211242F938F939F932091AD
:10002000620080916000909161002223A1F0213054
:1000300049F401979093610080936000009711F458
:10004000109262009BB8ABD9F918F912F911F904E
:100050001FBE1F90189501969093610080936000D9
:100060008F3F23E0920771F721E020936200EACFEF
:1000700083E88FBD8EB581608EBD1BBC1ABC82E04B
:1000800087BB84E089BF78940895F2DF85B78068E4
:1000900085BF889585B78F7785BFF8CFF89400C066
:0400A000F894FFCF02
: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.

23.39.7 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 $< $@

```

23.39.8 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`.

23.40 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.

23.40.1 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.

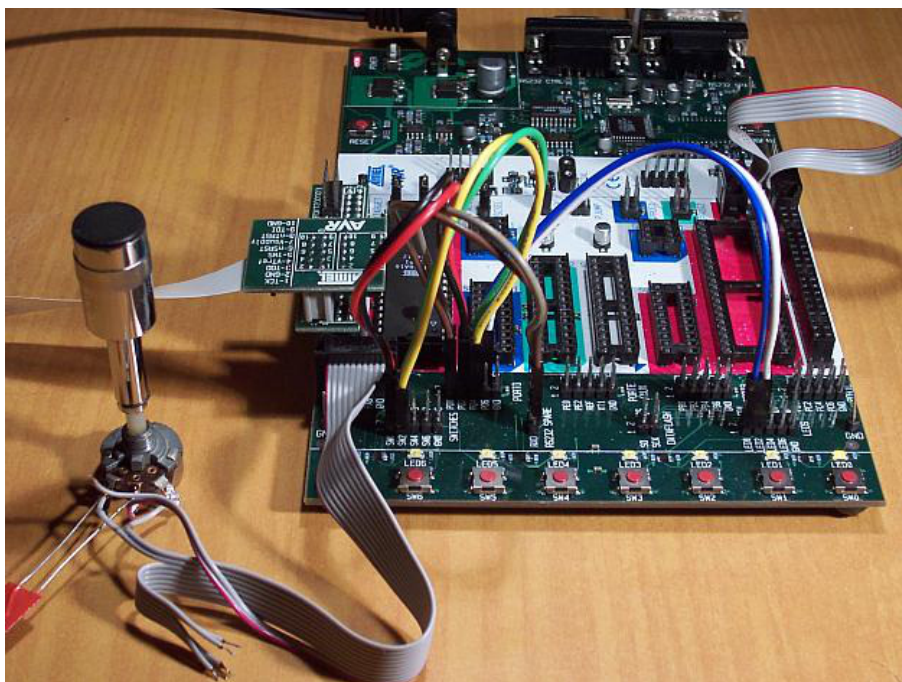


Figure 6 Setup of the STK500

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	

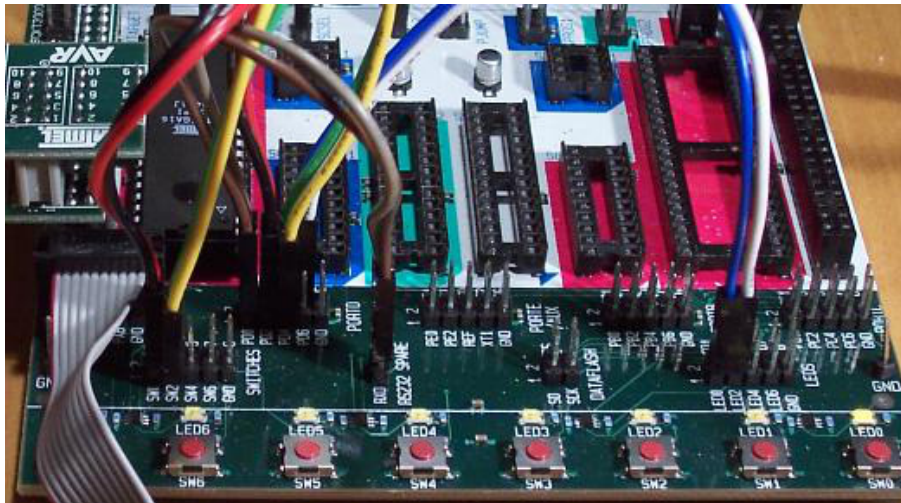


Figure 7 Wiring of the STK500

The following picture shows the alternate wiring where LED1 is connected but SW2 is not:

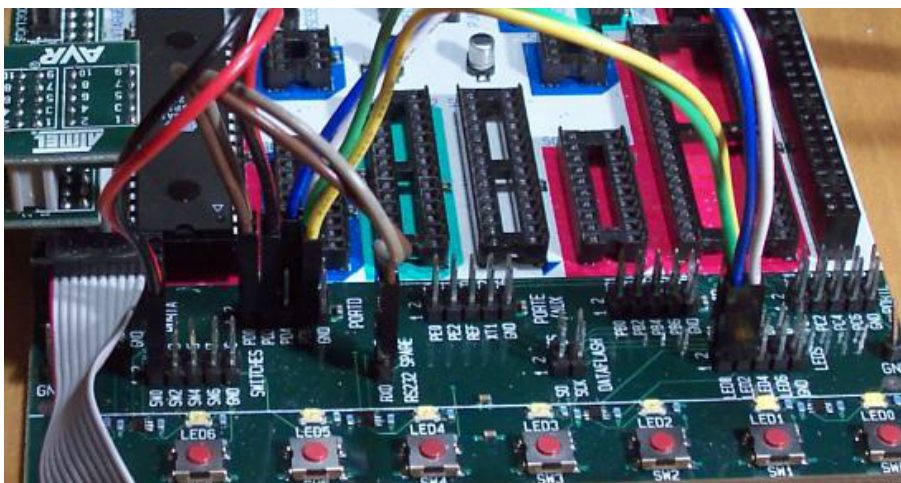


Figure 8 Wiring option #2 of the STK500

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.

23.40.2 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.

23.40.3 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.

23.40.3.1 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.

23.40.3.2 Part 2: Variable definitions The `intflags` 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.

23.40.3.3 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).

23.40.3.4 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 `eeeprom_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.

23.40.3.5 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()`.

23.40.4 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`.

23.41 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.

23.41.1 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

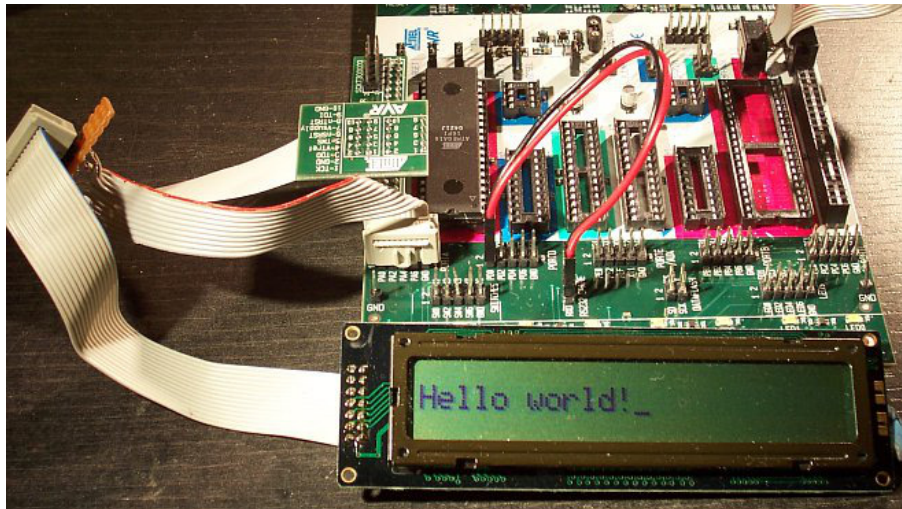


Figure 9 Wiring of the STK500

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.

23.41.2 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

23.41.3 A code walkthrough

23.41.3.1 `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 accomodate 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).

23.41.3.2 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.

23.41.3.3 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.

23.41.3.4 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).

23.41.3.5 lcd.h This function declares the public interface of the higher-level (character IO) LCD driver.

23.41.3.6 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()`.

23.41.3.7 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.

23.41.3.8 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.)

23.41.4 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`.

23.42 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

23.42.1 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.)

23.42.2 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.

23.42.3 The Source Code

The source code is installed under

```
$prefix/share/doc/avr-libc/examples/twittest/twittest.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 transfered within a single request, maintaining an internal address counter that is updated after each data byte transfered 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/~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/~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.

24 Data Structure Documentation

24.1 div_t Struct Reference

```
#include <stdlib.h>
```

Data Fields

- int [quot](#)
- int [rem](#)

24.1.1 Detailed Description

Result type for function [div\(\)](#).

24.1.2 Field Documentation

24.1.2.1 `quot` `int div_t::quot`

The Quotient.

24.1.2.2 `rem` `int div_t::rem`

The Remainder.

The documentation for this struct was generated from the following file:

- [stdlib.h](#)

24.2 `ldiv_t` Struct Reference

```
#include <stdlib.h>
```

Data Fields

- long [quot](#)
- long [rem](#)

24.2.1 Detailed Description

Result type for function [ldiv\(\)](#).

24.2.2 Field Documentation

24.2.2.1 `quot` `long ldiv_t::quot`

The Quotient.

24.2.2.2 `rem` `long ldiv_t::rem`

The Remainder.

The documentation for this struct was generated from the following file:

- [stdlib.h](#)

24.3 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](#)

24.3.1 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.

24.3.2 Field Documentation

24.3.2.1 tm_hour [int8_t](#) tm::tm_hour

hours since midnight - [0 to 23]

24.3.2.2 tm_isdst [int16_t](#) tm::tm_isdst

Daylight Saving Time flag

24.3.2.3 tm_mday [int8_t](#) tm::tm_mday

day of the month - [1 to 31]

24.3.2.4 tm_min [int8_t](#) tm::tm_min

minutes after the hour - [0 to 59]

24.3.2.5 `tm_mon` `int8_t` `tm::tm_mon`

months since January - [0 to 11]

24.3.2.6 `tm_sec` `int8_t` `tm::tm_sec`

seconds after the minute - [0 to 59]

24.3.2.7 `tm_wday` `int8_t` `tm::tm_wday`

days since Sunday - [0 to 6]

24.3.2.8 `tm_yday` `int16_t` `tm::tm_yday`

days since January 1 - [0 to 365]

24.3.2.9 `tm_year` `int16_t` `tm::tm_year`

years since 1900

The documentation for this struct was generated from the following file:

- [time.h](#)

24.4 week_date Struct Reference

```
#include <time.h>
```

Data Fields

- int [year](#)
- int [week](#)
- int [day](#)

24.4.1 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 for more information.

24.4.2 Field Documentation

24.4.2.1 day `int week_date::day`

day within week

24.4.2.2 week `int week_date::week`

week number (#1 is where first Thursday is in)

24.4.2.3 year `int week_date::year`

year number (Gregorian calendar)

The documentation for this struct was generated from the following file:

- [time.h](#)

25 File Documentation

25.1 project.h

```

1 /*
2 * -----
3 * "THE BEER-WARE LICENSE" (Revision 42):
4 * Joerg Wunsch wrote this file.  As long as you retain this notice you
5 * can do whatever you want with this stuff.  If we meet some day, and you think
6 * this stuff is worth it, you can buy me a beer in return.          Joerg Wunsch
7 * -----
8 *
9 * Demo combining C and assembly source files.
10 *
11 * $Id:  project.h 1124 2006-08-29 19:45:06Z joerg_wunsch $
12 */
13
14 /*
15 * Global register variables.
16 */
17 #ifdef __ASSEMBLER__
18
19 #  define sreg_save  r2
20 #  define flags      rl6
21 #  define counter_hi  r4
22
23 #else /* !ASSEMBLER */
24
25 #include <stdint.h>
26
27 register uint8_t sreg_save asm("r2");
28 register uint8_t flags    asm("r16");
29 register uint8_t counter_hi asm("r4");
30
31 #endif /* ASSEMBLER */

```

25.2 iocompat.h

```

1 /*
2 * -----
3 * "THE BEER-WARE LICENSE" (Revision 42):
4 * <joerg@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
5 * can do whatever you want with this stuff.  If we meet some day, and you think
6 * this stuff is worth it, you can buy me a beer in return.          Joerg Wunsch
7 * -----
8 *
9 * IO feature compatibility definitions for various AVRs.
10 *
11 * $Id:  iocompat.h 2384 2013-05-03 12:38:46Z joerg_wunsch $
12 */
13

```

```

14 #if !defined(IOCMPAT_H)
15 #define IOCMPAT_H 1
16
17 /*
18 * Device-specific adjustments:
19 *
20 * Supply definitions for the location of the OCR1[A] port/pin, the
21 * name of the OCR register controlling the PWM, and adjust interrupt
22 * vector names that differ from the one used in demo.c
23 * [TIMER1_OVF_vect].
24 */
25 #if defined(__AVR_AT90S2313__)
26 #   define OC1 PB3
27 #   define OCR OCR1
28 #   define DDROC DDRB
29 #   define TIMER1_OVF_vect TIMER1_OVF1_vect
30 #elif defined(__AVR_AT90S2333__) || defined(__AVR_AT90S4433__)
31 #   define OC1 PB1
32 #   define DDROC DDRB
33 #   define OCR OCR1
34 #elif defined(__AVR_AT90S4414__) || defined(__AVR_AT90S8515__) || \
35 defined(__AVR_AT90S4434__) || defined(__AVR_AT90S8535__) || \
36 defined(__AVR_ATmega163__) || defined(__AVR_ATmega8515__) || \
37 defined(__AVR_ATmega8535__) || \
38 defined(__AVR_ATmega164P__) || defined(__AVR_ATmega324P__) || \
39 defined(__AVR_ATmega644__) || defined(__AVR_ATmega644P__) || \
40 defined(__AVR_ATmega1284P__)
41 #   define OC1 PD5
42 #   define DDROC DDRD
43 #   define OCR OCR1A
44 #   if !defined(TIMSK)           /* new ATmegs */
45 #       define TIMSK TIMSK1
46 #   endif
47 #elif defined(__AVR_ATmega8__) || defined(__AVR_ATmega48__) || \
48 defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
49 #   define OC1 PB1
50 #   define DDROC DDRB
51 #   define OCR OCR1A
52 #   if !defined(TIMSK)           /* ATmega48/88/168 */
53 #       define TIMSK TIMSK1
54 #   endif /* !defined(TIMSK) */
55 #elif defined(__AVR_ATtiny2313__)
56 #   define OC1 PB3
57 #   define OCR OCR1A
58 #   define DDROC DDRB
59 #elif defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || \
60 defined(__AVR_ATtiny84__)
61 #   define OC1 PA6
62 #   define DDROC DDRA
63 #   if !defined(OCR1A)
64 #       /* work around misspelled name in avr-libc 1.4.[0..1] */
65 #       define OCR OCR1
66 #   else
67 #       define OCR OCR1A
68 #   endif
69 #   define TIMSK TIMSK1
70 #   define TIMER1_OVF_vect TIM1_OVF_vect /* XML and datasheet mismatch */
71 #elif defined(__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || \
72 defined(__AVR_ATtiny85__)
73 /* Timer 1 is only an 8-bit timer on these devices. */
74 #   define OC1 PB1
75 #   define DDROC DDRB
76 #   define OCR OCR1A
77 #   define TCCR1A TCCR1
78 #   define TCCR1B TCCR1
79 #   define TIMER1_OVF_vect TIM1_OVF_vect
80 #   define TIMER1_TOP 255 /* only 8-bit PWM possible */
81 #   define TIMER1_PWM_INIT _BV(PWM1A) | _BV(COM1A1)
82 #   define TIMER1_CLOCKSOURCE _BV(CS12) /* use 1/8 prescaler */
83 #elif defined(__AVR_ATtiny26__)
84 /* Rather close to ATtinyX5 but different enough for its own section. */
85 #   define OC1 PB1
86 #   define DDROC DDRB
87 #   define OCR OCR1A
88 #   define TIMER1_OVF_vect TIMER1_OVF1_vect
89 #   define TIMER1_TOP 255 /* only 8-bit PWM possible */
90 #   define TIMER1_PWM_INIT _BV(PWM1A) | _BV(COM1A1)
91 #   define TIMER1_CLOCKSOURCE _BV(CS12) /* use 1/8 prescaler */
92 /*
93 * Without setting OCR1C to TOP, the ATtiny26 does not trigger an
94 * overflow interrupt in PWM mode.
95 */
96 #   define TIMER1_SETUP_HOOK() OCR1C = 255
97 #elif defined(__AVR_ATtiny261__) || defined(__AVR_ATtiny461__) || \
98 defined(__AVR_ATtiny861__)
99 #   define OC1 PB1
100 #   define DDROC DDRB

```



```

101 # define OCR OCR1A
102 # define TIMER1_PWM_INIT _BV(WGM10) | _BV(PWM1A) | _BV(COM1A1)
103 /*
104 * While timer 1 could be operated in 10-bit mode on these devices,
105 * the handling of the 10-bit IO registers is more complicated than
106 * that of the 16-bit registers of other AVR devices (no combined
107 * 16-bit IO operations possible), so we restrict this demo to 8-bit
108 * mode which is pretty standard.
109 */
110 # define TIMER1_TOP 255
111 # define TIMER1_CLOCKSOURCE _BV(CS12) /* use 1/8 prescaler */
112 #elif defined(__AVR_ATmega32__) || defined(__AVR_ATmega16__)
113 # define OC1 PD5
114 # define DDROC DDRD
115 # define OCR OCR1A
116 #elif defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__) || \
117 defined(__AVR_ATmega165__) || defined(__AVR_ATmega169__) || \
118 defined(__AVR_ATmega325__) || defined(__AVR_ATmega3250__) || \
119 defined(__AVR_ATmega645__) || defined(__AVR_ATmega6450__) || \
120 defined(__AVR_ATmega329__) || defined(__AVR_ATmega3290__) || \
121 defined(__AVR_ATmega649__) || defined(__AVR_ATmega6490__) || \
122 defined(__AVR_ATmega640__) || \
123 defined(__AVR_ATmega1280__) || defined(__AVR_ATmega1281__) || \
124 defined(__AVR_ATmega2560__) || defined(__AVR_ATmega2561__) || \
125 defined(__AVR_ATmega32U4__)
126 # define OC1 PB5
127 # define DDROC DDRB
128 # define OCR OCR1A
129 # if !defined(PB5) /* work around missing bit definition */
130 # define PB5 5
131 # endif
132 # if !defined(TIMSK) /* new ATmegs */
133 # define TIMSK TIMSK1
134 # endif
135 #else
136 # error "Don't know what kind of MCU you are compiling for"
137 #endif
138
139 /*
140 * Map register names for older AVRs here.
141 */
142 #if !defined(COM1A1)
143 # define COM1A1 COM11
144 #endif
145
146 #if !defined(WGM10)
147 # define WGM10 PWM10
148 # define WGM11 PWM11
149 #endif
150
151 /*
152 * Provide defaults for device-specific macros unless overridden
153 * above.
154 */
155 #if !defined(TIMER1_TOP)
156 # define TIMER1_TOP 1023 /* 10-bit PWM */
157 #endif
158
159 #if !defined(TIMER1_PWM_INIT)
160 # define TIMER1_PWM_INIT _BV(WGM10) | _BV(WGM11) | _BV(COM1A1)
161 #endif
162
163 #if !defined(TIMER1_CLOCKSOURCE)
164 # define TIMER1_CLOCKSOURCE _BV(CS10) /* full clock */
165 #endif
166
167 #endif /* !defined(IOCMPAT_H) */

```

25.3 defines.h

```

1 /*
2 * -----
3 * "THE BEER-WARE LICENSE" (Revision 42):
4 * <joerg@FreeBSD.ORG> wrote this file. As long as you retain this notice you
5 * can do whatever you want with this stuff. If we meet some day, and you think
6 * this stuff is worth it, you can buy me a beer in return. Joerg Wunsch
7 * -----
8 *
9 * General stdiodemo defines
10 *
11 * $Id: defines.h 2186 2010-09-22 10:25:15Z aboyapati $
12 */
13
14 /* CPU frequency */

```

```

15 #define F_CPU 1000000UL
16
17 /* UART baud rate */
18 #define UART_BAUD 9600
19
20 /* HD44780 LCD port connections */
21 #define HD44780_RS A, 6
22 #define HD44780_RW A, 4
23 #define HD44780_E A, 5
24 /* The data bits have to be not only in ascending order but also consecutive. */
25 #define HD44780_D4 A, 0
26
27 /* Whether to read the busy flag, or fall back to
28 worst-time delays. */
29 #define USE_BUSY_BIT 1

```

25.4 hd44780.h

```

1 /*
2 * -----
3 * "THE BEER-WARE LICENSE" (Revision 42):
4 * <joerg@FreeBSD.ORG> wrote this file. As long as you retain this notice you
5 * can do whatever you want with this stuff. If we meet some day, and you think
6 * this stuff is worth it, you can buy me a beer in return. Joerg Wunsch
7 * -----
8 *
9 * HD44780 LCD display driver
10 *
11 * $Id: hd44780.h 2002 2009-06-25 20:21:16Z joerg_wunsch $
12 */
13
14 /*
15 * Send byte b to the LCD. rs is the RS signal (register select), 0
16 * selects instruction register, 1 selects the data register.
17 */
18 void hd44780_outbyte(uint8_t b, uint8_t rs);
19
20 /*
21 * Read one byte from the LCD controller. rs is the RS signal, 0
22 * selects busy flag (bit 7) and address counter, 1 selects the data
23 * register.
24 */
25 uint8_t hd44780_inbyte(uint8_t rs);
26
27 /*
28 * Wait for the busy flag to clear.
29 */
30 void hd44780_wait_ready(bool islong);
31
32 /*
33 * Initialize the LCD controller hardware.
34 */
35 void hd44780_init(void);
36
37 /*
38 * Prepare the LCD controller pins for powerdown.
39 */
40 void hd44780_powerdown(void);
41
42
43 /* Send a command to the LCD controller. */
44 #define hd44780_outcmd(n) hd44780_outbyte((n), 0)
45
46 /* Send a data byte to the LCD controller. */
47 #define hd44780_outdata(n) hd44780_outbyte((n), 1)
48
49 /* Read the address counter and busy flag from the LCD. */
50 #define hd44780_incmd() hd44780_inbyte(0)
51
52 /* Read the current data byte from the LCD. */
53 #define hd44780_indata() hd44780_inbyte(1)
54
55
56 /* Clear LCD display command. */
57 #define HD44780_CLR \
58 0x01
59
60 /* Home cursor command. */
61 #define HD44780_HOME \
62 0x02
63
64 /*
65 * Select the entry mode. inc determines whether the address counter
66 * auto-increments, shift selects an automatic display shift.

```

```

67 */
68 #define HD44780_ENTMODE(inc, shift) \
69 (0x04 | ((inc)? 0x02: 0) | ((shift)? 1: 0))
70
71 /*
72 * Selects disp[lay] on/off, cursor on/off, cursor blink[ing]
73 * on/off.
74 */
75 #define HD44780_DISPCTL(disp, cursor, blink) \
76 (0x08 | ((disp)? 0x04: 0) | ((cursor)? 0x02: 0) | ((blink)? 1: 0))
77
78 /*
79 * With shift = 1, shift display right or left.
80 * With shift = 0, move cursor right or left.
81 */
82 #define HD44780_SHIFT(shift, right) \
83 (0x10 | ((shift)? 0x08: 0) | ((right)? 0x04: 0))
84
85 /*
86 * Function set. if8bit selects an 8-bit data path, twoline arranges
87 * for a two-line display, font5x10 selects the 5x10 dot font (5x8
88 * dots if clear).
89 */
90 #define HD44780_FNSET(if8bit, twoline, font5x10) \
91 (0x20 | ((if8bit)? 0x10: 0) | ((twoline)? 0x08: 0) | \
92 ((font5x10)? 0x04: 0))
93
94 /*
95 * Set the next character generator address to addr.
96 */
97 #define HD44780_CGADDR(addr) \
98 (0x40 | ((addr) & 0x3f))
99
100 /*
101 * Set the next display address to addr.
102 */
103 #define HD44780_DDADDR(addr) \
104 (0x80 | ((addr) & 0x7f))
105

```

25.5 lcd.h

```

1 /*
2 * -----
3 * "THE BEER-WARE LICENSE" (Revision 42):
4 * <joerg@FreeBSD.ORG> wrote this file. As long as you retain this notice you
5 * can do whatever you want with this stuff. If we meet some day, and you think
6 * this stuff is worth it, you can buy me a beer in return. Joerg Wunsch
7 * -----
8 *
9 * Stdio demo, upper layer of LCD driver.
10 *
11 * $Id: lcd.h 1008 2005-12-28 21:38:59Z joerg_wunsch $
12 */
13
14 /*
15 * Initialize LCD controller. Performs a software reset.
16 */
17 void lcd_init(void);
18
19 /*
20 * Send one character to the LCD.
21 */
22 int lcd_putchar(char c, FILE *stream);

```

25.6 uart.h

```

1 /*
2 * -----
3 * "THE BEER-WARE LICENSE" (Revision 42):
4 * <joerg@FreeBSD.ORG> wrote this file. As long as you retain this notice you
5 * can do whatever you want with this stuff. If we meet some day, and you think
6 * this stuff is worth it, you can buy me a beer in return. Joerg Wunsch
7 * -----
8 *
9 * Stdio demo, UART declarations
10 *
11 * $Id: uart.h 1008 2005-12-28 21:38:59Z joerg_wunsch $
12 */
13
14 /*

```

```

15 * Perform UART startup initialization.
16 */
17 void    uart_init(void);
18
19 /*
20 * Send one character to the UART.
21 */
22 int uart_putchar(char c, FILE *stream);
23
24 /*
25 * Size of internal line buffer used by uart_getchar().
26 */
27 #define RX_BUFSIZE 80
28
29 /*
30 * Receive one character from the UART. The actual reception is
31 * line-buffered, and one character is returned from the buffer at
32 * each invocation.
33 */
34 int uart_getchar(FILE *stream);

```

25.7 alloca.h

```

1 /* Copyright (c) 2007, Dmitry Xmelkov
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9 * Redistributions in binary form must reproduce the above copyright
10 notice, this list of conditions and the following disclaimer in
11 the documentation and/or other materials provided with the
12 distribution.
13 * Neither the name of the copyright holders nor the names of
14 contributors may be used to endorse or promote products derived
15 from this software without specific prior written permission.
16
17 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
21 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
27 POSSIBILITY OF SUCH DAMAGE. */
28
29 /* $Id:  alloca.h 1508 2007-12-18 13:36:50Z dmix $ */
30
31 #ifndef _ALLOCA_H
32 #define _ALLOCA_H 1
33
34 #include <stddef.h>
35
36 /** \defgroup alloca <alloca.h>: Allocate space in the stack */
37
38 /** \ingroup alloca
39 \brief Allocate \a __size bytes of space in the stack frame of the caller.
40
41 This temporary space is automatically freed when the function that
42 called alloca() returns to its caller. Avr-libc defines the alloca() as
43 a macro, which is translated into the inlined \c __builtin_alloca()
44 function. The fact that the code is inlined, means that it is impossible
45 to take the address of this function, or to change its behaviour by
46 linking with a different library.
47
48 \return alloca() returns a pointer to the beginning of the allocated
49 space. If the allocation causes stack overflow, program behaviour is
50 undefined.
51
52 \warning Avoid use alloca() inside the list of arguments of a function
53 call.
54 */
55 extern void *alloca (size_t __size);
56
57 #define alloca(size)    __builtin_alloca (size)
58
59 #endif /* alloca.h */

```

25.8 assert.h File Reference

Macros

- #define [assert](#)(expression)

25.8.1 Macro Definition Documentation

25.8.1.1 assert #define assert(
 expression)

Parameters

<i>expression</i>	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).

25.9 assert.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2005,2007 Joerg Wunsch
2 All rights reserved.
3
4 Portions of documentation Copyright (c) 1991, 1993
5 The Regents of the University of California.
6
7 All rights reserved.
8
9 Redistribution and use in source and binary forms, with or without
10 modification, are permitted provided that the following conditions are met:
11
12 * Redistributions of source code must retain the above copyright
13 notice, this list of conditions and the following disclaimer.
14
15 * Redistributions in binary form must reproduce the above copyright
16 notice, this list of conditions and the following disclaimer in
17 the documentation and/or other materials provided with the
18 distribution.
19
20 * Neither the name of the copyright holders nor the names of
21 contributors may be used to endorse or promote products derived
22 from this software without specific prior written permission.
23
24 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
25 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
26 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
27 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
28 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
29 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
30 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
31 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
32 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
33 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
34 POSSIBILITY OF SUCH DAMAGE.
35
```

```

36 $Id:  assert.h 2526 2016-10-19 10:36:48Z pitchumani $
37 */
38
39 /** \file */
40 /** \defgroup avr_assert <assert.h>:  Diagnostics
41 \code #include <assert.h> \endcode
42
43 This header file defines a debugging aid.
44
45 As there is no standard error output stream available for many
46 applications using this library, the generation of a printable
47 error message is not enabled by default.  These messages will
48 only be generated if the application defines the macro
49
50 \code __ASSERT_USE_STDERR \endcode
51
52 before including the \c <assert.h> header file.  By default,
53 only abort() will be called to halt the application.
54 */
55
56 /*{*/
57
58 /*
59 * The ability to include this file (with or without NDEBUG) is a
60 * feature.
61 */
62
63 #undef assert
64
65 #include <stdlib.h>
66
67 #if defined(__DOXYGEN__)
68 /**
69 * \def assert
70 * \param expression Expression to test for.
71 *
72 * The assert() macro tests the given expression and if it is false,
73 * the calling process is terminated.  A diagnostic message is written
74 * to stderr and the function abort() is called, effectively
75 * terminating the program.
76 *
77 * If expression is true, the assert() macro does nothing.
78 *
79 * The assert() macro may be removed at compile time by defining
80 * NDEBUG as a macro (e.g., by using the compiler option -DNDEBUG).
81 */
82 #  define assert(expression)
83
84 #else /* !DOXYGEN */
85
86 #  if defined(NDEBUG)
87 #    define assert(e)  ((void)0)
88 #  else /* !NDEBUG */
89 #    if defined(__ASSERT_USE_STDERR)
90 #      define assert(e) ((e) ? (void)0 : \
91 __assert(__func__, __FILE__, __LINE__, #e))
92 #    else /* !__ASSERT_USE_STDERR */
93 #      define assert(e) ((e) ? (void)0 : abort())
94 #    endif /* __ASSERT_USE_STDERR */
95 #  endif /* NDEBUG */
96 #endif /* DOXYGEN */
97
98 #if (defined __STDC_VERSION__ && __STDC_VERSION__ >= 201112L) || \
99 ((_GNUC_ > 4 || (_GNUC_ == 4 && _GNUC_MINOR_ >= 6)) && !defined __cplusplus)
100 #  undef static_assert
101 #  define static_assert _Static_assert
102 #endif
103
104 #ifdef __cplusplus
105 extern "C" {
106 #endif
107
108 #if !defined(__DOXYGEN__)
109
110 extern void __assert(const char *__func, const char *__file,
111 int __lineno, const char *__sexp);
112
113 #endif /* not __DOXYGEN__ */
114
115 #ifdef __cplusplus
116 }
117 #endif
118
119 /*{*/
120 /* EOF */

```

25.10 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)`

25.11 boot.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002,2003,2004,2005,2006,2007,2008,2009 Eric B. Weddington
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9 * Redistributions in binary form must reproduce the above copyright
10 notice, this list of conditions and the following disclaimer in
11 the documentation and/or other materials provided with the
12 distribution.
13 * Neither the name of the copyright holders nor the names of
14 contributors may be used to endorse or promote products derived
15 from this software without specific prior written permission.
16
17 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
21 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
27 POSSIBILITY OF SUCH DAMAGE. */
28
29 /* $Id: boot.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
30
31 #ifndef _AVR_BOOT_H_
32 #define _AVR_BOOT_H_ 1
33
34 /** \file */
35 /** \defgroup avr_boot <avr/boot.h> Bootloader Support Utilities

```

```

36 \code
37 #include <avr/io.h>
38 #include <avr/boot.h>
39 \endcode
40
41 The macros in this module provide a C language interface to the
42 bootloader support functionality of certain AVR processors. These
43 macros are designed to work with all sizes of flash memory.
44
45 Global interrupts are not automatically disabled for these macros. It
46 is left up to the programmer to do this. See the code example below.
47 Also see the processor datasheet for caveats on having global interrupts
48 enabled during writing of the Flash.
49
50 \note Not all AVR processors provide bootloader support. See your
51 processor datasheet to see if it provides bootloader support.
52
53 \todo From email with Marek: On smaller devices (all except ATmega64/128),
54 __SPM_REG is in the I/O space, accessible with the shorter "in" and "out"
55 instructions - since the boot loader has a limited size, this could be an
56 important optimization.
57
58 \par API Usage Example
59 The following code shows typical usage of the boot API.
60
61 \code
62 #include <inttypes.h>
63 #include <avr/interrupt.h>
64 #include <avr/pgmspace.h>
65
66 void boot_program_page (uint32_t page, uint8_t *buf)
67 {
68     uint16_t i;
69     uint8_t sreg;
70
71     // Disable interrupts.
72
73     sreg = SREG;
74     cli();
75
76     eeprom_busy_wait ();
77
78     boot_page_erase (page);
79     boot_spm_busy_wait ();      // Wait until the memory is erased.
80
81     for (i=0; i<SPM_PAGESIZE; i+=2)
82     {
83         // Set up little-endian word.
84
85         uint16_t w = *buf++;
86         w += (*buf++) << 8;
87
88         boot_page_fill (page + i, w);
89     }
90
91     boot_page_write (page);      // Store buffer in flash page.
92     boot_spm_busy_wait();        // Wait until the memory is written.
93
94     // Reenable RWW-section again. We need this if we want to jump back
95     // to the application after bootloading.
96
97     boot_rww_enable ();
98
99     // Re-enable interrupts (if they were ever enabled).
100
101     SREG = sreg;
102 } \endcode */
103
104 #include <avr/eeprom.h>
105 #include <avr/io.h>
106 #include <inttypes.h>
107 #include <limits.h>
108
109 /* Check for SPM Control Register in processor. */
110 #if defined (SPMCSR)
111 #   define __SPM_REG    SPMCSR
112 #else
113 #   if defined (SPMCR)
114 #       define __SPM_REG    SPMCR
115 #   else
116 #       error AVR processor does not provide bootloader support!
117 #   endif
118 #endif
119
120
121 /* Check for SPM Enable bit. */
122 #if defined (SPMEN)

```



```

123 # define __SPM_ENABLE SPMEN
124 #elif defined(SELFPRGEN)
125 # define __SPM_ENABLE SELFPRGEN
126 #else
127 # error Cannot find SPM Enable bit definition!
128 #endif
129
130 /** \ingroup avr_boot
131 \def BOOTLOADER_SECTION
132
133 Used to declare a function or variable to be placed into a
134 new section called .bootloader. This section and its contents
135 can then be relocated to any address (such as the bootloader
136 NRWW area) at link-time. */
137
138 #define BOOTLOADER_SECTION __attribute__((section(".bootloader")))
139
140 #ifndef __DOXYGEN__
141 /* Create common bit definitions. */
142 #ifdef ASB
143 #define __COMMON_ASB ASB
144 #else
145 #define __COMMON_ASB RWWSB
146 #endif
147
148 #ifdef ASRE
149 #define __COMMON_ASRE ASRE
150 #else
151 #define __COMMON_ASRE RWWSRE
152 #endif
153
154 /* Define the bit positions of the Boot Lock Bits. */
155
156 #define BLB12 5
157 #define BLB11 4
158 #define BLB02 3
159 #define BLB01 2
160 #endif /* __DOXYGEN__ */
161
162 /** \ingroup avr_boot
163 \def boot_spm_interrupt_enable()
164 Enable the SPM interrupt. */
165
166 #define boot_spm_interrupt_enable() (__SPM_REG |= (uint8_t)_BV(SPMIE))
167
168 /** \ingroup avr_boot
169 \def boot_spm_interrupt_disable()
170 Disable the SPM interrupt. */
171
172 #define boot_spm_interrupt_disable() (__SPM_REG &= (uint8_t)~_BV(SPMIE))
173
174 /** \ingroup avr_boot
175 \def boot_is_spm_interrupt()
176 Check if the SPM interrupt is enabled. */
177
178 #define boot_is_spm_interrupt() (__SPM_REG & (uint8_t)_BV(SPMIE))
179
180 /** \ingroup avr_boot
181 \def boot_rww_busy()
182 Check if the RWW section is busy. */
183
184 #define boot_rww_busy() (__SPM_REG & (uint8_t)_BV(__COMMON_ASB))
185
186 /** \ingroup avr_boot
187 \def boot_spm_busy()
188 Check if the SPM instruction is busy. */
189
190 #define boot_spm_busy() (__SPM_REG & (uint8_t)_BV(__SPM_ENABLE))
191
192 /** \ingroup avr_boot
193 \def boot_spm_busy_wait()
194 Wait while the SPM instruction is busy. */
195
196 #define boot_spm_busy_wait() do{}while(boot_spm_busy())
197
198 #ifndef __DOXYGEN__
199 #define __BOOT_PAGE_ERASE (_BV(__SPM_ENABLE) | _BV(PGERS))
200 #define __BOOT_PAGE_WRITE (_BV(__SPM_ENABLE) | _BV(PGWRT))
201 #define __BOOT_PAGE_FILL _BV(__SPM_ENABLE)
202 #define __BOOT_RWW_ENABLE (_BV(__SPM_ENABLE) | _BV(__COMMON_ASRE))
203 #if defined(BLBSET)
204 #define __BOOT_LOCK_BITS_SET (_BV(__SPM_ENABLE) | _BV(BLBSET))
205 #elif defined(RFLB) /* Some devices have RFLB defined instead of BLBSET. */
206 #define __BOOT_LOCK_BITS_SET (_BV(__SPM_ENABLE) | _BV(RFLB))
207 #endif
208
209 #define __boot_page_fill_normal(address, data) \

```

```

210 (__extension__({
211     __asm__ __volatile__
212     (
213         "movw r0, %3\n\t"
214         "sts %0, %1\n\t"
215         "spm\n\t"
216         "clr r1\n\t"
217         :
218         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
219         "r" ((uint8_t)(__BOOT_PAGE_FILL)),
220         "z" ((uint16_t)(address)),
221         "r" ((uint16_t)(data))
222         : "r0"
223     );
224     })
225
226 #define __boot_page_fill_alternate(address, data) \
227 (__extension__({
228     __asm__ __volatile__
229     (
230         "movw r0, %3\n\t"
231         "sts %0, %1\n\t"
232         "spm\n\t"
233         ".word 0xffff\n\t"
234         "nop\n\t"
235         "clr r1\n\t"
236         :
237         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
238         "r" ((uint8_t)(__BOOT_PAGE_FILL)),
239         "z" ((uint16_t)(address)),
240         "r" ((uint16_t)(data))
241         : "r0"
242     );
243     })
244
245 #define __boot_page_fill_extended(address, data) \
246 (__extension__({
247     __asm__ __volatile__
248     (
249         "movw r0, %4\n\t"
250         "movw r30, %A3\n\t"
251         "sts %1, %C3\n\t"
252         "sts %0, %2\n\t"
253         "spm\n\t"
254         "clr r1\n\t"
255         :
256         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
257         "i" (_SFR_MEM_ADDR(RAMPZ)),
258         "r" ((uint8_t)(__BOOT_PAGE_FILL)),
259         "r" ((uint32_t)(address)),
260         "r" ((uint16_t)(data))
261         : "r0", "r30", "r31"
262     );
263     })
264
265 #define __boot_page_erase_normal(address) \
266 (__extension__({
267     __asm__ __volatile__
268     (
269         "sts %0, %1\n\t"
270         "spm\n\t"
271         :
272         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
273         "r" ((uint8_t)(__BOOT_PAGE_ERASE)),
274         "z" ((uint16_t)(address))
275     );
276     })
277
278 #define __boot_page_erase_alternate(address) \
279 (__extension__({
280     __asm__ __volatile__
281     (
282         "sts %0, %1\n\t"
283         "spm\n\t"
284         ".word 0xffff\n\t"
285         "nop\n\t"
286         :
287         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
288         "r" ((uint8_t)(__BOOT_PAGE_ERASE)),
289         "z" ((uint16_t)(address))
290     );
291     })
292
293 #define __boot_page_erase_extended(address) \
294 (__extension__({
295     __asm__ __volatile__
296     (

```

```

297 "movw r30, %A3\n\t"           \
298 "sts %1, %C3\n\t"           \
299 "sts %0, %2\n\t"           \
300 "spm\n\t"                     \
301 :                               \
302 : "i" (_SFR_MEM_ADDR(__SPM_REG)), \
303 "i" (_SFR_MEM_ADDR(RAMPZ)),      \
304 "r" ((uint8_t)(__BOOT_PAGE_ERASE)), \
305 "r" ((uint32_t)(address))        \
306 : "r30", "r31"                 \
307 );                               \
308 )))                             \
309
310 #define __boot_page_write_normal(address) \
311 (__extension__({                     \
312 __asm__ __volatile__                \
313 (                                     \
314 "sts %0, %1\n\t"                   \
315 "spm\n\t"                           \
316 :                                     \
317 : "i" (_SFR_MEM_ADDR(__SPM_REG)), \
318 "r" ((uint8_t)(__BOOT_PAGE_WRITE)), \
319 "z" ((uint16_t)(address))          \
320 );                                     \
321 )))
322
323 #define __boot_page_write_alternate(address) \
324 (__extension__({                     \
325 __asm__ __volatile__                \
326 (                                     \
327 "sts %0, %1\n\t"                   \
328 "spm\n\t"                           \
329 ".word 0xffff\n\t"                 \
330 "nop\n\t"                           \
331 :                                     \
332 : "i" (_SFR_MEM_ADDR(__SPM_REG)), \
333 "r" ((uint8_t)(__BOOT_PAGE_WRITE)), \
334 "z" ((uint16_t)(address))          \
335 );                                     \
336 )))
337
338 #define __boot_page_write_extended(address) \
339 (__extension__({                     \
340 __asm__ __volatile__                \
341 (                                     \
342 "movw r30, %A3\n\t"               \
343 "sts %1, %C3\n\t"                 \
344 "sts %0, %2\n\t"                 \
345 "spm\n\t"                           \
346 :                                     \
347 : "i" (_SFR_MEM_ADDR(__SPM_REG)), \
348 "i" (_SFR_MEM_ADDR(RAMPZ)),      \
349 "r" ((uint8_t)(__BOOT_PAGE_WRITE)), \
350 "r" ((uint32_t)(address))        \
351 : "r30", "r31"                 \
352 );                                     \
353 )))
354
355 #define __boot_rww_enable() \
356 (__extension__({           \
357 __asm__ __volatile__      \
358 (                           \
359 "sts %0, %1\n\t"           \
360 "spm\n\t"                   \
361 :                           \
362 : "i" (_SFR_MEM_ADDR(__SPM_REG)), \
363 "r" ((uint8_t)(__BOOT_RWW_ENABLE)) \
364 );                           \
365 )))
366
367 #define __boot_rww_enable_alternate() \
368 (__extension__({                     \
369 __asm__ __volatile__                \
370 (                                     \
371 "sts %0, %1\n\t"                   \
372 "spm\n\t"                           \
373 ".word 0xffff\n\t"                 \
374 "nop\n\t"                           \
375 :                                     \
376 : "i" (_SFR_MEM_ADDR(__SPM_REG)), \
377 "r" ((uint8_t)(__BOOT_RWW_ENABLE)) \
378 );                                     \
379 )))
380
381 /* From the mega16/mega128 data sheets (maybe others):
382
383 Bits by SPM To set the Boot Loader Lock bits, write the desired data to

```

```

384 R0, write "X0001001" to SPMCR and execute SPM within four clock cycles
385 after writing SPMCR. The only accessible Lock bits are the Boot Lock bits
386 that may prevent the Application and Boot Loader section from any
387 software update by the MCU.
388
389 If bits 5..2 in R0 are cleared (zero), the corresponding Boot Lock bit
390 will be programmed if an SPM instruction is executed within four cycles
391 after BLBSET and SPMEN (or SELFPRGEN) are set in SPMCR. The Z-pointer is
392 don't care during this operation, but for future compatibility it is
393 recommended to load the Z-pointer with $0001 (same as used for reading the
394 Lock bits). For future compatibility It is also recommended to set bits 7,
395 6, 1, and 0 in R0 to 1 when writing the Lock bits. When programming the
396 Lock bits the entire Flash can be read during the operation. */
397
398 #define __boot_lock_bits_set(lock_bits)
399 (__extension__({
400     uint8_t value = (uint8_t)(~(lock_bits));
401     __asm__ __volatile__
402     (
403         "ldi r30, 1\n\t"
404         "ldi r31, 0\n\t"
405         "mov r0, %2\n\t"
406         "sts %0, %1\n\t"
407         "spm\n\t"
408         :
409         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
410         "r" ((uint8_t)(__BOOT_LOCK_BITS_SET)),
411         "r" (value)
412         : "r0", "r30", "r31"
413     );
414 }));
415
416 #define __boot_lock_bits_set_alternate(lock_bits)
417 (__extension__({
418     uint8_t value = (uint8_t)(~(lock_bits));
419     __asm__ __volatile__
420     (
421         "ldi r30, 1\n\t"
422         "ldi r31, 0\n\t"
423         "mov r0, %2\n\t"
424         "sts %0, %1\n\t"
425         "spm\n\t"
426         ".word 0xffff\n\t"
427         "nop\n\t"
428         :
429         : "i" (_SFR_MEM_ADDR(__SPM_REG)),
430         "r" ((uint8_t)(__BOOT_LOCK_BITS_SET)),
431         "r" (value)
432         : "r0", "r30", "r31"
433     );
434 }));
435 #endif /* __DOXYGEN__ */
436
437 /*
438 Reading lock and fuse bits:
439
440 Similarly to writing the lock bits above, set BLBSET and SPMEN (or
441 SELFPRGEN) bits in __SPMREG, and then (within four clock cycles) issue an
442 LPM instruction.
443
444 Z address:          contents:
445 0x0000              low fuse bits
446 0x0001              lock bits
447 0x0002              extended fuse bits
448 0x0003              high fuse bits
449
450 Sounds confusing, doesn't it?
451
452 Unlike the macros in pgmspace.h, no need to care for non-enhanced
453 cores here as these old cores do not provide SPM support anyway.
454 */
455
456 /** \ingroup avr_boot
457  \def GET_LOW_FUSE_BITS
458  address to read the low fuse bits, using boot_lock_fuse_bits_get
459  */
460 #define GET_LOW_FUSE_BITS (0x0000)
461 /** \ingroup avr_boot
462  \def GET_LOCK_BITS
463  address to read the lock bits, using boot_lock_fuse_bits_get
464  */
465 #define GET_LOCK_BITS (0x0001)
466 /** \ingroup avr_boot
467  \def GET_EXTENDED_FUSE_BITS
468  address to read the extended fuse bits, using boot_lock_fuse_bits_get

```

```

469 */
470 #define GET_EXTENDED_FUSE_BITS      (0x0002)
471 /** \ingroup avr_boot
472 \def GET_HIGH_FUSE_BITS
473 address to read the high fuse bits, using boot_lock_fuse_bits_get
474 */
475 #define GET_HIGH_FUSE_BITS          (0x0003)
476
477 /** \ingroup avr_boot
478 \def boot_lock_fuse_bits_get(address)
479
480 Read the lock or fuse bits at \c address.
481
482 Parameter \c address can be any of GET_LOW_FUSE_BITS,
483 GET_LOCK_BITS, GET_EXTENDED_FUSE_BITS, or GET_HIGH_FUSE_BITS.
484
485 \note The lock and fuse bits returned are the physical values,
486 i.e. a bit returned as 0 means the corresponding fuse or lock bit
487 is programmed.
488 */
489 #define boot_lock_fuse_bits_get(address)
490 (__extension__({
491     uint8_t __result;
492     __asm__ __volatile__
493     (
494         "sts %1, %2\n\t"
495         "lpm %0, Z\n\t"
496         : "=r" (__result)
497         : "i" (__SFR_MEM_ADDR(__SPM_REG)),
498           "r" ((uint8_t)(__BOOT_LOCK_BITS_SET)),
499           "z" ((uint16_t)(address))
500         );
501     __result;
502 }))
503
504 #ifndef __DOXYGEN__
505 #define __BOOT_SIGROW_READ (_BV(__SPM_ENABLE) | _BV(SIGRD))
506 #endif
507 /** \ingroup avr_boot
508 \def boot_signature_byte_get(address)
509
510 Read the Signature Row byte at \c address. For some MCU types,
511 this function can also retrieve the factory-stored oscillator
512 calibration bytes.
513
514 Parameter \c address can be 0-0x1f as documented by the datasheet.
515 \note The values are MCU type dependent.
516 */
517
518 #define boot_signature_byte_get(addr)
519 (__extension__({
520     uint8_t __result;
521     __asm__ __volatile__
522     (
523         "sts %1, %2\n\t"
524         "lpm %0, Z" " " "\n\t"
525         : "=r" (__result)
526         : "i" (__SFR_MEM_ADDR(__SPM_REG)),
527           "r" ((uint8_t)(__BOOT_SIGROW_READ)),
528           "z" ((uint16_t)(addr))
529         );
530     __result;
531 }))
532
533 /** \ingroup avr_boot
534 \def boot_page_fill(address, data)
535
536 Fill the bootloader temporary page buffer for flash
537 address with data word.
538
539 \note The address is a byte address. The data is a word. The AVR
540 writes data to the buffer a word at a time, but addresses the buffer
541 per byte! So, increment your address by 2 between calls, and send 2
542 data bytes in a word format! The LSB of the data is written to the lower
543 address; the MSB of the data is written to the higher address.*/
544
545 /** \ingroup avr_boot
546 \def boot_page_erase(address)
547
548 Erase the flash page that contains address.
549
550 \note address is a byte address in flash, not a word address. */
551
552 /** \ingroup avr_boot
553 \def boot_page_write(address)
554
555 Write the bootloader temporary page buffer

```

```

556 to flash page that contains address.
557
558 \note address is a byte address in flash, not a word address. */
559
560 /** \ingroup avr_boot
561 \def boot_rww_enable()
562
563 Enable the Read-While-Write memory section. */
564
565 /** \ingroup avr_boot
566 \def boot_lock_bits_set(lock_bits)
567
568 Set the bootloader lock bits.
569
570 \param lock_bits A mask of which Boot Loader Lock Bits to set.
571
572 \note In this context, a 'set bit' will be written to a zero value.
573 Note also that only BLBxx bits can be programmed by this command.
574
575 For example, to disallow the SPM instruction from writing to the Boot
576 Loader memory section of flash, you would use this macro as such:
577
578 \code
579 boot_lock_bits_set (_BV (BLB11));
580 \endcode
581
582 \note Like any lock bits, the Boot Loader Lock Bits, once set,
583 cannot be cleared again except by a chip erase which will in turn
584 also erase the boot loader itself. */
585
586 /* Normal versions of the macros use 16-bit addresses.
587 Extended versions of the macros use 32-bit addresses.
588 Alternate versions of the macros use 16-bit addresses and require special
589 instruction sequences after LPM.
590
591 FLASHEND is defined in the ioXXXX.h file.
592 USHRT_MAX is defined in <limits.h>. */
593
594 #if defined(__AVR_ATmega161__) || defined(__AVR_ATmega163__) \
595 || defined(__AVR_ATmega323__)
596
597 /* Alternate: ATmega161/163/323 and 16 bit address */
598 #define boot_page_fill(address, data) __boot_page_fill_alternate(address, data)
599 #define boot_page_erase(address) __boot_page_erase_alternate(address)
600 #define boot_page_write(address) __boot_page_write_alternate(address)
601 #define boot_rww_enable() __boot_rww_enable_alternate()
602 #define boot_lock_bits_set(lock_bits) __boot_lock_bits_set_alternate(lock_bits)
603
604 #elif (FLASHEND > USHRT_MAX)
605
606 /* Extended: >16 bit address */
607 #define boot_page_fill(address, data) __boot_page_fill_extended(address, data)
608 #define boot_page_erase(address) __boot_page_erase_extended(address)
609 #define boot_page_write(address) __boot_page_write_extended(address)
610 #define boot_rww_enable() __boot_rww_enable()
611 #define boot_lock_bits_set(lock_bits) __boot_lock_bits_set(lock_bits)
612
613 #else
614
615 /* Normal: 16 bit address */
616 #define boot_page_fill(address, data) __boot_page_fill_normal(address, data)
617 #define boot_page_erase(address) __boot_page_erase_normal(address)
618 #define boot_page_write(address) __boot_page_write_normal(address)
619 #define boot_rww_enable() __boot_rww_enable()
620 #define boot_lock_bits_set(lock_bits) __boot_lock_bits_set(lock_bits)
621
622 #endif
623
624 /** \ingroup avr_boot
625
626 Same as boot_page_fill() except it waits for eeprom and spm operations to
627 complete before filling the page. */
628
629 #define boot_page_fill_safe(address, data) \
630 do { \
631 boot_spm_busy_wait(); \
632 eeprom_busy_wait(); \
633 boot_page_fill(address, data); \
634 } while (0)
635
636 /** \ingroup avr_boot
637
638 Same as boot_page_erase() except it waits for eeprom and spm operations to
639 complete before erasing the page. */
640
641 #define boot_page_erase_safe(address) \
642 do { \

```

```

643 boot_spm_busy_wait();           \
644 eeprom_busy_wait();           \
645 boot_page_erase (address);     \
646 } while (0)
647
648 /** \ingroup avr_boot
649
650 Same as boot_page_write() except it waits for eeprom and spm operations to
651 complete before writing the page. */
652
653 #define boot_page_write_safe(address) \
654 do { \
655 boot_spm_busy_wait();           \
656 eeprom_busy_wait();           \
657 boot_page_write (address);     \
658 } while (0)
659
660 /** \ingroup avr_boot
661
662 Same as boot_rww_enable() except waits for eeprom and spm operations to
663 complete before enabling the RWW mameory. */
664
665 #define boot_rww_enable_safe() \
666 do { \
667 boot_spm_busy_wait();           \
668 eeprom_busy_wait();           \
669 boot_rww_enable();             \
670 } while (0)
671
672 /** \ingroup avr_boot
673
674 Same as boot_lock_bits_set() except waits for eeprom and spm operations to
675 complete before setting the lock bits. */
676
677 #define boot_lock_bits_set_safe(lock_bits) \
678 do { \
679 boot_spm_busy_wait();           \
680 eeprom_busy_wait();           \
681 boot_lock_bits_set (lock_bits); \
682 } while (0)
683
684 #endif /* _AVR_BOOT_H_ */

```

25.12 cpufunc.h File Reference

Macros

- `#define _NOP()`
- `#define _MemoryBarrier()`

Functions

- void `ccp_write_io` (uint8_t *__ioaddr, uint8_t __value)

25.13 cpufunc.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2010, Joerg Wunsch
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of

```

```

16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: cpufunc.h 2529 2016-12-05 06:09:28Z pitchumani $ */
32
33 /* avr/cpufunc.h - Special CPU functions */
34
35 #ifndef _AVR_CPUFUNC_H_
36 #define _AVR_CPUFUNC_H_ 1
37
38 #include <stdint.h>
39
40 /** \file */
41 /** \defgroup avr_cpufunc <avr/cpufunc.h>: Special AVR CPU functions
42 \code #include <avr/cpufunc.h> \endcode
43
44 This header file contains macros that access special functions of
45 the AVR CPU which do not fit into any of the other header files.
46
47 */
48
49 #if defined(__DOXYGEN__)
50 /**
51 \ingroup avr_cpufunc
52 \def _NOP
53
54 Execute a <i>no operation</i> (NOP) CPU instruction. This
55 should not be used to implement delays, better use the functions
56 from <util/delay_basic.h> or <util/delay.h> for this. For
57 debugging purposes, a NOP can be useful to have an instruction that
58 is guaranteed to be not optimized away by the compiler, so it can
59 always become a breakpoint in the debugger.
60 */
61 #define _NOP()
62 #else /* real code */
63 #define _NOP() __asm__ __volatile__("nop")
64 #endif /* __DOXYGEN__ */
65
66 #if defined(__DOXYGEN__)
67 /**
68 \ingroup avr_cpufunc
69 \def _MemoryBarrier
70
71 Implement a read/write <i>memory barrier</i>. A memory
72 barrier instructs the compiler to not cache any memory data in
73 registers beyond the barrier. This can sometimes be more effective
74 than blocking certain optimizations by declaring some object with a
75 \c volatile qualifier.
76
77 See \ref optim_code_reorder for things to be taken into account
78 with respect to compiler optimizations.
79 */
80 #define _MemoryBarrier()
81 #else /* real code */
82 #define _MemoryBarrier() __asm__ __volatile__(":::memory")
83 #endif /* __DOXYGEN__ */
84
85 /**
86 \ingroup avr_cpufunc
87
88 Write \a __value to Configuration Change Protected (CCP) IO register
89 at \a __ioaddr.
90 */
91 void ccp_write_io (uint8_t *__ioaddr, uint8_t __value);
92
93 #endif /* _AVR_CPUFUNC_H_ */

```

25.14 eeprom.h

```

1 /* Copyright (c) 2002, 2003, 2004, 2007 Marek Michalkiewicz
2 Copyright (c) 2005, 2006 Bjoern Haase
3 Copyright (c) 2008 Atmel Corporation

```



```
4 Copyright (c) 2008 Wouter van Gulik
5 Copyright (c) 2009 Dmitry Xmelkov
6 All rights reserved.
7
8 Redistribution and use in source and binary forms, with or without
9 modification, are permitted provided that the following conditions are met:
10
11 * Redistributions of source code must retain the above copyright
12 notice, this list of conditions and the following disclaimer.
13 * Redistributions in binary form must reproduce the above copyright
14 notice, this list of conditions and the following disclaimer in
15 the documentation and/or other materials provided with the
16 distribution.
17 * Neither the name of the copyright holders nor the names of
18 contributors may be used to endorse or promote products derived
19 from this software without specific prior written permission.
20
21 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
22 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
23 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
24 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
25 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
26 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
27 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
28 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
29 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
30 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
31 POSSIBILITY OF SUCH DAMAGE. */
32
33 /* $Id: eeprom.h 2548 2018-06-29 06:38:56Z pitchumani $ */
34
35 #ifndef _AVR_EEPROM_H_
36 #define _AVR_EEPROM_H_ 1
37
38 #include <avr/io.h>
39
40 #if !E2END && !defined(__DOXYGEN__) && !defined(__COMPILING_AVR_LIBC__)
41 #warning "Device does not have EEPROM available."
42 #else
43
44 #if defined (EEAR) && !defined (EEARL) && !defined (EEARH)
45 #define EEARL EEAR
46 #endif
47
48 #ifndef __ASSEMBLER__
49
50 #include <stddef.h> /* size_t */
51 #include <stdint.h>
52
53 /** \defgroup avr_eeprom <avr/eeprom.h>: EEPROM handling
54 \code #include <avr/eeprom.h> \endcode
55
56 This header file declares the interface to some simple library
57 routines suitable for handling the data EEPROM contained in the
58 AVR microcontrollers. The implementation uses a simple polled
59 mode interface. Applications that require interrupt-controlled
60 EEPROM access to ensure that no time will be wasted in spinloops
61 will have to deploy their own implementation.
62
63 \par Notes:
64
65 - In addition to the write functions there is a set of update ones.
66 This functions read each byte first and skip the burning if the
67 old value is the same with new. The scanning direction is from
68 high address to low, to obtain quick return in common cases.
69
70 - All of the read/write functions first make sure the EEPROM is
71 ready to be accessed. Since this may cause long delays if a
72 write operation is still pending, time-critical applications
73 should first poll the EEPROM e. g. using eeprom_is_ready() before
74 attempting any actual I/O. But this functions are not wait until
75 SELFPRGEN in SPMCSR becomes zero. Do this manually, if your
76 software contains the Flash burning.
77
78 - As these functions modify IO registers, they are known to be
79 non-reentrant. If any of these functions are used from both,
80 standard and interrupt context, the applications must ensure
81 proper protection (e.g. by disabling interrupts before accessing
82 them).
83
84 - All write functions force erase_and_write programming mode.
85
86 - For Xmega the EEPROM start address is 0, like other architectures.
87 The reading functions add the 0x2000 value to use EEPROM mapping into
88 data space.
89 */
90
```

```

91 #ifdef __cplusplus
92 extern "C" {
93 #endif
94
95 #ifndef __ATTR_PURE__
96 #   ifdef __DOXYGEN__
97 #       define __ATTR_PURE__
98 #   else
99 #       define __ATTR_PURE__ __attribute__((__pure__))
100 #   endif
101 #endif
102
103 /** \def EEMEM
104 \ingroup avr_eeprom
105 Attribute expression causing a variable to be allocated within the
106 .eeprom section. */
107 #define EEMEM __attribute__((section(".eeprom")))
108
109 /** \def eeprom_is_ready
110 \ingroup avr_eeprom
111 \returns 1 if EEPROM is ready for a new read/write operation, 0 if not.
112 */
113 #if defined (__DOXYGEN__)
114 #   define eeprom_is_ready()
115 #elif defined (NVM_STATUS)
116 #   define eeprom_is_ready() bit_is_clear (NVM_STATUS, NVM_NVMBUSY_bp)
117 #elif defined (NVMCTRL_STATUS)
118 #   define eeprom_is_ready() bit_is_clear (NVMCTRL_STATUS, NVMCTRL_EEBUSY_bp)
119 #elif defined (DEECCR)
120 #   define eeprom_is_ready() bit_is_clear (DEECCR, BSY)
121 #elif defined (EEPE)
122 #   define eeprom_is_ready() bit_is_clear (EECR, EEPE)
123 #else
124 #   define eeprom_is_ready() bit_is_clear (EECR, EEWE)
125 #endif
126
127
128 /** \def eeprom_busy_wait
129 \ingroup avr_eeprom
130 Loops until the eeprom is no longer busy.
131 \returns Nothing.
132 */
133 #define eeprom_busy_wait() do {} while (!eeprom_is_ready())
134
135
136 /** \ingroup avr_eeprom
137 Read one byte from EEPROM address \a __p.
138 */
139 uint8_t eeprom_read_byte (const uint8_t *__p) __ATTR_PURE__;
140
141 /** \ingroup avr_eeprom
142 Read one 16-bit word (little endian) from EEPROM address \a __p.
143 */
144 uint16_t eeprom_read_word (const uint16_t *__p) __ATTR_PURE__;
145
146 /** \ingroup avr_eeprom
147 Read one 32-bit double word (little endian) from EEPROM address \a __p.
148 */
149 uint32_t eeprom_read_dword (const uint32_t *__p) __ATTR_PURE__;
150
151 /** \ingroup avr_eeprom
152 Read one float value (little endian) from EEPROM address \a __p.
153 */
154 float eeprom_read_float (const float *__p) __ATTR_PURE__;
155
156 /** \ingroup avr_eeprom
157 Read a block of \a __n bytes from EEPROM address \a __src to SRAM
158 \a __dst.
159 */
160 void eeprom_read_block (void *__dst, const void *__src, size_t __n);
161
162
163 /** \ingroup avr_eeprom
164 Write a byte \a __value to EEPROM address \a __p.
165 */
166 void eeprom_write_byte (uint8_t *__p, uint8_t __value);
167
168 /** \ingroup avr_eeprom
169 Write a word \a __value to EEPROM address \a __p.
170 */
171 void eeprom_write_word (uint16_t *__p, uint16_t __value);
172
173 /** \ingroup avr_eeprom
174 Write a 32-bit double word \a __value to EEPROM address \a __p.
175 */
176 void eeprom_write_dword (uint32_t *__p, uint32_t __value);
177

```

```

178 /** \ingroup avr_eeprom
179 Write a float \a __value to EEPROM address \a __p.
180 */
181 void eeprom_write_float (float *__p, float __value);
182
183 /** \ingroup avr_eeprom
184 Write a block of \a __n bytes to EEPROM address \a __dst from \a __src.
185 \note The argument order is mismatch with common functions like strcpy().
186 */
187 void eeprom_write_block (const void *__src, void *__dst, size_t __n);
188
189
190 /** \ingroup avr_eeprom
191 Update a byte \a __value to EEPROM address \a __p.
192 */
193 void eeprom_update_byte (uint8_t *__p, uint8_t __value);
194
195 /** \ingroup avr_eeprom
196 Update a word \a __value to EEPROM address \a __p.
197 */
198 void eeprom_update_word (uint16_t *__p, uint16_t __value);
199
200 /** \ingroup avr_eeprom
201 Update a 32-bit double word \a __value to EEPROM address \a __p.
202 */
203 void eeprom_update_dword (uint32_t *__p, uint32_t __value);
204
205 /** \ingroup avr_eeprom
206 Update a float \a __value to EEPROM address \a __p.
207 */
208 void eeprom_update_float (float *__p, float __value);
209
210 /** \ingroup avr_eeprom
211 Update a block of \a __n bytes to EEPROM address \a __dst from \a __src.
212 \note The argument order is mismatch with common functions like strcpy().
213 */
214 void eeprom_update_block (const void *__src, void *__dst, size_t __n);
215
216
217 /** \name IAR C compatibility defines */
218 /*{*/
219
220 /** \def _EEPUT
221 \ingroup avr_eeprom
222 Write a byte to EEPROM. Compatibility define for IAR C. */
223 #define _EEPUT(addr, val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))
224
225 /** \def __EEPUT
226 \ingroup avr_eeprom
227 Write a byte to EEPROM. Compatibility define for IAR C. */
228 #define __EEPUT(addr, val) eeprom_write_byte ((uint8_t *) (addr), (uint8_t) (val))
229
230 /** \def _EEGET
231 \ingroup avr_eeprom
232 Read a byte from EEPROM. Compatibility define for IAR C. */
233 #define _EEGET(var, addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))
234
235 /** \def __EEGET
236 \ingroup avr_eeprom
237 Read a byte from EEPROM. Compatibility define for IAR C. */
238 #define __EEGET(var, addr) (var) = eeprom_read_byte ((const uint8_t *) (addr))
239
240 /*}*/
241
242 #ifdef __cplusplus
243 }
244 #endif
245
246 #endif /* !__ASSEMBLER__ */
247 #endif /* E2END || defined(__DOXYGEN__) || defined(__COMPILING_AVR_LIBC__) */
248 #endif /* !_AVR_EEPROM_H_ */

```

25.15 fuse.h File Reference

25.16 fuse.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2007, Atmel Corporation
2 All rights reserved.
3

```

```

4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: fuse.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
32
33 /* avr/fuse.h - Fuse API */
34
35 #ifndef _AVR_FUSE_H_
36 #define _AVR_FUSE_H_ 1
37
38 /* This file must be explicitly included by <avr/io.h>. */
39 #if !defined(_AVR_IO_H_)
40 #error "You must #include <avr/io.h> and not <avr/fuse.h> by itself."
41 #endif
42
43
44 /** \file */
45 /** \defgroup avr_fuse <avr/fuse.h>: Fuse Support
46
47 \par Introduction
48
49 The Fuse API allows a user to specify the fuse settings for the specific
50 AVR device they are compiling for. These fuse settings will be placed
51 in a special section in the ELF output file, after linking.
52
53 Programming tools can take advantage of the fuse information embedded in
54 the ELF file, by extracting this information and determining if the fuses
55 need to be programmed before programming the Flash and EEPROM memories.
56 This also allows a single ELF file to contain all the
57 information needed to program an AVR.
58
59 To use the Fuse API, include the <avr/io.h> header file, which in turn
60 automatically includes the individual I/O header file and the <avr/fuse.h>
61 file. These other two files provides everything necessary to set the AVR
62 fuses.
63
64 \par Fuse API
65
66 Each I/O header file must define the FUSE_MEMORY_SIZE macro which is
67 defined to the number of fuse bytes that exist in the AVR device.
68
69 A new type, __fuse_t, is defined as a structure. The number of fields in
70 this structure are determined by the number of fuse bytes in the
71 FUSE_MEMORY_SIZE macro.
72
73 If FUSE_MEMORY_SIZE == 1, there is only a single field: byte, of type
74 unsigned char.
75
76 If FUSE_MEMORY_SIZE == 2, there are two fields: low, and high, of type
77 unsigned char.
78
79 If FUSE_MEMORY_SIZE == 3, there are three fields: low, high, and extended,
80 of type unsigned char.
81
82 If FUSE_MEMORY_SIZE > 3, there is a single field: byte, which is an array
83 of unsigned char with the size of the array being FUSE_MEMORY_SIZE.
84
85 A convenience macro, FUSEMEM, is defined as a GCC attribute for a
86 custom-named section of ".fuse".
87
88 A convenience macro, FUSES, is defined that declares a variable, __fuse, of
89 type __fuse_t with the attribute defined by FUSEMEM. This variable
90 allows the end user to easily set the fuse data.

```

```

91
92 \note If a device-specific I/O header file has previously defined FUSEMEM,
93 then FUSEMEM is not redefined. If a device-specific I/O header file has
94 previously defined FUSES, then FUSES is not redefined.
95
96 Each AVR device I/O header file has a set of defined macros which specify the
97 actual fuse bits available on that device. The AVR fuses have inverted
98 values, logical 1 for an unprogrammed (disabled) bit and logical 0 for a
99 programmed (enabled) bit. The defined macros for each individual fuse
100 bit represent this in their definition by a bit-wise inversion of a mask.
101 For example, the FUSE_EESAVE fuse in the ATmega128 is defined as:
102 \code
103 #define FUSE_EESAVE      ~_BV(3)
104 \endcode
105 \note The _BV macro creates a bit mask from a bit number. It is then
106 inverted to represent logical values for a fuse memory byte.
107
108 To combine the fuse bits macros together to represent a whole fuse byte,
109 use the bitwise AND operator, like so:
110 \code
111 (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN)
112 \endcode
113
114 Each device I/O header file also defines macros that provide default values
115 for each fuse byte that is available. LFUSE_DEFAULT is defined for a Low
116 Fuse byte. HFUSE_DEFAULT is defined for a High Fuse byte. EFUSE_DEFAULT
117 is defined for an Extended Fuse byte.
118
119 If FUSE_MEMORY_SIZE > 3, then the I/O header file defines macros that
120 provide default values for each fuse byte like so:
121 FUSE0_DEFAULT
122 FUSE1_DEFAULT
123 FUSE2_DEFAULT
124 FUSE3_DEFAULT
125 FUSE4_DEFAULT
126 ....
127
128 \par API Usage Example
129
130 Putting all of this together is easy. Using C99's designated initializers:
131
132 \code
133 #include <avr/io.h>
134
135 FUSES =
136 {
137     .low = LFUSE_DEFAULT,
138     .high = (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN),
139     .extended = EFUSE_DEFAULT,
140 };
141
142 int main(void)
143 {
144     return 0;
145 }
146 \endcode
147
148 Or, using the variable directly instead of the FUSES macro,
149
150 \code
151 #include <avr/io.h>
152
153 __fuse_t __fuse __attribute__((section (".fuse"))) =
154 {
155     .low = LFUSE_DEFAULT,
156     .high = (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN),
157     .extended = EFUSE_DEFAULT,
158 };
159
160 int main(void)
161 {
162     return 0;
163 }
164 \endcode
165
166 If you are compiling in C++, you cannot use the designated initializers so
167 you must do:
168
169 \code
170 #include <avr/io.h>
171
172 FUSES =
173 {
174     LFUSE_DEFAULT, // .low
175     (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN & FUSE_JTAGEN), // .high
176     EFUSE_DEFAULT, // .extended
177 };

```

```

178
179 int main(void)
180 {
181     return 0;
182 }
183 \endcode
184
185
186 However there are a number of caveats that you need to be aware of to
187 use this API properly.
188
189 Be sure to include <avr/io.h> to get all of the definitions for the API.
190 The FUSES macro defines a global variable to store the fuse data. This
191 variable is assigned to its own linker section. Assign the desired fuse
192 values immediately in the variable initialization.
193
194 The .fuse section in the ELF file will get its values from the initial
195 variable assignment ONLY. This means that you can NOT assign values to
196 this variable in functions and the new values will not be put into the
197 ELF .fuse section.
198
199 The global variable is declared in the FUSES macro has two leading
200 underscores, which means that it is reserved for the "implementation",
201 meaning the library, so it will not conflict with a user-named variable.
202
203 You must initialize ALL fields in the __fuse_t structure. This is because
204 the fuse bits in all bytes default to a logical 1, meaning unprogrammed.
205 Normal uninitialized data defaults to all logical zeros. So it is vital that
206 all fuse bytes are initialized, even with default data. If they are not,
207 then the fuse bits may not be programmed to the desired settings.
208
209 Be sure to have the -mmcu=<em>device</em> flag in your compile command line and
210 your linker command line to have the correct device selected and to have
211 the correct I/O header file included when you include <avr/io.h>.
212
213 You can print out the contents of the .fuse section in the ELF file by
214 using this command line:
215 \code
216 avr-objdump -s -j .fuse <ELF file>
217 \endcode
218 The section contents shows the address on the left, then the data going from
219 lower address to a higher address, left to right.
220
221 */
222
223 #if !(defined(__ASSEMBLER__) || defined(__DOXYGEN__))
224
225 #ifndef FUSEMEM
226 #define FUSEMEM __attribute__((__used__, __section__ (".fuse")))
227 #endif
228
229 #if FUSE_MEMORY_SIZE > 3
230
231 typedef struct
232 {
233     unsigned char byte[FUSE_MEMORY_SIZE];
234 } __fuse_t;
235
236
237 #elif FUSE_MEMORY_SIZE == 3
238
239 typedef struct
240 {
241     unsigned char low;
242     unsigned char high;
243     unsigned char extended;
244 } __fuse_t;
245
246 #elif FUSE_MEMORY_SIZE == 2
247
248 typedef struct
249 {
250     unsigned char low;
251     unsigned char high;
252 } __fuse_t;
253
254 #elif FUSE_MEMORY_SIZE == 1
255
256 typedef struct
257 {
258     unsigned char byte;
259 } __fuse_t;
260
261 #endif
262
263 #if !defined(FUSES)
264 #if defined(__AVR_XMEGA__)

```

```

265 #define FUSES NVM_FUSES_t __fuse FUSEMEM
266 #else
267 #define FUSES __fuse_t __fuse FUSEMEM
268 #endif
269 #endif
270
271
272 #endif /* !(__ASSEMBLER__ || __DOXYGEN__) */
273
274 #endif /* _AVR_FUSE_H_ */

```

25.17 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()`
- `#define cli()`

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()`
- `#define BADISR_vect`

ISR attributes

- `#define ISR_BLOCK`
- `#define ISR_NOBLOCK`
- `#define ISR_NAKED`
- `#define ISR_FLATTEN`
- `#define ISR_NOICF`
- `#define ISR_ALIASOF(target_vector)`

25.17.1 Detailed Description

@{

25.18 interrupt.h

[Go to the documentation of this file.](#)

```

1  /* Copyright (c) 2002,2005,2007 Marek Michalkiewicz
2  Copyright (c) 2007, Dean Camera
3
4  All rights reserved.
5
6  Redistribution and use in source and binary forms, with or without
7  modification, are permitted provided that the following conditions are met:
8
9  * Redistributions of source code must retain the above copyright
10 notice, this list of conditions and the following disclaimer.
11
12 * Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in
14 the documentation and/or other materials provided with the
15 distribution.
16
17 * Neither the name of the copyright holders nor the names of
18 contributors may be used to endorse or promote products derived
19 from this software without specific prior written permission.
20
21 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
22 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
23 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
24 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
25 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
26 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
27 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
28 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
29 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
30 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
31 POSSIBILITY OF SUCH DAMAGE. */
32
33 /* $Id: interrupt.h 2538 2017-06-11 15:16:05Z joerg_wunsch $ */
34
35 #ifndef _AVR_INTERRUPT_H_
36 #define _AVR_INTERRUPT_H_
37
38 #include <avr/io.h>
39
40 #if !defined(__DOXYGEN__) && !defined(__STRINGIFY)
41 /* Auxiliary macro for ISR_ALIAS(). */
42 #define __STRINGIFY(x) #x
43 #endif /* !defined(__DOXYGEN__) */
44
45 /**
46  \file
47  \@{
48  */
49
50
51 /** \name Global manipulation of the interrupt flag
52
53 The global interrupt flag is maintained in the I bit of the status
54 register (SREG).
55
56 Handling interrupts frequently requires attention regarding atomic
57 access to objects that could be altered by code running within an
58 interrupt context, see <util/atomic.h>.
59
60 Frequently, interrupts are being disabled for periods of time in
61 order to perform certain operations without being disturbed; see
62 \ref optim_code_reorder for things to be taken into account with
63 respect to compiler optimizations.
64 */
65
66 #if defined(__DOXYGEN__)
67 /** \def sei()
68  \ingroup avr_interrupts
69
70 Enables interrupts by setting the global interrupt mask. This function
71 actually compiles into a single line of assembly, so there is no function
72 call overhead. However, the macro also implies a <i>memory barrier</i>
73 which can cause additional loss of optimization.
74
75 In order to implement atomic access to multi-byte objects,
76 consider using the macros from <util/atomic.h>, rather than
77 implementing them manually with cli() and sei().
78 */
79 #define sei()
80 #else /* !DOXYGEN */
81 #define sei() __asm__ __volatile__ ("sei" ::: "memory")
82 #endif /* DOXYGEN */
83

```



```

84 #if defined(__DOXYGEN__)
85 /** \def cli()
86 \ingroup avr_interrupts
87
88 Disables all interrupts by clearing the global interrupt mask. This function
89 actually compiles into a single line of assembly, so there is no function
90 call overhead. However, the macro also implies a <i>memory barrier</i>
91 which can cause additional loss of optimization.
92
93 In order to implement atomic access to multi-byte objects,
94 consider using the macros from <util/atomic.h>, rather than
95 implementing them manually with cli() and sei().
96 */
97 #define cli()
98 #else /* !DOXYGEN */
99 # define cli() __asm__ __volatile__ ("cli" ::: "memory")
100 #endif /* DOXYGEN */
101
102
103 /** \name Macros for writing interrupt handler functions */
104
105
106 #if defined(__DOXYGEN__)
107 /** \def ISR(vector [, attributes])
108 \ingroup avr_interrupts
109
110 Introduces an interrupt handler function (interrupt service
111 routine) that runs with global interrupts initially disabled
112 by default with no attributes specified.
113
114 The attributes are optional and alter the behaviour and resultant
115 generated code of the interrupt routine. Multiple attributes may
116 be used for a single function, with a space separating each
117 attribute.
118
119 Valid attributes are ISR_BLOCK, ISR_NOBLOCK, ISR_NAKED and
120 ISR_ALIASOF(vect).
121
122 \c vector must be one of the interrupt vector names that are
123 valid for the particular MCU type.
124 */
125 # define ISR(vector, [attributes])
126 #else /* real code */
127
128 #if (__GNUC__ == 4 && __GNUC_MINOR__ >= 1) || (__GNUC__ > 4)
129 # define __INTR_ATTRS __used__, __externally_visible__
130 #else /* GCC < 4.1 */
131 # define __INTR_ATTRS __used__
132 #endif
133
134 #ifdef __cplusplus
135 # define ISR(vector, ...) \
136 extern "C" void vector(void) __attribute__((__signal__, __INTR_ATTRS)) __VA_ARGS__; \
137 void vector(void)
138 #else
139 # define ISR(vector, ...) \
140 void vector(void) __attribute__((__signal__, __INTR_ATTRS)) __VA_ARGS__; \
141 void vector(void)
142 #endif
143
144 #endif /* DOXYGEN */
145
146 #if defined(__DOXYGEN__)
147 /** \def SIGNAL(vector)
148 \ingroup avr_interrupts
149
150 Introduces an interrupt handler function that runs with global interrupts
151 initially disabled.
152
153 This is the same as the ISR macro without optional attributes.
154 \deprecated Do not use SIGNAL() in new code. Use ISR() instead.
155 */
156 # define SIGNAL(vector)
157 #else /* real code */
158
159 #ifdef __cplusplus
160 # define SIGNAL(vector) \
161 extern "C" void vector(void) __attribute__((__signal__, __INTR_ATTRS)); \
162 void vector(void)
163 #else
164 # define SIGNAL(vector) \
165 void vector(void) __attribute__((__signal__, __INTR_ATTRS)); \
166 void vector(void)
167 #endif
168
169 #endif /* DOXYGEN */
170

```

```

171 #if defined(__DOXYGEN__)
172 /** \def EMPTY_INTERRUPT(vector)
173 \ingroup avr_interruptions
174
175 Defines an empty interrupt handler function. This will not generate
176 any prolog or epilog code and will only return from the ISR. Do not
177 define a function body as this will define it for you.
178 Example:
179 \code EMPTY_INTERRUPT(ADC_vect);\endcode */
180 # define EMPTY_INTERRUPT(vector)
181 #else /* real code */
182
183 #ifdef __cplusplus
184 # define EMPTY_INTERRUPT(vector) \
185 extern "C" void vector(void) __attribute__((__signal__, __naked__, __INTR_ATTRS)); \
186 void vector(void) { __asm__ __volatile__ ("reti" ::: "memory"); }
187 #else
188 # define EMPTY_INTERRUPT(vector) \
189 void vector(void) __attribute__((__signal__, __naked__, __INTR_ATTRS)); \
190 void vector(void) { __asm__ __volatile__ ("reti" ::: "memory"); }
191 #endif
192
193 #endif /* DOXYGEN */
194
195 #if defined(__DOXYGEN__)
196 /** \def ISR_ALIAS(vector, target_vector)
197 \ingroup avr_interruptions
198
199 Aliases a given vector to another one in the same manner as the
200 ISR_ALIASOF attribute for the ISR() macro. Unlike the ISR_ALIASOF
201 attribute macro however, this is compatible for all versions of
202 GCC rather than just GCC version 4.2 onwards.
203
204 \note This macro creates a trampoline function for the aliased
205 macro. This will result in a two cycle penalty for the aliased
206 vector compared to the ISR the vector is aliased to, due to the
207 JMP/RJMP opcode used.
208
209 \deprecated
210 For new code, the use of ISR(..., ISR_ALIASOF(...)) is
211 recommended.
212
213 Example:
214 \code
215 ISR(INT0_vect)
216 {
217 PORTB = 42;
218 }
219
220 ISR_ALIAS(INT1_vect, INT0_vect);
221 \endcode
222
223 */
224 # define ISR_ALIAS(vector, target_vector)
225 #else /* real code */
226
227 #ifdef __cplusplus
228 # define ISR_ALIAS(vector, tgt) extern "C" void vector(void) \
229 __attribute__((__signal__, __naked__, __INTR_ATTRS)); \
230 void vector(void) { __asm__ __volatile__ ("%~jmp " __STRINGIFY(tgt) ::); }
231 #else /* !__cplusplus */
232 # define ISR_ALIAS(vector, tgt) void vector(void) \
233 __attribute__((__signal__, __naked__, __INTR_ATTRS)); \
234 void vector(void) { __asm__ __volatile__ ("%~jmp " __STRINGIFY(tgt) ::); }
235 #endif /* __cplusplus */
236
237 #endif /* DOXYGEN */
238
239 #if defined(__DOXYGEN__)
240 /** \def reti()
241 \ingroup avr_interruptions
242
243 Returns from an interrupt routine, enabling global interrupts. This should
244 be the last command executed before leaving an ISR defined with the ISR_NAKED
245 attribute.
246
247 This macro actually compiles into a single line of assembly, so there is
248 no function call overhead.
249 */
250 # define reti()
251 #else /* !DOXYGEN */
252 # define reti() __asm__ __volatile__ ("reti" ::: "memory")
253 #endif /* DOXYGEN */
254
255 #if defined(__DOXYGEN__)
256 /** \def BADISR_vect
257 \ingroup avr_interruptions

```

```

258
259 \code #include <avr/interrupt.h> \endcode
260
261 This is a vector which is aliased to __vector_default, the vector
262 executed when an ISR fires with no accompanying ISR handler. This
263 may be used along with the ISR() macro to create a catch-all for
264 undefined but used ISRs for debugging purposes.
265 */
266 # define BADISR_vect
267 #else /* !DOXYGEN */
268 # define BADISR_vect __vector_default
269 #endif /* DOXYGEN */
270
271 /** \name ISR attributes */
272
273 #if defined(__DOXYGEN__)
274 /** \def ISR_BLOCK
275 \ingroup avr_interrupts
276
277 Identical to an ISR with no attributes specified. Global
278 interrupts are initially disabled by the AVR hardware when
279 entering the ISR, without the compiler modifying this state.
280
281 Use this attribute in the attributes parameter of the ISR macro.
282 */
283 # define ISR_BLOCK
284
285 /** \def ISR_NOBLOCK
286 \ingroup avr_interrupts
287
288 ISR runs with global interrupts initially enabled. The interrupt
289 enable flag is activated by the compiler as early as possible
290 within the ISR to ensure minimal processing delay for nested
291 interrupts.
292
293 This may be used to create nested ISRs, however care should be
294 taken to avoid stack overflows, or to avoid infinitely entering
295 the ISR for those cases where the AVR hardware does not clear the
296 respective interrupt flag before entering the ISR.
297
298 Use this attribute in the attributes parameter of the ISR macro.
299 */
300 # define ISR_NOBLOCK
301
302 /** \def ISR_NAKED
303 \ingroup avr_interrupts
304
305 ISR is created with no prologue or epilogue code. The user code is
306 responsible for preservation of the machine state including the
307 SREG register, as well as placing a reti() at the end of the
308 interrupt routine.
309
310 Use this attribute in the attributes parameter of the ISR macro.
311 */
312 # define ISR_NAKED
313
314 /** \def ISR_FLATTEN
315 \ingroup avr_interrupts
316
317 The compiler will try to inline all called function into the ISR.
318 This has an effect with GCC 4.6 and newer only.
319
320 Use this attribute in the attributes parameter of the ISR macro.
321 */
322 # define ISR_FLATTEN
323
324 /** \def ISR_NOICF
325 \ingroup avr_interrupts
326
327 Avoid identical-code-folding optimization against this ISR.
328 This has an effect with GCC 5 and newer only.
329
330 Use this attribute in the attributes parameter of the ISR macro.
331 */
332 # define ISR_NOICF
333
334 /** \def ISR_ALIASOF(target_vector)
335 \ingroup avr_interrupts
336
337 The ISR is linked to another ISR, specified by the vect parameter.
338 This is compatible with GCC 4.2 and greater only.
339
340 Use this attribute in the attributes parameter of the ISR macro.
341 Example:
342 \code
343 ISR (INT0_vect)
344 {

```

```

345 PORTB = 42;
346 }
347
348 ISR (INT1_vect, ISR_ALIASOF (INT0_vect));
349 \endcode
350 */
351 # define ISR_ALIASOF(target_vector)
352 #else /* !DOXYGEN */
353 # define ISR_BLOCK /* empty */
354 /* FIXME: This won't work with older versions of avr-gcc as ISR_NOBLOCK
355 will use 'signal' and 'interrupt' at the same time. */
356 # define ISR_NOBLOCK __attribute__((__interrupt__))
357 # define ISR_NAKED __attribute__((__naked__))
358
359 #if (__GNUC__ == 4 && __GNUC_MINOR__ >= 6) || (__GNUC__ >= 5)
360 # define ISR_FLATTEN __attribute__((__flatten__))
361 #else
362 # define ISR_FLATTEN /* empty */
363 #endif /* has flatten (GCC 4.6+) */
364
365 #if defined (__has_attribute)
366 #if __has_attribute (__no_icf__)
367 # define ISR_NOICF __attribute__((__no_icf__))
368 #else
369 # define ISR_NOICF /* empty */
370 #endif /* has no_icf */
371 #endif /* has __has_attribute (GCC 5+) */
372
373 # define ISR_ALIASOF(v) __attribute__((__alias__((__STRINGIFY(v)))
374 #endif /* DOXYGEN */
375
376 /* @} */
377
378 #endif

```

25.19 io.h File Reference

25.20 io.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002,2003,2005,2006,2007 Marek Michalkiewicz, Joerg Wunsch
2 Copyright (c) 2007 Eric B. Weddington
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15
16 * Neither the name of the copyright holders nor the names of
17 contributors may be used to endorse or promote products derived
18 from this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE. */
31
32 /* $Id: io.h 2499 2016-01-28 14:41:31Z pitchumani $ */
33
34 /** \file */
35 /** \defgroup avr_io <avr/io.h>: AVR device-specific IO definitions
36 \code #include <avr/io.h> \endcode
37
38 This header file includes the appropriate IO definitions for the
39 device that has been specified by the <tt>-mmcu=</tt> compiler
40 command-line switch. This is done by diverting to the appropriate

```

```

41 file <tt>&lt;avr/io</tt><em>XXXX</em><tt>.h</tt> which should
42 never be included directly. Some register names common to all
43 AVR devices are defined directly within <tt>&lt;avr/common.h</tt>,
44 which is included in <tt>&lt;avr/io.h</tt>,
45 but most of the details come from the respective include file.
46
47 Note that this file always includes the following files:
48 \code
49 #include <avr/sfr_defs.h>
50 #include <avr/portpins.h>
51 #include <avr/common.h>
52 #include <avr/version.h>
53 \endcode
54 See \ref avr_sfr for more details about that header file.
55
56 Included are definitions of the IO register set and their
57 respective bit values as specified in the Atmel documentation.
58 Note that inconsistencies in naming conventions,
59 so even identical functions sometimes get different names on
60 different devices.
61
62 Also included are the specific names useable for interrupt
63 function definitions as documented
64 \ref avr_signames "here".
65
66 Finally, the following macros are defined:
67
68 - \b RAMEND
69 <br>
70 The last on-chip RAM address.
71 <br>
72 - \b XRAMEND
73 <br>
74 The last possible RAM location that is addressable. This is equal to
75 RAMEND for devices that do not allow for external RAM. For devices
76 that allow external RAM, this will be larger than RAMEND.
77 <br>
78 - \b E2END
79 <br>
80 The last EEPROM address.
81 <br>
82 - \b FLASHEND
83 <br>
84 The last byte address in the Flash program space.
85 <br>
86 - \b SPM_PAGESIZE
87 <br>
88 For devices with bootloader support, the flash pagesize
89 (in bytes) to be used for the \c SPM instruction.
90 - \b E2PAGESIZE
91 <br>
92 The size of the EEPROM page.
93
94 */
95
96 #ifndef _AVR_IO_H_
97 #define _AVR_IO_H_
98
99 #include <avr/sfr_defs.h>
100
101 #if defined (__AVR_AT94K__)
102 # include <avr/ioat94k.h>
103 #elif defined (__AVR_AT43USB320__)
104 # include <avr/io43u32x.h>
105 #elif defined (__AVR_AT43USB355__)
106 # include <avr/io43u35x.h>
107 #elif defined (__AVR_AT76C711__)
108 # include <avr/io76c711.h>
109 #elif defined (__AVR_AT86RF401__)
110 # include <avr/io86r401.h>
111 #elif defined (__AVR_AT90PWM1__)
112 # include <avr/io90pwm1.h>
113 #elif defined (__AVR_AT90PWM2__)
114 # include <avr/io90pwm2.h>
115 #elif defined (__AVR_AT90PWM2B__)
116 # include <avr/io90pwm2b.h>
117 #elif defined (__AVR_AT90PWM3__)
118 # include <avr/io90pwm3.h>
119 #elif defined (__AVR_AT90PWM3B__)
120 # include <avr/io90pwm3b.h>
121 #elif defined (__AVR_AT90PWM216__)
122 # include <avr/io90pwm216.h>
123 #elif defined (__AVR_AT90PWM316__)
124 # include <avr/io90pwm316.h>
125 #elif defined (__AVR_AT90PWM161__)
126 # include <avr/io90pwm161.h>
127 #elif defined (__AVR_AT90PWM81__)

```

```
128 # include <avr/io90pwm81.h>
129 #elif defined (__AVR_ATmega8U2__)
130 # include <avr/iom8u2.h>
131 #elif defined (__AVR_ATmega16M1__)
132 # include <avr/iom16m1.h>
133 #elif defined (__AVR_ATmega16U2__)
134 # include <avr/iom16u2.h>
135 #elif defined (__AVR_ATmega16U4__)
136 # include <avr/iom16u4.h>
137 #elif defined (__AVR_ATmega32C1__)
138 # include <avr/iom32c1.h>
139 #elif defined (__AVR_ATmega32M1__)
140 # include <avr/iom32m1.h>
141 #elif defined (__AVR_ATmega32U2__)
142 # include <avr/iom32u2.h>
143 #elif defined (__AVR_ATmega32U4__)
144 # include <avr/iom32u4.h>
145 #elif defined (__AVR_ATmega32U6__)
146 # include <avr/iom32u6.h>
147 #elif defined (__AVR_ATmega64C1__)
148 # include <avr/iom64c1.h>
149 #elif defined (__AVR_ATmega64M1__)
150 # include <avr/iom64m1.h>
151 #elif defined (__AVR_ATmega128__)
152 # include <avr/iom128.h>
153 #elif defined (__AVR_ATmega128A__)
154 # include <avr/iom128a.h>
155 #elif defined (__AVR_ATmega1280__)
156 # include <avr/iom1280.h>
157 #elif defined (__AVR_ATmega1281__)
158 # include <avr/iom1281.h>
159 #elif defined (__AVR_ATmega1284__)
160 # include <avr/iom1284.h>
161 #elif defined (__AVR_ATmega1284P__)
162 # include <avr/iom1284p.h>
163 #elif defined (__AVR_ATmega128RFA1__)
164 # include <avr/iom128rfal.h>
165 #elif defined (__AVR_ATmega128RFR2__)
166 # include <avr/iom1284rfr2.h>
167 #elif defined (__AVR_ATmega128RFR2__)
168 # include <avr/iom128rfr2.h>
169 #elif defined (__AVR_ATmega2564RFR2__)
170 # include <avr/iom2564rfr2.h>
171 #elif defined (__AVR_ATmega256RFR2__)
172 # include <avr/iom256rfr2.h>
173 #elif defined (__AVR_ATmega2560__)
174 # include <avr/iom2560.h>
175 #elif defined (__AVR_ATmega2561__)
176 # include <avr/iom2561.h>
177 #elif defined (__AVR_AT90CAN32__)
178 # include <avr/iocan32.h>
179 #elif defined (__AVR_AT90CAN64__)
180 # include <avr/iocan64.h>
181 #elif defined (__AVR_AT90CAN128__)
182 # include <avr/iocan128.h>
183 #elif defined (__AVR_AT90USB82__)
184 # include <avr/ioub82.h>
185 #elif defined (__AVR_AT90USB162__)
186 # include <avr/ioub162.h>
187 #elif defined (__AVR_AT90USB646__)
188 # include <avr/ioub646.h>
189 #elif defined (__AVR_AT90USB647__)
190 # include <avr/ioub647.h>
191 #elif defined (__AVR_AT90USB1286__)
192 # include <avr/ioub1286.h>
193 #elif defined (__AVR_AT90USB1287__)
194 # include <avr/ioub1287.h>
195 #elif defined (__AVR_ATmega644RFR2__)
196 # include <avr/iom644rfr2.h>
197 #elif defined (__AVR_ATmega64RFR2__)
198 # include <avr/iom64rfr2.h>
199 #elif defined (__AVR_ATmega64__)
200 # include <avr/iom64.h>
201 #elif defined (__AVR_ATmega64A__)
202 # include <avr/iom64a.h>
203 #elif defined (__AVR_ATmega640__)
204 # include <avr/iom640.h>
205 #elif defined (__AVR_ATmega644__)
206 # include <avr/iom644.h>
207 #elif defined (__AVR_ATmega644A__)
208 # include <avr/iom644a.h>
209 #elif defined (__AVR_ATmega644P__)
210 # include <avr/iom644p.h>
211 #elif defined (__AVR_ATmega644PA__)
212 # include <avr/iom644pa.h>
213 #elif defined (__AVR_ATmega645__) || defined (__AVR_ATmega645A__) || defined (__AVR_ATmega645P__)
214 # include <avr/iom645.h>
```

```
215 #elif defined (__AVR_ATmega6450__) || defined (__AVR_ATmega6450A__) || defined (__AVR_ATmega6450P__)
216 # include <avr/iom6450.h>
217 #elif defined (__AVR_ATmega649__) || defined (__AVR_ATmega649A__)
218 # include <avr/iom649.h>
219 #elif defined (__AVR_ATmega6490__) || defined (__AVR_ATmega6490A__) || defined (__AVR_ATmega6490P__)
220 # include <avr/iom6490.h>
221 #elif defined (__AVR_ATmega649P__)
222 # include <avr/iom649p.h>
223 #elif defined (__AVR_ATmega64HVE__)
224 # include <avr/iom64hve.h>
225 #elif defined (__AVR_ATmega64HVE2__)
226 # include <avr/iom64hve2.h>
227 #elif defined (__AVR_ATmega103__)
228 # include <avr/iom103.h>
229 #elif defined (__AVR_ATmega32__)
230 # include <avr/iom32.h>
231 #elif defined (__AVR_ATmega32A__)
232 # include <avr/iom32a.h>
233 #elif defined (__AVR_ATmega323__)
234 # include <avr/iom323.h>
235 #elif defined (__AVR_ATmega324P__) || defined (__AVR_ATmega324A__)
236 # include <avr/iom324.h>
237 #elif defined (__AVR_ATmega324PA__)
238 # include <avr/iom324pa.h>
239 #elif defined (__AVR_ATmega325__) || defined (__AVR_ATmega325A__)
240 # include <avr/iom325.h>
241 #elif defined (__AVR_ATmega325P__)
242 # include <avr/iom325.h>
243 #elif defined (__AVR_ATmega325PA__)
244 # include <avr/iom325pa.h>
245 #elif defined (__AVR_ATmega3250__) || defined (__AVR_ATmega3250A__)
246 # include <avr/iom3250.h>
247 #elif defined (__AVR_ATmega3250P__)
248 # include <avr/iom3250.h>
249 #elif defined (__AVR_ATmega3250PA__)
250 # include <avr/iom3250pa.h>
251 #elif defined (__AVR_ATmega328P__) || defined (__AVR_ATmega328__)
252 # include <avr/iom328p.h>
253 #elif defined (__AVR_ATmega329__) || defined (__AVR_ATmega329A__)
254 # include <avr/iom329.h>
255 #elif defined (__AVR_ATmega329P__) || defined (__AVR_ATmega329PA__)
256 # include <avr/iom329.h>
257 #elif defined (__AVR_ATmega3290__) || defined (__AVR_ATmega3290A__)
258 # include <avr/iom3290.h>
259 #elif defined (__AVR_ATmega3290P__)
260 # include <avr/iom3290.h>
261 #elif defined (__AVR_ATmega3290PA__)
262 # include <avr/iom3290pa.h>
263 #elif defined (__AVR_ATmega32HVB__)
264 # include <avr/iom32hvb.h>
265 #elif defined (__AVR_ATmega32HVBREVB__)
266 # include <avr/iom32hvbrevb.h>
267 #elif defined (__AVR_ATmega406__)
268 # include <avr/iom406.h>
269 #elif defined (__AVR_ATmega16__)
270 # include <avr/iom16.h>
271 #elif defined (__AVR_ATmega16A__)
272 # include <avr/iom16a.h>
273 #elif defined (__AVR_ATmega161__)
274 # include <avr/iom161.h>
275 #elif defined (__AVR_ATmega162__)
276 # include <avr/iom162.h>
277 #elif defined (__AVR_ATmega163__)
278 # include <avr/iom163.h>
279 #elif defined (__AVR_ATmega164P__) || defined (__AVR_ATmega164A__)
280 # include <avr/iom164.h>
281 #elif defined (__AVR_ATmega164PA__)
282 # include <avr/iom164pa.h>
283 #elif defined (__AVR_ATmega165__)
284 # include <avr/iom165.h>
285 #elif defined (__AVR_ATmega165A__)
286 # include <avr/iom165a.h>
287 #elif defined (__AVR_ATmega165P__)
288 # include <avr/iom165p.h>
289 #elif defined (__AVR_ATmega165PA__)
290 # include <avr/iom165pa.h>
291 #elif defined (__AVR_ATmega168__)
292 # include <avr/iom168.h>
293 #elif defined (__AVR_ATmega168A__)
294 # include <avr/iom168a.h>
295 #elif defined (__AVR_ATmega168P__)
296 # include <avr/iom168p.h>
297 #elif defined (__AVR_ATmega168PA__)
298 # include <avr/iom168pa.h>
299 #elif defined (__AVR_ATmega169__) || defined (__AVR_ATmega169A__)
300 # include <avr/iom169.h>
301 #elif defined (__AVR_ATmega169P__)
```

```
302 # include <avr/iom169p.h>
303 #elif defined (__AVR_ATmega169PA__)
304 # include <avr/iom169pa.h>
305 #elif defined (__AVR_ATmega8HVA__)
306 # include <avr/iom8hva.h>
307 #elif defined (__AVR_ATmega16HVA__)
308 # include <avr/iom16hva.h>
309 #elif defined (__AVR_ATmega16HVA2__)
310 # include <avr/iom16hva2.h>
311 #elif defined (__AVR_ATmega16HVB__)
312 # include <avr/iom16hvb.h>
313 #elif defined (__AVR_ATmega16HVBREVB__)
314 # include <avr/iom16hvbrevb.h>
315 #elif defined (__AVR_ATmega8__)
316 # include <avr/iom8.h>
317 #elif defined (__AVR_ATmega8A__)
318 # include <avr/iom8a.h>
319 #elif defined (__AVR_ATmega48__)
320 # include <avr/iom48.h>
321 #elif defined (__AVR_ATmega48A__)
322 # include <avr/iom48a.h>
323 #elif defined (__AVR_ATmega48PA__)
324 # include <avr/iom48pa.h>
325 #elif defined (__AVR_ATmega48PB__)
326 # include <avr/iom48pb.h>
327 #elif defined (__AVR_ATmega48P__)
328 # include <avr/iom48p.h>
329 #elif defined (__AVR_ATmega88__)
330 # include <avr/iom88.h>
331 #elif defined (__AVR_ATmega88A__)
332 # include <avr/iom88a.h>
333 #elif defined (__AVR_ATmega88P__)
334 # include <avr/iom88p.h>
335 #elif defined (__AVR_ATmega88PA__)
336 # include <avr/iom88pa.h>
337 #elif defined (__AVR_ATmega88PB__)
338 # include <avr/iom88pb.h>
339 #elif defined (__AVR_ATmega8515__)
340 # include <avr/iom8515.h>
341 #elif defined (__AVR_ATmega8535__)
342 # include <avr/iom8535.h>
343 #elif defined (__AVR_AT90S8535__)
344 # include <avr/io8535.h>
345 #elif defined (__AVR_AT90C8534__)
346 # include <avr/io8534.h>
347 #elif defined (__AVR_AT90S8515__)
348 # include <avr/io8515.h>
349 #elif defined (__AVR_AT90S4434__)
350 # include <avr/io4434.h>
351 #elif defined (__AVR_AT90S4433__)
352 # include <avr/io4433.h>
353 #elif defined (__AVR_AT90S4414__)
354 # include <avr/io4414.h>
355 #elif defined (__AVR_ATtiny22__)
356 # include <avr/iotn22.h>
357 #elif defined (__AVR_ATtiny26__)
358 # include <avr/iotn26.h>
359 #elif defined (__AVR_AT90S2343__)
360 # include <avr/io2343.h>
361 #elif defined (__AVR_AT90S2333__)
362 # include <avr/io2333.h>
363 #elif defined (__AVR_AT90S2323__)
364 # include <avr/io2323.h>
365 #elif defined (__AVR_AT90S2313__)
366 # include <avr/io2313.h>
367 #elif defined (__AVR_ATtiny4__)
368 # include <avr/iotn4.h>
369 #elif defined (__AVR_ATtiny5__)
370 # include <avr/iotn5.h>
371 #elif defined (__AVR_ATtiny9__)
372 # include <avr/iotn9.h>
373 #elif defined (__AVR_ATtiny10__)
374 # include <avr/iotn10.h>
375 #elif defined (__AVR_ATtiny20__)
376 # include <avr/iotn20.h>
377 #elif defined (__AVR_ATtiny40__)
378 # include <avr/iotn40.h>
379 #elif defined (__AVR_ATtiny2313__)
380 # include <avr/iotn2313.h>
381 #elif defined (__AVR_ATtiny2313A__)
382 # include <avr/iotn2313a.h>
383 #elif defined (__AVR_ATtiny13__)
384 # include <avr/iotn13.h>
385 #elif defined (__AVR_ATtiny13A__)
386 # include <avr/iotn13a.h>
387 #elif defined (__AVR_ATtiny25__)
388 # include <avr/iotn25.h>
```



```
389 #elif defined (__AVR_ATtiny4313__)
390 # include <avr/iotn4313.h>
391 #elif defined (__AVR_ATtiny45__)
392 # include <avr/iotn45.h>
393 #elif defined (__AVR_ATtiny85__)
394 # include <avr/iotn85.h>
395 #elif defined (__AVR_ATtiny24__)
396 # include <avr/iotn24.h>
397 #elif defined (__AVR_ATtiny24A__)
398 # include <avr/iotn24a.h>
399 #elif defined (__AVR_ATtiny44__)
400 # include <avr/iotn44.h>
401 #elif defined (__AVR_ATtiny44A__)
402 # include <avr/iotn44a.h>
403 #elif defined (__AVR_ATtiny441__)
404 # include <avr/iotn441.h>
405 #elif defined (__AVR_ATtiny84__)
406 # include <avr/iotn84.h>
407 #elif defined (__AVR_ATtiny84A__)
408 # include <avr/iotn84a.h>
409 #elif defined (__AVR_ATtiny841__)
410 # include <avr/iotn841.h>
411 #elif defined (__AVR_ATtiny261__)
412 # include <avr/iotn261.h>
413 #elif defined (__AVR_ATtiny261A__)
414 # include <avr/iotn261a.h>
415 #elif defined (__AVR_ATtiny461__)
416 # include <avr/iotn461.h>
417 #elif defined (__AVR_ATtiny461A__)
418 # include <avr/iotn461a.h>
419 #elif defined (__AVR_ATtiny861__)
420 # include <avr/iotn861.h>
421 #elif defined (__AVR_ATtiny861A__)
422 # include <avr/iotn861a.h>
423 #elif defined (__AVR_ATtiny43U__)
424 # include <avr/iotn43u.h>
425 #elif defined (__AVR_ATtiny48__)
426 # include <avr/iotn48.h>
427 #elif defined (__AVR_ATtiny88__)
428 # include <avr/iotn88.h>
429 #elif defined (__AVR_ATtiny828__)
430 # include <avr/iotn828.h>
431 #elif defined (__AVR_ATtiny87__)
432 # include <avr/iotn87.h>
433 #elif defined (__AVR_ATtiny167__)
434 # include <avr/iotn167.h>
435 #elif defined (__AVR_ATtiny1634__)
436 # include <avr/iotn1634.h>
437 #elif defined (__AVR_AT90SCR100__)
438 # include <avr/io90scr100.h>
439 #elif defined (__AVR_ATxmega8E5__)
440 # include <avr/iox8e5.h>
441 #elif defined (__AVR_ATxmega16A4__)
442 # include <avr/iox16a4.h>
443 #elif defined (__AVR_ATxmega16A4U__)
444 # include <avr/iox16a4u.h>
445 #elif defined (__AVR_ATxmega16C4__)
446 # include <avr/iox16c4.h>
447 #elif defined (__AVR_ATxmega16D4__)
448 # include <avr/iox16d4.h>
449 #elif defined (__AVR_ATxmega32A4__)
450 # include <avr/iox32a4.h>
451 #elif defined (__AVR_ATxmega32A4U__)
452 # include <avr/iox32a4u.h>
453 #elif defined (__AVR_ATxmega32C3__)
454 # include <avr/iox32c3.h>
455 #elif defined (__AVR_ATxmega32C4__)
456 # include <avr/iox32c4.h>
457 #elif defined (__AVR_ATxmega32D3__)
458 # include <avr/iox32d3.h>
459 #elif defined (__AVR_ATxmega32D4__)
460 # include <avr/iox32d4.h>
461 #elif defined (__AVR_ATxmega32E5__)
462 # include <avr/iox32e5.h>
463 #elif defined (__AVR_ATxmega64A1__)
464 # include <avr/iox64a1.h>
465 #elif defined (__AVR_ATxmega64A1U__)
466 # include <avr/iox64a1u.h>
467 #elif defined (__AVR_ATxmega64A3__)
468 # include <avr/iox64a3.h>
469 #elif defined (__AVR_ATxmega64A3U__)
470 # include <avr/iox64a3u.h>
471 #elif defined (__AVR_ATxmega64A4U__)
472 # include <avr/iox64a4u.h>
473 #elif defined (__AVR_ATxmega64B1__)
474 # include <avr/iox64b1.h>
475 #elif defined (__AVR_ATxmega64B3__)
```

```
476 # include <avr/iox64b3.h>
477 #elif defined (__AVR_ATxmega64C3__)
478 # include <avr/iox64c3.h>
479 #elif defined (__AVR_ATxmega64D3__)
480 # include <avr/iox64d3.h>
481 #elif defined (__AVR_ATxmega64D4__)
482 # include <avr/iox64d4.h>
483 #elif defined (__AVR_ATxmega128A1__)
484 # include <avr/iox128a1.h>
485 #elif defined (__AVR_ATxmega128A1U__)
486 # include <avr/iox128alu.h>
487 #elif defined (__AVR_ATxmega128A4U__)
488 # include <avr/iox128a4u.h>
489 #elif defined (__AVR_ATxmega128A3__)
490 # include <avr/iox128a3.h>
491 #elif defined (__AVR_ATxmega128A3U__)
492 # include <avr/iox128a3u.h>
493 #elif defined (__AVR_ATxmega128B1__)
494 # include <avr/iox128b1.h>
495 #elif defined (__AVR_ATxmega128B3__)
496 # include <avr/iox128b3.h>
497 #elif defined (__AVR_ATxmega128C3__)
498 # include <avr/iox128c3.h>
499 #elif defined (__AVR_ATxmega128D3__)
500 # include <avr/iox128d3.h>
501 #elif defined (__AVR_ATxmega128D4__)
502 # include <avr/iox128d4.h>
503 #elif defined (__AVR_ATxmega192A3__)
504 # include <avr/iox192a3.h>
505 #elif defined (__AVR_ATxmega192A3U__)
506 # include <avr/iox192a3u.h>
507 #elif defined (__AVR_ATxmega192C3__)
508 # include <avr/iox192c3.h>
509 #elif defined (__AVR_ATxmega192D3__)
510 # include <avr/iox192d3.h>
511 #elif defined (__AVR_ATxmega256A3__)
512 # include <avr/iox256a3.h>
513 #elif defined (__AVR_ATxmega256A3U__)
514 # include <avr/iox256a3u.h>
515 #elif defined (__AVR_ATxmega256A3B__)
516 # include <avr/iox256a3b.h>
517 #elif defined (__AVR_ATxmega256A3BU__)
518 # include <avr/iox256a3bu.h>
519 #elif defined (__AVR_ATxmega256C3__)
520 # include <avr/iox256c3.h>
521 #elif defined (__AVR_ATxmega256D3__)
522 # include <avr/iox256d3.h>
523 #elif defined (__AVR_ATxmega384C3__)
524 # include <avr/iox384c3.h>
525 #elif defined (__AVR_ATxmega384D3__)
526 # include <avr/iox384d3.h>
527 #elif defined (__AVR_ATA5702M322__)
528 # include <avr/ioa5702m322.h>
529 #elif defined (__AVR_ATA5782__)
530 # include <avr/ioa5782.h>
531 #elif defined (__AVR_ATA5790__)
532 # include <avr/ioa5790.h>
533 #elif defined (__AVR_ATA5790N__)
534 # include <avr/ioa5790n.h>
535 #elif defined (__AVR_ATA5831__)
536 # include <avr/ioa5831.h>
537 #elif defined (__AVR_ATA5272__)
538 # include <avr/ioa5272.h>
539 #elif defined (__AVR_ATA5505__)
540 # include <avr/ioa5505.h>
541 #elif defined (__AVR_ATA5795__)
542 # include <avr/ioa5795.h>
543 #elif defined (__AVR_ATA6285__)
544 # include <avr/ioa6285.h>
545 #elif defined (__AVR_ATA6286__)
546 # include <avr/ioa6286.h>
547 #elif defined (__AVR_ATA6289__)
548 # include <avr/ioa6289.h>
549 #elif defined (__AVR_ATA6612C__)
550 # include <avr/ioa6612c.h>
551 #elif defined (__AVR_ATA6613C__)
552 # include <avr/ioa6613c.h>
553 #elif defined (__AVR_ATA6614Q__)
554 # include <avr/ioa6614q.h>
555 #elif defined (__AVR_ATA6616C__)
556 # include <avr/ioa6616c.h>
557 #elif defined (__AVR_ATA6617C__)
558 # include <avr/ioa6617c.h>
559 #elif defined (__AVR_ATA664251__)
560 # include <avr/ioa664251.h>
561 /* avr1: the following only supported for assembler programs */
562 #elif defined (__AVR_ATtiny28__)
```

```

563 # include <avr/iotn28.h>
564 #elif defined (__AVR_AT90S1200__)
565 # include <avr/io1200.h>
566 #elif defined (__AVR_ATtiny15__)
567 # include <avr/iotn15.h>
568 #elif defined (__AVR_ATtiny12__)
569 # include <avr/iotn12.h>
570 #elif defined (__AVR_ATtiny11__)
571 # include <avr/iotn11.h>
572 #elif defined (__AVR_M3000__)
573 # include <avr/iom3000.h>
574 #elif defined (__AVR_DEV_LIB_NAME__)
575 # define __concat__(a,b) a##b
576 # define __header1__(a,b) __concat__(a,b)
577 # define __AVR_DEVICE_HEADER__ <avr/__header1__(io,__AVR_DEV_LIB_NAME__)>.h>
578 # include __AVR_DEVICE_HEADER__
579 #else
580 # if !defined(__COMPILING_AVR_LIBC__)
581 # warning "device type not defined"
582 # endif
583 #endif
584
585 #include <avr/portpins.h>
586
587 #include <avr/common.h>
588
589 #include <avr/version.h>
590
591 #if __AVR_ARCH__ >= 100
592 # include <avr/xmega.h>
593 #endif
594
595 /* Include fuse.h after individual IO header files. */
596 #include <avr/fuse.h>
597
598 /* Include lock.h after individual IO header files. */
599 #include <avr/lock.h>
600
601 #endif /* _AVR_IO_H_ */

```

25.21 lock.h File Reference

25.22 lock.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2007, Atmel Corporation
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: lock.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
32
33 /* avr/lock.h - Lock Bits API */
34
35 #ifndef _AVR_LOCK_H_

```

```

36 #define _AVR_LOCK_H_ 1
37
38
39 /** \file */
40 /** \defgroup avr_lock <avr/lock.h>: Lockbit Support
41
42 \par Introduction
43
44 The Lockbit API allows a user to specify the lockbit settings for the
45 specific AVR device they are compiling for. These lockbit settings will be
46 placed in a special section in the ELF output file, after linking.
47
48 Programming tools can take advantage of the lockbit information embedded in
49 the ELF file, by extracting this information and determining if the lockbits
50 need to be programmed after programming the Flash and EEPROM memories.
51 This also allows a single ELF file to contain all the
52 information needed to program an AVR.
53
54 To use the Lockbit API, include the <avr/io.h> header file, which in turn
55 automatically includes the individual I/O header file and the <avr/lock.h>
56 file. These other two files provides everything necessary to set the AVR
57 lockbits.
58
59 \par Lockbit API
60
61 Each I/O header file may define up to 3 macros that controls what kinds
62 of lockbits are available to the user.
63
64 If __LOCK_BITS_EXIST is defined, then two lock bits are available to the
65 user and 3 mode settings are defined for these two bits.
66
67 If __BOOT_LOCK_BITS_0_EXIST is defined, then the two BLB0 lock bits are
68 available to the user and 4 mode settings are defined for these two bits.
69
70 If __BOOT_LOCK_BITS_1_EXIST is defined, then the two BLB1 lock bits are
71 available to the user and 4 mode settings are defined for these two bits.
72
73 If __BOOT_LOCK_APPLICATION_TABLE_BITS_EXIST is defined then two lock bits
74 are available to set the locking mode for the Application Table Section
75 (which is used in the XMEGA family).
76
77 If __BOOT_LOCK_APPLICATION_BITS_EXIST is defined then two lock bits are
78 available to set the locking mode for the Application Section (which is used
79 in the XMEGA family).
80
81 If __BOOT_LOCK_BOOT_BITS_EXIST is defined then two lock bits are available
82 to set the locking mode for the Boot Loader Section (which is used in the
83 XMEGA family).
84
85 The AVR lockbit modes have inverted values, logical 1 for an unprogrammed
86 (disabled) bit and logical 0 for a programmed (enabled) bit. The defined
87 macros for each individual lock bit represent this in their definition by a
88 bit-wise inversion of a mask. For example, the LB_MODE_3 macro is defined
89 as:
90 \code
91 #define LB_MODE_3 (0xFC)
92 ` \endcode
93
94 To combine the lockbit mode macros together to represent a whole byte,
95 use the bitwise AND operator, like so:
96 \code
97 (LB_MODE_3 & BLB0_MODE_2)
98 \endcode
99
100 <avr/lock.h> also defines a macro that provides a default lockbit value:
101 LOCKBITS_DEFAULT which is defined to be 0xFF.
102
103 See the AVR device specific datasheet for more details about these
104 lock bits and the available mode settings.
105
106 A convenience macro, LOCKMEM, is defined as a GCC attribute for a
107 custom-named section of ".lock".
108
109 A convenience macro, LOCKBITS, is defined that declares a variable, __lock,
110 of type unsigned char with the attribute defined by LOCKMEM. This variable
111 allows the end user to easily set the lockbit data.
112
113 \note If a device-specific I/O header file has previously defined LOCKMEM,
114 then LOCKMEM is not redefined. If a device-specific I/O header file has
115 previously defined LOCKBITS, then LOCKBITS is not redefined. LOCKBITS is
116 currently known to be defined in the I/O header files for the XMEGA devices.
117
118 \par API Usage Example
119
120 Putting all of this together is easy:
121
122 \code

```

```

123 #include <avr/io.h>
124
125 LOCKBITS = (LB_MODE_1 & BLB0_MODE_3 & BLB1_MODE_4);
126
127 int main(void)
128 {
129     return 0;
130 }
131 \endcode
132
133 Or:
134
135 \code
136 #include <avr/io.h>
137
138 unsigned char __lock __attribute__((section (".lock"))) =
139 (LB_MODE_1 & BLB0_MODE_3 & BLB1_MODE_4);
140
141 int main(void)
142 {
143     return 0;
144 }
145 \endcode
146
147
148
149 However there are a number of caveats that you need to be aware of to
150 use this API properly.
151
152 Be sure to include <avr/io.h> to get all of the definitions for the API.
153 The LOCKBITS macro defines a global variable to store the lockbit data. This
154 variable is assigned to its own linker section. Assign the desired lockbit
155 values immediately in the variable initialization.
156
157 The .lock section in the ELF file will get its values from the initial
158 variable assignment ONLY. This means that you can NOT assign values to
159 this variable in functions and the new values will not be put into the
160 ELF .lock section.
161
162 The global variable is declared in the LOCKBITS macro has two leading
163 underscores, which means that it is reserved for the "implementation",
164 meaning the library, so it will not conflict with a user-named variable.
165
166 You must initialize the lockbit variable to some meaningful value, even
167 if it is the default value. This is because the lockbits default to a
168 logical 1, meaning unprogrammed. Normal uninitialized data defaults to all
169 logical zeros. So it is vital that all lockbits are initialized, even with
170 default data. If they are not, then the lockbits may not be programmed to the
171 desired settings and can possibly put your device into an unrecoverable
172 state.
173
174 Be sure to have the -mmcu=<em>device</em> flag in your compile command line and
175 your linker command line to have the correct device selected and to have
176 the correct I/O header file included when you include <avr/io.h>.
177
178 You can print out the contents of the .lock section in the ELF file by
179 using this command line:
180 \code
181 avr-objdump -s -j .lock <ELF file>
182 \endcode
183
184 */
185
186
187 #if !(defined(__ASSEMBLER__) || defined(__DOXYGEN__))
188
189 #ifndef LOCKMEM
190 #define LOCKMEM __attribute__((__used__, __section__ (".lock")))
191 #endif
192
193 #ifndef LOCKBITS
194 #define LOCKBITS unsigned char __lock LOCKMEM
195 #endif
196
197 /* Lock Bit Modes */
198 #if defined(__LOCK_BITS_EXIST)
199 #define LB_MODE_1 (0xFF)
200 #define LB_MODE_2 (0xFE)
201 #define LB_MODE_3 (0xFC)
202 #endif
203
204 #if defined(__BOOT_LOCK_BITS_0_EXIST)
205 #define BLB0_MODE_1 (0xFF)
206 #define BLB0_MODE_2 (0xFB)
207 #define BLB0_MODE_3 (0xF3)
208 #define BLB0_MODE_4 (0xF7)
209 #endif

```

```

210
211 #if defined(__BOOT_LOCK_BITS_1_EXIST)
212 #define BLB1_MODE_1    (0xFF)
213 #define BLB1_MODE_2    (0xEF)
214 #define BLB1_MODE_3    (0xCF)
215 #define BLB1_MODE_4    (0xDF)
216 #endif
217
218 #if defined(__BOOT_LOCK_APPLICATION_TABLE_BITS_EXIST)
219 #define BLBAT0 ~_BV(2)
220 #define BLBAT1 ~_BV(3)
221 #endif
222
223 #if defined(__BOOT_LOCK_APPLICATION_BITS_EXIST)
224 #define BLBA0 ~_BV(4)
225 #define BLBA1 ~_BV(5)
226 #endif
227
228 #if defined(__BOOT_LOCK_BOOT_BITS_EXIST)
229 #define BLBB0 ~_BV(6)
230 #define BLBB1 ~_BV(7)
231 #endif
232
233
234 #define LOCKBITS_DEFAULT (0xFF)
235
236 #endif /* !(__ASSEMBLER || __DOXYGEN__) */
237
238
239 #endif /* _AVR_LOCK_H_ */

```

25.23 pgmspace.h File Reference

Macros

- #define [PROGMEM](#) __ATTR_PROGMEM__
- #define [PGM_P](#) const char *
- #define [PGM_VOID_P](#) const void *
- #define [PSTR](#)(s) ((const [PROGMEM](#) char *) (s))
- #define [pgm_read_byte_near](#)(address_short) __LPM__((uint16_t)(address_short))
- #define [pgm_read_word_near](#)(address_short) __LPM_word__((uint16_t)(address_short))
- #define [pgm_read_dword_near](#)(address_short) __LPM_dword__((uint16_t)(address_short))
- #define [pgm_read_float_near](#)(address_short) __LPM_float__((uint16_t)(address_short))
- #define [pgm_read_ptr_near](#)(address_short) (void*) __LPM_word__((uint16_t)(address_short))
- #define [pgm_read_byte_far](#)(address_long) __ELPM__((uint32_t)(address_long))
- #define [pgm_read_word_far](#)(address_long) __ELPM_word__((uint32_t)(address_long))
- #define [pgm_read_dword_far](#)(address_long) __ELPM_dword__((uint32_t)(address_long))
- #define [pgm_read_float_far](#)(address_long) __ELPM_float__((uint32_t)(address_long))
- #define [pgm_read_ptr_far](#)(address_long) (void*) __ELPM_word__((uint32_t)(address_long))
- #define [pgm_read_byte](#)(address_short) [pgm_read_byte_near](#)(address_short)
- #define [pgm_read_word](#)(address_short) [pgm_read_word_near](#)(address_short)
- #define [pgm_read_dword](#)(address_short) [pgm_read_dword_near](#)(address_short)
- #define [pgm_read_float](#)(address_short) [pgm_read_float_near](#)(address_short)
- #define [pgm_read_ptr](#)(address_short) [pgm_read_ptr_near](#)(address_short)
- #define [pgm_get_far_address](#)(var)

Typedefs

- typedef void [PROGMEM](#) [prog_void](#)
- typedef char [PROGMEM](#) [prog_char](#)
- typedef unsigned char [PROGMEM](#) [prog_uchar](#)
- typedef int8_t [PROGMEM](#) [prog_int8_t](#)
- typedef uint8_t [PROGMEM](#) [prog_uint8_t](#)

- typedef [int16_t](#) [PROGMEM prog_int16_t](#)
- typedef [uint16_t](#) [PROGMEM prog_uint16_t](#)
- typedef [int32_t](#) [PROGMEM prog_int32_t](#)
- typedef [uint32_t](#) [PROGMEM prog_uint32_t](#)
- typedef [int64_t](#) [PROGMEM prog_int64_t](#)
- typedef [uint64_t](#) [PROGMEM prog_uint64_t](#)

Functions

- const void * [memchr_P](#) (const void *, int __val, size_t __len)
- int [memcmp_P](#) (const void *, const void *, size_t) [__ATTR_PURE__](#)
- void * [memcpy_P](#) (void *, const void *, int __val, size_t)
- void * [memcpy_P](#) (void *, const void *, size_t)
- void * [memset_P](#) (const void *, size_t, const void *, size_t) [__ATTR_PURE__](#)
- 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 *) [__ATTR_PURE__](#)
- char * [strcpy_P](#) (char *, const char *)
- int [strcasecmp_P](#) (const char *, const char *) [__ATTR_PURE__](#)
- char * [strcasestr_P](#) (const char *, const char *) [__ATTR_PURE__](#)
- size_t [strcspn_P](#) (const char *__s, const char *__reject) [__ATTR_PURE__](#)
- 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) [__ATTR_PURE__](#)
- int [strncasecmp_P](#) (const char *, const char *, size_t) [__ATTR_PURE__](#)
- 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) [__ATTR_PURE__](#)
- 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) [__ATTR_PURE__](#)
- char * [strstr_P](#) (const char *, const char *) [__ATTR_PURE__](#)
- char * [strtok_P](#) (char *__s, const char *__delim)
- char * [strtok_r_P](#) (char *__s, const char *__delim, char **__last)
- size_t [strlen_PF](#) ([uint_farptr_t](#) src)
- size_t [strlen_P](#) ([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) [__ATTR_PURE__](#)
- int [strncmp_PF](#) (const char *s1, [uint_farptr_t](#) s2, size_t n) [__ATTR_PURE__](#)
- int [strcasecmp_PF](#) (const char *s1, [uint_farptr_t](#) s2) [__ATTR_PURE__](#)
- int [strncasecmp_PF](#) (const char *s1, [uint_farptr_t](#) s2, size_t n) [__ATTR_PURE__](#)
- char * [strstr_PF](#) (const char *s1, [uint_farptr_t](#) s2)
- size_t [strncpy_PF](#) (char *dst, [uint_farptr_t](#) src, size_t siz)
- int [memcmp_PF](#) (const void *, [uint_farptr_t](#), size_t) [__ATTR_PURE__](#)
- static size_t [strlen_P](#) (const char *s)

25.24 pgmspace.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002-2007 Marek Michalkiewicz
2 Copyright (c) 2006, Carlos Lamas
3 Copyright (c) 2009-2010, Jan Wacławek
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 * Redistributions of source code must retain the above copyright
10 notice, this list of conditions and the following disclaimer.
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: pgmspace.h 2544 2017-08-04 08:50:27Z pitchumani $ */
32
33 /*
34 pgmspace.h
35
36 Contributors:
37 Created by Marek Michalkiewicz <marekm@linux.org.pl>
38 Eric B. Weddington <eric@ecentral.com>
39 Wolfgang Haidinger <wh@vmars.tuwien.ac.at> (pgm_read_dword())
40 Ivanov Anton <anton@arc.com.ru> (pgm_read_float())
41 */
42
43 /** \file */
44 /** \defgroup avr_pgmspace <avr/pgmspace.h>: Program Space Utilities
45 \code
46 #include <avr/io.h>
47 #include <avr/pgmspace.h>
48 \endcode
49
50 The functions in this module provide interfaces for a program to access
51 data stored in program space (flash memory) of the device. In order to
52 use these functions, the target device must support either the \c LPM or
53 \c ELPM instructions.
54
55 \note These functions are an attempt to provide some compatibility with
56 header files that come with IAR C, to make porting applications between
57 different compilers easier. This is not 100% compatibility though (GCC
58 does not have full support for multiple address spaces yet).
59
60 \note If you are working with strings which are completely based in ram,
61 use the standard string functions described in \ref avr_string.
62
63 \note If possible, put your constant tables in the lower 64 KB and use
64 pgm_read_byte_near() or pgm_read_word_near() instead of
65 pgm_read_byte_far() or pgm_read_word_far() since it is more efficient that
66 way, and you can still use the upper 64K for executable code.
67 All functions that are suffixed with a \c _P \e require their
68 arguments to be in the lower 64 KB of the flash ROM, as they do
69 not use ELPM instructions. This is normally not a big concern as
70 the linker setup arranges any program space constants declared
71 using the macros from this header file so they are placed right after
72 the interrupt vectors, and in front of any executable code. However,
73 it can become a problem if there are too many of these constants, or
74 for bootloaders on devices with more than 64 KB of ROM.
75 <em>All these functions will not work in that situation.</em>
76
77 \note For <b>Xmega</b> devices, make sure the NVM controller
78 command register (\c NVM_CMD or \c NVM_CMD) is set to 0x00 (NOP)
79 before using any of these functions.
80 */
81
82 #ifndef __PGMSPACE_H_
83 #define __PGMSPACE_H_ 1

```



```

84
85 #ifndef __DOXYGEN__
86 #define __need_size_t
87 #endif
88 #include <inttypes.h>
89 #include <stddef.h>
90 #include <avr/io.h>
91
92 #ifndef __DOXYGEN__
93 #ifndef __ATTR_CONST__
94 #define __ATTR_CONST__ __attribute__((__const__))
95 #endif
96
97 #ifndef __ATTR_PROGMEM__
98 #define __ATTR_PROGMEM__ __attribute__((__progmem__))
99 #endif
100
101 #ifndef __ATTR_PURE__
102 #define __ATTR_PURE__ __attribute__((__pure__))
103 #endif
104 #endif /* !__DOXYGEN__ */
105
106 /**
107 \ingroup avr_pgmspace
108 \def PROGMEM
109
110 Attribute to use in order to declare an object being located in
111 flash ROM.
112 */
113 #define PROGMEM __ATTR_PROGMEM__
114
115 #ifdef __cplusplus
116 extern "C" {
117 #endif
118
119 #if defined(__DOXYGEN__)
120 /*
121 * Doxygen doesn't grok the appended attribute syntax of
122 * GCC, and confuses the typedefs with function decls, so
123 * supply a doxygen-friendly view.
124 */
125
126 /**
127 \ingroup avr_pgmspace
128 \typedef prog_void
129 \note DEPRECATED
130
131 This typedef is now deprecated because the usage of the __progmem__
132 attribute on a type is not supported in GCC. However, the use of the
133 __progmem__ attribute on a variable declaration is supported, and this is
134 now the recommended usage.
135
136 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
137 has been defined before including <avr/pgmspace.h> (either by a
138 \c \#define directive, or by a -D compiler option.)
139
140 Type of a "void" object located in flash ROM. Does not make much
141 sense by itself, but can be used to declare a "void *" object in
142 flash ROM.
143 */
144 typedef void PROGMEM prog_void;
145
146 /**
147 \ingroup avr_pgmspace
148 \typedef prog_char
149 \note DEPRECATED
150
151 This typedef is now deprecated because the usage of the __progmem__
152 attribute on a type is not supported in GCC. However, the use of the
153 __progmem__ attribute on a variable declaration is supported, and this is
154 now the recommended usage.
155
156 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
157 has been defined before including <avr/pgmspace.h> (either by a
158 \c \#define directive, or by a -D compiler option.)
159
160 Type of a "char" object located in flash ROM.
161 */
162 typedef char PROGMEM prog_char;
163
164 /**
165 \ingroup avr_pgmspace
166 \typedef prog_uchar
167 \note DEPRECATED
168
169 This typedef is now deprecated because the usage of the __progmem__
170 attribute on a type is not supported in GCC. However, the use of the

```

```

171 __progmem__ attribute on a variable declaration is supported, and this is
172 now the recommended usage.
173
174 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
175 has been defined before including <avr/pgmspace.h> (either by a
176 \c \#define directive, or by a -D compiler option.)
177
178 Type of an "unsigned char" object located in flash ROM.
179 */
180 typedef unsigned char PROGMEM prog_uchar;
181
182 /**
183 \ingroup avr_pgmspace
184 \typedef prog_int8_t
185 \note DEPRECATED
186
187 This typedef is now deprecated because the usage of the __progmem__
188 attribute on a type is not supported in GCC. However, the use of the
189 __progmem__ attribute on a variable declaration is supported, and this is
190 now the recommended usage.
191
192 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
193 has been defined before including <avr/pgmspace.h> (either by a
194 \c \#define directive, or by a -D compiler option.)
195
196 Type of an "int8_t" object located in flash ROM.
197 */
198 typedef int8_t PROGMEM prog_int8_t;
199
200 /**
201 \ingroup avr_pgmspace
202 \typedef prog_uint8_t
203 \note DEPRECATED
204
205 This typedef is now deprecated because the usage of the __progmem__
206 attribute on a type is not supported in GCC. However, the use of the
207 __progmem__ attribute on a variable declaration is supported, and this is
208 now the recommended usage.
209
210 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
211 has been defined before including <avr/pgmspace.h> (either by a
212 \c \#define directive, or by a -D compiler option.)
213
214 Type of an "uint8_t" object located in flash ROM.
215 */
216 typedef uint8_t PROGMEM prog_uint8_t;
217
218 /**
219 \ingroup avr_pgmspace
220 \typedef prog_int16_t
221 \note DEPRECATED
222
223 This typedef is now deprecated because the usage of the __progmem__
224 attribute on a type is not supported in GCC. However, the use of the
225 __progmem__ attribute on a variable declaration is supported, and this is
226 now the recommended usage.
227
228 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
229 has been defined before including <avr/pgmspace.h> (either by a
230 \c \#define directive, or by a -D compiler option.)
231
232 Type of an "int16_t" object located in flash ROM.
233 */
234 typedef int16_t PROGMEM prog_int16_t;
235
236 /**
237 \ingroup avr_pgmspace
238 \typedef prog_uint16_t
239 \note DEPRECATED
240
241 This typedef is now deprecated because the usage of the __progmem__
242 attribute on a type is not supported in GCC. However, the use of the
243 __progmem__ attribute on a variable declaration is supported, and this is
244 now the recommended usage.
245
246 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
247 has been defined before including <avr/pgmspace.h> (either by a
248 \c \#define directive, or by a -D compiler option.)
249
250 Type of an "uint16_t" object located in flash ROM.
251 */
252 typedef uint16_t PROGMEM prog_uint16_t;
253
254 /**
255 \ingroup avr_pgmspace
256 \typedef prog_int32_t
257 \note DEPRECATED

```

```

258
259 This typedef is now deprecated because the usage of the __progmem__
260 attribute on a type is not supported in GCC. However, the use of the
261 __progmem__ attribute on a variable declaration is supported, and this is
262 now the recommended usage.
263
264 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
265 has been defined before including <avr/pgmspace.h> (either by a
266 \c \#define directive, or by a -D compiler option.)
267
268 Type of an "int32_t" object located in flash ROM.
269 */
270 typedef int32_t PROGMEM prog_int32_t;
271
272 /**
273 \ingroup avr_pgmspace
274 \typedef prog_uint32_t
275 \note DEPRECATED
276
277 This typedef is now deprecated because the usage of the __progmem__
278 attribute on a type is not supported in GCC. However, the use of the
279 __progmem__ attribute on a variable declaration is supported, and this is
280 now the recommended usage.
281
282 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
283 has been defined before including <avr/pgmspace.h> (either by a
284 \c \#define directive, or by a -D compiler option.)
285
286 Type of an "uint32_t" object located in flash ROM.
287 */
288 typedef uint32_t PROGMEM prog_uint32_t;
289
290 /**
291 \ingroup avr_pgmspace
292 \typedef prog_int64_t
293 \note DEPRECATED
294
295 This typedef is now deprecated because the usage of the __progmem__
296 attribute on a type is not supported in GCC. However, the use of the
297 __progmem__ attribute on a variable declaration is supported, and this is
298 now the recommended usage.
299
300 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
301 has been defined before including <avr/pgmspace.h> (either by a
302 \c \#define directive, or by a -D compiler option.)
303
304 Type of an "int64_t" object located in flash ROM.
305
306 \note This type is not available when the compiler
307 option -mint8 is in effect.
308 */
309 typedef int64_t PROGMEM prog_int64_t;
310
311 /**
312 \ingroup avr_pgmspace
313 \typedef prog_uint64_t
314 \note DEPRECATED
315
316 This typedef is now deprecated because the usage of the __progmem__
317 attribute on a type is not supported in GCC. However, the use of the
318 __progmem__ attribute on a variable declaration is supported, and this is
319 now the recommended usage.
320
321 The typedef is only visible if the macro __PROG_TYPES_COMPAT__
322 has been defined before including <avr/pgmspace.h> (either by a
323 \c \#define directive, or by a -D compiler option.)
324
325 Type of an "uint64_t" object located in flash ROM.
326
327 \note This type is not available when the compiler
328 option -mint8 is in effect.
329 */
330 typedef uint64_t PROGMEM prog_uint64_t;
331
332 /** \ingroup avr_pgmspace
333 \def PGM_P
334
335 Used to declare a variable that is a pointer to a string in program
336 space. */
337
338 #ifndef PGM_P
339 #define PGM_P const char *
340 #endif
341
342 /** \ingroup avr_pgmspace
343 \def PGM_VOID_P
344

```

```

345 Used to declare a generic pointer to an object in program space.  */
346
347 #ifndef PGM_VOID_P
348 #define PGM_VOID_P const void *
349 #endif
350
351 #elif defined(__PROG_TYPES_COMPAT__) /* !DOXYGEN */
352
353 typedef void prog_void __attribute__((__progmem__, deprecated("prog_void type is deprecated.")));
354 typedef char prog_char __attribute__((__progmem__, deprecated("prog_char type is deprecated.")));
355 typedef unsigned char prog_uchar __attribute__((__progmem__, deprecated("prog_uchar type is
    deprecated.")));
356 typedef int8_t prog_int8_t __attribute__((__progmem__, deprecated("prog_int8_t type is
    deprecated.")));
357 typedef uint8_t prog_uint8_t __attribute__((__progmem__, deprecated("prog_uint8_t type is
    deprecated.")));
358 typedef int16_t prog_int16_t __attribute__((__progmem__, deprecated("prog_int16_t type is
    deprecated.")));
359 typedef uint16_t prog_uint16_t __attribute__((__progmem__, deprecated("prog_uint16_t type is
    deprecated.")));
360 typedef int32_t prog_int32_t __attribute__((__progmem__, deprecated("prog_int32_t type is
    deprecated.")));
361 typedef uint32_t prog_uint32_t __attribute__((__progmem__, deprecated("prog_uint32_t type is
    deprecated.")));
362 #if !__USING_MINT8
363 typedef int64_t prog_int64_t __attribute__((__progmem__, deprecated("prog_int64_t type is
    deprecated.")));
364 typedef uint64_t prog_uint64_t __attribute__((__progmem__, deprecated("prog_uint64_t type is
    deprecated.")));
365 #endif
366
367 #ifndef PGM_P
368 #define PGM_P const prog_char *
369 #endif
370
371 #ifndef PGM_VOID_P
372 #define PGM_VOID_P const prog_void *
373 #endif
374
375 #else /* !defined(__DOXYGEN__), !defined(__PROG_TYPES_COMPAT__) */
376
377 #ifndef PGM_P
378 #define PGM_P const char *
379 #endif
380
381 #ifndef PGM_VOID_P
382 #define PGM_VOID_P const void *
383 #endif
384 #endif /* defined(__DOXYGEN__), defined(__PROG_TYPES_COMPAT__) */
385
386 /* Although in C, we can get away with just using __c, it does not work in
387 C++. We need to use &__c[0] to avoid the compiler puking. Dave Hylands
388 explained it thusly,
389
390 Let's suppose that we use PSTR("Test"). In this case, the type returned
391 by __c is a prog_char[5] and not a prog_char *. While these are
392 compatible, they aren't the same thing (especially in C++). The type
393 returned by &__c[0] is a prog_char *, which explains why it works
394 fine. */
395
396 #if defined(__DOXYGEN__)
397 /*
398 * The #define below is just a dummy that serves documentation
399 * purposes only.
400 */
401 /** \ingroup avr_pgmspace
402 \def PSTR(s)
403
404 Used to declare a static pointer to a string in program space.  */
405 # define PSTR(s) ((const PROGMEM char *) (s))
406 #else /* !DOXYGEN */
407 /* The real thing. */
408 # define PSTR(s) (__extension__({static const char __c[] PROGMEM = (s); &__c[0];}))
409 #endif /* DOXYGEN */
410
411 #ifndef __DOXYGEN__ /* Internal macros, not documented. */
412 #define __LPM_classic__(addr) \
413   (__extension__({ \
414     uint16_t __addr16 = (uint16_t) (addr); \
415     uint8_t __result; \
416     __asm__ __volatile__ \
417     ( \
418       "lpm" "\n\t" \
419       "mov %0, r0" "\n\t" \
420       : "=r" (__result) \
421       : "z" (__addr16) \
422       : "r0"

```

```

423     );
424     __result;
425 )))
426
427 #define __LPM_enhanced__(addr) \
428 (__extension__({ \
429 uint16_t __addr16 = (uint16_t)(addr); \
430 uint8_t __result; \
431 __asm__ __volatile__ \
432 ( \
433 "lpm %0, Z" "\n\t" \
434 : "=r" (__result) \
435 : "z" (__addr16) \
436 ); \
437 __result; \
438 )))
439
440 #define __LPM_word_classic__(addr) \
441 (__extension__({ \
442 uint16_t __addr16 = (uint16_t)(addr); \
443 uint16_t __result; \
444 __asm__ __volatile__ \
445 ( \
446 "lpm" "\n\t" \
447 "mov %A0, r0" "\n\t" \
448 "adiw r30, 1" "\n\t" \
449 "lpm" "\n\t" \
450 "mov %B0, r0" "\n\t" \
451 : "=r" (__result), "=z" (__addr16) \
452 : "1" (__addr16) \
453 : "r0" \
454 ); \
455 __result; \
456 )))
457
458 #define __LPM_word_enhanced__(addr) \
459 (__extension__({ \
460 uint16_t __addr16 = (uint16_t)(addr); \
461 uint16_t __result; \
462 __asm__ __volatile__ \
463 ( \
464 "lpm %A0, Z+" "\n\t" \
465 "lpm %B0, Z" "\n\t" \
466 : "=r" (__result), "=z" (__addr16) \
467 : "1" (__addr16) \
468 ); \
469 __result; \
470 )))
471
472 #define __LPM_dword_classic__(addr) \
473 (__extension__({ \
474 uint16_t __addr16 = (uint16_t)(addr); \
475 uint32_t __result; \
476 __asm__ __volatile__ \
477 ( \
478 "lpm" "\n\t" \
479 "mov %A0, r0" "\n\t" \
480 "adiw r30, 1" "\n\t" \
481 "lpm" "\n\t" \
482 "mov %B0, r0" "\n\t" \
483 "adiw r30, 1" "\n\t" \
484 "lpm" "\n\t" \
485 "mov %C0, r0" "\n\t" \
486 "adiw r30, 1" "\n\t" \
487 "lpm" "\n\t" \
488 "mov %D0, r0" "\n\t" \
489 : "=r" (__result), "=z" (__addr16) \
490 : "1" (__addr16) \
491 : "r0" \
492 ); \
493 __result; \
494 )))
495
496 #define __LPM_dword_enhanced__(addr) \
497 (__extension__({ \
498 uint16_t __addr16 = (uint16_t)(addr); \
499 uint32_t __result; \
500 __asm__ __volatile__ \
501 ( \
502 "lpm %A0, Z+" "\n\t" \
503 "lpm %B0, Z+" "\n\t" \
504 "lpm %C0, Z+" "\n\t" \
505 "lpm %D0, Z" "\n\t" \
506 : "=r" (__result), "=z" (__addr16) \
507 : "1" (__addr16) \
508 ); \
509 __result;

```

```

510 )))
511
512 #define __LPM_float_classic__(addr)      \
513 (__extension__({                          \
514   uint16_t __addr16 = (uint16_t)(addr);  \
515   float __result;                        \
516   __asm__ __volatile__                   \
517   (                                       \
518     "lpm"                                \
519     "mov %A0, r0"                        \
520     "adiw r30, 1"                        \
521     "lpm"                                \
522     "mov %B0, r0"                        \
523     "adiw r30, 1"                        \
524     "lpm"                                \
525     "mov %C0, r0"                        \
526     "adiw r30, 1"                        \
527     "lpm"                                \
528     "mov %D0, r0"                        \
529     : "=r" (__result), "=z" (__addr16) \
530     : "1" (__addr16)                   \
531     : "r0"                             \
532   );                                     \
533   __result;                             \
534   )))
535
536 #define __LPM_float_enhanced__(addr)      \
537 (__extension__({                          \
538   uint16_t __addr16 = (uint16_t)(addr);  \
539   float __result;                        \
540   __asm__ __volatile__                   \
541   (                                       \
542     "lpm %A0, Z+"                        \
543     "lpm %B0, Z+"                        \
544     "lpm %C0, Z+"                        \
545     "lpm %D0, Z"                         \
546     : "=r" (__result), "=z" (__addr16) \
547     : "1" (__addr16)                   \
548   );                                     \
549   __result;                             \
550   )))
551
552 #if defined (__AVR_HAVE_LPMX__)
553 #define __LPM(addr) __LPM_enhanced__(addr)
554 #define __LPM_word(addr) __LPM_word_enhanced__(addr)
555 #define __LPM_dword(addr) __LPM_dword_enhanced__(addr)
556 #define __LPM_float(addr) __LPM_float_enhanced__(addr)
557 #else
558 #define __LPM(addr) __LPM_classic__(addr)
559 #define __LPM_word(addr) __LPM_word_classic__(addr)
560 #define __LPM_dword(addr) __LPM_dword_classic__(addr)
561 #define __LPM_float(addr) __LPM_float_classic__(addr)
562 #endif
563
564 #endif /* !__DOXYGEN__ */
565
566 /** \ingroup avr_pgmSPACE
567 \def pgm_read_byte_near(address_short)
568 Read a byte from the program space with a 16-bit (near) address.
569 \note The address is a byte address.
570 The address is in the program space. */
571
572 #define pgm_read_byte_near(address_short) __LPM((uint16_t)(address_short))
573
574 /** \ingroup avr_pgmSPACE
575 \def pgm_read_word_near(address_short)
576 Read a word from the program space with a 16-bit (near) address.
577 \note The address is a byte address.
578 The address is in the program space. */
579
580 #define pgm_read_word_near(address_short) __LPM_word((uint16_t)(address_short))
581
582 /** \ingroup avr_pgmSPACE
583 \def pgm_read_dword_near(address_short)
584 Read a double word from the program space with a 16-bit (near) address.
585 \note The address is a byte address.
586 The address is in the program space. */
587
588 #define pgm_read_dword_near(address_short) \
589   __LPM_dword((uint16_t)(address_short))
590
591 /** \ingroup avr_pgmSPACE
592 \def pgm_read_float_near(address_short)
593 Read a float from the program space with a 16-bit (near) address.
594 \note The address is a byte address.
595 The address is in the program space. */
596

```

```

597 #define pgm_read_float_near(address_short) \
598   __LPM_float((uint16_t)(address_short))
599
600 /** \ingroup avr_pgmspace
601 \def pgm_read_ptr_near(address_short)
602 Read a pointer from the program space with a 16-bit (near) address.
603 \note The address is a byte address.
604 The address is in the program space. */
605
606 #define pgm_read_ptr_near(address_short) \
607   (void*)__LPM_word((uint16_t)(address_short))
608
609 #if defined(RAMPZ) || defined(__DOXYGEN__)
610
611 /* Only for devices with more than 64K of program memory.
612 RAMPZ must be defined (see iom103.h, iom128.h).
613 */
614
615 /* The classic functions are needed for ATmega103. */
616 #ifndef __DOXYGEN__ /* These are internal macros, avoid "is
617 not documented" warnings. */
618 #define __ELPM_classic__(addr) \
619   (__extension__({ \
620     uint32_t __addr32 = (uint32_t)(addr); \
621     uint8_t __result; \
622     __asm__ __volatile__ \
623     ( \
624       "out %2, %C1" "\n\t" \
625       "mov r31, %B1" "\n\t" \
626       "mov r30, %A1" "\n\t" \
627       "elpm" "\n\t" \
628       "mov %0, r0" "\n\t" \
629       : "=r" (__result) \
630       : "r" (__addr32), \
631       "I" (_SFR_IO_ADDR(RAMPZ)) \
632       : "r0", "r30", "r31" \
633     ); \
634     __result; \
635   }))
636
637 #define __ELPM_enhanced__(addr) \
638   (__extension__({ \
639     uint32_t __addr32 = (uint32_t)(addr); \
640     uint8_t __result; \
641     __asm__ __volatile__ \
642     ( \
643       "out %2, %C1" "\n\t" \
644       "movw r30, %1" "\n\t" \
645       "elpm %0, Z+" "\n\t" \
646       : "=r" (__result) \
647       : "r" (__addr32), \
648       "I" (_SFR_IO_ADDR(RAMPZ)) \
649       : "r30", "r31" \
650     ); \
651     __result; \
652   }))
653
654 #define __ELPM_xmega__(addr) \
655   (__extension__({ \
656     uint32_t __addr32 = (uint32_t)(addr); \
657     uint8_t __result; \
658     __asm__ __volatile__ \
659     ( \
660       "in __tmp_reg__, %2" "\n\t" \
661       "out %2, %C1" "\n\t" \
662       "movw r30, %1" "\n\t" \
663       "elpm %0, Z+" "\n\t" \
664       "out %2, __tmp_reg__" \
665       : "=r" (__result) \
666       : "r" (__addr32), \
667       "I" (_SFR_IO_ADDR(RAMPZ)) \
668       : "r30", "r31" \
669     ); \
670     __result; \
671   }))
672
673 #define __ELPM_word_classic__(addr) \
674   (__extension__({ \
675     uint32_t __addr32 = (uint32_t)(addr); \
676     uint16_t __result; \
677     __asm__ __volatile__ \
678     ( \
679       "out %2, %C1" "\n\t" \
680       "mov r31, %B1" "\n\t" \
681       "mov r30, %A1" "\n\t" \
682       "elpm" "\n\t" \
683       "mov %A0, r0" "\n\t"

```

```

684         "in r0, %2"          "\n\t"
685         "adiw r30, 1"         "\n\t"
686         "adc r0, __zero_reg__" "\n\t"
687         "out %2, r0"          "\n\t"
688         "elpm"                "\n\t"
689         "mov %B0, r0"          "\n\t"
690         : "=r" (__result)
691         : "r" (__addr32),
692           "I" (_SFR_IO_ADDR(RAMPZ))
693         : "r0", "r30", "r31"
694     );
695     __result;
696 })
697
698 #define __ELPM_word_enhanced__(addr)
699 (__extension__({
700     uint32_t __addr32 = (uint32_t)(addr);
701     uint16_t __result;
702     __asm__ __volatile__
703     (
704         "out %2, %C1"          "\n\t"
705         "movw r30, %1"         "\n\t"
706         "elpm %A0, Z+"         "\n\t"
707         "elpm %B0, Z"          "\n\t"
708         : "=r" (__result)
709         : "r" (__addr32),
710           "I" (_SFR_IO_ADDR(RAMPZ))
711         : "r30", "r31"
712     );
713     __result;
714 })
715
716 #define __ELPM_word_xmega__(addr)
717 (__extension__({
718     uint32_t __addr32 = (uint32_t)(addr);
719     uint16_t __result;
720     __asm__ __volatile__
721     (
722         "in __tmp_reg__, %2"   "\n\t"
723         "out %2, %C1"          "\n\t"
724         "movw r30, %1"         "\n\t"
725         "elpm %A0, Z+"         "\n\t"
726         "elpm %B0, Z"          "\n\t"
727         "out %2, __tmp_reg__"
728         : "=r" (__result)
729         : "r" (__addr32),
730           "I" (_SFR_IO_ADDR(RAMPZ))
731         : "r30", "r31"
732     );
733     __result;
734 })
735
736 #define __ELPM_dword_classic__(addr)
737 (__extension__({
738     uint32_t __addr32 = (uint32_t)(addr);
739     uint32_t __result;
740     __asm__ __volatile__
741     (
742         "out %2, %C1"          "\n\t"
743         "mov r31, %B1"         "\n\t"
744         "mov r30, %A1"         "\n\t"
745         "elpm"                "\n\t"
746         "mov %A0, r0"          "\n\t"
747         "in r0, %2"            "\n\t"
748         "adiw r30, 1"          "\n\t"
749         "adc r0, __zero_reg__" "\n\t"
750         "out %2, r0"          "\n\t"
751         "elpm"                "\n\t"
752         "mov %B0, r0"          "\n\t"
753         "in r0, %2"            "\n\t"
754         "adiw r30, 1"          "\n\t"
755         "adc r0, __zero_reg__" "\n\t"
756         "out %2, r0"          "\n\t"
757         "elpm"                "\n\t"
758         "mov %C0, r0"          "\n\t"
759         "in r0, %2"            "\n\t"
760         "adiw r30, 1"          "\n\t"
761         "adc r0, __zero_reg__" "\n\t"
762         "out %2, r0"          "\n\t"
763         "elpm"                "\n\t"
764         "mov %D0, r0"          "\n\t"
765         : "=r" (__result)
766         : "r" (__addr32),
767           "I" (_SFR_IO_ADDR(RAMPZ))
768         : "r0", "r30", "r31"
769     );
770     __result;

```



```

771 )))
772
773 #define __ELPM_dword_enhanced__(addr) \
774 (__extension__({ \
775     uint32_t __addr32 = (uint32_t)(addr); \
776     uint32_t __result; \
777     __asm__ __volatile__ \
778     ( \
779         "out %2, %C1"      "\n\t" \
780         "movw r30, %1"     "\n\t" \
781         "elpm %A0, Z+"     "\n\t" \
782         "elpm %B0, Z+"     "\n\t" \
783         "elpm %C0, Z+"     "\n\t" \
784         "elpm %D0, Z"      "\n\t" \
785         : "=r" (__result) \
786         : "r" (__addr32), \
787         "I" (_SFR_IO_ADDR(RAMPZ)) \
788         : "r30", "r31" \
789     ); \
790     __result; \
791     }) \
792
793 #define __ELPM_dword_xmega__(addr) \
794 (__extension__({ \
795     uint32_t __addr32 = (uint32_t)(addr); \
796     uint32_t __result; \
797     __asm__ __volatile__ \
798     ( \
799         "in __tmp_reg__, %2" "\n\t" \
800         "out %2, %C1"      "\n\t" \
801         "movw r30, %1"     "\n\t" \
802         "elpm %A0, Z+"     "\n\t" \
803         "elpm %B0, Z+"     "\n\t" \
804         "elpm %C0, Z+"     "\n\t" \
805         "elpm %D0, Z"      "\n\t" \
806         "out %2, __tmp_reg__" \
807         : "=r" (__result) \
808         : "r" (__addr32), \
809         "I" (_SFR_IO_ADDR(RAMPZ)) \
810         : "r30", "r31" \
811     ); \
812     __result; \
813     }) \
814
815 #define __ELPM_float_classic__(addr) \
816 (__extension__({ \
817     uint32_t __addr32 = (uint32_t)(addr); \
818     float __result; \
819     __asm__ __volatile__ \
820     ( \
821         "out %2, %C1"      "\n\t" \
822         "mov r31, %B1"     "\n\t" \
823         "mov r30, %A1"     "\n\t" \
824         "elpm"             "\n\t" \
825         "mov %A0, r0"      "\n\t" \
826         "in r0, %2"        "\n\t" \
827         "adiw r30, 1"      "\n\t" \
828         "adc r0, __zero_reg__" "\n\t" \
829         "out %2, r0"       "\n\t" \
830         "elpm"             "\n\t" \
831         "mov %B0, r0"      "\n\t" \
832         "in r0, %2"        "\n\t" \
833         "adiw r30, 1"      "\n\t" \
834         "adc r0, __zero_reg__" "\n\t" \
835         "out %2, r0"       "\n\t" \
836         "elpm"             "\n\t" \
837         "mov %C0, r0"      "\n\t" \
838         "in r0, %2"        "\n\t" \
839         "adiw r30, 1"      "\n\t" \
840         "adc r0, __zero_reg__" "\n\t" \
841         "out %2, r0"       "\n\t" \
842         "elpm"             "\n\t" \
843         "mov %D0, r0"      "\n\t" \
844         : "=r" (__result) \
845         : "r" (__addr32), \
846         "I" (_SFR_IO_ADDR(RAMPZ)) \
847         : "r0", "r30", "r31" \
848     ); \
849     __result; \
850     }) \
851
852 #define __ELPM_float_enhanced__(addr) \
853 (__extension__({ \
854     uint32_t __addr32 = (uint32_t)(addr); \
855     float __result; \
856     __asm__ __volatile__ \
857     ( \

```

```

858 "out %2, %C1" "\n\t" \
859 "movw r30, %1" "\n\t" \
860 "elpm %A0, Z+" "\n\t" \
861 "elpm %B0, Z+" "\n\t" \
862 "elpm %C0, Z+" "\n\t" \
863 "elpm %D0, Z" "\n\t" \
864 : "=r" (__result) \
865 : "r" (__addr32), \
866 "I" (_SFR_IO_ADDR(RAMPZ)) \
867 : "r30", "r31" \
868 ); \
869 __result; \
870 ))) \
871 \
872 #define __ELPM_float_xmega__(addr) \
873 (__extension__({ \
874 uint32_t __addr32 = (uint32_t)(addr); \
875 float __result; \
876 __asm__ __volatile__ \
877 ( \
878 "in __tmp_reg__, %2" "\n\t" \
879 "out %2, %C1" "\n\t" \
880 "movw r30, %1" "\n\t" \
881 "elpm %A0, Z+" "\n\t" \
882 "elpm %B0, Z+" "\n\t" \
883 "elpm %C0, Z+" "\n\t" \
884 "elpm %D0, Z" "\n\t" \
885 "out %2, __tmp_reg__" \
886 : "=r" (__result) \
887 : "r" (__addr32), \
888 "I" (_SFR_IO_ADDR(RAMPZ)) \
889 : "r30", "r31" \
890 ); \
891 __result; \
892 ))) \
893 \
894 /* \
895 Check for architectures that implement RAMPD (avrxmega5, avrxmega7) \
896 as they need to save/restore RAMPZ for ELPM macros so it does \
897 not interfere with data accesses. \
898 */ \
899 #if defined (__AVR_HAVE_RAMPD__) \
900 \
901 #define __ELPM(addr) __ELPM_xmega__(addr) \
902 #define __ELPM_word(addr) __ELPM_word_xmega__(addr) \
903 #define __ELPM_dword(addr) __ELPM_dword_xmega__(addr) \
904 #define __ELPM_float(addr) __ELPM_float_xmega__(addr) \
905 \
906 #else \
907 \
908 #if defined (__AVR_HAVE_LPMX__) \
909 \
910 #define __ELPM(addr) __ELPM_enhanced__(addr) \
911 #define __ELPM_word(addr) __ELPM_word_enhanced__(addr) \
912 #define __ELPM_dword(addr) __ELPM_dword_enhanced__(addr) \
913 #define __ELPM_float(addr) __ELPM_float_enhanced__(addr) \
914 \
915 #else \
916 \
917 #define __ELPM(addr) __ELPM_classic__(addr) \
918 #define __ELPM_word(addr) __ELPM_word_classic__(addr) \
919 #define __ELPM_dword(addr) __ELPM_dword_classic__(addr) \
920 #define __ELPM_float(addr) __ELPM_float_classic__(addr) \
921 \
922 #endif /* __AVR_HAVE_LPMX__ */ \
923 \
924 #endif /* __AVR_HAVE_RAMPD__ */ \
925 \
926 #endif /* !__DOXYGEN__ */ \
927 \
928 /** \ingroup avr_pgmspace \
929 \def pgm_read_byte_far(address_long) \
930 Read a byte from the program space with a 32-bit (far) address. \
931 \
932 \note The address is a byte address. \
933 The address is in the program space. */ \
934 \
935 #define pgm_read_byte_far(address_long) __ELPM__((uint32_t)(address_long)) \
936 \
937 /** \ingroup avr_pgmspace \
938 \def pgm_read_word_far(address_long) \
939 Read a word from the program space with a 32-bit (far) address. \
940 \
941 \note The address is a byte address. \
942 The address is in the program space. */ \
943 \
944 #define pgm_read_word_far(address_long) __ELPM_word__((uint32_t)(address_long))

```

```

945
946 /** \ingroup avr_pgmspace
947 \def pgm_read_dword_far(address_long)
948 Read a double word from the program space with a 32-bit (far) address.
949
950 \note The address is a byte address.
951 The address is in the program space. */
952
953 #define pgm_read_dword_far(address_long) __ELPM_dword((uint32_t)(address_long))
954
955 /** \ingroup avr_pgmspace
956 \def pgm_read_float_far(address_long)
957 Read a float from the program space with a 32-bit (far) address.
958
959 \note The address is a byte address.
960 The address is in the program space. */
961
962 #define pgm_read_float_far(address_long) __ELPM_float((uint32_t)(address_long))
963
964 /** \ingroup avr_pgmspace
965 \def pgm_read_ptr_far(address_long)
966 Read a pointer from the program space with a 32-bit (far) address.
967
968 \note The address is a byte address.
969 The address is in the program space. */
970
971 #define pgm_read_ptr_far(address_long) (void*)__ELPM_word((uint32_t)(address_long))
972
973 #endif /* RAMPZ or __DOXYGEN__ */
974
975 /** \ingroup avr_pgmspace
976 \def pgm_read_byte(address_short)
977 Read a byte from the program space with a 16-bit (near) address.
978
979 \note The address is a byte address.
980 The address is in the program space. */
981
982 #define pgm_read_byte(address_short) pgm_read_byte_near(address_short)
983
984 /** \ingroup avr_pgmspace
985 \def pgm_read_word(address_short)
986 Read a word from the program space with a 16-bit (near) address.
987
988 \note The address is a byte address.
989 The address is in the program space. */
990
991 #define pgm_read_word(address_short) pgm_read_word_near(address_short)
992
993 /** \ingroup avr_pgmspace
994 \def pgm_read_dword(address_short)
995 Read a double word from the program space with a 16-bit (near) address.
996
997 \note The address is a byte address.
998 The address is in the program space. */
999
1000 #define pgm_read_dword(address_short) pgm_read_dword_near(address_short)
1001
1002 /** \ingroup avr_pgmspace
1003 \def pgm_read_float(address_short)
1004 Read a float from the program space with a 16-bit (near) address.
1005
1006 \note The address is a byte address.
1007 The address is in the program space. */
1008
1009 #define pgm_read_float(address_short) pgm_read_float_near(address_short)
1010
1011 /** \ingroup avr_pgmspace
1012 \def pgm_read_ptr(address_short)
1013 Read a pointer from the program space with a 16-bit (near) address.
1014
1015 \note The address is a byte address.
1016 The address is in the program space. */
1017
1018 #define pgm_read_ptr(address_short) pgm_read_ptr_near(address_short)
1019
1020 /** \ingroup avr_pgmspace
1021 \def pgm_get_far_address(var)
1022
1023 This macro facilitates the obtention of a 32 bit "far" pointer (only 24 bits
1024 used) to data even passed the 64KB limit for the 16 bit ordinary pointer. It
1025 is similar to the '&' operator, with some limitations.
1026
1027 Comments:
1028
1029 - The overhead is minimal and it's mainly due to the 32 bit size operation.
1030
1031 - 24 bit sizes guarantees the code compatibility for use in future devices.

```

```

1032
1033 - hh8() is an undocumented feature but seems to give the third significant byte
1034 of a 32 bit data and accepts symbols, complementing the functionality of hi8()
1035 and lo8(). There is not an equivalent assembler function to get the high
1036 significant byte.
1037
1038 - 'var' has to be resolved at linking time as an existing symbol, i.e., a simple
1039 type variable name, an array name (not an indexed element of the array, if the
1040 index is a constant the compiler does not complain but fails to get the address
1041 if optimization is enabled), a struct name or a struct field name, a function
1042 identifier, a linker defined identifier,...
1043
1044 - The returned value is the identifier's VMA (virtual memory address) determined
1045 by the linker and falls in the corresponding memory region. The AVR Harvard
1046 architecture requires non overlapping VMA areas for the multiple address spaces
1047 in the processor: Flash ROM, RAM, and EEPROM. Typical offset for this are
1048 0x00000000, 0x00800xx0, and 0x00810000 respectively, derived from the linker
1049 script used and linker options. The value returned can be seen then as a
1050 universal pointer.
1051 */
1052
1053 #define pgm_get_far_address(var)
1054 {
1055     uint_farptr_t tmp;
1056     \
1057     __asm__ __volatile__(
1058     \
1059     "ldi    %A0, lo8(%1)"           "\n\t" \
1060     "ldi    %B0, hi8(%1)"           "\n\t" \
1061     "ldi    %C0, hh8(%1)"           "\n\t" \
1062     "clr    %D0"                     "\n\t" \
1063     :
1064     "d" (tmp)
1065     :
1066     "p" (&(var))
1067     );
1068     tmp;
1069 }
1070
1071
1072
1073 /** \ingroup avr_pgmspace
1074 \fn const void * memchr_P(const void *s, int val, size_t len)
1075 \brief Scan flash memory for a character.
1076
1077 The memchr_P() function scans the first \p len bytes of the flash
1078 memory area pointed to by \p s for the character \p val. The first
1079 byte to match \p val (interpreted as an unsigned character) stops
1080 the operation.
1081
1082 \return The memchr_P() function returns a pointer to the matching
1083 byte or \c NULL if the character does not occur in the given memory
1084 area. */
1085 extern const void * memchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;
1086
1087 /** \ingroup avr_pgmspace
1088 \fn int memcmp_P(const void *s1, const void *s2, size_t len)
1089 \brief Compare memory areas
1090
1091 The memcmp_P() function compares the first \p len bytes of the memory
1092 areas \p s1 and flash \p s2. The comparison is performed using unsigned
1093 char operations.
1094
1095 \returns The memcmp_P() function returns an integer less than, equal
1096 to, or greater than zero if the first \p len bytes of \p s1 is found,
1097 respectively, to be less than, to match, or be greater than the first
1098 \p len bytes of \p s2. */
1099 extern int memcmp_P(const void *, const void *, size_t) __ATTR_PURE__;
1100
1101 /** \ingroup avr_pgmspace
1102 \fn void *memcpy_P(void *dest, const void *src, int val, size_t len)
1103
1104 This function is similar to memcpy() except that \p src is pointer
1105 to a string in program space. */
1106 extern void *memcpy_P(void *, const void *, int __val, size_t);
1107
1108 /** \ingroup avr_pgmspace
1109 \fn void *memcpy_P(void *dest, const void *src, size_t n)
1110
1111 The memcpy_P() function is similar to memcpy(), except the src string
1112 resides in program space.
1113
1114 \returns The memcpy_P() function returns a pointer to dest. */
1115 extern void *memcpy_P(void *, const void *, size_t);
1116
1117 /** \ingroup avr_pgmspace
1118 \fn void *memmem_P(const void *s1, size_t len1, const void *s2, size_t len2)

```

```

1119
1120 The memmem_P() function is similar to memmem() except that \p s2 is
1121 pointer to a string in program space. */
1122 extern void *memmem_P(const void *, size_t, const void *, size_t) __ATTR_PURE__;
1123
1124 /** \ingroup avr_pgmspace
1125 \fn const void *memrchr_P(const void *src, int val, size_t len)
1126
1127 The memrchr_P() function is like the memchr_P() function, except
1128 that it searches backwards from the end of the \p len bytes pointed
1129 to by \p src instead of forwards from the front. (Glibc, GNU extension.)
1130
1131 \return The memrchr_P() function returns a pointer to the matching
1132 byte or \c NULL if the character does not occur in the given memory
1133 area. */
1134 extern const void * memrchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;
1135
1136 /** \ingroup avr_pgmspace
1137 \fn char *strcat_P(char *dest, const char *src)
1138
1139 The strcat_P() function is similar to strcat() except that the \e src
1140 string must be located in program space (flash).
1141
1142 \returns The strcat() function returns a pointer to the resulting string
1143 \e dest. */
1144 extern char *strcat_P(char *, const char *);
1145
1146 /** \ingroup avr_pgmspace
1147 \fn const char *strchr_P(const char *s, int val)
1148 \brief Locate character in program space string.
1149
1150 The strchr_P() function locates the first occurrence of \p val
1151 (converted to a char) in the string pointed to by \p s in program
1152 space. The terminating null character is considered to be part of
1153 the string.
1154
1155 The strchr_P() function is similar to strchr() except that \p s is
1156 pointer to a string in program space.
1157
1158 \returns The strchr_P() function returns a pointer to the matched
1159 character or \c NULL if the character is not found. */
1160 extern const char * strchr_P(const char *, int __val) __ATTR_CONST__;
1161
1162 /** \ingroup avr_pgmspace
1163 \fn const char *strchrnul_P(const char *s, int c)
1164
1165 The strchrnul_P() function is like strchr_P() except that if \p c is
1166 not found in \p s, then it returns a pointer to the null byte at the
1167 end of \p s, rather than \c NULL. (Glibc, GNU extension.)
1168
1169 \return The strchrnul_P() function returns a pointer to the matched
1170 character, or a pointer to the null byte at the end of \p s (i.e.,
1171 \c s+strlen(s)) if the character is not found. */
1172 extern const char * strchrnul_P(const char *, int __val) __ATTR_CONST__;
1173
1174 /** \ingroup avr_pgmspace
1175 \fn int strcmp_P(const char *s1, const char *s2)
1176
1177 The strcmp_P() function is similar to strcmp() except that \p s2 is
1178 pointer to a string in program space.
1179
1180 \returns The strcmp_P() function returns an integer less than, equal
1181 to, or greater than zero if \p s1 is found, respectively, to be less
1182 than, to match, or be greater than \p s2. A consequence of the
1183 ordering used by strcmp_P() is that if \p s1 is an initial substring
1184 of \p s2, then \p s1 is considered to be "less than" \p s2. */
1185 extern int strcmp_P(const char *, const char *) __ATTR_PURE__;
1186
1187 /** \ingroup avr_pgmspace
1188 \fn char *strcpy_P(char *dest, const char *src)
1189
1190 The strcpy_P() function is similar to strcpy() except that src is a
1191 pointer to a string in program space.
1192
1193 \returns The strcpy_P() function returns a pointer to the destination
1194 string dest. */
1195 extern char *strcpy_P(char *, const char *);
1196
1197 /** \ingroup avr_pgmspace
1198 \fn int strcasecmp_P(const char *s1, const char *s2)
1199 \brief Compare two strings ignoring case.
1200
1201 The strcasecmp_P() function compares the two strings \p s1 and \p s2,
1202 ignoring the case of the characters.
1203
1204 \param s1 A pointer to a string in the devices SRAM.
1205 \param s2 A pointer to a string in the devices Flash.

```

```

1206
1207 \returns The strcasecmp_P() function returns an integer less than,
1208 equal to, or greater than zero if \p s1 is found, respectively, to
1209 be less than, to match, or be greater than \p s2. A consequence of
1210 the ordering used by strcasecmp_P() is that if \p s1 is an initial
1211 substring of \p s2, then \p s1 is considered to be "less than" \p s2. */
1212 extern int strcasecmp_P(const char *, const char *) __ATTR_PURE__;
1213
1214 /** \ingroup avr_pgmspace
1215 \fn char *strcasestr_P(const char *s1, const char *s2)
1216
1217 This function is similar to strcasestr() except that \p s2 is pointer
1218 to a string in program space. */
1219 extern char *strcasestr_P(const char *, const char *) __ATTR_PURE__;
1220
1221 /** \ingroup avr_pgmspace
1222 \fn size_t strcspn_P(const char *s, const char *reject)
1223
1224 The strcspn_P() function calculates the length of the initial segment
1225 of \p s which consists entirely of characters not in \p reject. This
1226 function is similar to strcspn() except that \p reject is a pointer
1227 to a string in program space.
1228
1229 \return The strcspn_P() function returns the number of characters in
1230 the initial segment of \p s which are not in the string \p reject.
1231 The terminating zero is not considered as a part of string. */
1232 extern size_t strcspn_P(const char *__s, const char * __reject) __ATTR_PURE__;
1233
1234 /** \ingroup avr_pgmspace
1235 \fn size_t strlcat_P(char *dst, const char *src, size_t siz)
1236 \brief Concatenate two strings.
1237
1238 The strlcat_P() function is similar to strlcat(), except that the \p src
1239 string must be located in program space (flash).
1240
1241 Appends \p src to string \p dst of size \p siz (unlike strncat(),
1242 \p siz is the full size of \p dst, not space left). At most \p siz-1
1243 characters will be copied. Always NULL terminates (unless \p siz <=
1244 \p strlen(dst)).
1245
1246 \returns The strlcat_P() function returns strlen(src) + MIN(siz,
1247 strlen(initial dst)). If retval >= siz, truncation occurred. */
1248 extern size_t strlcat_P(char *, const char *, size_t);
1249
1250 /** \ingroup avr_pgmspace
1251 \fn size_t strlcpy_P(char *dst, const char *src, size_t siz)
1252 \brief Copy a string from progmem to RAM.
1253
1254 Copy \p src to string \p dst of size \p siz. At most \p siz-1
1255 characters will be copied. Always NULL terminates (unless \p siz == 0).
1256 The strlcpy_P() function is similar to strlcpy() except that the
1257 \p src is pointer to a string in memory space.
1258
1259 \returns The strlcpy_P() function returns strlen(src). If
1260 retval >= siz, truncation occurred. */
1261 extern size_t strlcpy_P(char *, const char *, size_t);
1262
1263 /** \ingroup avr_pgmspace
1264 \fn size_t strnlen_P(const char *src, size_t len)
1265 \brief Determine the length of a fixed-size string.
1266
1267 The strnlen_P() function is similar to strnlen(), except that \c src is a
1268 pointer to a string in program space.
1269
1270 \returns The strnlen_P function returns strlen_P(src), if that is less than
1271 \c len, or \c len if there is no '\0' character among the first \c len
1272 characters pointed to by \c src. */
1273 extern size_t strnlen_P(const char *, size_t) __ATTR_CONST__; /* program memory can't change */
1274
1275 /** \ingroup avr_pgmspace
1276 \fn int strncmp_P(const char *s1, const char *s2, size_t n)
1277
1278 The strncmp_P() function is similar to strcmp_P() except it only compares
1279 the first (at most) n characters of s1 and s2.
1280
1281 \returns The strncmp_P() function returns an integer less than, equal to,
1282 or greater than zero if s1 (or the first n bytes thereof) is found,
1283 respectively, to be less than, to match, or be greater than s2. */
1284 extern int strncmp_P(const char *, const char *, size_t) __ATTR_PURE__;
1285
1286 /** \ingroup avr_pgmspace
1287 \fn int strncasecmp_P(const char *s1, const char *s2, size_t n)
1288 \brief Compare two strings ignoring case.
1289
1290 The strncasecmp_P() function is similar to strcasecmp_P(), except it
1291 only compares the first \p n characters of \p s1.
1292

```

```

1293 \param s1 A pointer to a string in the devices SRAM.
1294 \param s2 A pointer to a string in the devices Flash.
1295 \param n The maximum number of bytes to compare.
1296
1297 \returns The strncasecmp_P() function returns an integer less than,
1298 equal to, or greater than zero if \p s1 (or the first \p n bytes
1299 thereof) is found, respectively, to be less than, to match, or be
1300 greater than \p s2. A consequence of the ordering used by
1301 strncasecmp_P() is that if \p s1 is an initial substring of \p s2,
1302 then \p s1 is considered to be "less than" \p s2. */
1303 extern int strncasecmp_P(const char *, const char *, size_t) __ATTR_PURE__;
1304
1305 /** \ingroup avr_pgmspace
1306 \fn char *strncat_P(char *dest, const char *src, size_t len)
1307 \brief Concatenate two strings.
1308
1309 The strncat_P() function is similar to strncat(), except that the \e src
1310 string must be located in program space (flash).
1311
1312 \returns The strncat_P() function returns a pointer to the resulting string
1313 dest. */
1314 extern char *strncat_P(char *, const char *, size_t);
1315
1316 /** \ingroup avr_pgmspace
1317 \fn char *strncpy_P(char *dest, const char *src, size_t n)
1318
1319 The strncpy_P() function is similar to strcpy_P() except that not more
1320 than n bytes of src are copied. Thus, if there is no null byte among the
1321 first n bytes of src, the result will not be null-terminated.
1322
1323 In the case where the length of src is less than that of n, the remainder
1324 of dest will be padded with nulls.
1325
1326 \returns The strncpy_P() function returns a pointer to the destination
1327 string dest. */
1328 extern char *strncpy_P(char *, const char *, size_t);
1329
1330 /** \ingroup avr_pgmspace
1331 \fn char *strpbrk_P(const char *s, const char *accept)
1332
1333 The strpbrk_P() function locates the first occurrence in the string
1334 \p s of any of the characters in the flash string \p accept. This
1335 function is similar to strpbrk() except that \p accept is a pointer
1336 to a string in program space.
1337
1338 \return The strpbrk_P() function returns a pointer to the character
1339 in \p s that matches one of the characters in \p accept, or \c NULL
1340 if no such character is found. The terminating zero is not considered
1341 as a part of string: if one or both args are empty, the result will
1342 \c NULL. */
1343 extern char *strpbrk_P(const char *__s, const char * __accept) __ATTR_PURE__;
1344
1345 /** \ingroup avr_pgmspace
1346 \fn const char *strrchr_P(const char *s, int val)
1347 \brief Locate character in string.
1348
1349 The strrchr_P() function returns a pointer to the last occurrence of
1350 the character \p val in the flash string \p s.
1351
1352 \return The strrchr_P() function returns a pointer to the matched
1353 character or \c NULL if the character is not found. */
1354 extern const char *strrchr_P(const char *, int __val) __ATTR_CONST__;
1355
1356 /** \ingroup avr_pgmspace
1357 \fn char *strsep_P(char **sp, const char *delim)
1358 \brief Parse a string into tokens.
1359
1360 The strsep_P() function locates, in the string referenced by \p *sp,
1361 the first occurrence of any character in the string \p delim (or the
1362 terminating '\\0' character) and replaces it with a '\\0'. The
1363 location of the next character after the delimiter (or \c
1364 NULL, if the end of the string was reached) is stored in \p *sp. An
1365 "empty" field, i.e. one caused by two adjacent delimiter
1366 characters, can be detected by comparing the location referenced by
1367 the pointer returned in \p *sp to '\\0'. This function is similar to
1368 strsep() except that \p delim is a pointer to a string in program
1369 space.
1370
1371 \return The strsep_P() function returns a pointer to the original
1372 value of \p *sp. If \p *sp is initially \c NULL, strsep_P() returns
1373 \c NULL. */
1374 extern char *strsep_P(char **__sp, const char * __delim);
1375
1376 /** \ingroup avr_pgmspace
1377 \fn size_t strspn_P(const char *s, const char *accept)
1378
1379 The strspn_P() function calculates the length of the initial segment

```

```

1380 of \p s which consists entirely of characters in \p accept. This
1381 function is similar to strspn() except that \p accept is a pointer
1382 to a string in program space.
1383
1384 \return The strspn_P() function returns the number of characters in
1385 the initial segment of \p s which consist only of characters from \p
1386 accept. The terminating zero is not considered as a part of string. */
1387 extern size_t strspn_P(const char *__s, const char * __accept) __ATTR_PURE__;
1388
1389 /** \ingroup avr_pgmspace
1390 \fn char *strstr_P(const char *s1, const char *s2)
1391 \brief Locate a substring.
1392
1393 The strstr_P() function finds the first occurrence of the substring
1394 \p s2 in the string \p s1. The terminating '\\0' characters are not
1395 compared. The strstr_P() function is similar to strstr() except that
1396 \p s2 is pointer to a string in program space.
1397
1398 \returns The strstr_P() function returns a pointer to the beginning
1399 of the substring, or NULL if the substring is not found. If \p s2
1400 points to a string of zero length, the function returns \p s1. */
1401 extern char *strstr_P(const char *, const char *) __ATTR_PURE__;
1402
1403 /** \ingroup avr_pgmspace
1404 \fn char *strtok_P(char *s, const char * delim)
1405 \brief Parses the string into tokens.
1406
1407 strtok_P() parses the string \p s into tokens. The first call to
1408 strtok_P() should have \p s as its first argument. Subsequent calls
1409 should have the first argument set to NULL. If a token ends with a
1410 delimiter, this delimiting character is overwritten with a '\\0' and a
1411 pointer to the next character is saved for the next call to strtok_P().
1412 The delimiter string \p delim may be different for each call.
1413
1414 The strtok_P() function is similar to strtok() except that \p delim
1415 is pointer to a string in program space.
1416
1417 \returns The strtok_P() function returns a pointer to the next token or
1418 NULL when no more tokens are found.
1419
1420 \note strtok_P() is NOT reentrant. For a reentrant version of this
1421 function see strtok_rP().
1422 */
1423 extern char *strtok_P(char *__s, const char * __delim);
1424
1425 /** \ingroup avr_pgmspace
1426 \fn char *strtok_rP(char *string, const char *delim, char **last)
1427 \brief Parses string into tokens.
1428
1429 The strtok_rP() function parses \p string into tokens. The first call to
1430 strtok_rP() should have string as its first argument. Subsequent calls
1431 should have the first argument set to NULL. If a token ends with a
1432 delimiter, this delimiting character is overwritten with a '\\0' and a
1433 pointer to the next character is saved for the next call to strtok_rP().
1434 The delimiter string \p delim may be different for each call. \p last is
1435 a user allocated char* pointer. It must be the same while parsing the
1436 same string. strtok_rP() is a reentrant version of strtok_P().
1437
1438 The strtok_rP() function is similar to strtok_r() except that \p delim
1439 is pointer to a string in program space.
1440
1441 \returns The strtok_rP() function returns a pointer to the next token or
1442 NULL when no more tokens are found. */
1443 extern char *strtok_rP(char *__s, const char * __delim, char **__last);
1444
1445 /** \ingroup avr_pgmspace
1446 \fn size_t strlen_PF(uint_farptr_t s)
1447 \brief Obtain the length of a string
1448
1449 The strlen_PF() function is similar to strlen(), except that \e s is a
1450 far pointer to a string in program space.
1451
1452 \param s A far pointer to the string in flash
1453
1454 \returns The strlen_PF() function returns the number of characters in
1455 \e s. The contents of RAMPZ SFR are undefined when the function returns. */
1456 extern size_t strlen_PF(uint_farptr_t src) __ATTR_CONST__; /* program memory can't change */
1457
1458 /** \ingroup avr_pgmspace
1459 \fn size_t strnlen_PF(uint_farptr_t s, size_t len)
1460 \brief Determine the length of a fixed-size string
1461
1462 The strnlen_PF() function is similar to strnlen(), except that \e s is a
1463 far pointer to a string in program space.
1464
1465 \param s A far pointer to the string in Flash
1466 \param len The maximum number of length to return

```



```

1467
1468 \returns The strlen_PF function returns strlen_P(\e s), if that is less
1469 than \e len, or \e len if there is no '\\0' character among the first \e
1470 len characters pointed to by \e s. The contents of RAMPZ SFR are
1471 undefined when the function returns. */
1472 extern size_t strlen_PF(uint_farptr_t src, size_t len) __ATTR_CONST__; /* program memory can't change
    */
1473
1474 /** \ingroup avr_pgmspace
1475 \fn void *memcpy_PF(void *dest, uint_farptr_t src, size_t n)
1476 \brief Copy a memory block from flash to SRAM
1477
1478 The memcpy_PF() function is similar to memcpy(), except the data
1479 is copied from the program space and is addressed using a far pointer.
1480
1481 \param dest A pointer to the destination buffer
1482 \param src A far pointer to the origin of data in flash memory
1483 \param n The number of bytes to be copied
1484
1485 \returns The memcpy_PF() function returns a pointer to \e dst. The contents
1486 of RAMPZ SFR are undefined when the function returns. */
1487 extern void *memcpy_PF(void *dest, uint_farptr_t src, size_t len);
1488
1489 /** \ingroup avr_pgmspace
1490 \fn char *strcpy_PF(char *dst, uint_farptr_t src)
1491 \brief Duplicate a string
1492
1493 The strcpy_PF() function is similar to strcpy() except that \e src is a far
1494 pointer to a string in program space.
1495
1496 \param dst A pointer to the destination string in SRAM
1497 \param src A far pointer to the source string in Flash
1498
1499 \returns The strcpy_PF() function returns a pointer to the destination
1500 string \e dst. The contents of RAMPZ SFR are undefined when the function
1501 returns. */
1502 extern char *strcpy_PF(char *dst, uint_farptr_t src);
1503
1504 /** \ingroup avr_pgmspace
1505 \fn char *strncpy_PF(char *dst, uint_farptr_t src, size_t n)
1506 \brief Duplicate a string until a limited length
1507
1508 The strncpy_PF() function is similar to strcpy_PF() except that not more
1509 than \e n bytes of \e src are copied. Thus, if there is no null byte among
1510 the first \e n bytes of \e src, the result will not be null-terminated.
1511
1512 In the case where the length of \e src is less than that of \e n, the
1513 remainder of \e dst will be padded with nulls.
1514
1515 \param dst A pointer to the destination string in SRAM
1516 \param src A far pointer to the source string in Flash
1517 \param n The maximum number of bytes to copy
1518
1519 \returns The strncpy_PF() function returns a pointer to the destination
1520 string \e dst. The contents of RAMPZ SFR are undefined when the function
1521 returns. */
1522 extern char *strncpy_PF(char *dst, uint_farptr_t src, size_t len);
1523
1524 /** \ingroup avr_pgmspace
1525 \fn char *strcat_PF(char *dst, uint_farptr_t src)
1526 \brief Concatenates two strings
1527
1528 The strcat_PF() function is similar to strcat() except that the \e src
1529 string must be located in program space (flash) and is addressed using
1530 a far pointer
1531
1532 \param dst A pointer to the destination string in SRAM
1533 \param src A far pointer to the string to be appended in Flash
1534
1535 \returns The strcat_PF() function returns a pointer to the resulting
1536 string \e dst. The contents of RAMPZ SFR are undefined when the function
1537 returns */
1538 extern char *strcat_PF(char *dst, uint_farptr_t src);
1539
1540 /** \ingroup avr_pgmspace
1541 \fn size_t strlcat_PF(char *dst, uint_farptr_t src, size_t n)
1542 \brief Concatenate two strings
1543
1544 The strlcat_PF() function is similar to strlcat(), except that the \e src
1545 string must be located in program space (flash) and is addressed using
1546 a far pointer.
1547
1548 Appends src to string dst of size \e n (unlike strncat(), \e n is the
1549 full size of \e dst, not space left). At most \e n-1 characters
1550 will be copied. Always NULL terminates (unless \e n <= strlen(\e dst)).
1551
1552 \param dst A pointer to the destination string in SRAM

```

```

1553 \param src A far pointer to the source string in Flash
1554 \param n The total number of bytes allocated to the destination string
1555
1556 \returns The strlcat_PF() function returns strlen(\e src) + MIN(\e n,
1557 strlen(initial \e dst)). If retval >= \e n, truncation occurred. The
1558 contents of RAMPZ SFR are undefined when the function returns. */
1559 extern size_t strlcat_PF(char *dst, uint_farptr_t src, size_t siz);
1560
1561 /** \ingroup avr_pgmspace
1562 \fn char *strncat_PF(char *dst, uint_farptr_t src, size_t n)
1563 \brief Concatenate two strings
1564
1565 The strncat_PF() function is similar to strncat(), except that the \e src
1566 string must be located in program space (flash) and is addressed using a
1567 far pointer.
1568
1569 \param dst A pointer to the destination string in SRAM
1570 \param src A far pointer to the source string in Flash
1571 \param n The maximum number of bytes to append
1572
1573 \returns The strncat_PF() function returns a pointer to the resulting
1574 string \e dst. The contents of RAMPZ SFR are undefined when the function
1575 returns. */
1576 extern char *strncat_PF(char *dest, uint_farptr_t src, size_t len);
1577
1578 /** \ingroup avr_pgmspace
1579 \fn int strcmp_PF(const char *s1, uint_farptr_t s2)
1580 \brief Compares two strings
1581
1582 The strcmp_PF() function is similar to strcmp() except that \e s2 is a far
1583 pointer to a string in program space.
1584
1585 \param s1 A pointer to the first string in SRAM
1586 \param s2 A far pointer to the second string in Flash
1587
1588 \returns The strcmp_PF() function returns an integer less than, equal to,
1589 or greater than zero if \e s1 is found, respectively, to be less than, to
1590 match, or be greater than \e s2. The contents of RAMPZ SFR are undefined
1591 when the function returns. */
1592 extern int strcmp_PF(const char *s1, uint_farptr_t s2) __ATTR_PURE__;
1593
1594 /** \ingroup avr_pgmspace
1595 \fn int strncmp_PF(const char *s1, uint_farptr_t s2, size_t n)
1596 \brief Compare two strings with limited length
1597
1598 The strncmp_PF() function is similar to strcmp_PF() except it only
1599 compares the first (at most) \e n characters of \e s1 and \e s2.
1600
1601 \param s1 A pointer to the first string in SRAM
1602 \param s2 A far pointer to the second string in Flash
1603 \param n The maximum number of bytes to compare
1604
1605 \returns The strncmp_PF() function returns an integer less than, equal
1606 to, or greater than zero if \e s1 (or the first \e n bytes thereof) is found,
1607 respectively, to be less than, to match, or be greater than \e s2. The
1608 contents of RAMPZ SFR are undefined when the function returns. */
1609 extern int strncmp_PF(const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;
1610
1611 /** \ingroup avr_pgmspace
1612 \fn int strcasecmp_PF(const char *s1, uint_farptr_t s2)
1613 \brief Compare two strings ignoring case
1614
1615 The strcasecmp_PF() function compares the two strings \e s1 and \e s2, ignoring
1616 the case of the characters.
1617
1618 \param s1 A pointer to the first string in SRAM
1619 \param s2 A far pointer to the second string in Flash
1620
1621 \returns The strcasecmp_PF() function returns an integer less than, equal
1622 to, or greater than zero if \e s1 is found, respectively, to be less than, to
1623 match, or be greater than \e s2. The contents of RAMPZ SFR are undefined
1624 when the function returns. */
1625 extern int strcasecmp_PF(const char *s1, uint_farptr_t s2) __ATTR_PURE__;
1626
1627 /** \ingroup avr_pgmspace
1628 \fn int strncasecmp_PF(const char *s1, uint_farptr_t s2, size_t n)
1629 \brief Compare two strings ignoring case
1630
1631 The strncasecmp_PF() function is similar to strcasecmp_PF(), except it
1632 only compares the first \e n characters of \e s1 and the string in flash is
1633 addressed using a far pointer.
1634
1635 \param s1 A pointer to a string in SRAM
1636 \param s2 A far pointer to a string in Flash
1637 \param n The maximum number of bytes to compare
1638
1639 \returns The strncasecmp_PF() function returns an integer less than, equal

```

```

1640 to, or greater than zero if \e s1 (or the first \e n bytes thereof) is found,
1641 respectively, to be less than, to match, or be greater than \e s2. The
1642 contents of RAMPZ SFR are undefined when the function returns. */
1643 extern int strncasecmp_PF(const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;
1644
1645 /** \ingroup avr_pgmspace
1646 \fn char *strstr_PF(const char *s1, uint_farptr_t s2)
1647 \brief Locate a substring.
1648
1649 The strstr_PF() function finds the first occurrence of the substring \c s2
1650 in the string \c s1. The terminating '\\0' characters are not
1651 compared.
1652 The strstr_PF() function is similar to strstr() except that \c s2 is a
1653 far pointer to a string in program space.
1654
1655 \returns The strstr_PF() function returns a pointer to the beginning of the
1656 substring, or NULL if the substring is not found.
1657 If \c s2 points to a string of zero length, the function returns \c s1. The
1658 contents of RAMPZ SFR are undefined when the function returns. */
1659 extern char *strstr_PF(const char *s1, uint_farptr_t s2);
1660
1661 /** \ingroup avr_pgmspace
1662 \fn size_t strlcpy_PF(char *dst, uint_farptr_t src, size_t siz)
1663 \brief Copy a string from progmem to RAM.
1664
1665 Copy src to string dst of size siz. At most siz-1 characters will be
1666 copied. Always NULL terminates (unless siz == 0).
1667
1668 \returns The strlcpy_PF() function returns strlen(src). If retval >= siz,
1669 truncation occurred. The contents of RAMPZ SFR are undefined when the
1670 function returns. */
1671 extern size_t strlcpy_PF(char *dst, uint_farptr_t src, size_t siz);
1672
1673 /** \ingroup avr_pgmspace
1674 \fn int memcmp_PF(const void *s1, uint_farptr_t s2, size_t len)
1675 \brief Compare memory areas
1676
1677 The memcmp_PF() function compares the first \p len bytes of the memory
1678 areas \p s1 and flash \p s2. The comparison is performed using unsigned
1679 char operations. It is an equivalent of memcmp_P() function, except
1680 that it is capable working on all FLASH including the extended area
1681 above 64kB.
1682
1683 \returns The memcmp_PF() function returns an integer less than, equal
1684 to, or greater than zero if the first \p len bytes of \p s1 is found,
1685 respectively, to be less than, to match, or be greater than the first
1686 \p len bytes of \p s2. */
1687 extern int memcmp_PF(const void *, uint_farptr_t, size_t) __ATTR_PURE__;
1688
1689 #ifdef __DOXYGEN__
1690 /** \ingroup avr_pgmspace
1691 \fn size_t strlen_P(const char *src)
1692
1693 The strlen_P() function is similar to strlen(), except that src is a
1694 pointer to a string in program space.
1695
1696 \returns The strlen_P() function returns the number of characters in src.
1697
1698 \note strlen_P() is implemented as an inline function in the avr/pgmspace.h
1699 header file, which will check if the length of the string is a constant
1700 and known at compile time. If it is not known at compile time, the macro
1701 will issue a call to __strlen_P() which will then calculate the length
1702 of the string as normal.
1703 */
1704 static inline size_t strlen_P(const char *s);
1705 #else
1706 extern size_t __strlen_P(const char *) __ATTR_CONST__; /* internal helper function */
1707 __attribute__((always_inline)) static __inline__ size_t strlen_P(const char *s);
1708 static __inline__ size_t strlen_P(const char *s) {
1709     return __builtin_constant_p(__builtin_strlen(s))
1710         ? __builtin_strlen(s) : __strlen_P(s);
1711 }
1712 #endif
1713
1714 #ifdef __cplusplus
1715 }
1716 #endif
1717
1718 #endif /* __PGMSPACE_H_ */

```

25.25 portpins.h

```

1 /* Copyright (c) 2003 Theodore A. Roth
2 All rights reserved.

```

```

3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: portpins.h 1936 2009-03-19 22:19:26Z arcanum $ */
32
33 #ifndef _AVR_PORTPINS_H_
34 #define _AVR_PORTPINS_H_ 1
35
36 /* This file should only be included from <avr/io.h>, never directly. */
37
38 #ifndef _AVR_IO_H_
39 # error "Include <avr/io.h> instead of this file."
40 #endif
41
42 /* Define Generic PORTn, DDn, and PINn values. */
43
44 /* Port Data Register (generic) */
45 #define PORT7 7
46 #define PORT6 6
47 #define PORT5 5
48 #define PORT4 4
49 #define PORT3 3
50 #define PORT2 2
51 #define PORT1 1
52 #define PORT0 0
53
54 /* Port Data Direction Register (generic) */
55 #define DD7 7
56 #define DD6 6
57 #define DD5 5
58 #define DD4 4
59 #define DD3 3
60 #define DD2 2
61 #define DD1 1
62 #define DD0 0
63
64 /* Port Input Pins (generic) */
65 #define PIN7 7
66 #define PIN6 6
67 #define PIN5 5
68 #define PIN4 4
69 #define PIN3 3
70 #define PIN2 2
71 #define PIN1 1
72 #define PIN0 0
73
74 /* Define PORTxn an Pxn values for all possible port pins if not defined already by io.h. */
75
76 /* PORT A */
77
78 #if defined(PA0) && !defined(PORTA0)
79 # define PORTA0 PA0
80 #elif defined(PORTA0) && !defined(PA0)
81 # define PA0 PORTA0
82 #endif
83 #if defined(PA1) && !defined(PORTA1)
84 # define PORTA1 PA1
85 #elif defined(PORTA1) && !defined(PA1)
86 # define PA1 PORTA1
87 #endif
88 #if defined(PA2) && !defined(PORTA2)
89 # define PORTA2 PA2

```

```
90 #elif defined(PORTA2) && !defined(PA2)
91 #   define PA2 PORTA2
92 #endif
93 #if defined(PA3) && !defined(PORTA3)
94 #   define PORTA3 PA3
95 #elif defined(PORTA3) && !defined(PA3)
96 #   define PA3 PORTA3
97 #endif
98 #if defined(PA4) && !defined(PORTA4)
99 #   define PORTA4 PA4
100 #elif defined(PORTA4) && !defined(PA4)
101 #   define PA4 PORTA4
102 #endif
103 #if defined(PA5) && !defined(PORTA5)
104 #   define PORTA5 PA5
105 #elif defined(PORTA5) && !defined(PA5)
106 #   define PA5 PORTA5
107 #endif
108 #if defined(PA6) && !defined(PORTA6)
109 #   define PORTA6 PA6
110 #elif defined(PORTA6) && !defined(PA6)
111 #   define PA6 PORTA6
112 #endif
113 #if defined(PA7) && !defined(PORTA7)
114 #   define PORTA7 PA7
115 #elif defined(PORTA7) && !defined(PA7)
116 #   define PA7 PORTA7
117 #endif
118
119 /* PORT B */
120
121 #if defined(PB0) && !defined(PORTB0)
122 #   define PORTB0 PB0
123 #elif defined(PORTB0) && !defined(PB0)
124 #   define PB0 PORTB0
125 #endif
126 #if defined(PB1) && !defined(PORTB1)
127 #   define PORTB1 PB1
128 #elif defined(PORTB1) && !defined(PB1)
129 #   define PB1 PORTB1
130 #endif
131 #if defined(PB2) && !defined(PORTB2)
132 #   define PORTB2 PB2
133 #elif defined(PORTB2) && !defined(PB2)
134 #   define PB2 PORTB2
135 #endif
136 #if defined(PB3) && !defined(PORTB3)
137 #   define PORTB3 PB3
138 #elif defined(PORTB3) && !defined(PB3)
139 #   define PB3 PORTB3
140 #endif
141 #if defined(PB4) && !defined(PORTB4)
142 #   define PORTB4 PB4
143 #elif defined(PORTB4) && !defined(PB4)
144 #   define PB4 PORTB4
145 #endif
146 #if defined(PB5) && !defined(PORTB5)
147 #   define PORTB5 PB5
148 #elif defined(PORTB5) && !defined(PB5)
149 #   define PB5 PORTB5
150 #endif
151 #if defined(PB6) && !defined(PORTB6)
152 #   define PORTB6 PB6
153 #elif defined(PORTB6) && !defined(PB6)
154 #   define PB6 PORTB6
155 #endif
156 #if defined(PB7) && !defined(PORTB7)
157 #   define PORTB7 PB7
158 #elif defined(PORTB7) && !defined(PB7)
159 #   define PB7 PORTB7
160 #endif
161
162 /* PORT C */
163
164 #if defined(PC0) && !defined(PORTC0)
165 #   define PORTC0 PC0
166 #elif defined(PORTC0) && !defined(PC0)
167 #   define PC0 PORTC0
168 #endif
169 #if defined(PC1) && !defined(PORTC1)
170 #   define PORTC1 PC1
171 #elif defined(PORTC1) && !defined(PC1)
172 #   define PC1 PORTC1
173 #endif
174 #if defined(PC2) && !defined(PORTC2)
175 #   define PORTC2 PC2
176 #elif defined(PORTC2) && !defined(PC2)
```

```
177 # define PC2 PORTC2
178 #endif
179 #if defined(PC3) && !defined(PORTC3)
180 # define PORTC3 PC3
181 #elif defined(PORTC3) && !defined(PC3)
182 # define PC3 PORTC3
183 #endif
184 #if defined(PC4) && !defined(PORTC4)
185 # define PORTC4 PC4
186 #elif defined(PORTC4) && !defined(PC4)
187 # define PC4 PORTC4
188 #endif
189 #if defined(PC5) && !defined(PORTC5)
190 # define PORTC5 PC5
191 #elif defined(PORTC5) && !defined(PC5)
192 # define PC5 PORTC5
193 #endif
194 #if defined(PC6) && !defined(PORTC6)
195 # define PORTC6 PC6
196 #elif defined(PORTC6) && !defined(PC6)
197 # define PC6 PORTC6
198 #endif
199 #if defined(PC7) && !defined(PORTC7)
200 # define PORTC7 PC7
201 #elif defined(PORTC7) && !defined(PC7)
202 # define PC7 PORTC7
203 #endif
204
205 /* PORT D */
206
207 #if defined(PD0) && !defined(PORTD0)
208 # define PORTD0 PD0
209 #elif defined(PORTD0) && !defined(PD0)
210 # define PD0 PORTD0
211 #endif
212 #if defined(PD1) && !defined(PORTD1)
213 # define PORTD1 PD1
214 #elif defined(PORTD1) && !defined(PD1)
215 # define PD1 PORTD1
216 #endif
217 #if defined(PD2) && !defined(PORTD2)
218 # define PORTD2 PD2
219 #elif defined(PORTD2) && !defined(PD2)
220 # define PD2 PORTD2
221 #endif
222 #if defined(PD3) && !defined(PORTD3)
223 # define PORTD3 PD3
224 #elif defined(PORTD3) && !defined(PD3)
225 # define PD3 PORTD3
226 #endif
227 #if defined(PD4) && !defined(PORTD4)
228 # define PORTD4 PD4
229 #elif defined(PORTD4) && !defined(PD4)
230 # define PD4 PORTD4
231 #endif
232 #if defined(PD5) && !defined(PORTD5)
233 # define PORTD5 PD5
234 #elif defined(PORTD5) && !defined(PD5)
235 # define PD5 PORTD5
236 #endif
237 #if defined(PD6) && !defined(PORTD6)
238 # define PORTD6 PD6
239 #elif defined(PORTD6) && !defined(PD6)
240 # define PD6 PORTD6
241 #endif
242 #if defined(PD7) && !defined(PORTD7)
243 # define PORTD7 PD7
244 #elif defined(PORTD7) && !defined(PD7)
245 # define PD7 PORTD7
246 #endif
247
248 /* PORT E */
249
250 #if defined(PE0) && !defined(PORTE0)
251 # define PORTE0 PE0
252 #elif defined(PORTE0) && !defined(PE0)
253 # define PE0 PORTE0
254 #endif
255 #if defined(PE1) && !defined(PORTE1)
256 # define PORTE1 PE1
257 #elif defined(PORTE1) && !defined(PE1)
258 # define PE1 PORTE1
259 #endif
260 #if defined(PE2) && !defined(PORTE2)
261 # define PORTE2 PE2
262 #elif defined(PORTE2) && !defined(PE2)
263 # define PE2 PORTE2
```

```
264 #endif
265 #if defined(PE3) && !defined(PORTE3)
266 #   define PORTE3 PE3
267 #elif defined(PORTE3) && !defined(PE3)
268 #   define PE3 PORTE3
269 #endif
270 #if defined(PE4) && !defined(PORTE4)
271 #   define PORTE4 PE4
272 #elif defined(PORTE4) && !defined(PE4)
273 #   define PE4 PORTE4
274 #endif
275 #if defined(PE5) && !defined(PORTE5)
276 #   define PORTE5 PE5
277 #elif defined(PORTE5) && !defined(PE5)
278 #   define PE5 PORTE5
279 #endif
280 #if defined(PE6) && !defined(PORTE6)
281 #   define PORTE6 PE6
282 #elif defined(PORTE6) && !defined(PE6)
283 #   define PE6 PORTE6
284 #endif
285 #if defined(PE7) && !defined(PORTE7)
286 #   define PORTE7 PE7
287 #elif defined(PORTE7) && !defined(PE7)
288 #   define PE7 PORTE7
289 #endif
290
291 /* PORT F */
292
293 #if defined(PF0) && !defined(PORTF0)
294 #   define PORTF0 PF0
295 #elif defined(PORTF0) && !defined(PF0)
296 #   define PF0 PORTF0
297 #endif
298 #if defined(PF1) && !defined(PORTF1)
299 #   define PORTF1 PF1
300 #elif defined(PORTF1) && !defined(PF1)
301 #   define PF1 PORTF1
302 #endif
303 #if defined(PF2) && !defined(PORTF2)
304 #   define PORTF2 PF2
305 #elif defined(PORTF2) && !defined(PF2)
306 #   define PF2 PORTF2
307 #endif
308 #if defined(PF3) && !defined(PORTF3)
309 #   define PORTF3 PF3
310 #elif defined(PORTF3) && !defined(PF3)
311 #   define PF3 PORTF3
312 #endif
313 #if defined(PF4) && !defined(PORTF4)
314 #   define PORTF4 PF4
315 #elif defined(PORTF4) && !defined(PF4)
316 #   define PF4 PORTF4
317 #endif
318 #if defined(PF5) && !defined(PORTF5)
319 #   define PORTF5 PF5
320 #elif defined(PORTF5) && !defined(PF5)
321 #   define PF5 PORTF5
322 #endif
323 #if defined(PF6) && !defined(PORTF6)
324 #   define PORTF6 PF6
325 #elif defined(PORTF6) && !defined(PF6)
326 #   define PF6 PORTF6
327 #endif
328 #if defined(PF7) && !defined(PORTF7)
329 #   define PORTF7 PF7
330 #elif defined(PORTF7) && !defined(PF7)
331 #   define PF7 PORTF7
332 #endif
333
334 /* PORT G */
335
336 #if defined(PG0) && !defined(PORTG0)
337 #   define PORTG0 PG0
338 #elif defined(PORTG0) && !defined(PG0)
339 #   define PG0 PORTG0
340 #endif
341 #if defined(PG1) && !defined(PORTG1)
342 #   define PORTG1 PG1
343 #elif defined(PORTG1) && !defined(PG1)
344 #   define PG1 PORTG1
345 #endif
346 #if defined(PG2) && !defined(PORTG2)
347 #   define PORTG2 PG2
348 #elif defined(PORTG2) && !defined(PG2)
349 #   define PG2 PORTG2
350 #endif
```

```
351 #if defined(PG3) && !defined(PORTG3)
352 #   define PORTG3 PG3
353 #elif defined(PORTG3) && !defined(PG3)
354 #   define PG3 PORTG3
355 #endif
356 #if defined(PG4) && !defined(PORTG4)
357 #   define PORTG4 PG4
358 #elif defined(PORTG4) && !defined(PG4)
359 #   define PG4 PORTG4
360 #endif
361 #if defined(PG5) && !defined(PORTG5)
362 #   define PORTG5 PG5
363 #elif defined(PORTG5) && !defined(PG5)
364 #   define PG5 PORTG5
365 #endif
366 #if defined(PG6) && !defined(PORTG6)
367 #   define PORTG6 PG6
368 #elif defined(PORTG6) && !defined(PG6)
369 #   define PG6 PORTG6
370 #endif
371 #if defined(PG7) && !defined(PORTG7)
372 #   define PORTG7 PG7
373 #elif defined(PORTG7) && !defined(PG7)
374 #   define PG7 PORTG7
375 #endif
376
377 /* PORT H */
378
379 #if defined(PH0) && !defined(PORTH0)
380 #   define PORTH0 PH0
381 #elif defined(PORTH0) && !defined(PH0)
382 #   define PH0 PORTH0
383 #endif
384 #if defined(PH1) && !defined(PORTH1)
385 #   define PORTH1 PH1
386 #elif defined(PORTH1) && !defined(PH1)
387 #   define PH1 PORTH1
388 #endif
389 #if defined(PH2) && !defined(PORTH2)
390 #   define PORTH2 PH2
391 #elif defined(PORTH2) && !defined(PH2)
392 #   define PH2 PORTH2
393 #endif
394 #if defined(PH3) && !defined(PORTH3)
395 #   define PORTH3 PH3
396 #elif defined(PORTH3) && !defined(PH3)
397 #   define PH3 PORTH3
398 #endif
399 #if defined(PH4) && !defined(PORTH4)
400 #   define PORTH4 PH4
401 #elif defined(PORTH4) && !defined(PH4)
402 #   define PH4 PORTH4
403 #endif
404 #if defined(PH5) && !defined(PORTH5)
405 #   define PORTH5 PH5
406 #elif defined(PORTH5) && !defined(PH5)
407 #   define PH5 PORTH5
408 #endif
409 #if defined(PH6) && !defined(PORTH6)
410 #   define PORTH6 PH6
411 #elif defined(PORTH6) && !defined(PH6)
412 #   define PH6 PORTH6
413 #endif
414 #if defined(PH7) && !defined(PORTH7)
415 #   define PORTH7 PH7
416 #elif defined(PORTH7) && !defined(PH7)
417 #   define PH7 PORTH7
418 #endif
419
420 /* PORT J */
421
422 #if defined(PJ0) && !defined(PORTJ0)
423 #   define PORTJ0 PJ0
424 #elif defined(PORTJ0) && !defined(PJ0)
425 #   define PJ0 PORTJ0
426 #endif
427 #if defined(PJ1) && !defined(PORTJ1)
428 #   define PORTJ1 PJ1
429 #elif defined(PORTJ1) && !defined(PJ1)
430 #   define PJ1 PORTJ1
431 #endif
432 #if defined(PJ2) && !defined(PORTJ2)
433 #   define PORTJ2 PJ2
434 #elif defined(PORTJ2) && !defined(PJ2)
435 #   define PJ2 PORTJ2
436 #endif
437 #if defined(PJ3) && !defined(PORTJ3)
```



```
438 # define PORTJ3 PJ3
439 #elif defined(PORTJ3) && !defined(PJ3)
440 # define PJ3 PORTJ3
441 #endif
442 #if defined(PJ4) && !defined(PORTJ4)
443 # define PORTJ4 PJ4
444 #elif defined(PORTJ4) && !defined(PJ4)
445 # define PJ4 PORTJ4
446 #endif
447 #if defined(PJ5) && !defined(PORTJ5)
448 # define PORTJ5 PJ5
449 #elif defined(PORTJ5) && !defined(PJ5)
450 # define PJ5 PORTJ5
451 #endif
452 #if defined(PJ6) && !defined(PORTJ6)
453 # define PORTJ6 PJ6
454 #elif defined(PORTJ6) && !defined(PJ6)
455 # define PJ6 PORTJ6
456 #endif
457 #if defined(PJ7) && !defined(PORTJ7)
458 # define PORTJ7 PJ7
459 #elif defined(PORTJ7) && !defined(PJ7)
460 # define PJ7 PORTJ7
461 #endif
462
463 /* PORT K */
464
465 #if defined(PK0) && !defined(PORTK0)
466 # define PORTK0 PK0
467 #elif defined(PORTK0) && !defined(PK0)
468 # define PK0 PORTK0
469 #endif
470 #if defined(PK1) && !defined(PORTK1)
471 # define PORTK1 PK1
472 #elif defined(PORTK1) && !defined(PK1)
473 # define PK1 PORTK1
474 #endif
475 #if defined(PK2) && !defined(PORTK2)
476 # define PORTK2 PK2
477 #elif defined(PORTK2) && !defined(PK2)
478 # define PK2 PORTK2
479 #endif
480 #if defined(PK3) && !defined(PORTK3)
481 # define PORTK3 PK3
482 #elif defined(PORTK3) && !defined(PK3)
483 # define PK3 PORTK3
484 #endif
485 #if defined(PK4) && !defined(PORTK4)
486 # define PORTK4 PK4
487 #elif defined(PORTK4) && !defined(PK4)
488 # define PK4 PORTK4
489 #endif
490 #if defined(PK5) && !defined(PORTK5)
491 # define PORTK5 PK5
492 #elif defined(PORTK5) && !defined(PK5)
493 # define PK5 PORTK5
494 #endif
495 #if defined(PK6) && !defined(PORTK6)
496 # define PORTK6 PK6
497 #elif defined(PORTK6) && !defined(PK6)
498 # define PK6 PORTK6
499 #endif
500 #if defined(PK7) && !defined(PORTK7)
501 # define PORTK7 PK7
502 #elif defined(PORTK7) && !defined(PK7)
503 # define PK7 PORTK7
504 #endif
505
506 /* PORT L */
507
508 #if defined(PL0) && !defined(PORTL0)
509 # define PORTL0 PL0
510 #elif defined(PORTL0) && !defined(PL0)
511 # define PL0 PORTL0
512 #endif
513 #if defined(PL1) && !defined(PORTL1)
514 # define PORTL1 PL1
515 #elif defined(PORTL1) && !defined(PL1)
516 # define PL1 PORTL1
517 #endif
518 #if defined(PL2) && !defined(PORTL2)
519 # define PORTL2 PL2
520 #elif defined(PORTL2) && !defined(PL2)
521 # define PL2 PORTL2
522 #endif
523 #if defined(PL3) && !defined(PORTL3)
524 # define PORTL3 PL3
```

```

525 #elif defined(PORTL3) && !defined(PL3)
526 #   define PL3 PORTL3
527 #endif
528 #if defined(PL4) && !defined(PORTL4)
529 #   define PORTL4 PL4
530 #elif defined(PORTL4) && !defined(PL4)
531 #   define PL4 PORTL4
532 #endif
533 #if defined(PL5) && !defined(PORTL5)
534 #   define PORTL5 PL5
535 #elif defined(PORTL5) && !defined(PL5)
536 #   define PL5 PORTL5
537 #endif
538 #if defined(PL6) && !defined(PORTL6)
539 #   define PORTL6 PL6
540 #elif defined(PORTL6) && !defined(PL6)
541 #   define PL6 PORTL6
542 #endif
543 #if defined(PL7) && !defined(PORTL7)
544 #   define PORTL7 PL7
545 #elif defined(PORTL7) && !defined(PL7)
546 #   define PL7 PORTL7
547 #endif
548
549 #endif /* _AVR_PORTPINS_H_ */

```

25.26 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 __inline void [__attribute__](#) ((__always_inline__)) [__power_all_enable\(\)](#)
- void [clock_prescale_set](#) (clock_div_t __x)

25.26.1 Macro Definition Documentation

25.26.1.1 [clock_prescale_get](#) #define [clock_prescale_get](#) () (clock_div_t)(CLKPR & (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))

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`.

25.26.2 Function Documentation

25.26.2.1 [__attribute__](#)() static __inline void [__attribute__](#) ((__always_inline__)) [static]

25.27 power.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2006, 2007, 2008 Eric B. Weddington
2 Copyright (c) 2011 Frédéric Nadeau
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14 * Neither the name of the copyright holders nor the names of
15 contributors may be used to endorse or promote products derived
16 from this software without specific prior written permission.
17
18 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
19 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
20 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
21 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
22 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
23 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
24 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
25 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
26 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
28 POSSIBILITY OF SUCH DAMAGE. */
29
30 /* $Id: power.h 2525 2016-09-15 10:24:25Z pitchumani $ */
31
32 #ifndef _AVR_POWER_H_
33 #define _AVR_POWER_H_ 1
34
35 #include <avr/io.h>
36 #include <stdint.h>
37
38 /** \file */
39 /** \defgroup avr_power <avr/power.h>: Power Reduction Management
40
41 \code #include <avr/power.h>\endcode
42
43 Many AVR's contain a Power Reduction Register (PRR) or Registers (PRRx) that
44 allow you to reduce power consumption by disabling or enabling various on-board
45 peripherals as needed. Some devices have the XTAL Divide Control Register
46 (XDIV) which offer similar functionality as System Clock Prescaler
47 Register (CLKPR).
48
49 There are many macros in this header file that provide an easy interface
50 to enable or disable on-board peripherals to reduce power. See the table below.
51
52 \note Not all AVR devices have a Power Reduction Register (for example
53 the ATmega8). On those devices without a Power Reduction Register, the
54 power reduction macros are not available..
55
56 \note Not all AVR devices contain the same peripherals (for example, the LCD
57 interface), or they will be named differently (for example, USART and
58 USART0). Please consult your device's datasheet, or the header file, to
59 find out which macros are applicable to your device.
60
61 \note For device using the XTAL Divide Control Register (XDIV), when prescaler
62 is used, Timer/Counter0 can only be used in asynchronous mode. Keep in mind
63 that Timer/Counter0 source shall be less than ¼th of peripheral clock.
64 Therefore, when using a typical 32.768 kHz crystal, one shall not scale
65 the clock below 131.072 kHz.
66
67 */
68
69
70 /** \addtogroup avr_power
71
72 \anchor avr_powermacros
73 <small>
74 <center>
75 <table border="3">
76 <tr>
77 <td width="10%"><strong>Power Macro</strong></td>
78 <td width="15%"><strong>Description</strong></td>
79 </tr>
80 <tr>
81 <td>power_aca_disable()</td>
82 <td></td>
83 </tr>

```

```
84 <td>Disable the Analog Comparator on PortA.</td>
85 </tr>
86
87 <tr>
88 <td>power_aca_enable()</td>
89 <td>Enable the Analog Comparator on PortA.</td>
90 </tr>
91
92 <tr>
93 <td>power_adc_enable()</td>
94 <td>Enable the Analog to Digital Converter module.</td>
95 </tr>
96
97 <tr>
98 <td>power_adc_disable()</td>
99 <td>Disable the Analog to Digital Converter module.</td>
100 </tr>
101
102 <tr>
103 <td>power_adca_disable()</td>
104 <td>Disable the Analog to Digital Converter module on PortA</td>
105 </tr>
106
107 <tr>
108 <td>power_adca_enable()</td>
109 <td>Enable the Analog to Digital Converter module on PortA</td>
110 </tr>
111
112 <tr>
113 <td>power_evsys_disable()</td>
114 <td>Disable the EVSYS module</td>
115 </tr>
116
117 <tr>
118 <td>power_evsys_enable()</td>
119 <td>Enable the EVSYS module</td>
120 </tr>
121
122 <tr>
123 <td>power_hiresc_disable()</td>
124 <td>Disable the HIRES module on PortC</td>
125 </tr>
126
127 <tr>
128 <td>power_hiresc_enable()</td>
129 <td>Enable the HIRES module on PortC</td>
130 </tr>
131
132 <tr>
133 <td>power_lcd_enable()</td>
134 <td>Enable the LCD module.</td>
135 </tr>
136
137 <tr>
138 <td>power_lcd_disable()</td>
139 <td>Disable the LCD module.</td>
140 </tr>
141
142 <tr>
143 <td>power_pga_enable()</td>
144 <td>Enable the Programmable Gain Amplifier module.</td>
145 </tr>
146
147 <tr>
148 <td>power_pga_disable()</td>
149 <td>Disable the Programmable Gain Amplifier module.</td>
150 </tr>
151
152 <tr>
153 <td>power_pscr_enable()</td>
154 <td>Enable the Reduced Power Stage Controller module.</td>
155 </tr>
156
157 <tr>
158 <td>power_pscr_disable()</td>
159 <td>Disable the Reduced Power Stage Controller module.</td>
160 </tr>
161
162 <tr>
163 <td>power_psc0_enable()</td>
164 <td>Enable the Power Stage Controller 0 module.</td>
165 </tr>
166
167 <tr>
168 <td>power_psc0_disable()</td>
169 <td>Disable the Power Stage Controller 0 module.</td>
170 </tr>
```

```
171
172 <tr>
173 <td>power_psc1_enable()</td>
174 <td>Enable the Power Stage Controller 1 module.</td>
175 </tr>
176
177 <tr>
178 <td>power_psc1_disable()</td>
179 <td>Disable the Power Stage Controller 1 module.</td>
180 </tr>
181
182 <tr>
183 <td>power_psc2_enable()</td>
184 <td>Enable the Power Stage Controller 2 module.</td>
185 </tr>
186
187 <tr>
188 <td>power_psc2_disable()</td>
189 <td>Disable the Power Stage Controller 2 module.</td>
190 </tr>
191
192 <tr>
193 <td>power_ram0_enable()</td>
194 <td>Enable the SRAM block 0 .</td>
195 </tr>
196
197 <tr>
198 <td>power_ram0_disable()</td>
199 <td>Disable the SRAM block 0. </td>
200 </tr>
201
202 <tr>
203 <td>power_ram1_enable()</td>
204 <td>Enable the SRAM block 1 .</td>
205 </tr>
206
207 <tr>
208 <td>power_ram1_disable()</td>
209 <td>Disable the SRAM block 1. </td>
210 </tr>
211
212 <tr>
213 <td>power_ram2_enable()</td>
214 <td>Enable the SRAM block 2 .</td>
215 </tr>
216
217 <tr>
218 <td>power_ram2_disable()</td>
219 <td>Disable the SRAM block 2. </td>
220 </tr>
221
222 <tr>
223 <td>power_ram3_enable()</td>
224 <td>Enable the SRAM block 3 .</td>
225 </tr>
226
227 <tr>
228 <td>power_ram3_disable()</td>
229 <td>Disable the SRAM block 3. </td>
230 </tr>
231
232 <tr>
233 <td>power_rtc_disable()</td>
234 <td>Disable the RTC module</td>
235 </tr>
236
237 <tr>
238 <td>power_rtc_enable()</td>
239 <td>Enable the RTC module</td>
240 </tr>
241
242 <tr>
243 <td>power_spi_enable()</td>
244 <td>Enable the Serial Peripheral Interface module.</td>
245 </tr>
246
247 <tr>
248 <td>power_spi_disable()</td>
249 <td>Disable the Serial Peripheral Interface module.</td>
250 </tr>
251
252 <tr>
253 <td>power_spic_disable()</td>
254 <td>Disable the SPI module on PortC</td>
255 </tr>
256
257 <tr>
```

```
258 <td>power_spic_enable()</td>
259 <td>Enable the SPI module on PortC</td>
260 </tr>
261
262 <tr>
263 <td>power_spid_disable()</td>
264 <td>Disable the SPI module on PortD</td>
265 </tr>
266
267 <tr>
268 <td>power_spid_enable()</td>
269 <td>Enable the SPI module on PortD</td>
270 </tr>
271
272 <tr>
273 <td>power_tc0c_disable()</td>
274 <td>Disable the TC0 module on PortC</td>
275 </tr>
276
277 <tr>
278 <td>power_tc0c_enable()</td>
279 <td>Enable the TC0 module on PortC</td>
280 </tr>
281
282 <tr>
283 <td>power_tc0d_disable()</td>
284 <td>Disable the TC0 module on PortD</td>
285 </tr>
286
287 <tr>
288 <td>power_tc0d_enable()</td>
289 <td>Enable the TC0 module on PortD</td>
290 </tr>
291
292 <tr>
293 <td>power_tc0e_disable()</td>
294 <td>Disable the TC0 module on PortE</td>
295 </tr>
296
297 <tr>
298 <td>power_tc0e_enable()</td>
299 <td>Enable the TC0 module on PortE</td>
300 </tr>
301
302 <tr>
303 <td>power_tc0f_disable()</td>
304 <td>Disable the TC0 module on PortF</td>
305 </tr>
306
307 <tr>
308 <td>power_tc0f_enable()</td>
309 <td>Enable the TC0 module on PortF</td>
310 </tr>
311
312 <tr>
313 <td>power_tcl1c_disable()</td>
314 <td>Disable the TC1 module on PortC</td>
315 </tr>
316
317 <tr>
318 <td>power_tcl1c_enable()</td>
319 <td>Enable the TC1 module on PortC</td>
320 </tr>
321
322 <tr>
323 <td>power_twic_disable()</td>
324 <td>Disable the Two Wire Interface module on PortC</td>
325 </tr>
326
327 <tr>
328 <td>power_twic_enable()</td>
329 <td>Enable the Two Wire Interface module on PortC</td>
330 </tr>
331
332 <tr>
333 <td>power_twie_disable()</td>
334 <td>Disable the Two Wire Interface module on PortE</td>
335 </tr>
336
337 <tr>
338 <td>power_twie_enable()</td>
339 <td>Enable the Two Wire Interface module on PortE</td>
340 </tr>
341
342 <tr>
343 <td>power_timer0_enable()</td>
344 <td>Enable the Timer 0 module.</td>
```

```
345 </tr>
346
347 <tr>
348 <td>power_timer0_disable()</td>
349 <td>Disable the Timer 0 module.</td>
350 </tr>
351
352 <tr>
353 <td>power_timer1_enable()</td>
354 <td>Enable the Timer 1 module.</td>
355 </tr>
356
357 <tr>
358 <td>power_timer1_disable()</td>
359 <td>Disable the Timer 1 module.</td>
360 </tr>
361
362 <tr>
363 <td>power_timer2_enable()</td>
364 <td>Enable the Timer 2 module.</td>
365 </tr>
366
367 <tr>
368 <td>power_timer2_disable()</td>
369 <td>Disable the Timer 2 module.</td>
370 </tr>
371
372 <tr>
373 <td>power_timer3_enable()</td>
374 <td>Enable the Timer 3 module.</td>
375 </tr>
376
377 <tr>
378 <td>power_timer3_disable()</td>
379 <td>Disable the Timer 3 module.</td>
380 </tr>
381
382 <tr>
383 <td>power_timer4_enable()</td>
384 <td>Enable the Timer 4 module.</td>
385 </tr>
386
387 <tr>
388 <td>power_timer4_disable()</td>
389 <td>Disable the Timer 4 module.</td>
390 </tr>
391
392 <tr>
393 <td>power_timer5_enable()</td>
394 <td>Enable the Timer 5 module.</td>
395 </tr>
396
397 <tr>
398 <td>power_timer5_disable()</td>
399 <td>Disable the Timer 5 module.</td>
400 </tr>
401
402 <tr>
403 <td>power_twi_enable()</td>
404 <td>Enable the Two Wire Interface module.</td>
405 </tr>
406
407 <tr>
408 <td>power_twi_disable()</td>
409 <td>Disable the Two Wire Interface module.</td>
410 </tr>
411
412 <tr>
413 <td>power_usart_enable()</td>
414 <td>Enable the USART module.</td>
415 </tr>
416
417 <tr>
418 <td>power_usart_disable()</td>
419 <td>Disable the USART module.</td>
420 </tr>
421
422 <tr>
423 <td>power_usart0_enable()</td>
424 <td>Enable the USART 0 module.</td>
425 </tr>
426
427 <tr>
428 <td>power_usart0_disable()</td>
429 <td>Disable the USART 0 module.</td>
430 </tr>
431
```

```
432 <tr>
433 <td>power_usart1_enable()</td>
434 <td>Enable the USART 1 module.</td>
435 </tr>
436
437 <tr>
438 <td>power_usart1_disable()</td>
439 <td>Disable the USART 1 module.</td>
440 </tr>
441
442 <tr>
443 <td>power_usart2_enable()</td>
444 <td>Enable the USART 2 module.</td>
445 </tr>
446
447 <tr>
448 <td>power_usart2_disable()</td>
449 <td>Disable the USART 2 module.</td>
450 </tr>
451
452 <tr>
453 <td>power_usart3_enable()</td>
454 <td>Enable the USART 3 module.</td>
455 </tr>
456
457 <tr>
458 <td>power_usart3_disable()</td>
459 <td>Disable the USART 3 module.</td>
460 </tr>
461
462 <tr>
463 <td>power_usartc0_disable()</td>
464 <td> Disable the USART0 module on PortC</td>
465 </tr>
466
467 <tr>
468 <td>power_usartc0_enable()</td>
469 <td> Enable the USART0 module on PortC</td>
470 </tr>
471
472 <tr>
473 <td>power_usartd0_disable()</td>
474 <td> Disable the USART0 module on PortD</td>
475 </tr>
476
477 <tr>
478 <td>power_usartd0_enable()</td>
479 <td> Enable the USART0 module on PortD</td>
480 </tr>
481
482 <tr>
483 <td>power_usarte0_disable()</td>
484 <td> Disable the USART0 module on PortE</td>
485 </tr>
486
487 <tr>
488 <td>power_usarte0_enable()</td>
489 <td> Enable the USART0 module on PortE</td>
490 </tr>
491
492 <tr>
493 <td>power_usartf0_disable()</td>
494 <td> Disable the USART0 module on PortF</td>
495 </tr>
496
497 <tr>
498 <td>power_usartf0_enable()</td>
499 <td> Enable the USART0 module on PortF</td>
500 </tr>
501
502 <tr>
503 <td>power_usb_enable()</td>
504 <td>Enable the USB module.</td>
505 </tr>
506
507 <tr>
508 <td>power_usb_disable()</td>
509 <td>Disable the USB module.</td>
510 </tr>
511
512 <tr>
513 <td>power_usi_enable()</td>
514 <td>Enable the Universal Serial Interface module.</td>
515 </tr>
516
517 <tr>
518 <td>power_usi_disable()</td>
```



```

519 <td>Disable the Universal Serial Interface module.</td>
520 </tr>
521
522 <tr>
523 <td>power_vadc_enable()</td>
524 <td>Enable the Voltage ADC module.</td>
525 </tr>
526
527 <tr>
528 <td>power_vadc_disable()</td>
529 <td>Disable the Voltage ADC module.</td>
530 </tr>
531
532 <tr>
533 <td>power_all_enable()</td>
534 <td>Enable all modules.</td>
535 </tr>
536
537 <tr>
538 <td>power_all_disable()</td>
539 <td>Disable all modules.</td>
540 </tr>
541 </table>
542 </center>
543 </small>
544
545 @} */
546
547 #if defined(__AVR_HAVE_PRR_PRADC)
548 #define power_adc_enable()      (PRR &= (uint8_t)~(1 << PRADC))
549 #define power_adc_disable()    (PRR |= (uint8_t)(1 << PRADC))
550 #endif
551
552 #if defined(__AVR_HAVE_PRR_PRCAN)
553 #define power_can_enable()      (PRR &= (uint8_t)~(1 << PRCAN))
554 #define power_can_disable()    (PRR |= (uint8_t)(1 << PRCAN))
555 #endif
556
557 #if defined(__AVR_HAVE_PRR_PRLCD)
558 #define power_lcd_enable()      (PRR &= (uint8_t)~(1 << PRLCD))
559 #define power_lcd_disable()    (PRR |= (uint8_t)(1 << PRLCD))
560 #endif
561
562 #if defined(__AVR_HAVE_PRR_PRLIN)
563 #define power_lin_enable()      (PRR &= (uint8_t)~(1 << PRLIN))
564 #define power_lin_disable()    (PRR |= (uint8_t)(1 << PRLIN))
565 #endif
566
567 #if defined(__AVR_HAVE_PRR_PRPSC)
568 #define power_psc_enable()      (PRR &= (uint8_t)~(1 << PRPSC))
569 #define power_psc_disable()    (PRR |= (uint8_t)(1 << PRPSC))
570 #endif
571
572 #if defined(__AVR_HAVE_PRR_PRPSC0)
573 #define power_psc0_enable()     (PRR &= (uint8_t)~(1 << PRPSC0))
574 #define power_psc0_disable()   (PRR |= (uint8_t)(1 << PRPSC0))
575 #endif
576
577 #if defined(__AVR_HAVE_PRR_PRPSC1)
578 #define power_psc1_enable()     (PRR &= (uint8_t)~(1 << PRPSC1))
579 #define power_psc1_disable()   (PRR |= (uint8_t)(1 << PRPSC1))
580 #endif
581
582 #if defined(__AVR_HAVE_PRR_PRPSC2)
583 #define power_psc2_enable()     (PRR &= (uint8_t)~(1 << PRPSC2))
584 #define power_psc2_disable()   (PRR |= (uint8_t)(1 << PRPSC2))
585 #endif
586
587 #if defined(__AVR_HAVE_PRR_PRPSCR)
588 #define power_pscr_enable()     (PRR &= (uint8_t)~(1 << PRPSCR))
589 #define power_pscr_disable()   (PRR |= (uint8_t)(1 << PRPSCR))
590 #endif
591
592 #if defined(__AVR_HAVE_PRR_PRSPI)
593 #define power_spi_enable()      (PRR &= (uint8_t)~(1 << PRSPI))
594 #define power_spi_disable()    (PRR |= (uint8_t)(1 << PRSPI))
595 #endif
596
597 #if defined(__AVR_HAVE_PRR_PRTIM0)
598 #define power_timer0_enable()   (PRR &= (uint8_t)~(1 << PRTIM0))
599 #define power_timer0_disable() (PRR |= (uint8_t)(1 << PRTIM0))
600 #endif
601
602 #if defined(__AVR_HAVE_PRR_PRTIM1)
603 #define power_timer1_enable()   (PRR &= (uint8_t)~(1 << PRTIM1))
604 #define power_timer1_disable() (PRR |= (uint8_t)(1 << PRTIM1))
605 #endif

```

```

606
607 #if defined(__AVR_HAVE_PRR_PRTIM2)
608 #define power_timer2_enable() (PRR &= (uint8_t)~(1 << PRTIM2))
609 #define power_timer2_disable() (PRR |= (uint8_t)(1 << PRTIM2))
610 #endif
611
612 #if defined(__AVR_HAVE_PRR_PRTWI)
613 #define power_twi_enable() (PRR &= (uint8_t)~(1 << PRTWI))
614 #define power_twi_disable() (PRR |= (uint8_t)(1 << PRTWI))
615 #endif
616
617 #if defined(__AVR_HAVE_PRR_PRUSART)
618 #define power_usart_enable() (PRR &= (uint8_t)~(1 << PRUSART))
619 #define power_usart_disable() (PRR |= (uint8_t)(1 << PRUSART))
620 #endif
621
622 #if defined(__AVR_HAVE_PRR_PRUSART0)
623 #define power_usart0_enable() (PRR &= (uint8_t)~(1 << PRUSART0))
624 #define power_usart0_disable() (PRR |= (uint8_t)(1 << PRUSART0))
625 #endif
626
627 #if defined(__AVR_HAVE_PRR_PRUSART1)
628 #define power_usart1_enable() (PRR &= (uint8_t)~(1 << PRUSART1))
629 #define power_usart1_disable() (PRR |= (uint8_t)(1 << PRUSART1))
630 #endif
631
632 #if defined(__AVR_HAVE_PRR_PRUSI)
633 #define power_usi_enable() (PRR &= (uint8_t)~(1 << PRUSI))
634 #define power_usi_disable() (PRR |= (uint8_t)(1 << PRUSI))
635 #endif
636
637 #if defined(__AVR_HAVE_PRR0_PRADC)
638 #define power_adc_enable() (PRR0 &= (uint8_t)~(1 << PRADC))
639 #define power_adc_disable() (PRR0 |= (uint8_t)(1 << PRADC))
640 #endif
641
642 #if defined(__AVR_HAVE_PRR0_PRCO)
643 #define power_clock_output_enable() (PRR0 &= (uint8_t)~(1 << PRCO))
644 #define power_clock_output_disable() (PRR0 |= (uint8_t)(1 << PRCO))
645 #endif
646
647 #if defined(__AVR_HAVE_PRR0_PRCRC)
648 #define power_crc_enable() (PRR0 &= (uint8_t)~(1 << PRCRC))
649 #define power_crc_disable() (PRR0 |= (uint8_t)(1 << PRCRC))
650 #endif
651
652 #if defined(__AVR_HAVE_PRR0_PRCU)
653 #define power_crypto_enable() (PRR0 &= (uint8_t)~(1 << PRCU))
654 #define power_crypto_disable() (PRR0 |= (uint8_t)(1 << PRCU))
655 #endif
656
657 #if defined(__AVR_HAVE_PRR0_PRDS)
658 #define power_irdriver_enable() (PRR0 &= (uint8_t)~(1 << PRDS))
659 #define power_irdriver_disable() (PRR0 |= (uint8_t)(1 << PRDS))
660 #endif
661
662 #if defined(__AVR_HAVE_PRR0_PRLFR)
663 #define power_lfreceiver_enable() (PRR0 &= (uint8_t)~(1 << PRLFR))
664 #define power_lfreceiver_disable() (PRR0 |= (uint8_t)(1 << PRLFR))
665 #endif
666
667 #if defined(__AVR_HAVE_PRR0_PRLFRS)
668 #define power_lfrs_enable() (PRR0 &= (uint8_t)~(1 << PRLFRS))
669 #define power_lfrs_disable() (PRR0 |= (uint8_t)(1 << PRLFRS))
670 #endif
671
672 #if defined(__AVR_HAVE_PRR0_PRLIN)
673 #define power_lin_enable() (PRR0 &= (uint8_t)~(1 << PRLIN))
674 #define power_lin_disable() (PRR0 |= (uint8_t)(1 << PRLIN))
675 #endif
676
677 #if defined(__AVR_HAVE_PRR0_PRPGA)
678 #define power_pga_enable() (PRR0 &= (uint8_t)~(1 << PRPGA))
679 #define power_pga_disable() (PRR0 |= (uint8_t)(1 << PRPGA))
680 #endif
681
682 #if defined(__AVR_HAVE_PRR0_PRRXDC)
683 #define power_receive_dsp_control_enable() (PRR0 &= (uint8_t)~(1 << PRRXDC))
684 #define power_receive_dsp_control_disable() (PRR0 |= (uint8_t)(1 << PRRXDC))
685 #endif
686
687 #if defined(__AVR_HAVE_PRR0_PRSPI)
688 #define power_spi_enable() (PRR0 &= (uint8_t)~(1 << PRSPI))
689 #define power_spi_disable() (PRR0 |= (uint8_t)(1 << PRSPI))
690 #endif
691
692 #if defined(__AVR_HAVE_PRR0_PRT0)

```

```

693 #define power_timer0_enable()      (PRR0 &= (uint8_t)~(1 << PRT0))
694 #define power_timer0_disable()    (PRR0 |= (uint8_t) (1 << PRT0))
695 #endif
696
697 #if defined(__AVR_HAVE_PRR0_PRTIM0)
698 #define power_timer0_enable()      (PRR0 &= (uint8_t)~(1 << PRTIM0))
699 #define power_timer0_disable()    (PRR0 |= (uint8_t) (1 << PRTIM0))
700 #endif
701
702 #if defined(__AVR_HAVE_PRR0_PRT1)
703 #define power_timer1_enable()      (PRR0 &= (uint8_t)~(1 << PRT1))
704 #define power_timer1_disable()    (PRR0 |= (uint8_t) (1 << PRT1))
705 #endif
706
707 #if defined(__AVR_HAVE_PRR0_PRTIM1)
708 #define power_timer1_enable()      (PRR0 &= (uint8_t)~(1 << PRTIM1))
709 #define power_timer1_disable()    (PRR0 |= (uint8_t) (1 << PRTIM1))
710 #endif
711
712 #if defined(__AVR_HAVE_PRR0_PRT2)
713 #define power_timer2_enable()      (PRR0 &= (uint8_t)~(1 << PRT2))
714 #define power_timer2_disable()    (PRR0 |= (uint8_t) (1 << PRT2))
715 #endif
716
717 #if defined(__AVR_HAVE_PRR0_PRTIM2)
718 #define power_timer2_enable()      (PRR0 &= (uint8_t)~(1 << PRTIM2))
719 #define power_timer2_disable()    (PRR0 |= (uint8_t) (1 << PRTIM2))
720 #endif
721
722 #if defined(__AVR_HAVE_PRR0_PRT3)
723 #define power_timer3_enable()      (PRR0 &= (uint8_t)~(1 << PRT3))
724 #define power_timer3_disable()    (PRR0 |= (uint8_t) (1 << PRT3))
725 #endif
726
727 #if defined(__AVR_HAVE_PRR0_PRTM)
728 #define power_timermodulator_enable() (PRR0 &= (uint8_t)~(1 << PRTM))
729 #define power_timermodulator_disable() (PRR0 |= (uint8_t) (1 << PRTM))
730 #endif
731
732 #if defined(__AVR_HAVE_PRR0_PRTWI)
733 #define power_twi_enable()         (PRR0 &= (uint8_t)~(1 << PRTWI))
734 #define power_twi_disable()       (PRR0 |= (uint8_t) (1 << PRTWI))
735 #endif
736
737 #if defined(__AVR_HAVE_PRR0_PRTWI1)
738 #define power_twi1_enable()        (PRR0 &= (uint8_t)~(1 << PRTWI1))
739 #define power_twi1_disable()       (PRR0 |= (uint8_t) (1 << PRTWI1))
740 #endif
741
742 #if defined(__AVR_HAVE_PRR0_PRTXDC)
743 #define power_transmit_dsp_control_enable() (PRR0 &= (uint8_t)~(1 << PRTXDC))
744 #define power_transmit_dsp_control_disable() (PRR0 |= (uint8_t) (1 << PRTXDC))
745 #endif
746
747 #if defined(__AVR_HAVE_PRR0_PRUSART0)
748 #define power_usart0_enable()      (PRR0 &= (uint8_t)~(1 << PRUSART0))
749 #define power_usart0_disable()    (PRR0 |= (uint8_t) (1 << PRUSART0))
750 #endif
751
752 #if defined(__AVR_HAVE_PRR0_PRUSART1)
753 #define power_usart1_enable()      (PRR0 &= (uint8_t)~(1 << PRUSART1))
754 #define power_usart1_disable()    (PRR0 |= (uint8_t) (1 << PRUSART1))
755 #endif
756
757 #if defined(__AVR_HAVE_PRR0_PRVADC)
758 #define power_vadc_enable()        (PRR0 &= (uint8_t)~(1 << PRVADC))
759 #define power_vadc_disable()      (PRR0 |= (uint8_t) (1 << PRVADC))
760 #endif
761
762 #if defined(__AVR_HAVE_PRR0_PRVM)
763 #define power_voltage_monitor_enable() (PRR0 &= (uint8_t)~(1 << PRVM))
764 #define power_voltage_monitor_disable() (PRR0 |= (uint8_t) (1 << PRVM))
765 #endif
766
767 #if defined(__AVR_HAVE_PRR0_PRVRM)
768 #define power_vrm_enable()         (PRR0 &= (uint8_t)~(1 << PRVRM))
769 #define power_vrm_disable()       (PRR0 |= (uint8_t) (1 << PRVRM))
770 #endif
771
772 #if defined(__AVR_HAVE_PRR1_PRAES)
773 #define power_aes_enable()         (PRR1 &= (uint8_t)~(1 << PRAES))
774 #define power_aes_disable()       (PRR1 |= (uint8_t) (1 << PRAES))
775 #endif
776
777 #if defined(__AVR_HAVE_PRR1_PRCI)
778 #define power_cinterface_enable()  (PRR1 &= (uint8_t)~(1 << PRCI))
779 #define power_cinterface_disable() (PRR1 |= (uint8_t) (1 << PRCI))

```

```

780 #endif
781
782 #if defined(__AVR_HAVE_PRR1_PRHSSPI)
783 #define power_hsspi_enable() (PRR1 &= (uint8_t)~(1 << PRHSSPI))
784 #define power_hsspi_disable() (PRR1 |= (uint8_t) (1 << PRHSSPI))
785 #endif
786
787 #if defined(__AVR_HAVE_PRR1_PRKB)
788 #define power_kb_enable() (PRR1 &= (uint8_t)~(1 << PRKB))
789 #define power_kb_disable() (PRR1 |= (uint8_t) (1 << PRKB))
790 #endif
791
792 #if defined(__AVR_HAVE_PRR1_PRLFPH)
793 #define power_lfph_enable() (PRR1 &= (uint8_t)~(1 << PRLFPH))
794 #define power_lfph_disable() (PRR1 |= (uint8_t) (1 << PRLFPH))
795 #endif
796
797 #if defined(__AVR_HAVE_PRR1_PRLFR)
798 #define power_lfreceiver_enable() (PRR1 &= (uint8_t)~(1 << PRLFR))
799 #define power_lfreceiver_disable() (PRR1 |= (uint8_t) (1 << PRLFR))
800 #endif
801
802 #if defined(__AVR_HAVE_PRR1_PRLFTP)
803 #define power_lftp_enable() (PRR1 &= (uint8_t)~(1 << PRLFTP))
804 #define power_lftp_disable() (PRR1 |= (uint8_t) (1 << PRLFTP))
805 #endif
806
807 #if defined(__AVR_HAVE_PRR1_PRSCI)
808 #define power_sci_enable() (PRR1 &= (uint8_t)~(1 << PRSCI))
809 #define power_sci_disable() (PRR1 |= (uint8_t) (1 << PRSCI))
810 #endif
811
812 #if defined(__AVR_HAVE_PRR1_PRSPI)
813 #define power_spi_enable() (PRR1 &= (uint8_t)~(1 << PRSPI))
814 #define power_spi_disable() (PRR1 |= (uint8_t) (1 << PRSPI))
815 #endif
816
817 #if defined(__AVR_HAVE_PRR1_PRT1)
818 #define power_timer1_enable() (PRR1 &= (uint8_t)~(1 << PRT1))
819 #define power_timer1_disable() (PRR1 |= (uint8_t) (1 << PRT1))
820 #endif
821
822 #if defined(__AVR_HAVE_PRR1_PRT2)
823 #define power_timer2_enable() (PRR1 &= (uint8_t)~(1 << PRT2))
824 #define power_timer2_disable() (PRR1 |= (uint8_t) (1 << PRT2))
825 #endif
826
827 #if defined(__AVR_HAVE_PRR1_PRT3)
828 #define power_timer3_enable() (PRR1 &= (uint8_t)~(1 << PRT3))
829 #define power_timer3_disable() (PRR1 |= (uint8_t) (1 << PRT3))
830 #endif
831
832 #if defined(__AVR_HAVE_PRR1_PRT4)
833 #define power_timer4_enable() (PRR1 &= (uint8_t)~(1 << PRT4))
834 #define power_timer4_disable() (PRR1 |= (uint8_t) (1 << PRT4))
835 #endif
836
837 #if defined(__AVR_HAVE_PRR1_PRT5)
838 #define power_timer5_enable() (PRR1 &= (uint8_t)~(1 << PRT5))
839 #define power_timer5_disable() (PRR1 |= (uint8_t) (1 << PRT5))
840 #endif
841
842 #if defined(__AVR_HAVE_PRR1_PRTIM3)
843 #define power_timer3_enable() (PRR1 &= (uint8_t)~(1 << PRTIM3))
844 #define power_timer3_disable() (PRR1 |= (uint8_t) (1 << PRTIM3))
845 #endif
846
847 #if defined(__AVR_HAVE_PRR1_PRTIM4)
848 #define power_timer4_enable() (PRR1 &= (uint8_t)~(1 << PRTIM4))
849 #define power_timer4_disable() (PRR1 |= (uint8_t) (1 << PRTIM4))
850 #endif
851
852 #if defined(__AVR_HAVE_PRR1_PRTIM5)
853 #define power_timer5_enable() (PRR1 &= (uint8_t)~(1 << PRTIM5))
854 #define power_timer5_disable() (PRR1 |= (uint8_t) (1 << PRTIM5))
855 #endif
856
857 #if defined(__AVR_HAVE_PRR1_PRTRX24)
858 #define power_transceiver_enable() (PRR1 &= (uint8_t)~(1 << PRTRX24))
859 #define power_transceiver_disable() (PRR1 |= (uint8_t) (1 << PRTRX24))
860 #endif
861
862 #if defined(__AVR_HAVE_PRR1_PRUSART1)
863 #define power_usart1_enable() (PRR1 &= (uint8_t)~(1 << PRUSART1))
864 #define power_usart1_disable() (PRR1 |= (uint8_t) (1 << PRUSART1))
865 #endif
866

```

```

867 #if defined(__AVR_HAVE_PRR1_PRUSART2)
868 #define power_usart2_enable() (PRR1 &= (uint8_t)~(1 << PRUSART2))
869 #define power_usart2_disable() (PRR1 |= (uint8_t)(1 << PRUSART2))
870 #endif
871
872 #if defined(__AVR_HAVE_PRR1_PRUSART3)
873 #define power_usart3_enable() (PRR1 &= (uint8_t)~(1 << PRUSART3))
874 #define power_usart3_disable() (PRR1 |= (uint8_t)(1 << PRUSART3))
875 #endif
876
877 #if defined(__AVR_HAVE_PRR1_PRUSB)
878 #define power_usb_enable() (PRR1 &= (uint8_t)~(1 << PRUSB))
879 #define power_usb_disable() (PRR1 |= (uint8_t)(1 << PRUSB))
880 #endif
881
882 #if defined(__AVR_HAVE_PRR1_PRUSBH)
883 #define power_usbh_enable() (PRR1 &= (uint8_t)~(1 << PRUSBH))
884 #define power_usbh_disable() (PRR1 |= (uint8_t)(1 << PRUSBH))
885 #endif
886
887 #if defined(__AVR_HAVE_PRR2_PRDF)
888 #define power_data_fifo_enable() (PRR2 &= (uint8_t)~(1 << PRDF))
889 #define power_data_fifo_disable() (PRR2 |= (uint8_t)(1 << PRDF))
890 #endif
891
892 #if defined(__AVR_HAVE_PRR2_PRIDS)
893 #define power_id_scan_enable() (PRR2 &= (uint8_t)~(1 << PRIDS))
894 #define power_id_scan_disable() (PRR2 |= (uint8_t)(1 << PRIDS))
895 #endif
896
897 #if defined(__AVR_HAVE_PRR2_PRRAM0)
898 #define power_ram0_enable() (PRR2 &= (uint8_t)~(1 << PRRAM0))
899 #define power_ram0_disable() (PRR2 |= (uint8_t)(1 << PRRAM0))
900 #endif
901
902 #if defined(__AVR_HAVE_PRR2_PRRAM1)
903 #define power_ram1_enable() (PRR2 &= (uint8_t)~(1 << PRRAM1))
904 #define power_ram1_disable() (PRR2 |= (uint8_t)(1 << PRRAM1))
905 #endif
906
907 #if defined(__AVR_HAVE_PRR2_PRRAM2)
908 #define power_ram2_enable() (PRR2 &= (uint8_t)~(1 << PRRAM2))
909 #define power_ram2_disable() (PRR2 |= (uint8_t)(1 << PRRAM2))
910 #endif
911
912 #if defined(__AVR_HAVE_PRR2_PRRAM3)
913 #define power_ram3_enable() (PRR2 &= (uint8_t)~(1 << PRRAM3))
914 #define power_ram3_disable() (PRR2 |= (uint8_t)(1 << PRRAM3))
915 #endif
916
917 #if defined(__AVR_HAVE_PRR2_PRRS)
918 #define power_rssi_buffer_enable() (PRR2 &= (uint8_t)~(1 << PRRS))
919 #define power_rssi_buffer_disable() (PRR2 |= (uint8_t)(1 << PRRS))
920 #endif
921
922 #if defined(__AVR_HAVE_PRR2_PRSF)
923 #define power_preamble_rssi_fifo_enable() (PRR2 &= (uint8_t)~(1 << PRSF))
924 #define power_preamble_rssi_fifo_disable() (PRR2 |= (uint8_t)(1 << PRSF))
925 #endif
926
927 #if defined(__AVR_HAVE_PRR2_PRSPI2)
928 #define power_spi2_enable() (PRR2 &= (uint8_t)~(1 << PRSPI2))
929 #define power_spi2_disable() (PRR2 |= (uint8_t)(1 << PRSPI2))
930 #endif
931
932 #if defined(__AVR_HAVE_PRR2_PRSSM)
933 #define power_sequencer_state_machine_enable() (PRR2 &= (uint8_t)~(1 << PRSSM))
934 #define power_sequencer_state_machine_disable() (PRR2 |= (uint8_t)(1 << PRSSM))
935 #endif
936
937 #if defined(__AVR_HAVE_PRR2_PRTM)
938 #define power_tx_modulator_enable() (PRR2 &= (uint8_t)~(1 << PRTM))
939 #define power_tx_modulator_disable() (PRR2 |= (uint8_t)(1 << PRTM))
940 #endif
941
942 #if defined(__AVR_HAVE_PRR2_PRTWI2)
943 #define power_tw_i2_enable() (PRR2 &= (uint8_t)~(1 << PRTWI2))
944 #define power_tw_i2_disable() (PRR2 |= (uint8_t)(1 << PRTWI2))
945 #endif
946
947 #if defined(__AVR_HAVE_PRR2_PRXA)
948 #define power_rx_buffer_A_enable() (PRR2 &= (uint8_t)~(1 << PRXA))
949 #define power_rx_buffer_A_disable() (PRR2 |= (uint8_t)(1 << PRXA))
950 #endif
951
952 #if defined(__AVR_HAVE_PRR2_PRXB)
953 #define power_rx_buffer_B_enable() (PRR2 &= (uint8_t)~(1 << PRXB))

```

```

954 #define power_rx_buffer_B_disable()      (PRR2 |= (uint8_t)(1 << PRXB))
955 #endif
956
957 #if defined(__AVR_HAVE_PRGEN_AES)
958 #define power_aes_enable()                (PR_PRGEN &= (uint8_t)~(PR_AES_bm))
959 #define power_aes_disable()              (PR_PRGEN |= (uint8_t)PR_AES_bm)
960 #endif
961
962 #if defined(__AVR_HAVE_PRGEN_DMA)
963 #define power_dma_enable()                (PR_PRGEN &= (uint8_t)~(PR_DMA_bm))
964 #define power_dma_disable()              (PR_PRGEN |= (uint8_t)PR_DMA_bm)
965 #endif
966
967 #if defined(__AVR_HAVE_PRGEN_EBI)
968 #define power_ebi_enable()                (PR_PRGEN &= (uint8_t)~(PR_EBI_bm))
969 #define power_ebi_disable()              (PR_PRGEN |= (uint8_t)PR_EBI_bm)
970 #endif
971
972 #if defined(__AVR_HAVE_PRGEN_EDMA)
973 #define power_edma_enable()               (PR_PRGEN &= (uint8_t)~(PR_EDMA_bm))
974 #define power_edma_disable()             (PR_PRGEN |= (uint8_t)PR_EDMA_bm)
975 #endif
976
977 #if defined(__AVR_HAVE_PRGEN_EVSYS)
978 #define power_evsys_enable()              (PR_PRGEN &= (uint8_t)~(PR_EVSYS_bm))
979 #define power_evsys_disable()            (PR_PRGEN |= (uint8_t)PR_EVSYS_bm)
980 #endif
981
982 #if defined(__AVR_HAVE_PRGEN_LCD)
983 #define power_lcd_enable()                (PR_PRGEN &= (uint8_t)~(PR_LCD_bm))
984 #define power_lcd_disable()              (PR_PRGEN |= (uint8_t)PR_LCD_bm)
985 #endif
986
987 #if defined(__AVR_HAVE_PRGEN_RTC)
988 #define power_rtc_enable()                (PR_PRGEN &= (uint8_t)~(PR_RTC_bm))
989 #define power_rtc_disable()              (PR_PRGEN |= (uint8_t)PR_RTC_bm)
990 #endif
991
992 #if defined(__AVR_HAVE_PRGEN_USB)
993 #define power_usb_enable()                (PR_PRGEN &= (uint8_t)~(PR_USB_bm))
994 #define power_usb_disable()              (PR_PRGEN &= (uint8_t)PR_USB_bm))
995 #endif
996
997 #if defined(__AVR_HAVE_PRGEN_XCL)
998 #define power_xcl_enable()                (PR_PRGEN &= (uint8_t)~(PR_XCL_bm))
999 #define power_xcl_disable()              (PR_PRGEN |= (uint8_t)PR_XCL_bm)
1000 #endif
1001
1002 #if defined(__AVR_HAVE_PRPA_AC)
1003 #define power_aca_enable()                (PR_PRPA &= (uint8_t)~(PR_AC_bm))
1004 #define power_aca_disable()              (PR_PRPA |= (uint8_t)PR_AC_bm)
1005 #endif
1006
1007 #if defined(__AVR_HAVE_PRPA_ADC)
1008 #define power_adca_enable()               (PR_PRPA &= (uint8_t)~(PR_ADC_bm))
1009 #define power_adca_disable()             (PR_PRPA |= (uint8_t)PR_ADC_bm)
1010 #endif
1011
1012 #if defined(__AVR_HAVE_PRPA_DAC)
1013 #define power_daca_enable()               (PR_PRPA &= (uint8_t)~(PR_DAC_bm))
1014 #define power_daca_disable()             (PR_PRPA |= (uint8_t)PR_DAC_bm)
1015 #endif
1016
1017 #if defined(__AVR_HAVE_PRPB_AC)
1018 #define power_acb_enable()                (PR_PRPB &= (uint8_t)~(PR_AC_bm))
1019 #define power_acb_disable()              (PR_PRPB |= (uint8_t)PR_AC_bm)
1020 #endif
1021
1022 #if defined(__AVR_HAVE_PRPB_ADC)
1023 #define power_adcb_enable()               (PR_PRPB &= (uint8_t)~(PR_ADC_bm))
1024 #define power_adcb_disable()             (PR_PRPB |= (uint8_t)PR_ADC_bm)
1025 #endif
1026
1027 #if defined(__AVR_HAVE_PRPB_DAC)
1028 #define power_dacb_enable()               (PR_PRPB &= (uint8_t)~(PR_DAC_bm))
1029 #define power_dacb_disable()             (PR_PRPB |= (uint8_t)PR_DAC_bm)
1030 #endif
1031
1032 #if defined(__AVR_HAVE_PRPC_HIRES)
1033 #define power_hiresc_enable()             (PR_PRPC &= (uint8_t)~(PR_HIRES_bm))
1034 #define power_hiresc_disable()           (PR_PRPC |= (uint8_t)PR_HIRES_bm)
1035 #endif
1036
1037 #if defined(__AVR_HAVE_PRPC_SPI)
1038 #define power_spic_enable()               (PR_PRPC &= (uint8_t)~(PR_SPI_bm))
1039 #define power_spic_disable()             (PR_PRPC |= (uint8_t)PR_SPI_bm)
1040 #endif

```

```
1041
1042 #if defined(__AVR_HAVE_PRPC_TC0)
1043 #define power_tc0c_enable() (PR_PRPC &= (uint8_t)~(PR_TC0_bm))
1044 #define power_tc0c_disable() (PR_PRPC |= (uint8_t)PR_TC0_bm)
1045 #endif
1046
1047 #if defined(__AVR_HAVE_PRPC_TC1)
1048 #define power_tc1c_enable() (PR_PRPC &= (uint8_t)~(PR_TC1_bm))
1049 #define power_tc1c_disable() (PR_PRPC |= (uint8_t)PR_TC1_bm)
1050 #endif
1051
1052 #if defined(__AVR_HAVE_PRPC_TC4)
1053 #define power_tc4c_enable() (PR_PRPC &= (uint8_t)~(PR_TC4_bm))
1054 #define power_tc4c_disable() (PR_PRPC |= (uint8_t)PR_TC4_bm)
1055 #endif
1056
1057 #if defined(__AVR_HAVE_PRPC_TC5)
1058 #define power_tc5c_enable() (PR_PRPC &= (uint8_t)~(PR_TC5_bm))
1059 #define power_tc5c_disable() (PR_PRPC |= (uint8_t)PR_TC5_bm)
1060 #endif
1061
1062 #if defined(__AVR_HAVE_PRPC_TWI)
1063 #define power_twic_enable() (PR_PRPC &= (uint8_t)~(PR_TWI_bm))
1064 #define power_twic_disable() (PR_PRPC |= (uint8_t)PR_TWI_bm)
1065 #endif
1066
1067 #if defined(__AVR_HAVE_PRPC_USART0)
1068 #define power_usartc0_enable() (PR_PRPC &= (uint8_t)~(PR_USART0_bm))
1069 #define power_usartc0_disable() (PR_PRPC |= (uint8_t)PR_USART0_bm)
1070 #endif
1071
1072 #if defined(__AVR_HAVE_PRPC_USART1)
1073 #define power_usartc1_enable() (PR_PRPC &= (uint8_t)~(PR_USART1_bm))
1074 #define power_usartc1_disable() (PR_PRPC |= (uint8_t)PR_USART1_bm)
1075 #endif
1076
1077 #if defined(__AVR_HAVE_PRPD_HIRES)
1078 #define power_hiresd_enable() (PR_PRPD &= (uint8_t)~(PR_HIRES_bm))
1079 #define power_hiresd_disable() (PR_PRPD |= (uint8_t)PR_HIRES_bm)
1080 #endif
1081
1082 #if defined(__AVR_HAVE_PRPD_SPI)
1083 #define power_spid_enable() (PR_PRPD &= (uint8_t)~(PR_SPI_bm))
1084 #define power_spid_disable() (PR_PRPD |= (uint8_t)PR_SPI_bm)
1085 #endif
1086
1087 #if defined(__AVR_HAVE_PRPD_TC0)
1088 #define power_tc0d_enable() (PR_PRPD &= (uint8_t)~(PR_TC0_bm))
1089 #define power_tc0d_disable() (PR_PRPD |= (uint8_t)PR_TC0_bm)
1090 #endif
1091
1092 #if defined(__AVR_HAVE_PRPD_TC1)
1093 #define power_tc1d_enable() (PR_PRPD &= (uint8_t)~(PR_TC1_bm))
1094 #define power_tc1d_disable() (PR_PRPD |= (uint8_t)PR_TC1_bm)
1095 #endif
1096
1097 #if defined(__AVR_HAVE_PRPD_TC5)
1098 #define power_tc5d_enable() (PR_PRPD &= (uint8_t)~(PR_TC5_bm))
1099 #define power_tc5d_disable() (PR_PRPD |= (uint8_t)PR_TC5_bm)
1100 #endif
1101
1102 #if defined(__AVR_HAVE_PRPD_TWI)
1103 #define power_twid_enable() (PR_PRPD &= (uint8_t)~(PR_TWI_bm))
1104 #define power_twid_disable() (PR_PRPD |= (uint8_t)PR_TWI_bm)
1105 #endif
1106
1107 #if defined(__AVR_HAVE_PRPD_USART0)
1108 #define power_usartd0_enable() (PR_PRPD &= (uint8_t)~(PR_USART0_bm))
1109 #define power_usartd0_disable() (PR_PRPD |= (uint8_t)PR_USART0_bm)
1110 #endif
1111
1112 #if defined(__AVR_HAVE_PRPD_USART1)
1113 #define power_usartd1_enable() (PR_PRPD &= (uint8_t)~(PR_USART1_bm))
1114 #define power_usartd1_disable() (PR_PRPD |= (uint8_t)PR_USART1_bm)
1115 #endif
1116
1117 #if defined(__AVR_HAVE_PRPE_HIRES)
1118 #define power_hirese_enable() (PR_PRPE &= (uint8_t)~(PR_HIRES_bm))
1119 #define power_hirese_disable() (PR_PRPE |= (uint8_t)PR_HIRES_bm)
1120 #endif
1121
1122 #if defined(__AVR_HAVE_PRPE_SPI)
1123 #define power_spie_enable() (PR_PRPE &= (uint8_t)~(PR_SPI_bm))
1124 #define power_spie_disable() (PR_PRPE |= (uint8_t)PR_SPI_bm)
1125 #endif
1126
1127 #if defined(__AVR_HAVE_PRPE_TC0)
```

```

1128 #define power_tc0e_enable()      (PR_PRPE &= (uint8_t)~(PR_TC0_bm))
1129 #define power_tc0e_disable()     (PR_PRPE |= (uint8_t)PR_TC0_bm)
1130 #endif
1131
1132 #if defined(__AVR_HAVE_PRPE_TC1)
1133 #define power_tc1e_enable()      (PR_PRPE &= (uint8_t)~(PR_TC1_bm))
1134 #define power_tc1e_disable()     (PR_PRPE |= (uint8_t)PR_TC1_bm)
1135 #endif
1136
1137 #if defined(__AVR_HAVE_PRPE_TWI)
1138 #define power_twie_enable()      (PR_PRPE &= (uint8_t)~(PR_TWI_bm))
1139 #define power_twie_disable()     (PR_PRPE |= (uint8_t)PR_TWI_bm)
1140 #endif
1141
1142 #if defined(__AVR_HAVE_PRPE_USART0)
1143 #define power_usarte0_enable()   (PR_PRPE &= (uint8_t)~(PR_USART0_bm))
1144 #define power_usarte0_disable() (PR_PRPE |= (uint8_t)PR_USART0_bm)
1145 #endif
1146
1147 #if defined(__AVR_HAVE_PRPE_USART1)
1148 #define power_usart1e_enable()   (PR_PRPE &= (uint8_t)~(PR_USART1_bm))
1149 #define power_usart1e_disable() (PR_PRPE |= (uint8_t)PR_USART1_bm)
1150 #endif
1151
1152 #if defined(__AVR_HAVE_PRPF_HIRES)
1153 #define power_hiresf_enable()    (PR_PRPF &= (uint8_t)~(PR_HIRES_bm))
1154 #define power_hiresf_disable()   (PR_PRPF |= (uint8_t)PR_HIRES_bm)
1155 #endif
1156
1157 #if defined(__AVR_HAVE_PRPF_SPI)
1158 #define power_spif_enable()      (PR_PRPF &= (uint8_t)~(PR_SPI_bm))
1159 #define power_spif_disable()     (PR_PRPF |= (uint8_t)PR_SPI_bm)
1160 #endif
1161
1162 #if defined(__AVR_HAVE_PRPF_TC0)
1163 #define power_tc0f_enable()      (PR_PRPF &= (uint8_t)~(PR_TC0_bm))
1164 #define power_tc0f_disable()     (PR_PRPF |= (uint8_t)PR_TC0_bm)
1165 #endif
1166
1167 #if defined(__AVR_HAVE_PRPF_TC1)
1168 #define power_tc1f_enable()      (PR_PRPF &= (uint8_t)~(PR_TC1_bm))
1169 #define power_tc1f_disable()     (PR_PRPF |= (uint8_t)PR_TC1_bm)
1170 #endif
1171
1172 #if defined(__AVR_HAVE_PRPF_TWI)
1173 #define power_twif_enable()      (PR_PRPF &= (uint8_t)~(PR_TWI_bm))
1174 #define power_twif_disable()     (PR_PRPF |= (uint8_t)PR_TWI_bm)
1175 #endif
1176
1177 #if defined(__AVR_HAVE_PRPF_USART0)
1178 #define power_usartf0_enable()   (PR_PRPF &= (uint8_t)~(PR_USART0_bm))
1179 #define power_usartf0_disable() (PR_PRPF |= (uint8_t)PR_USART0_bm)
1180 #endif
1181
1182 #if defined(__AVR_HAVE_PRPF_USART1)
1183 #define power_usartf1_enable()   (PR_PRPF &= (uint8_t)~(PR_USART1_bm))
1184 #define power_usartf1_disable() (PR_PRPF |= (uint8_t)PR_USART1_bm)
1185 #endif
1186
1187 static __inline void
1188 __attribute__((always_inline))
1189 __power_all_enable()
1190 {
1191 #ifdef __AVR_HAVE_PRR
1192     PRR &= (uint8_t)~(__AVR_HAVE_PRR);
1193 #endif
1194
1195 #ifdef __AVR_HAVE_PRR0
1196     PRR0 &= (uint8_t)~(__AVR_HAVE_PRR0);
1197 #endif
1198
1199 #ifdef __AVR_HAVE_PRR1
1200     PRR1 &= (uint8_t)~(__AVR_HAVE_PRR1);
1201 #endif
1202
1203 #ifdef __AVR_HAVE_PRR2
1204     PRR2 &= (uint8_t)~(__AVR_HAVE_PRR2);
1205 #endif
1206
1207 #ifdef __AVR_HAVE_PRGEN
1208     PR_PRGEN &= (uint8_t)~(__AVR_HAVE_PRGEN);
1209 #endif
1210
1211 #ifdef __AVR_HAVE_PRPA
1212     PR_PRPA &= (uint8_t)~(__AVR_HAVE_PRPA);
1213 #endif
1214

```



```

1215 #ifdef __AVR_HAVE_PRPB
1216     PR_PRPB &= (uint8_t)~(__AVR_HAVE_PRPB);
1217 #endif
1218
1219 #ifdef __AVR_HAVE_PRPC
1220     PR_PRPC &= (uint8_t)~(__AVR_HAVE_PRPC);
1221 #endif
1222
1223 #ifdef __AVR_HAVE_PRPD
1224     PR_PRPD &= (uint8_t)~(__AVR_HAVE_PRPD);
1225 #endif
1226
1227 #ifdef __AVR_HAVE_PRPE
1228     PR_PRPE &= (uint8_t)~(__AVR_HAVE_PRPE);
1229 #endif
1230
1231 #ifdef __AVR_HAVE_PRPF
1232     PR_PRPF &= (uint8_t)~(__AVR_HAVE_PRPF);
1233 #endif
1234 }
1235
1236 static __inline void
1237 __attribute__((always_inline))
1238 __power_all_disable()
1239 {
1240     #ifdef __AVR_HAVE_PRR
1241         PRR |= (uint8_t)(__AVR_HAVE_PRR);
1242     #endif
1243
1244     #ifdef __AVR_HAVE_PRR0
1245         PRR0 |= (uint8_t)(__AVR_HAVE_PRR0);
1246     #endif
1247
1248     #ifdef __AVR_HAVE_PRR1
1249         PRR1 |= (uint8_t)(__AVR_HAVE_PRR1);
1250     #endif
1251
1252     #ifdef __AVR_HAVE_PRR2
1253         PRR2 |= (uint8_t)(__AVR_HAVE_PRR2);
1254     #endif
1255
1256     #ifdef __AVR_HAVE_PRGEN
1257         PR_PRGEN |= (uint8_t)(__AVR_HAVE_PRGEN);
1258     #endif
1259
1260     #ifdef __AVR_HAVE_PRPA
1261         PR_PRPA |= (uint8_t)(__AVR_HAVE_PRPA);
1262     #endif
1263
1264     #ifdef __AVR_HAVE_PRPB
1265         PR_PRPB |= (uint8_t)(__AVR_HAVE_PRPB);
1266     #endif
1267
1268     #ifdef __AVR_HAVE_PRPC
1269         PR_PRPC |= (uint8_t)(__AVR_HAVE_PRPC);
1270     #endif
1271
1272     #ifdef __AVR_HAVE_PRPD
1273         PR_PRPD |= (uint8_t)(__AVR_HAVE_PRPD);
1274     #endif
1275
1276     #ifdef __AVR_HAVE_PRPE
1277         PR_PRPE |= (uint8_t)(__AVR_HAVE_PRPE);
1278     #endif
1279
1280     #ifdef __AVR_HAVE_PRPF
1281         PR_PRPF |= (uint8_t)(__AVR_HAVE_PRPF);
1282     #endif
1283 }
1284
1285 #ifndef __DOXYGEN__
1286 #ifndef power_all_enable
1287 #define power_all_enable() __power_all_enable()
1288 #endif
1289
1290 #ifndef power_all_disable
1291 #define power_all_disable() __power_all_disable()
1292 #endif
1293 #endif /* !__DOXYGEN__ */
1294
1295
1296 #if defined(__AVR_AT90CAN32__) \
1297 || defined(__AVR_AT90CAN64__) \
1298 || defined(__AVR_AT90CAN128__) \
1299 || defined(__AVR_AT90PWM1__) \
1300 || defined(__AVR_AT90PWM2__) \
1301 || defined(__AVR_AT90PWM2B__) \

```

```
1302 || defined(__AVR_AT90PWM3__) \
1303 || defined(__AVR_AT90PWM3B__) \
1304 || defined(__AVR_AT90PWM81__) \
1305 || defined(__AVR_AT90PWM161__) \
1306 || defined(__AVR_AT90PWM216__) \
1307 || defined(__AVR_AT90PWM316__) \
1308 || defined(__AVR_AT90SCR100__) \
1309 || defined(__AVR_AT90USB646__) \
1310 || defined(__AVR_AT90USB647__) \
1311 || defined(__AVR_AT90USB82__) \
1312 || defined(__AVR_AT90USB1286__) \
1313 || defined(__AVR_AT90USB1287__) \
1314 || defined(__AVR_AT90USB162__) \
1315 || defined(__AVR_ATA5505__) \
1316 || defined(__AVR_ATA5272__) \
1317 || defined(__AVR_ATmega1280__) \
1318 || defined(__AVR_ATmega1281__) \
1319 || defined(__AVR_ATmega1284__) \
1320 || defined(__AVR_ATmega128RFA1__) \
1321 || defined(__AVR_ATmega1284RFR2__) \
1322 || defined(__AVR_ATmega128RFR2__) \
1323 || defined(__AVR_ATmega1284P__) \
1324 || defined(__AVR_ATmega162__) \
1325 || defined(__AVR_ATmega164A__) \
1326 || defined(__AVR_ATmega164P__) \
1327 || defined(__AVR_ATmega164PA__) \
1328 || defined(__AVR_ATmega165__) \
1329 || defined(__AVR_ATmega165A__) \
1330 || defined(__AVR_ATmega165P__) \
1331 || defined(__AVR_ATmega165PA__) \
1332 || defined(__AVR_ATmega168__) \
1333 || defined(__AVR_ATmega168P__) \
1334 || defined(__AVR_ATmega168PA__) \
1335 || defined(__AVR_ATmega169__) \
1336 || defined(__AVR_ATmega169A__) \
1337 || defined(__AVR_ATmega169P__) \
1338 || defined(__AVR_ATmega169PA__) \
1339 || defined(__AVR_ATmega16U4__) \
1340 || defined(__AVR_ATmega2560__) \
1341 || defined(__AVR_ATmega2561__) \
1342 || defined(__AVR_ATmega2564RFR2__) \
1343 || defined(__AVR_ATmega256RFR2__) \
1344 || defined(__AVR_ATmega324A__) \
1345 || defined(__AVR_ATmega324P__) \
1346 || defined(__AVR_ATmega324PA__) \
1347 || defined(__AVR_ATmega325__) \
1348 || defined(__AVR_ATmega325A__) \
1349 || defined(__AVR_ATmega325PA__) \
1350 || defined(__AVR_ATmega3250__) \
1351 || defined(__AVR_ATmega3250A__) \
1352 || defined(__AVR_ATmega3250PA__) \
1353 || defined(__AVR_ATmega328__) \
1354 || defined(__AVR_ATmega328P__) \
1355 || defined(__AVR_ATmega329__) \
1356 || defined(__AVR_ATmega329A__) \
1357 || defined(__AVR_ATmega329P__) \
1358 || defined(__AVR_ATmega329PA__) \
1359 || defined(__AVR_ATmega3290__) \
1360 || defined(__AVR_ATmega3290A__) \
1361 || defined(__AVR_ATmega3290PA__) \
1362 || defined(__AVR_ATmega32C1__) \
1363 || defined(__AVR_ATmega32M1__) \
1364 || defined(__AVR_ATmega32U2__) \
1365 || defined(__AVR_ATmega32U4__) \
1366 || defined(__AVR_ATmega32U6__) \
1367 || defined(__AVR_ATmega48__) \
1368 || defined(__AVR_ATmega48A__) \
1369 || defined(__AVR_ATmega48PA__) \
1370 || defined(__AVR_ATmega48P__) \
1371 || defined(__AVR_ATmega640__) \
1372 || defined(__AVR_ATmega640P__) \
1373 || defined(__AVR_ATmega644__) \
1374 || defined(__AVR_ATmega644A__) \
1375 || defined(__AVR_ATmega644P__) \
1376 || defined(__AVR_ATmega644PA__) \
1377 || defined(__AVR_ATmega645__) \
1378 || defined(__AVR_ATmega645A__) \
1379 || defined(__AVR_ATmega645P__) \
1380 || defined(__AVR_ATmega6450__) \
1381 || defined(__AVR_ATmega6450A__) \
1382 || defined(__AVR_ATmega6450P__) \
1383 || defined(__AVR_ATmega649__) \
1384 || defined(__AVR_ATmega649A__) \
1385 || defined(__AVR_ATmega6490__) \
1386 || defined(__AVR_ATmega6490A__) \
1387 || defined(__AVR_ATmega6490P__) \
1388 || defined(__AVR_ATmega644RFR2__) \
```

```

1389 || defined(__AVR_ATmega64RFR2__) \
1390 || defined(__AVR_ATmega88__) \
1391 || defined(__AVR_ATmega88P__) \
1392 || defined(__AVR_ATmega8U2__) \
1393 || defined(__AVR_ATmega16U2__) \
1394 || defined(__AVR_ATmega32U2__) \
1395 || defined(__AVR_ATtiny48__) \
1396 || defined(__AVR_ATtiny167__) \
1397 || defined(__DOXYGEN__)
1398
1399
1400 /** \addtogroup avr_power
1401
1402 Some of the newer AVR's contain a System Clock Prescale Register (CLKPR) that
1403 allows you to decrease the system clock frequency and the power consumption
1404 when the need for processing power is low.
1405 On some earlier AVR's (ATmega103, ATmega64, ATmega128), similar
1406 functionality can be achieved through the XTAL Divide Control Register.
1407 Below are two macros and an enumerated type that can be used to
1408 interface to the Clock Prescale Register or
1409 XTAL Divide Control Register.
1410
1411 \note Not all AVR devices have a clock prescaler. On those devices
1412 without a Clock Prescale Register or XTAL Divide Control Register, these
1413 macros are not available.
1414 */
1415
1416
1417 /** \addtogroup avr_power
1418 \code
1419 typedef enum
1420 {
1421     clock_div_1 = 0,
1422     clock_div_2 = 1,
1423     clock_div_4 = 2,
1424     clock_div_8 = 3,
1425     clock_div_16 = 4,
1426     clock_div_32 = 5,
1427     clock_div_64 = 6,
1428     clock_div_128 = 7,
1429     clock_div_256 = 8,
1430     clock_div_1_rc = 15, // ATmega128RFA1 only
1431 } clock_div_t;
1432 \endcode
1433 Clock prescaler setting enumerations for device using
1434 System Clock Prescale Register.
1435
1436 \code
1437 typedef enum
1438 {
1439     clock_div_1 = 1,
1440     clock_div_2 = 2,
1441     clock_div_4 = 4,
1442     clock_div_8 = 8,
1443     clock_div_16 = 16,
1444     clock_div_32 = 32,
1445     clock_div_64 = 64,
1446     clock_div_128 = 128
1447 } clock_div_t;
1448 \endcode
1449 Clock prescaler setting enumerations for device using
1450 XTAL Divide Control Register.
1451
1452 */
1453 #ifndef __DOXYGEN__
1454 typedef enum
1455 {
1456     clock_div_1 = 0,
1457     clock_div_2 = 1,
1458     clock_div_4 = 2,
1459     clock_div_8 = 3,
1460     clock_div_16 = 4,
1461     clock_div_32 = 5,
1462     clock_div_64 = 6,
1463     clock_div_128 = 7,
1464     clock_div_256 = 8
1465 #if defined(__AVR_ATmega128RFA1__) \
1466 || defined(__AVR_ATmega2564RFR2__) \
1467 || defined(__AVR_ATmega1284RFR2__) \
1468 || defined(__AVR_ATmega644RFR2__) \
1469 || defined(__AVR_ATmega256RFR2__) \
1470 || defined(__AVR_ATmega128RFR2__) \
1471 || defined(__AVR_ATmega64RFR2__)
1472     , clock_div_1_rc = 15
1473 #endif
1474 } clock_div_t;
1475

```

```

1476 static __inline__ void clock_prescale_set(clock_div_t) __attribute__((__always_inline__));
1477 #endif /* !__DOXYGEN__ */
1478
1479 /**
1480 \ingroup avr_power
1481 \fn clock_prescale_set(clock_div_t x)
1482
1483 Set the clock prescaler register select bits, selecting a system clock
1484 division setting. This function is inlined, even if compiler
1485 optimizations are disabled.
1486
1487 The type of \c x is \c clock_div_t.
1488
1489 \note For device with XTAL Divide Control Register (XDIV), \c x can actually range
1490 from 1 to 129. Thus, one does not need to use \c clock_div_t type as argument.
1491 */
1492 void clock_prescale_set(clock_div_t __x)
1493 {
1494     uint8_t __tmp = _BV(CLKPCE);
1495     __asm__ __volatile__ (
1496         "in __tmp_reg__, __SREG__" "\n\t"
1497         "cli" "\n\t"
1498         "sts %1, %0" "\n\t"
1499         "sts %1, %2" "\n\t"
1500         "out __SREG__, __tmp_reg__"
1501         : /* no outputs */
1502         : "d" (__tmp),
1503         "M" (_SFR_MEM_ADDR(CLKPR)),
1504         "d" (__x)
1505         : "r0");
1506 }
1507
1508 /** \addtogroup avr_power
1509 \def clock_prescale_get()
1510 Gets and returns the clock prescaler register setting. The return type is \c clock_div_t.
1511
1512 \note For device with XTAL Divide Control Register (XDIV), return can actually
1513 range from 1 to 129. Care should be taken has the return value could differ from the
1514 typedef enum clock_div_t. This should only happen if clock_prescale_set was previously
1515 called with a value other than those defined by \c clock_div_t.
1516 */
1517 #define clock_prescale_get() (clock_div_t)(CLKPR &
    (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))
1518
1519 #elif defined(__AVR_ATmega16HVB__) \
1520 || defined(__AVR_ATmega16HVBREVB__) \
1521 || defined(__AVR_ATmega32HVB__) \
1522 || defined(__AVR_ATmega32HVBREVB__)
1523
1524 typedef enum
1525 {
1526     clock_div_1 = 0,
1527     clock_div_2 = 1,
1528     clock_div_4 = 2,
1529     clock_div_8 = 3
1530 } clock_div_t;
1531
1532 static __inline__ void clock_prescale_set(clock_div_t) __attribute__((__always_inline__));
1533
1534 void clock_prescale_set(clock_div_t __x)
1535 {
1536     uint8_t __tmp = _BV(CLKPCE);
1537     __asm__ __volatile__ (
1538         "in __tmp_reg__, __SREG__" "\n\t"
1539         "cli" "\n\t"
1540         "sts %1, %0" "\n\t"
1541         "sts %1, %2" "\n\t"
1542         "out __SREG__, __tmp_reg__"
1543         : /* no outputs */
1544         : "d" (__tmp),
1545         "M" (_SFR_MEM_ADDR(CLKPR)),
1546         "d" (__x)
1547         : "r0");
1548 }
1549
1550 #define clock_prescale_get() (clock_div_t)(CLKPR & (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)))
1551
1552 #elif defined(__AVR_AT5790__) \
1553 || defined(__AVR_AT5795__)
1554
1555 typedef enum
1556 {
1557     clock_div_1 = 0,
1558     clock_div_2 = 1,
1559     clock_div_4 = 2,
1560     clock_div_8 = 3,
1561     clock_div_16 = 4,

```

```

1562     clock_div_32 = 5,
1563     clock_div_64 = 6,
1564     clock_div_128 = 7,
1565 } clock_div_t;
1566
1567 static __inline__ void system_clock_prescale_set(clock_div_t __attribute__((__always_inline__)));
1568
1569 void system_clock_prescale_set(clock_div_t __x)
1570 {
1571     uint8_t __tmp = _BV(CLKPCE);
1572     __asm__ __volatile__ (
1573         "in __tmp_reg__, __SREG__" "\n\t"
1574         "cli" "\n\t"
1575         "out %1, %0" "\n\t"
1576         "out %1, %2" "\n\t"
1577         "out __SREG__, __tmp_reg__"
1578         : /* no outputs */
1579         : "d" (__tmp),
1580           "I" (_SFR_IO_ADDR(CLKPR)),
1581           "d" (__x)
1582         : "r0");
1583 }
1584
1585 #define system_clock_prescale_get() (clock_div_t)(CLKPR & (uint8_t)((1<<CLKPS0) | (1<<CLKPS1) | (1<<CLKPS2)))
1586
1587 typedef enum
1588 {
1589     timer_clock_div_reset = 0,
1590     timer_clock_div_1 = 1,
1591     timer_clock_div_2 = 2,
1592     timer_clock_div_4 = 3,
1593     timer_clock_div_8 = 4,
1594     timer_clock_div_16 = 5,
1595     timer_clock_div_32 = 6,
1596     timer_clock_div_64 = 7
1597 } timer_clock_div_t;
1598
1599 static __inline__ void timer_clock_prescale_set(timer_clock_div_t __attribute__((__always_inline__)));
1600
1601 void timer_clock_prescale_set(timer_clock_div_t __x)
1602 {
1603     uint8_t __t;
1604     __asm__ __volatile__ (
1605         "in __tmp_reg__, __SREG__" "\n\t"
1606         "cli" "\n\t"
1607         "in %[temp], %[clkpr]" "\n\t"
1608         "out %[clkpr], %[enable]" "\n\t"
1609         "andi %[temp], %[not_CLTPS]" "\n\t"
1610         "or %[temp], %[set_value]" "\n\t"
1611         "out %[clkpr], %[temp]" "\n\t"
1612         "sei" "\n\t"
1613         "out __SREG__, __tmp_reg__" "\n\t"
1614         : /* no outputs */
1615         : [temp] "r" (__t),
1616           [clkpr] "I" (_SFR_IO_ADDR(CLKPR)),
1617           [enable] "r" (_BV(CLKPCE)),
1618           [not_CLTPS] "M" (0xFF & ~(1 << CLTPS2) | (1 << CLTPS1) | (1 << CLTPS0))),
1619           [set_value] "r" ((__x & 7) << 3)
1620         : "r0");
1621 }
1622
1623 #define timer_clock_prescale_get() (timer_clock_div_t)(CLKPR &
1624     (uint8_t)((1<<CLTPS0) | (1<<CLTPS1) | (1<<CLTPS2)))
1625
1626 #elif defined(__AVR_ATmega6285__) \
1627 || defined(__AVR_ATmega6286__)
1628
1629 typedef enum
1630 {
1631     clock_div_1 = 0,
1632     clock_div_2 = 1,
1633     clock_div_4 = 2,
1634     clock_div_8 = 3,
1635     clock_div_16 = 4,
1636     clock_div_32 = 5,
1637     clock_div_64 = 6,
1638     clock_div_128 = 7
1639 } clock_div_t;
1640
1641 static __inline__ void system_clock_prescale_set(clock_div_t __attribute__((__always_inline__)));
1642
1643 void system_clock_prescale_set(clock_div_t __x)
1644 {
1645     uint8_t __t;
1646     __asm__ __volatile__ (
1647         "in __tmp_reg__, __SREG__" "\n\t"
1648         "cli" "\n\t"

```

```

1648     "in %[temp], %[clpr]" "\n\t"
1649     "out %[clpr], %[enable]" "\n\t"
1650     "andi %[temp], %[not_CLKPS]" "\n\t"
1651     "or %[temp], %[set_value]" "\n\t"
1652     "out %[clpr], %[temp]" "\n\t"
1653     "sei" "\n\t"
1654     "out __SREG__, __tmp_reg__" "\n\t"
1655     : /* no outputs */
1656     : [temp] "r" (__t),
1657       [clpr] "I" (_SFR_IO_ADDR(CLKPR)),
1658       [enable] "r" _BV(CLPCE),
1659       [not_CLKPS] "M" (0xFF & (~ ((1 << CLKPS2) | (1 << CLKPS1) | (1 << CLKPS0)))),
1660       [set_value] "r" (__x & 7)
1661     : "r0");
1662 }
1663
1664 #define system_clock_prescale_get() (clock_div_t)(CLKPR & (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)))
1665
1666 typedef enum
1667 {
1668     timer_clock_div_reset = 0,
1669     timer_clock_div_1 = 1,
1670     timer_clock_div_2 = 2,
1671     timer_clock_div_4 = 3,
1672     timer_clock_div_8 = 4,
1673     timer_clock_div_16 = 5,
1674     timer_clock_div_32 = 6,
1675     timer_clock_div_64 = 7
1676 } timer_clock_div_t;
1677
1678 static __inline__ void timer_clock_prescale_set(timer_clock_div_t __attribute__((__always_inline__));
1679 void timer_clock_prescale_set(timer_clock_div_t __x)
1680 {
1681     uint8_t __t;
1682     __asm__ __volatile__ (
1683         "in __tmp_reg__, __SREG__" "\n\t"
1684         "cli" "\n\t"
1685         "in %[temp], %[clpr]" "\n\t"
1686         "out %[clpr], %[enable]" "\n\t"
1687         "andi %[temp], %[not_CLTPS]" "\n\t"
1688         "or %[temp], %[set_value]" "\n\t"
1689         "out %[clpr], %[temp]" "\n\t"
1690         "sei" "\n\t"
1691         "out __SREG__, __tmp_reg__" "\n\t"
1692         : /* no outputs */
1693         : [temp] "r" (__t),
1694           [clpr] "I" (_SFR_IO_ADDR(CLKPR)),
1695           [enable] "r" _BV(CLPCE),
1696           [not_CLTPS] "M" (0xFF & (~ ((1 << CLTPS2) | (1 << CLTPS1) | (1 << CLTPS0)))),
1697           [set_value] "r" ((__x & 7) << 3)
1698         : "r0");
1699     }
1700 }
1701
1702 #define timer_clock_prescale_get() (timer_clock_div_t)(CLKPR &
(uint8_t)((1<<CLTPS0)|(1<<CLTPS1)|(1<<CLTPS2)))
1703
1704 #elif defined(__AVR_ATtiny24__) \
1705 || defined(__AVR_ATtiny24A__) \
1706 || defined(__AVR_ATtiny44__) \
1707 || defined(__AVR_ATtiny44A__) \
1708 || defined(__AVR_ATtiny84__) \
1709 || defined(__AVR_ATtiny84A__) \
1710 || defined(__AVR_ATtiny25__) \
1711 || defined(__AVR_ATtiny45__) \
1712 || defined(__AVR_ATtiny85__) \
1713 || defined(__AVR_ATtiny261A__) \
1714 || defined(__AVR_ATtiny261__) \
1715 || defined(__AVR_ATtiny461__) \
1716 || defined(__AVR_ATtiny461A__) \
1717 || defined(__AVR_ATtiny861__) \
1718 || defined(__AVR_ATtiny861A__) \
1719 || defined(__AVR_ATtiny2313__) \
1720 || defined(__AVR_ATtiny2313A__) \
1721 || defined(__AVR_ATtiny4313__) \
1722 || defined(__AVR_ATtiny13__) \
1723 || defined(__AVR_ATtiny13A__) \
1724 || defined(__AVR_ATtiny43U__) \
1725
1726 typedef enum
1727 {
1728     clock_div_1 = 0,
1729     clock_div_2 = 1,
1730     clock_div_4 = 2,
1731     clock_div_8 = 3,
1732     clock_div_16 = 4,
1733     clock_div_32 = 5,

```

```

1734     clock_div_64 = 6,
1735     clock_div_128 = 7,
1736     clock_div_256 = 8
1737 } clock_div_t;
1738
1739 static __inline__ void clock_prescale_set(clock_div_t) __attribute__((__always_inline__));
1740
1741 void clock_prescale_set(clock_div_t __x)
1742 {
1743     uint8_t __tmp = _BV(CLKPCE);
1744     __asm__ __volatile__ (
1745         "in __tmp_reg__, __SREG__" "\n\t"
1746         "cli" "\n\t"
1747         "out %1, %0" "\n\t"
1748         "out %1, %2" "\n\t"
1749         "out __SREG__, __tmp_reg__"
1750         : /* no outputs */
1751         : "d" (__tmp),
1752         "I" (_SFR_IO_ADDR(CLKPR)),
1753         "d" (__x)
1754         : "r0");
1755 }
1756
1757
1758 #define clock_prescale_get() (clock_div_t)(CLKPR &
1759 (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))
1760
1761 #elif defined(__AVR_ATmega64__) \
1762 || defined(__AVR_ATmega103__) \
1763 || defined(__AVR_ATmega128__)
1764 //Enum is declared for code compatibility
1765 typedef enum
1766 {
1767     clock_div_1 = 1,
1768     clock_div_2 = 2,
1769     clock_div_4 = 4,
1770     clock_div_8 = 8,
1771     clock_div_16 = 16,
1772     clock_div_32 = 32,
1773     clock_div_64 = 64,
1774     clock_div_128 = 128
1775 } clock_div_t;
1776
1777 static __inline__ void clock_prescale_set(clock_div_t) __attribute__((__always_inline__));
1778
1779 void clock_prescale_set(clock_div_t __x)
1780 {
1781     if((__x <= 0) || (__x > 129))
1782     {
1783         return; //Invalid value.
1784     }
1785     else
1786     {
1787         uint8_t __tmp = 0;
1788         //Algo explained:
1789         //1 - Clear XDIV in order for it to accept a new value (actually only
1790         //    XDIVEN need to be cleared, but clearing XDIV is faster than
1791         //    read-modify-write since we will rewrite XDIV later anyway)
1792         //2 - wait 8 clock cycle for stability, see datasheet errata
1793         //3 - Exist if requested prescaler is 1
1794         //4 - Calculate XDIV6..0 value = 129 - __x
1795         //5 - Set XDIVEN bit in calculated value
1796         //6 - write XDIV with calculated value
1797         //7 - wait 8 clock cycle for stability, see datasheet errata
1798         __asm__ __volatile__ (
1799             "in __tmp_reg__, __SREG__" "\n\t"
1800             "cli" "\n\t"
1801             "out %1, __zero_reg__" "\n\t"
1802             "nop" "\n\t"
1803             "nop" "\n\t"
1804             "nop" "\n\t"
1805             "nop" "\n\t"
1806             "nop" "\n\t"
1807             "nop" "\n\t"
1808             "nop" "\n\t"
1809             "nop" "\n\t"
1810             "cpi %0, 0x01" "\n\t"
1811             "breq L_%" "\n\t"
1812             "ldi %2, 0x81" "\n\t" //129
1813             "sub %2, %0" "\n\t"
1814             "ori %2, 0x80" "\n\t" //128
1815             "out %1, %2" "\n\t"
1816             "nop" "\n\t"
1817             "nop" "\n\t"
1818             "nop" "\n\t"
1819             "nop" "\n\t"

```

```

1820         "nop" "\n\t"
1821         "nop" "\n\t"
1822         "nop" "\n\t"
1823         "nop" "\n\t"
1824         "L_%=: " "out __SREG__, __tmp_reg__"
1825         : /* no outputs */
1826         : "d" (__x),
1827         "I" (__SFR_IO_ADDR(XDIV)),
1828         "d" (__tmp)
1829         : "r0");
1830     }
1831 }
1832
1833 static __inline__ clock_div_t clock_prescale_get(void) __attribute__((__always_inline__));
1834
1835 clock_div_t clock_prescale_get(void)
1836 {
1837     if(bit_is_clear(XDIV, XDIVEN))
1838     {
1839         return 1;
1840     }
1841     else
1842     {
1843         return (clock_div_t)(129 - (XDIV & 0x7F));
1844     }
1845 }
1846
1847 #elif defined(__AVR_ATtiny4__) \
1848 || defined(__AVR_ATtiny5__) \
1849 || defined(__AVR_ATtiny9__) \
1850 || defined(__AVR_ATtiny10__) \
1851 || defined(__AVR_ATtiny20__) \
1852 || defined(__AVR_ATtiny40__) \
1853
1854 typedef enum
1855 {
1856     clock_div_1 = 0,
1857     clock_div_2 = 1,
1858     clock_div_4 = 2,
1859     clock_div_8 = 3,
1860     clock_div_16 = 4,
1861     clock_div_32 = 5,
1862     clock_div_64 = 6,
1863     clock_div_128 = 7,
1864     clock_div_256 = 8
1865 } clock_div_t;
1866
1867 static __inline__ void clock_prescale_set(clock_div_t) __attribute__((__always_inline__));
1868
1869 void clock_prescale_set(clock_div_t __x)
1870 {
1871     uint8_t __tmp = 0xD8;
1872     __asm__ __volatile__ (
1873         "in __tmp_reg__, __SREG__" "\n\t"
1874         "cli" "\n\t"
1875         "out %1, %0" "\n\t"
1876         "out %2, %3" "\n\t"
1877         "out __SREG__, __tmp_reg__"
1878         : /* no outputs */
1879         : "d" (__tmp),
1880         "I" (__SFR_IO_ADDR(CCP)),
1881         "I" (__SFR_IO_ADDR(CLKPSR)),
1882         "d" (__x)
1883         : "r16");
1884 }
1885
1886 #define clock_prescale_get() (clock_div_t)(CLKPSR &
1887     (uint8_t)((1<<CLKPS0)|(1<<CLKPS1)|(1<<CLKPS2)|(1<<CLKPS3)))
1888 #endif
1889
1890 #endif /* _AVR_POWER_H_ */

```

25.28 sfr_defs.h

```

1 /* Copyright (c) 2002, Marek Michalkiewicz <marekm@amelek.gda.pl>
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9

```



```

10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* avr/sfr_defs.h - macros for accessing AVR special function registers */
32
33 /* $Id: sfr_defs.h 1691 2008-04-28 22:05:42Z arcanum $ */
34
35 #ifndef _AVR_SFR_DEFS_H_
36 #define _AVR_SFR_DEFS_H_ 1
37
38 /** \defgroup avr_sfr_notes Additional notes from <avr/sfr_defs.h>
39 \ingroup avr_sfr
40
41 The \c <avr/sfr_defs.h> file is included by all of the \c <avr/ioXXXX.h>
42 files, which use macros defined here to make the special function register
43 definitions look like C variables or simple constants, depending on the
44 <tt>_SFR_ASM_COMPAT</tt> define. Some examples from \c <avr/iocanxx.h> to
45 show how to define such macros:
46
47 \code
48 #define PORTA _SFR_IO8(0x02)
49 #define EEAR _SFR_IO16(0x21)
50 #define UDR0 _SFR_MEM8(0xC6)
51 #define TCNT3 _SFR_MEM16(0x94)
52 #define CANIDT _SFR_MEM32(0xF0)
53 \endcode
54
55 If \c _SFR_ASM_COMPAT is not defined, C programs can use names like
56 <tt>PORTA</tt> directly in C expressions (also on the left side of
57 assignment operators) and GCC will do the right thing (use short I/O
58 instructions if possible). The \c __SFR_OFFSET definition is not used in
59 any way in this case.
60
61 Define \c _SFR_ASM_COMPAT as 1 to make these names work as simple constants
62 (addresses of the I/O registers). This is necessary when included in
63 preprocessed assembler (*.S) source files, so it is done automatically if
64 \c __ASSEMBLER__ is defined. By default, all addresses are defined as if
65 they were memory addresses (used in \c lds/sts instructions). To use these
66 addresses in \c in/out instructions, you must subtract 0x20 from them.
67
68 For more backwards compatibility, insert the following at the start of your
69 old assembler source file:
70
71 \code
72 #define __SFR_OFFSET 0
73 \endcode
74
75 This automatically subtracts 0x20 from I/O space addresses, but it's a
76 hack, so it is recommended to change your source: wrap such addresses in
77 macros defined here, as shown below. After this is done, the
78 <tt>__SFR_OFFSET</tt> definition is no longer necessary and can be removed.
79
80 Real example - this code could be used in a boot loader that is portable
81 between devices with \c SPMCR at different addresses.
82
83 \verbatim
84 <avr/iom163.h>: #define SPMCR _SFR_IO8(0x37)
85 <avr/iom128.h>: #define SPMCR _SFR_MEM8(0x68)
86 \endverbatim
87
88 \code
89 #if _SFR_IO_REG_P(SPMCR)
90 out _SFR_IO_ADDR(SPMCR), r24
91 #else
92 sts _SFR_MEM_ADDR(SPMCR), r24
93 #endif
94 \endcode
95
96 You can use the \c in/out/cbi/sbi/sbic/sbis instructions, without the

```

```

97 <tt>_SFR_IO_REG_P</tt> test, if you know that the register is in the I/O
98 space (as with \c SREG, for example). If it isn't, the assembler will
99 complain (I/O address out of range 0...0x3f), so this should be fairly
100 safe.
101
102 If you do not define \c __SFR_OFFSET (so it will be 0x20 by default), all
103 special register addresses are defined as memory addresses (so \c SREG is
104 0x5f), and (if code size and speed are not important, and you don't like
105 the ugly \#if above) you can always use lds/sts to access them. But, this
106 will not work if <tt>__SFR_OFFSET</tt> != 0x20, so use a different macro
107 (defined only if <tt>__SFR_OFFSET</tt> == 0x20) for safety:
108
109 \code
110 sts _SFR_ADDR(SPMCR), r24
111 \endcode
112
113 In C programs, all 3 combinations of \c _SFR_ASM_COMPAT and
114 <tt>__SFR_OFFSET</tt> are supported - the \c _SFR_ADDR(SPMCR) macro can be
115 used to get the address of the \c SPMCR register (0x57 or 0x68 depending on
116 device). */
117
118 #ifdef __ASSEMBLER__
119 #define _SFR_ASM_COMPAT 1
120 #elif !defined(_SFR_ASM_COMPAT)
121 #define _SFR_ASM_COMPAT 0
122 #endif
123
124 #ifndef __ASSEMBLER__
125 /* These only work in C programs. */
126 #include <inttypes.h>
127
128 #define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *) (mem_addr))
129 #define _MMIO_WORD(mem_addr) (*(volatile uint16_t *) (mem_addr))
130 #define _MMIO_DWORD(mem_addr) (*(volatile uint32_t *) (mem_addr))
131 #endif
132
133 #if _SFR_ASM_COMPAT
134
135 #ifndef __SFR_OFFSET
136 /* Define as 0 before including this file for compatibility with old asm
137 sources that don't subtract __SFR_OFFSET from symbolic I/O addresses. */
138 # if __AVR_ARCH__ >= 100
139 #   define __SFR_OFFSET 0x00
140 # else
141 #   define __SFR_OFFSET 0x20
142 # endif
143 #endif
144
145 #if (__SFR_OFFSET != 0) && (__SFR_OFFSET != 0x20)
146 #error "__SFR_OFFSET must be 0 or 0x20"
147 #endif
148
149 #define _SFR_MEM8(mem_addr) (mem_addr)
150 #define _SFR_MEM16(mem_addr) (mem_addr)
151 #define _SFR_MEM32(mem_addr) (mem_addr)
152 #define _SFR_IO8(io_addr) ((io_addr) + __SFR_OFFSET)
153 #define _SFR_IO16(io_addr) ((io_addr) + __SFR_OFFSET)
154
155 #define _SFR_IO_ADDR(sfr) ((sfr) - __SFR_OFFSET)
156 #define _SFR_MEM_ADDR(sfr) (sfr)
157 #define _SFR_IO_REG_P(sfr) ((sfr) < 0x40 + __SFR_OFFSET)
158
159 #if (__SFR_OFFSET == 0x20)
160 /* No need to use ?: operator, so works in assembler too. */
161 #define _SFR_ADDR(sfr) _SFR_MEM_ADDR(sfr)
162 #elif !defined(__ASSEMBLER__)
163 #define _SFR_ADDR(sfr) (_SFR_IO_REG_P(sfr) ? (_SFR_IO_ADDR(sfr) + 0x20) : _SFR_MEM_ADDR(sfr))
164 #endif
165
166 #else /* !_SFR_ASM_COMPAT */
167
168 #ifndef __SFR_OFFSET
169 # if __AVR_ARCH__ >= 100
170 #   define __SFR_OFFSET 0x00
171 # else
172 #   define __SFR_OFFSET 0x20
173 # endif
174 #endif
175
176 #define _SFR_MEM8(mem_addr) _MMIO_BYTE(mem_addr)
177 #define _SFR_MEM16(mem_addr) _MMIO_WORD(mem_addr)
178 #define _SFR_MEM32(mem_addr) _MMIO_DWORD(mem_addr)
179 #define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + __SFR_OFFSET)
180 #define _SFR_IO16(io_addr) _MMIO_WORD((io_addr) + __SFR_OFFSET)
181
182 #define _SFR_MEM_ADDR(sfr) ((uint16_t) &(sfr))
183 #define _SFR_IO_ADDR(sfr) (_SFR_MEM_ADDR(sfr) - __SFR_OFFSET)

```

```

184 #define _SFR_IO_REG_P(sfr) (_SFR_MEM_ADDR(sfr) < 0x40 + __SFR_OFFSET)
185
186 #define _SFR_ADDR(sfr) _SFR_MEM_ADDR(sfr)
187
188 #endif /* !_SFR_ASM_COMPAT */
189
190 #define _SFR_BYTE(sfr) _MMIO_BYTE(_SFR_ADDR(sfr))
191 #define _SFR_WORD(sfr) _MMIO_WORD(_SFR_ADDR(sfr))
192 #define _SFR_DWORD(sfr) _MMIO_DWORD(_SFR_ADDR(sfr))
193
194 /** \name Bit manipulation */
195
196 /*@{*/
197 /** \def _BV
198 \ingroup avr_sfr
199
200 \code #include <avr/io.h>\endcode
201
202 Converts a bit number into a byte value.
203
204 \note The bit shift is performed by the compiler which then inserts the
205 result into the code. Thus, there is no run-time overhead when using
206 _BV(). */
207
208 #define _BV(bit) (1 << (bit))
209
210 /*@}*/
211
212 #ifndef _VECTOR
213 #define _VECTOR(N) __vector_ ## N
214 #endif
215
216 #ifndef __ASSEMBLER__
217
218 /** \name IO register bit manipulation */
219
220 /*@{*/
221
222 /** \def bit_is_set
223 \ingroup avr_sfr
224
225 \code #include <avr/io.h>\endcode
226
227 Test whether bit \c bit in IO register \c sfr is set.
228 This will return a 0 if the bit is clear, and non-zero
229 if the bit is set. */
230
231 #define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
232
233 /** \def bit_is_clear
234 \ingroup avr_sfr
235
236 \code #include <avr/io.h>\endcode
237
238 Test whether bit \c bit in IO register \c sfr is clear.
239 This will return non-zero if the bit is clear, and a 0
240 if the bit is set. */
241
242 #define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))
243
244 /** \def loop_until_bit_is_set
245 \ingroup avr_sfr
246
247 \code #include <avr/io.h>\endcode
248
249 Wait until bit \c bit in IO register \c sfr is set. */
250
251 #define loop_until_bit_is_set(sfr, bit) do { } while (bit_is_clear(sfr, bit))
252
253 /** \def loop_until_bit_is_clear
254 \ingroup avr_sfr
255
256 \code #include <avr/io.h>\endcode
257
258 Wait until bit \c bit in IO register \c sfr is clear. */
259
260 #define loop_until_bit_is_clear(sfr, bit) do { } while (bit_is_set(sfr, bit))
261
262 /*@}*/
263
264 #endif /* !__ASSEMBLER__ */
265
266 #endif /* _SFR_DEFS_H */

```

25.29 signal.h

```

1 /* Copyright (c) 2002,2005,2006 Marek Michalkiewicz
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: signal.h 1059 2006-02-02 19:42:12Z aesok $ */
32
33 #ifndef _AVR_SIGNAL_H_
34 #define _AVR_SIGNAL_H_
35
36 #warning "This header file is obsolete. Use <avr/interrupt.h>."
37 #include <avr/interrupt.h>
38
39 #endif /* _AVR_SIGNAL_H_ */

```

25.30 signature.h File Reference

25.31 signature.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2009, Atmel Corporation
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: signature.h 2296 2012-06-19 06:52:59Z joerg_wunsch $ */
32
33 /* avr/signature.h - Signature API */

```

```

34
35 #ifndef _AVR_SIGNATURE_H_
36 #define _AVR_SIGNATURE_H_ 1
37
38 /** \file */
39 /** \defgroup avr_signature <avr/signature.h>: Signature Support
40
41 \par Introduction
42
43 The <avr/signature.h> header file allows the user to automatically
44 and easily include the device's signature data in a special section of
45 the final linked ELF file.
46
47 This value can then be used by programming software to compare the on-device
48 signature with the signature recorded in the ELF file to look for a match
49 before programming the device.
50
51 \par API Usage Example
52
53 Usage is very simple; just include the header file:
54
55 \code
56 #include <avr/signature.h>
57 \endcode
58
59 This will declare a constant unsigned char array and it is initialized with
60 the three signature bytes, MSB first, that are defined in the device I/O
61 header file. This array is then placed in the .signature section in the
62 resulting linked ELF file.
63
64 The three signature bytes that are used to initialize the array are
65 these defined macros in the device I/O header file, from MSB to LSB:
66 SIGNATURE_2, SIGNATURE_1, SIGNATURE_0.
67
68 This header file should only be included once in an application.
69 */
70
71 #ifndef __ASSEMBLER__
72
73 #include <avr/io.h>
74
75 #if defined(SIGNATURE_0) && defined(SIGNATURE_1) && defined(SIGNATURE_2)
76
77 const unsigned char __signature[3]
78 __attribute__((__used__, __section__(".signature"))) =
79 { SIGNATURE_2, SIGNATURE_1, SIGNATURE_0 };
80
81 #endif /* defined(SIGNATURE_0) && defined(SIGNATURE_1) && defined(SIGNATURE_2) */
82
83 #endif /* __ASSEMBLER__ */
84
85 #endif /* _AVR_SIGNATURE_H_ */

```

25.32 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)

25.33 sleep.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, 2004 Theodore A. Roth
2 Copyright (c) 2004, 2007, 2008 Eric B. Weddington
3 Copyright (c) 2005, 2006, 2007 Joerg Wunsch
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8

```

```

9 * Redistributions of source code must retain the above copyright
10 notice, this list of conditions and the following disclaimer.
11
12 * Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in
14 the documentation and/or other materials provided with the
15 distribution.
16
17 * Neither the name of the copyright holders nor the names of
18 contributors may be used to endorse or promote products derived
19 from this software without specific prior written permission.
20
21 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
22 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
23 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
24 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
25 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
26 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
27 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
28 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
29 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
30 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
31 POSSIBILITY OF SUCH DAMAGE. */
32
33 /* $Id: sleep.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
34
35 #ifndef _AVR_SLEEP_H_
36 #define _AVR_SLEEP_H_ 1
37
38 #include <avr/io.h>
39 #include <stdint.h>
40
41
42 /** \file */
43
44 /** \defgroup avr_sleep <avr/sleep.h>: Power Management and Sleep Modes
45
46 \code #include <avr/sleep.h>\endcode
47
48 Use of the \c SLEEP instruction can allow an application to reduce its
49 power consumption considerably. AVR devices can be put into different
50 sleep modes. Refer to the datasheet for the details relating to the device
51 you are using.
52
53 There are several macros provided in this header file to actually
54 put the device into sleep mode. The simplest way is to optionally
55 set the desired sleep mode using \c set_sleep_mode() (it usually
56 defaults to idle mode where the CPU is put on sleep but all
57 peripheral clocks are still running), and then call
58 \c sleep_mode(). This macro automatically sets the sleep enable bit, goes
59 to sleep, and clears the sleep enable bit.
60
61 Example:
62 \code
63 #include <avr/sleep.h>
64
65 ...
66 set_sleep_mode(<mode>);
67 sleep_mode();
68 \endcode
69
70 Note that unless your purpose is to completely lock the CPU (until a
71 hardware reset), interrupts need to be enabled before going to sleep.
72
73 As the \c sleep_mode() macro might cause race conditions in some
74 situations, the individual steps of manipulating the sleep enable
75 (SE) bit, and actually issuing the \c SLEEP instruction, are provided
76 in the macros \c sleep_enable(), \c sleep_disable(), and
77 \c sleep_cpu(). This also allows for test-and-sleep scenarios that
78 take care of not missing the interrupt that will awake the device
79 from sleep.
80
81 Example:
82 \code
83 #include <avr/interrupt.h>
84 #include <avr/sleep.h>
85
86 ...
87 set_sleep_mode(<mode>);
88 cli();
89 if (some_condition)
90 {
91 sleep_enable();
92 sei();
93 sleep_cpu();
94 sleep_disable();
95 }

```

```

96 sei();
97 \endcode
98
99 This sequence ensures an atomic test of \c some_condition with
100 interrupts being disabled. If the condition is met, sleep mode
101 will be prepared, and the \c SLEEP instruction will be scheduled
102 immediately after an \c SEI instruction. As the instruction right
103 after the \c SEI is guaranteed to be executed before an interrupt
104 could trigger, it is sure the device will really be put to sleep.
105
106 Some devices have the ability to disable the Brown Out Detector (BOD) before
107 going to sleep. This will also reduce power while sleeping. If the
108 specific AVR device has this ability then an additional macro is defined:
109 \c sleep_bod_disable(). This macro generates inlined assembly code
110 that will correctly implement the timed sequence for disabling the BOD
111 before sleeping. However, there is a limited number of cycles after the
112 BOD has been disabled that the device can be put into sleep mode, otherwise
113 the BOD will not truly be disabled. Recommended practice is to disable
114 the BOD (\c sleep_bod_disable()), set the interrupts (\c sei()), and then
115 put the device to sleep (\c sleep_cpu()), like so:
116
117 \code
118 #include <avr/interrupt.h>
119 #include <avr/sleep.h>
120
121 ...
122 set_sleep_mode(<mode>);
123 cli();
124 if (some_condition)
125 {
126     sleep_enable();
127     sleep_bod_disable();
128     sei();
129     sleep_cpu();
130     sleep_disable();
131 }
132 sei();
133 \endcode
134 */
135
136
137 /* Define an internal sleep control register and an internal sleep enable bit mask. */
138 #if defined(SLEEP_CTRL)
139
140     /* XMEGA devices */
141     #define _SLEEP_CONTROL_REG    SLEEP_CTRL
142     #define _SLEEP_ENABLE_MASK    SLEEP_SEN_bm
143
144 #elif defined(SMCR)
145
146     #define _SLEEP_CONTROL_REG    SMCR
147     #define _SLEEP_ENABLE_MASK    _BV(SE)
148
149 #elif defined(__AVR_AT94K__)
150
151     #define _SLEEP_CONTROL_REG    MCUR
152     #define _SLEEP_ENABLE_MASK    _BV(SE)
153
154 #elif !defined(__DOXYGEN__)
155
156     #define _SLEEP_CONTROL_REG    MCUCR
157     #define _SLEEP_ENABLE_MASK    _BV(SE)
158
159 #endif
160
161
162 /* Special casing these three devices - they are the
163 only ones that need to write to more than one register. */
164 #if defined(__AVR_ATmega161__)
165
166     #define set_sleep_mode(mode) \
167     do { \
168         MCUCR = ((MCUCR & ~_BV(SM1)) | ((mode) == SLEEP_MODE_PWR_DOWN || (mode) == SLEEP_MODE_PWR_SAVE ? \
169             _BV(SM1) : 0)); \
169         EMCUCR = ((EMCUCR & ~_BV(SM0)) | ((mode) == SLEEP_MODE_PWR_SAVE ? _BV(SM0) : 0)); \
170     } while(0)
171
172
173 #elif defined(__AVR_ATmega162__) \
174 || defined(__AVR_ATmega8515__)
175
176     #define set_sleep_mode(mode) \
177     do { \
178         MCUCR = ((MCUCR & ~_BV(SM1)) | ((mode) == SLEEP_MODE_IDLE ? 0 : _BV(SM1))); \
179         MCUCSR = ((MCUCSR & ~_BV(SM2)) | ((mode) == SLEEP_MODE_STANDBY || (mode) == SLEEP_MODE_EXT_STANDBY ? \
180             _BV(SM2) : 0)); \
181         EMCUCR = ((EMCUCR & ~_BV(SM0)) | ((mode) == SLEEP_MODE_PWR_SAVE || (mode) == SLEEP_MODE_EXT_STANDBY ?

```

```

    _BV(SM0) : 0)); \
181 } while(0)
182
183 /* For xmega, check presence of SLEEP_SMODE<n>_bm and define set_sleep_mode accordingly. */
184 #elif defined(__AVR_XMEGA__)
185 #if defined(SLEEP_SMODE2_bm)
186
187 #define set_sleep_mode(mode) \
188 do { \
189 _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~(SLEEP_SMODE2_bm | SLEEP_SMODE1_bm | SLEEP_SMODE0_bm)) |
    (mode)); \
190 } while(0)
191
192 #elif defined(SLEEP_SMODE1_bm)
193
194 #define set_sleep_mode(mode) \
195 do { \
196 _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~(SLEEP_SMODE1_bm | SLEEP_SMODE0_bm)) | (mode)); \
197 } while(0)
198
199 #else
200
201 #define set_sleep_mode(mode) \
202 do { \
203 _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~( SLEEP_SMODE0_bm)) | (mode)); \
204 } while(0)
205
206
207 #endif /* #if defined(SLEEP_SMODE2_bm) */
208
209 /* For everything else, check for presence of SM<n> and define set_sleep_mode accordingly. */
210 #else
211 #if defined(SM2)
212
213 #define set_sleep_mode(mode) \
214 do { \
215 _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~(_BV(SM0) | _BV(SM1) | _BV(SM2))) | (mode)); \
216 } while(0)
217
218 #elif defined(SM1)
219
220 #define set_sleep_mode(mode) \
221 do { \
222 _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~(_BV(SM0) | _BV(SM1))) | (mode)); \
223 } while(0)
224
225 #elif defined(SM)
226
227 #define set_sleep_mode(mode) \
228 do { \
229 _SLEEP_CONTROL_REG = ((_SLEEP_CONTROL_REG & ~_BV(SM)) | (mode)); \
230 } while(0)
231
232 #else
233
234 #error "No SLEEP mode defined for this device."
235
236 #endif /* if defined(SM2) */
237 #endif /* #if defined(__AVR_ATmega161__) */
238
239
240
241 /** \ingroup avr_sleep
242
243 Put the device in sleep mode. How the device is brought out of sleep mode
244 depends on the specific mode selected with the set_sleep_mode() function.
245 See the data sheet for your device for more details. */
246
247
248 #if defined(__DOXYGEN__)
249
250 /** \ingroup avr_sleep
251
252 Set the SE (sleep enable) bit.
253 */
254 extern void sleep_enable (void);
255
256 #else
257
258 #define sleep_enable() \
259 do { \
260 _SLEEP_CONTROL_REG |= (uint8_t)_SLEEP_ENABLE_MASK; \
261 } while(0)
262
263 #endif
264
265

```



```

266 #if defined(__DOXYGEN__)
267
268 /** \ingroup avr_sleep
269
270 Clear the SE (sleep enable) bit.
271 */
272 extern void sleep_disable (void);
273
274 #else
275
276 #define sleep_disable() \
277 do { \
278     _SLEEP_CONTROL_REG &= (uint8_t)(~_SLEEP_ENABLE_MASK); \
279 } while(0)
280
281 #endif
282
283
284 /** \ingroup avr_sleep
285
286 Put the device into sleep mode. The SE bit must be set
287 beforehand, and it is recommended to clear it afterwards.
288 */
289 #if defined(__DOXYGEN__)
290
291 extern void sleep_cpu (void);
292
293 #else
294
295 #define sleep_cpu() \
296 do { \
297     __asm__ __volatile__ ( "sleep" "\n\t" :: ); \
298 } while(0)
299
300 #endif
301
302
303 #if defined(__DOXYGEN__)
304
305 /** \ingroup avr_sleep
306
307 Put the device into sleep mode, taking care of setting
308 the SE bit before, and clearing it afterwards. */
309 extern void sleep_mode (void);
310
311 #else
312
313 #define sleep_mode() \
314 do { \
315     sleep_enable(); \
316     sleep_cpu(); \
317     sleep_disable(); \
318 } while (0)
319
320 #endif
321
322
323 #if defined(__DOXYGEN__)
324
325 /** \ingroup avr_sleep
326
327 Disable BOD before going to sleep.
328 Not available on all devices.
329 */
330 extern void sleep_bod_disable (void);
331
332 #else
333
334 #if defined(BODS) && defined(BODSE)
335
336 #ifdef BODCR
337
338 #define BOD_CONTROL_REG BODCR
339
340 #else
341
342 #define BOD_CONTROL_REG MCUCR
343
344 #endif
345
346 #define sleep_bod_disable() \
347 do { \
348     uint8_t tempreg; \
349     __asm__ __volatile__ ("in %[tempreg], %[mcucr]" "\n\t" \
350                          "ori %[tempreg], %[bods_bodse]" "\n\t" \
351                          "out %[mcucr], %[tempreg]" "\n\t" \
352                          "andi %[tempreg], %[not_bodse]" "\n\t" \

```

```

353         "out %[mcucr], %[tempreg]" \
354         : [tempreg] "=&d" (tempreg) \
355         : [mcucr] "I" _SFR_IO_ADDR(BOD_CONTROL_REG), \
356         [bods_bodse] "i" (_BV(BODS) | _BV(BODSE)), \
357         [not_bodse] "i" (~_BV(BODSE)); \
358 } while (0)
359 #endif
360 #endif
361 #endif
362 #endif
363 #endif
364 #endif
365 /* @} */
366 #endif /* _AVR_SLEEP_H */

```

25.34 version.h

```

1  /* Copyright (c) 2005, Joerg Wunsch                               -*- c -*-
2  All rights reserved.
3
4  Redistribution and use in source and binary forms, with or without
5  modification, are permitted provided that the following conditions are met:
6
7  * Redistributions of source code must retain the above copyright
8  notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: version.h.in 870 2005-09-12 20:18:12Z joerg_wunsch $ */
32
33 /** \defgroup avr_version <avr/version.h>: avr-libc version macros
34 \code #include <avr/version.h> \endcode
35
36 This header file defines macros that contain version numbers and
37 strings describing the current version of avr-libc.
38
39 The version number itself basically consists of three pieces that
40 are separated by a dot: the major number, the minor number, and
41 the revision number. For development versions (which use an odd
42 minor number), the string representation additionally gets the
43 date code (YYYYMMDD) appended.
44
45 This file will also be included by \c <avr/io.h>. That way,
46 portable tests can be implemented using \c <avr/io.h> that can be
47 used in code that wants to remain backwards-compatible to library
48 versions prior to the date when the library version API had been
49 added, as referenced but undefined C preprocessor macros
50 automatically evaluate to 0.
51 */
52
53 #ifndef _AVR_VERSION_H_
54 #define _AVR_VERSION_H_
55
56 /** \ingroup avr_version
57 String literal representation of the current library version. */
58 #define __AVR_LIBC_VERSION_STRING__ "2.1.0"
59
60 /** \ingroup avr_version
61 Numerical representation of the current library version.
62
63 In the numerical representation, the major number is multiplied by
64 10000, the minor number by 100, and all three parts are then
65 added. It is intended to provide a monotonically increasing
66 numerical value that can easily be used in numerical checks.

```

```

67 */
68 #define __AVR_LIBC_VERSION__      20100UL
69
70 /** \ingroup avr_version
71 String literal representation of the release date. */
72 #define __AVR_LIBC_DATE_STRING__  "20220128"
73
74 /** \ingroup avr_version
75 Numerical representation of the release date. */
76 #define __AVR_LIBC_DATE__         20220128UL
77
78 /** \ingroup avr_version
79 Library major version number. */
80 #define __AVR_LIBC_MAJOR__        2
81
82 /** \ingroup avr_version
83 Library minor version number. */
84 #define __AVR_LIBC_MINOR__        1
85
86 /** \ingroup avr_version
87 Library revision number. */
88 #define __AVR_LIBC_REVISION__     0
89
90 #endif /* _AVR_VERSION_H_ */

```

25.35 wdt.h File Reference

Macros

- #define [wdt_reset\(\)](#) `__asm__ __volatile__ ("wdr")`
- #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 `__inline__` [__attribute__](#) ((`__always_inline__`)) void [wdt_enable](#)(const [uint8_t](#) value)

25.36 wdt.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, 2004 Marek Michalkiewicz
2 Copyright (c) 2005, 2006, 2007 Eric B. Weddington
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15
16 * Neither the name of the copyright holders nor the names of
17 contributors may be used to endorse or promote products derived
18 from this software without specific prior written permission.
19

```

```

20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE. */
31
32 /* $Id: wdt.h 2522 2016-04-20 05:43:23Z joerg_wunsch $ */
33
34 /*
35 avr/wdt.h - macros for AVR watchdog timer
36 */
37
38 #ifndef _AVR_WDT_H_
39 #define _AVR_WDT_H_
40
41 #include <avr/io.h>
42 #include <stdint.h>
43
44 /** \file */
45 /** \defgroup avr_watchdog <avr/wdt.h>: Watchdog timer handling
46 \code #include <avr/wdt.h> \endcode
47
48 This header file declares the interface to some inline macros
49 handling the watchdog timer present in many AVR devices. In order
50 to prevent the watchdog timer configuration from being
51 accidentally altered by a crashing application, a special timed
52 sequence is required in order to change it. The macros within
53 this header file handle the required sequence automatically
54 before changing any value. Interrupts will be disabled during
55 the manipulation.
56
57 \note Depending on the fuse configuration of the particular
58 device, further restrictions might apply, in particular it might
59 be disallowed to turn off the watchdog timer.
60
61 Note that for newer devices (ATmega88 and newer, effectively any
62 AVR that has the option to also generate interrupts), the watchdog
63 timer remains active even after a system reset (except a power-on
64 condition), using the fastest prescaler value (approximately 15
65 ms). It is therefore required to turn off the watchdog early
66 during program startup, the datasheet recommends a sequence like
67 the following:
68
69 \code
70 #include <stdint.h>
71 #include <avr/wdt.h>
72
73 uint8_t mcusr_mirror __attribute__((section(".noinit")));
74
75 void get_mcusr(void) \
76 __attribute__((naked)) \
77 __attribute__((section(".init3")));
78 void get_mcusr(void)
79 {
80 mcusr_mirror = MCUSR;
81 MCUSR = 0;
82 wdt_disable();
83 }
84 \endcode
85
86 Saving the value of MCUSR in \c mcusr_mirror is only needed if the
87 application later wants to examine the reset source, but in particular,
88 clearing the watchdog reset flag before disabling the
89 watchdog is required, according to the datasheet.
90 */
91
92 /**
93 \ingroup avr_watchdog
94 Reset the watchdog timer. When the watchdog timer is enabled,
95 a call to this instruction is required before the timer expires,
96 otherwise a watchdog-initiated device reset will occur.
97 */
98
99 #define wdt_reset() __asm__ __volatile__ ("wdr")
100
101 #ifndef __DOXYGEN__
102
103 #if defined(WDP3)
104 # define _WD_PS3_MASK _BV(WDP3)
105 #else
106 # define _WD_PS3_MASK 0x00

```

```

107 #endif
108
109 #if defined(WDTCSR)
110 # define _WD_CONTROL_REG      WDTCSR
111 #elif defined(WDTCR)
112 # define _WD_CONTROL_REG      WDTCSR
113 #else
114 # define _WD_CONTROL_REG      WDT
115 #endif
116
117 #if defined(WDTOE)
118 #define _WD_CHANGE_BIT        WDTOE
119 #else
120 #define _WD_CHANGE_BIT        WDCE
121 #endif
122
123 #endif /* !__DOXYGEN__ */
124
125
126 /**
127 \ingroup avr_watchdog
128 Enable the watchdog timer, configuring it for expiry after
129 \c timeout (which is a combination of the \c WDP0 through
130 \c WDP2 bits to write into the \c WDTCR register; For those devices
131 that have a \c WDTCSR register, it uses the combination of the \c WDP0
132 through \c WDP3 bits).
133
134 See also the symbolic constants \c WDTO_15MS et al.
135 */
136
137
138 #if defined(__AVR_XMEGA__)
139
140 /*
141 wdt_enable(timeout) for xmega devices
142 ** write signature (CCP_IOREG_gc) that enables change of protected I/O
143 registers to the CCP register
144 ** At the same time,
145 1) set WDT change enable (WDT_CEN_bm)
146 2) enable WDT (WDT_ENABLE_bm)
147 3) set timeout (timeout)
148 ** Synchronization starts when ENABLE bit of WDT is set. So, wait till it
149 finishes (SYNDBUSY of STATUS register is automatically cleared after the
150 sync is finished).
151 */
152 #define wdt_enable(timeout) \
153 do { \
154     uint8_t tmp; \
155     __asm__ __volatile__ ( \
156         "in __tmp_reg__, %[rampd]"           "\n\t" \
157         "out %[rampd], __zero_reg__"         "\n\t" \
158         "out %[ccp_reg], %[ioreg_cen_mask]"   "\n\t" \
159         "sts %[wdt_reg], %[wdt_enable_timeout]" "\n\t" \
160         "l:lds %[tmp], %[wdt_status_reg]"     "\n\t" \
161         "sbrc %[tmp], %[wdt_syncbusy_bit]"    "\n\t" \
162         "rjmp 1b"                             "\n\t" \
163         "out %[rampd], __tmp_reg__"           "\n\t" \
164         : [tmp]                               "=r" (tmp) \
165         : [rampd]                             "I" (_SFR_IO_ADDR(RAMPD)), \
166         [ccp_reg]                             "I" (_SFR_IO_ADDR(CCP)), \
167         [ioreg_cen_mask]                       "r" ((uint8_t)CCP_IOREG_gc), \
168         [wdt_reg]                             "n" (_SFR_MEM_ADDR(WDT_CTRL)), \
169         [wdt_enable_timeout]                   "r" ((uint8_t)(WDT_CEN_bm | WDT_ENABLE_bm | timeout)), \
170         [wdt_status_reg]                       "n" (_SFR_MEM_ADDR(WDT_STATUS)), \
171         [wdt_syncbusy_bit]                     "I" (WDT_SYNCBUSY_bm) \
172         : "r0" \
173     ); \
174 } while(0)
175
176 #define wdt_disable() \
177 __asm__ __volatile__ ( \
178     "in __tmp_reg__, %[rampd]"           "\n\t" \
179     "out %[rampd], __zero_reg__"         "\n\t" \
180     "out %[ccp_reg], %[ioreg_cen_mask]"   "\n\t" \
181     "sts %[wdt_reg], %[disable_mask]"     "\n\t" \
182     "out %[rampd], __tmp_reg__"           "\n\t" \
183     : \
184     : [rampd]                             "I" (_SFR_IO_ADDR(RAMPD)), \
185     [ccp_reg]                             "I" (_SFR_IO_ADDR(CCP)), \
186     [ioreg_cen_mask]                       "r" ((uint8_t)CCP_IOREG_gc), \
187     [wdt_reg]                             "n" (_SFR_MEM_ADDR(WDT_CTRL)), \
188     [disable_mask]                         "r" ((uint8_t)((~WDT_ENABLE_bm) | WDT_CEN_bm)) \
189     : "r0" \
190 );
191
192 #elif defined(__AVR_TINY__)
193

```

```

194 #define wdt_enable(value) \
195 __asm__ __volatile__ ( \
196 "in __tmp_reg__, __SREG__" "\n\t" \
197 "cli" "\n\t" \
198 "wdr" "\n\t" \
199 "out %[CCPADDRESS], %[SIGNATURE]" "\n\t" \
200 "out %[WDTREG], %[WDVALUE]" "\n\t" \
201 "out __SREG__, __tmp_reg__" "\n\t" \
202 : /* no outputs */ \
203 : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)), \
204 [SIGNATURE] "r" ((uint8_t)0xD8), \
205 [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)), \
206 [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00) \
207 | _BV(WDE) | (value & 0x07) )) \
208 : "r16" \
209 )
210
211 #define wdt_disable() \
212 do { \
213 uint8_t temp_wd; \
214 __asm__ __volatile__ ( \
215 "in __tmp_reg__, __SREG__" "\n\t" \
216 "cli" "\n\t" \
217 "wdr" "\n\t" \
218 "out %[CCPADDRESS], %[SIGNATURE]" "\n\t" \
219 "in %[TEMP_WD], %[WDTREG]" "\n\t" \
220 "cbr %[TEMP_WD], %[WDVALUE]" "\n\t" \
221 "out %[WDTREG], %[TEMP_WD]" "\n\t" \
222 "out __SREG__, __tmp_reg__" "\n\t" \
223 : /*no output */ \
224 : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)), \
225 [SIGNATURE] "r" ((uint8_t)0xD8), \
226 [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)), \
227 [TEMP_WD] "d" (temp_wd), \
228 [WDVALUE] "n" (1 << WDE) \
229 : "r16" \
230 ); \
231 }while(0)
232
233 #elif defined(CCP)
234
235 static __inline__
236 __attribute__((always_inline))
237 void wdt_enable (const uint8_t value)
238 {
239     if (!_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
240     {
241         __asm__ __volatile__ (
242             "in __tmp_reg__, __SREG__" "\n\t"
243             "cli" "\n\t"
244             "wdr" "\n\t"
245             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
246             "sts %[WDTREG], %[WDVALUE]" "\n\t"
247             "out __SREG__, __tmp_reg__" "\n\t"
248             : /* no outputs */
249             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
250             [SIGNATURE] "r" ((uint8_t)0xD8),
251             [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
252             [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00)
253 | _BV(WDE) | (value & 0x07) ))
254             : "r0"
255             );
256     }
257     else if (!_SFR_IO_REG_P (CCP) && _SFR_IO_REG_P (_WD_CONTROL_REG))
258     {
259         __asm__ __volatile__ (
260             "in __tmp_reg__, __SREG__" "\n\t"
261             "cli" "\n\t"
262             "wdr" "\n\t"
263             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
264             "out %[WDTREG], %[WDVALUE]" "\n\t"
265             "out __SREG__, __tmp_reg__" "\n\t"
266             : /* no outputs */
267             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
268             [SIGNATURE] "r" ((uint8_t)0xD8),
269             [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
270             [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00)
271 | _BV(WDE) | (value & 0x07) ))
272             : "r0"
273             );
274     }
275     else if (_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
276     {
277         __asm__ __volatile__ (
278             "in __tmp_reg__, __SREG__" "\n\t"
279             "cli" "\n\t"
280             "wdr" "\n\t"

```

```

281         "out %[CCPADDRESS], %[SIGNATURE]" "\n\t"
282         "sts %[WDTREG], %[WDVALUE]" "\n\t"
283         "out __SREG__, __tmp_reg__" "\n\t"
284         : /* no outputs */
285         : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
286         : [SIGNATURE] "r" ((uint8_t)0xD8),
287         : [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
288         : [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00)
289         | _BV(WDE) | (value & 0x07) ));
290         : "r0"
291     );
292 }
293 else
294 {
295     __asm__ __volatile__ (
296         "in __tmp_reg__, __SREG__" "\n\t"
297         "cli" "\n\t"
298         "wdr" "\n\t"
299         "out %[CCPADDRESS], %[SIGNATURE]" "\n\t"
300         "out %[WDTREG], %[WDVALUE]" "\n\t"
301         "out __SREG__, __tmp_reg__" "\n\t"
302         : /* no outputs */
303         : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
304         : [SIGNATURE] "r" ((uint8_t)0xD8),
305         : [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
306         : [WDVALUE] "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00)
307         | _BV(WDE) | (value & 0x07) ));
308         : "r0"
309     );
310 }
311 }
312
313 static __inline__
314 __attribute__((always_inline))
315 void wdt_disable (void)
316 {
317     if (!_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
318     {
319         uint8_t temp_wd;
320         __asm__ __volatile__ (
321             "in __tmp_reg__, __SREG__" "\n\t"
322             "cli" "\n\t"
323             "wdr" "\n\t"
324             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
325             "lds %[TEMP_WD], %[WDTREG]" "\n\t"
326             "cbr %[TEMP_WD], %[WDVALUE]" "\n\t"
327             "sts %[WDTREG], %[TEMP_WD]" "\n\t"
328             "out __SREG__, __tmp_reg__" "\n\t"
329             : /*no output */
330             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
331             : [SIGNATURE] "r" ((uint8_t)0xD8),
332             : [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
333             : [TEMP_WD] "d" (temp_wd),
334             : [WDVALUE] "n" (1 << WDE)
335             : "r0"
336         );
337     }
338     else if (!_SFR_IO_REG_P (CCP) && _SFR_IO_REG_P (_WD_CONTROL_REG))
339     {
340         uint8_t temp_wd;
341         __asm__ __volatile__ (
342             "in __tmp_reg__, __SREG__" "\n\t"
343             "cli" "\n\t"
344             "wdr" "\n\t"
345             "sts %[CCPADDRESS], %[SIGNATURE]" "\n\t"
346             "in %[TEMP_WD], %[WDTREG]" "\n\t"
347             "cbr %[TEMP_WD], %[WDVALUE]" "\n\t"
348             "out %[WDTREG], %[TEMP_WD]" "\n\t"
349             "out __SREG__, __tmp_reg__" "\n\t"
350             : /*no output */
351             : [CCPADDRESS] "n" (_SFR_MEM_ADDR(CCP)),
352             : [SIGNATURE] "r" ((uint8_t)0xD8),
353             : [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
354             : [TEMP_WD] "d" (temp_wd),
355             : [WDVALUE] "n" (1 << WDE)
356             : "r0"
357         );
358     }
359     else if (_SFR_IO_REG_P (CCP) && !_SFR_IO_REG_P (_WD_CONTROL_REG))
360     {
361         uint8_t temp_wd;
362         __asm__ __volatile__ (
363             "in __tmp_reg__, __SREG__" "\n\t"
364             "cli" "\n\t"
365             "wdr" "\n\t"
366             "out %[CCPADDRESS], %[SIGNATURE]" "\n\t"
367             "lds %[TEMP_WD], %[WDTREG]" "\n\t"

```

```

368         "cbr %[TEMP_WD], %[WDVALUE]" "\n\t"
369         "sts %[WDTREG], %[TEMP_WD]" "\n\t"
370         "out __SREG__, __tmp_reg__" "\n\t"
371         : /*no output */
372         : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
373         : [SIGNATURE] "r" ((uint8_t)0xD8),
374         : [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
375         : [TEMP_WD] "d" (temp_wd),
376         : [WDVALUE] "n" (1 << WDE)
377         : "r0"
378         );
379     }
380     else
381     {
382         uint8_t temp_wd;
383         __asm__ __volatile__ (
384             "in __tmp_reg__, __SREG__" "\n\t"
385             "cli" "\n\t"
386             "wdr" "\n\t"
387             "out %[CCPADDRESS], %[SIGNATURE]" "\n\t"
388             "in %[TEMP_WD], %[WDTREG]" "\n\t"
389             "cbr %[TEMP_WD], %[WDVALUE]" "\n\t"
390             "out %[WDTREG], %[TEMP_WD]" "\n\t"
391             "out __SREG__, __tmp_reg__" "\n\t"
392             : /*no output */
393             : [CCPADDRESS] "I" (_SFR_IO_ADDR(CCP)),
394             : [SIGNATURE] "r" ((uint8_t)0xD8),
395             : [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
396             : [TEMP_WD] "d" (temp_wd),
397             : [WDVALUE] "n" (1 << WDE)
398             : "r0"
399             );
400     }
401 }
402
403 #else
404
405 static __inline__
406 __attribute__((always_inline))
407 void wdt_enable (const uint8_t value)
408 {
409     if (_SFR_IO_REG_P (_WD_CONTROL_REG))
410     {
411         __asm__ __volatile__ (
412             "in __tmp_reg__, __SREG__" "\n\t"
413             "cli" "\n\t"
414             "wdr" "\n\t"
415             "out %0, %1" "\n\t"
416             "out __SREG__, __tmp_reg__" "\n\t"
417             "out %0, %2" "\n\t"
418             : /* no outputs */
419             : "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
420             : "r" ((uint8_t)(_BV(_WD_CHANGE_BIT) | _BV(WDE))),
421             : "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00) |
422                 _BV(WDE) | (value & 0x07)))
423             : "r0"
424             );
425     }
426     else
427     {
428         __asm__ __volatile__ (
429             "in __tmp_reg__, __SREG__" "\n\t"
430             "cli" "\n\t"
431             "wdr" "\n\t"
432             "sts %0, %1" "\n\t"
433             "out __SREG__, __tmp_reg__" "\n\t"
434             "sts %0, %2" "\n\t"
435             : /* no outputs */
436             : "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
437             : "r" ((uint8_t)(_BV(_WD_CHANGE_BIT) | _BV(WDE))),
438             : "r" ((uint8_t)((value & 0x08 ? _WD_PS3_MASK : 0x00) |
439                 _BV(WDE) | (value & 0x07)))
440             : "r0"
441             );
442     }
443 }
444
445 static __inline__
446 __attribute__((always_inline))
447 void wdt_disable (void)
448 {
449     if (_SFR_IO_REG_P (_WD_CONTROL_REG))
450     {
451         uint8_t register temp_reg;
452         __asm__ __volatile__ (
453             "in __tmp_reg__, __SREG__" "\n\t"
454             "cli" "\n\t"

```



```

455         "wdr"                                "\n\t"
456         "in %[TEMPREG], %[WDTREG]"            "\n\t"
457         "ori %[TEMPREG], %[WDCE_WDE]"         "\n\t"
458         "out %[WDTREG], %[TEMPREG]"           "\n\t"
459         "out %[WDTREG], __zero_reg__"          "\n\t"
460         "out __SREG__, __tmp_reg__"           "\n\t"
461         : [TEMPREG] "=d" (temp_reg)
462         : [WDTREG] "I" (_SFR_IO_ADDR(_WD_CONTROL_REG)),
463         [WDCE_WDE] "n" ((uint8_t) (_BV(_WD_CHANGE_BIT) | _BV(WDE)))
464         : "r0"
465     );
466 }
467 else
468 {
469     uint8_t register temp_reg;
470     __asm__ __volatile__ (
471         "in __tmp_reg__, __SREG__"            "\n\t"
472         "cli"                                  "\n\t"
473         "wdr"                                  "\n\t"
474         "lds %[TEMPREG], %[WDTREG]"            "\n\t"
475         "ori %[TEMPREG], %[WDCE_WDE]"         "\n\t"
476         "sts %[WDTREG], %[TEMPREG]"           "\n\t"
477         "sts %[WDTREG], __zero_reg__"          "\n\t"
478         "out __SREG__, __tmp_reg__"           "\n\t"
479         : [TEMPREG] "=d" (temp_reg)
480         : [WDTREG] "n" (_SFR_MEM_ADDR(_WD_CONTROL_REG)),
481         [WDCE_WDE] "n" ((uint8_t) (_BV(_WD_CHANGE_BIT) | _BV(WDE)))
482         : "r0"
483     );
484 }
485 }
486
487 #endif
488
489 /**
490 \ingroup avr_watchdog
491 Symbolic constants for the watchdog timeout. Since the watchdog
492 timer is based on a free-running RC oscillator, the times are
493 approximate only and apply to a supply voltage of 5 V. At lower
494 supply voltages, the times will increase. For older devices, the
495 times will be as large as three times when operating at Vcc = 3 V,
496 while the newer devices (e. g. ATmega128, ATmega8) only experience
497 a negligible change.
498
499 Possible timeout values are: 15 ms, 30 ms, 60 ms, 120 ms, 250 ms,
500 500 ms, 1 s, 2 s. (Some devices also allow for 4 s and 8 s.)
501 Symbolic constants are formed by the prefix
502 \c WDTO_, followed by the time.
503
504 Example that would select a watchdog timer expiry of approximately
505 500 ms:
506 \code
507 wdt_enable(WDTO_500MS);
508 \endcode
509 */
510 #define WDTO_15MS 0
511
512 /** \ingroup avr_watchdog
513 See \c WDTO_15MS */
514 #define WDTO_30MS 1
515
516 /** \ingroup avr_watchdog
517 See \c WDTO_15MS */
518 #define WDTO_60MS 2
519
520 /** \ingroup avr_watchdog
521 See \c WDTO_15MS */
522 #define WDTO_120MS 3
523
524 /** \ingroup avr_watchdog
525 See \c WDTO_15MS */
526 #define WDTO_250MS 4
527
528 /** \ingroup avr_watchdog
529 See \c WDTO_15MS */
530 #define WDTO_500MS 5
531
532 /** \ingroup avr_watchdog
533 See \c WDTO_15MS */
534 #define WDTO_1S 6
535
536 /** \ingroup avr_watchdog
537 See \c WDTO_15MS */
538 #define WDTO_2S 7
539
540 #if defined(__DOXYGEN__) || defined(WDP3)

```

```

542
543 /** \ingroup avr_watchdog
544 See \c WDTO_15MS
545 Note: This is only available on the
546 ATtiny2313,
547 ATtiny24, ATtiny44, ATtiny84, ATtiny84A,
548 ATtiny25, ATtiny45, ATtiny85,
549 ATtiny261, ATtiny461, ATtiny861,
550 ATmega48, ATmega88, ATmega168,
551 ATmega48P, ATmega88P, ATmega168P, ATmega328P,
552 ATmega164P, ATmega324P, ATmega324PA, ATmega644P, ATmega644,
553 ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561,
554 ATmega8HVA, ATmega16HVA, ATmega32HVB,
555 ATmega406, ATmega1284P,
556 AT90PWM1, AT90PWM2, AT90PWM2B, AT90PWM3, AT90PWM3B, AT90PWM216, AT90PWM316,
557 AT90PWM81, AT90PWM161,
558 AT90USB82, AT90USB162,
559 AT90USB646, AT90USB647, AT90USB1286, AT90USB1287,
560 ATtiny48, ATtiny88.
561
562 Note: This value does <em>not</em> match the bit pattern of the
563 respective control register. It is solely meant to be used together
564 with wdt_enable().
565 */
566 #define WDTO_4S      8
567
568 /** \ingroup avr_watchdog
569 See \c WDTO_15MS
570 Note: This is only available on the
571 ATtiny2313,
572 ATtiny24, ATtiny44, ATtiny84, ATtiny84A,
573 ATtiny25, ATtiny45, ATtiny85,
574 ATtiny261, ATtiny461, ATtiny861,
575 ATmega48, ATmega48A, ATmega48PA, ATmega88, ATmega168,
576 ATmega48P, ATmega88P, ATmega168P, ATmega328P,
577 ATmega164P, ATmega324P, ATmega324PA, ATmega644P, ATmega644,
578 ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561,
579 ATmega8HVA, ATmega16HVA, ATmega32HVB,
580 ATmega406, ATmega1284P,
581 ATmega2564RFR2, ATmega256RFR2, ATmega1284RFR2, ATmega128RFR2, ATmega644RFR2, ATmega64RFR2
582 AT90PWM1, AT90PWM2, AT90PWM2B, AT90PWM3, AT90PWM3B, AT90PWM216, AT90PWM316,
583 AT90PWM81, AT90PWM161,
584 AT90USB82, AT90USB162,
585 AT90USB646, AT90USB647, AT90USB1286, AT90USB1287,
586 ATtiny48, ATtiny88,
587 ATxmega16a4u, ATxmega32a4u,
588 ATxmega16c4, ATxmega32c4,
589 ATxmega128c3, ATxmega192c3, ATxmega256c3.
590
591 Note: This value does <em>not</em> match the bit pattern of the
592 respective control register. It is solely meant to be used together
593 with wdt_enable().
594 */
595 #define WDTO_8S      9
596
597 #endif /* defined(__DOXYGEN__) || defined(WDP3) */
598
599
600 #endif /* _AVR_WDT_H_ */

```

25.37 xmega.h

```

1 /* Copyright (c) 2012 Joerg Wunsch
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

```

```

23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: xmega.h 2500 2016-02-05 06:01:20Z pitchumani $ */
32
33 /*
34  * This file is included by <avr/io.h> whenever compiling for an Xmega
35  * device. It abstracts certain features common to the Xmega device
36  * families.
37  */
38
39 #ifndef _AVR_XMEGA_H
40 #define _AVR_XMEGA_H
41
42 #ifdef __DOXYGEN__
43 /**
44  \def _PROTECTED_WRITE
45  \ingroup avr_io
46
47  Write value \c value to IO register \c reg that is protected through
48  the Xmega configuration change protection (CCP) mechanism. This
49  implements the timed sequence that is required for CCP.
50
51  Example to modify the CPU clock:
52  \code
53  #include <avr/io.h>
54
55  _PROTECTED_WRITE(CLK_PSCTRL, CLK_PSADIV0_bm);
56  _PROTECTED_WRITE(CLK_CTRL, CLK_SCLKSEL0_bm);
57  \endcode
58  */
59 #define _PROTECTED_WRITE(reg, value)
60 #else /* !__DOXYGEN__ */
61 #define _PROTECTED_WRITE(reg, value)
62 __asm__ __volatile__ ("out %[ccp], %[ccp_ioreg]" "\n\t" \
63                      "sts %[ioreg], %[val]" \
64                      : \
65                      : [ccp] "I" (_SFR_IO_ADDR(CCP)), \
66                      [ccp_ioreg] "d" ((uint8_t)CCP_IOREG_gc), \
67                      [ioreg] "n" (_SFR_MEM_ADDR(reg)), \
68                      [val] "r" ((uint8_t)value))
69 #endif /* DOXYGEN */
70
71 #endif /* _AVR_XMEGA_H */

```

25.38 deprecated.h

```

1  /* Copyright (c) 2005,2006 Joerg Wunsch
2  All rights reserved.
3
4  Redistribution and use in source and binary forms, with or without
5  modification, are permitted provided that the following conditions are met:
6
7  * Redistributions of source code must retain the above copyright
8  notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: deprecated.h 1724 2008-07-30 21:31:33Z arcanum $ */
32

```

```

33 #ifndef _COMPAT_DEPRECATED_H_
34 #define _COMPAT_DEPRECATED_H_
35
36 /** \defgroup deprecated_items <compat/deprecated.h>:  Deprecated items
37
38 This header file contains several items that used to be available
39 in previous versions of this library, but have eventually been
40 deprecated over time.
41
42 \code #include <compat/deprecated.h> \endcode
43
44 These items are supplied within that header file for backward
45 compatibility reasons only, so old source code that has been
46 written for previous library versions could easily be maintained
47 until its end-of-life.  Use of any of these items in new code is
48 strongly discouraged.
49 */
50
51 /** \name Allowing specific system-wide interrupts
52
53 In addition to globally enabling interrupts, each device's particular
54 interrupt needs to be enabled separately if interrupts for this device are
55 desired.  While some devices maintain their interrupt enable bit inside
56 the device's register set, external and timer interrupts have system-wide
57 configuration registers.
58
59 Example:
60
61 \code
62 // Enable timer 1 overflow interrupts.
63 timer_enable_int(_BV(TOIE1));
64
65 // Do some work...
66
67 // Disable all timer interrupts.
68 timer_enable_int(0);
69 \endcode
70
71 \note Be careful when you use these functions.  If you already have a
72 different interrupt enabled, you could inadvertently disable it by
73 enabling another interrupt.  */
74
75 /*{*/
76
77 /** \ingroup deprecated_items
78 \def enable_external_int(mask)
79 \deprecated
80
81 This macro gives access to the \c GIMSK register (or \c EIMSK register
82 if using an AVR Mega device or \c GICR register for others).  Although this
83 macro is essentially the same as assigning to the register, it does
84 adapt slightly to the type of device being used.  This macro is
85 unavailable if none of the registers listed above are defined.  */
86
87 /* Define common register definition if available.  */
88 #if defined(EIMSK)
89 #   define __EICR EIMSK
90 #elif defined(GIMSK)
91 #   define __EICR GIMSK
92 #elif defined(GICR)
93 #   define __EICR GICR
94 #endif
95
96 /* If common register defined, define macro.  */
97 #if defined(__EICR) || defined(__DOXYGEN__)
98 #define enable_external_int(mask)          (__EICR = mask)
99 #endif
100
101 /** \ingroup deprecated_items
102 \deprecated
103
104 This function modifies the \c tmsk register.
105 The value you pass via \c ints is device specific.  */
106
107 static __inline__ void timer_enable_int (unsigned char ints)
108 {
109     #ifdef TIMSK
110         TIMSK = ints;
111     #endif
112 }
113
114 /** \def INTERRUPT(signame)
115 \ingroup deprecated_items
116 \deprecated
117
118 Introduces an interrupt handler function that runs with global interrupts
119 initially enabled.  This allows interrupt handlers to be interrupted.

```

```

120
121 As this macro has been used by too many unsuspecting people in the
122 past, it has been deprecated, and will be removed in a future
123 version of the library. Users who want to legitimately re-enable
124 interrupts in their interrupt handlers as quickly as possible are
125 encouraged to explicitly declare their handlers as described
126 \ref attr_interrupt "above".
127 */
128
129 #if (__GNUC__ == 4 && __GNUC_MINOR__ >= 1) || (__GNUC__ > 4)
130 # define __INTR_ATTRS used, externally_visible
131 #else /* GCC < 4.1 */
132 # define __INTR_ATTRS used
133 #endif
134
135 #ifdef __cplusplus
136 #define INTERRUPT(signame) \
137 extern "C" void signame(void); \
138 void signame(void) __attribute__((interrupt,__INTR_ATTRS)); \
139 void signame(void)
140 #else
141 #define INTERRUPT(signame) \
142 void signame(void) __attribute__((interrupt,__INTR_ATTRS)); \
143 void signame(void)
144 #endif
145
146 /* @} */
147
148 /**
149 \name Obsolete IO macros
150
151 Back in a time when AVR-GCC and avr-libc could not handle IO port
152 access in the direct assignment form as they are handled now, all
153 IO port access had to be done through specific macros that
154 eventually resulted in inline assembly instructions performing the
155 desired action.
156
157 These macros became obsolete, as reading and writing IO ports can
158 be done by simply using the IO port name in an expression, and all
159 bit manipulation (including those on IO ports) can be done using
160 generic C bit manipulation operators.
161
162 The macros in this group simulate the historical behaviour. While
163 they are supposed to be applied to IO ports, the emulation actually
164 uses standard C methods, so they could be applied to arbitrary
165 memory locations as well.
166 */
167
168 /* @{ */
169
170 /**
171 \ingroup deprecated_items
172 \def inp(port)
173 \deprecated
174
175 Read a value from an IO port \c port.
176 */
177 #define inp(port) (port)
178
179 /**
180 \ingroup deprecated_items
181 \def outp(val, port)
182 \deprecated
183
184 Write \c val to IO port \c port.
185 */
186 #define outp(val, port) (port) = (val)
187
188 /**
189 \ingroup deprecated_items
190 \def inb(port)
191 \deprecated
192
193 Read a value from an IO port \c port.
194 */
195 #define inb(port) (port)
196
197 /**
198 \ingroup deprecated_items
199 \def outb(port, val)
200 \deprecated
201
202 Write \c val to IO port \c port.
203 */
204 #define outb(port, val) (port) = (val)
205
206 /**

```

```

207 \ingroup deprecated_items
208 \def sbi(port, bit)
209 \deprecated
210
211 Set \c bit in IO port \c port.
212 */
213 #define sbi(port, bit) (port) |= (1 « (bit))
214
215 /**
216 \ingroup deprecated_items
217 \def cbi(port, bit)
218 \deprecated
219
220 Clear \c bit in IO port \c port.
221 */
222 #define cbi(port, bit) (port) &= ~(1 « (bit))
223
224 /*@}*/
225
226 #endif /* _COMPAT_DEPRECATED_H_ */

```

25.39 ina90.h

```

1 /* Copyright (c) 2002,2004 Marek Michalkiewicz
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: ina90.h 932 2005-11-05 21:22:33Z joerg_wunsch $ */
32 /* copied from: Id: avr/ina90.h,v 1.8 2004/11/09 19:16:09 arcanum Exp */
33
34 /*
35 ina90.h
36
37 Contributors:
38 Created by Marek Michalkiewicz <marekm@linux.org.pl>
39 */
40
41 /**
42 \defgroup compat_ina90 <compat/ina90.h>: Compatibility with IAR EWB 3.x
43
44 \code #include <compat/ina90.h> \endcode
45
46 This is an attempt to provide some compatibility with
47 header files that come with IAR C, to make porting applications
48 between different compilers easier. No 100% compatibility though.
49
50 \note For actual documentation, please see the IAR manual.
51 */
52
53 #ifndef _INA90_H_
54 #define _INA90_H_ 1
55
56 #define _CLI() do { __asm__ __volatile__ ("cli"); } while (0)
57 #define _SEI() do { __asm__ __volatile__ ("sei"); } while (0)
58 #define _NOP() do { __asm__ __volatile__ ("nop"); } while (0)
59 #define _WDR() do { __asm__ __volatile__ ("wdr"); } while (0)
60 #define _SLEEP() do { __asm__ __volatile__ ("sleep"); } while (0)
61 #define _OPC(op) do { __asm__ __volatile__ (".word %0" : : "n" (op)); } while (0)

```

```

62
63 /* _LPM, _ELPM */
64 #include <avr/pgmspace.h>
65 #define _LPM(x) do { __LPM(x); } while (0)
66 #define _ELPM(x) do { __ELPM(x); } while (0)
67
68 /* _EGET, _EPUT */
69 #include <avr/eeprom.h>
70
71 #define input(port) (port)
72 #define output(port, val) do { (port) = (val); } while (0)
73
74 #define __inp_blk__(port, addr, cnt, op) do { \
75 unsigned char __i = (cnt); \
76 unsigned char *__addr = (addr); \
77 while (__i) { \
78 *(__addr op) = input(port); \
79 __i--; \
80 } \
81 } while (0)
82
83 #define input_block_inc(port, addr, cnt) __inp_blk__(port, addr, cnt, ++)
84 #define input_block_dec(port, addr, cnt) __inp_blk__(port, addr, cnt, --)
85
86 #define __out_blk__(port, addr, cnt, op) do { \
87 unsigned char __i = (cnt); \
88 const unsigned char *__addr = (addr); \
89 while (__i) { \
90 output(port, *(__addr op)); \
91 __i--; \
92 } \
93 } while (0)
94
95 #define output_block_inc(port, addr, cnt) __out_blk__(port, addr, cnt, ++)
96 #define output_block_dec(port, addr, cnt) __out_blk__(port, addr, cnt, --)
97
98 #endif
99

```

25.40 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)

25.41 ctype.h

[Go to the documentation of this file.](#)

```

1  /* Copyright (c) 2002,2007 Michael Stumpf
2  All rights reserved.
3
4  Redistribution and use in source and binary forms, with or without
5  modification, are permitted provided that the following conditions are met:
6
7  * Redistributions of source code must retain the above copyright
8  notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id:  ctype.h 1504 2007-12-16 07:34:00Z dmix $ */
32
33 /*
34 ctype.h - character conversion macros and ctype macros
35
36 Author :  Michael Stumpf
37 Michael.Stumpf@t-online.de
38 */
39
40 #ifndef __CTYPE_H__
41 #define __CTYPE_H__ 1
42
43 #ifndef __ATTR_CONST__
44 #define __ATTR_CONST__ __attribute__((__const__))
45 #endif
46
47 #ifdef __cplusplus
48 extern "C" {
49 #endif
50
51 /** \file */
52 /** \defgroup ctype <ctype.h>: Character Operations
53 These functions perform various operations on characters.
54
55 \code #include <ctype.h>\endcode
56
57 */
58
59 /** \name Character classification routines
60
61 These functions perform character classification. They return true or
62 false status depending whether the character passed to the function falls
63 into the function's classification (i.e. isdigit() returns true if its
64 argument is any value '0' though '9', inclusive). If the input is not
65 an unsigned char value, all of this function return false.  */
66
67 /* @{ */
68
69 /** \ingroup ctype
70
71 Checks for an alphanumeric character. It is equivalent to <tt>(isalpha(c) ||
72 isdigit(c))</tt>.  */
73
74 extern int isalnum(int __c) __ATTR_CONST__;
75
76 /** \ingroup ctype
77
78 Checks for an alphabetic character. It is equivalent to <tt>(isupper(c) ||
79 islower(c))</tt>.  */
80
81 extern int isalpha(int __c) __ATTR_CONST__;
82
83 /** \ingroup ctype

```



```

84
85 Checks whether \c c is a 7-bit unsigned char value that fits into the
86 ASCII character set. */
87
88 extern int isascii(int __c) __ATTR_CONST__;
89
90 /** \ingroup ctype
91
92 Checks for a blank character, that is, a space or a tab. */
93
94 extern int isblank(int __c) __ATTR_CONST__;
95
96 /** \ingroup ctype
97
98 Checks for a control character. */
99
100 extern int iscntrl(int __c) __ATTR_CONST__;
101
102 /** \ingroup ctype
103
104 Checks for a digit (0 through 9). */
105
106 extern int isdigit(int __c) __ATTR_CONST__;
107
108 /** \ingroup ctype
109
110 Checks for any printable character except space. */
111
112 extern int isgraph(int __c) __ATTR_CONST__;
113
114 /** \ingroup ctype
115
116 Checks for a lower-case character. */
117
118 extern int islower(int __c) __ATTR_CONST__;
119
120 /** \ingroup ctype
121
122 Checks for any printable character including space. */
123
124 extern int isprint(int __c) __ATTR_CONST__;
125
126 /** \ingroup ctype
127
128 Checks for any printable character which is not a space or an alphanumeric
129 character. */
130
131 extern int ispunct(int __c) __ATTR_CONST__;
132
133 /** \ingroup ctype
134
135 Checks for white-space characters. For the avr-libc library, these are:
136 space, form-feed ('\f'), newline ('\n'), carriage return ('\r'),
137 horizontal tab ('\t'), and vertical tab ('\v'). */
138
139 extern int isspace(int __c) __ATTR_CONST__;
140
141 /** \ingroup ctype
142
143 Checks for an uppercase letter. */
144
145 extern int isupper(int __c) __ATTR_CONST__;
146
147 /** \ingroup ctype
148
149 Checks for a hexadecimal digits, i.e. one of 0 1 2 3 4 5 6 7 8 9 a b c d e
150 f A B C D E F. */
151
152 extern int isxdigit(int __c) __ATTR_CONST__;
153
154 /* @} */
155
156 /** \name Character conversion routines
157
158 This realization permits all possible values of integer argument.
159 The toascii() function clears all highest bits. The tolower() and
160 toupper() functions return an input argument as is, if it is not an
161 unsigned char value. */
162
163 /* @{ */
164
165 /** \ingroup ctype
166
167 Converts \c c to a 7-bit unsigned char value that fits into the ASCII
168 character set, by clearing the high-order bits.
169
170 \warning Many people will be unhappy if you use this function. This

```

```

171 function will convert accented letters into random characters.  */
172
173 extern int toascii(int __c) __ATTR_CONST__;
174
175 /** \ingroup ctype
176
177 Converts the letter \c c to lower case, if possible.  */
178
179 extern int tolower(int __c) __ATTR_CONST__;
180
181 /** \ingroup ctype
182
183 Converts the letter \c c to upper case, if possible.  */
184
185 extern int toupper(int __c) __ATTR_CONST__;
186
187 /* @} */
188
189 #ifdef __cplusplus
190 }
191 #endif
192
193 #endif

```

25.42 errno.h File Reference

Macros

- #define [EDOM](#) 33
- #define [ERANGE](#) 34

Variables

- int [errno](#)

25.43 errno.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002,2007 Marek Michalkiewicz
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id:  errno.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
32
33 #ifndef __ERRNO_H_
34 #define __ERRNO_H_ 1

```

```

35
36 /** \file */
37 /** \defgroup avr_errno <errno.h>: System Errors
38
39 \code #include <errno.h>\endcode
40
41 Some functions in the library set the global variable \c errno when an
42 error occurs. The file, \c <errno.h>, provides symbolic names for various
43 error codes.
44 */
45
46 #ifdef __cplusplus
47 extern "C" {
48 #endif
49
50 /** \ingroup avr_errno
51 \brief Error code for last error encountered by library
52
53 The variable \c errno holds the last error code encountered by
54 a library function. This variable must be cleared by the
55 user prior to calling a library function.
56
57 \warning The \c errno global variable is not safe to use in a threaded or
58 multi-task system. A race condition can occur if a task is interrupted
59 between the call which sets \c error and when the task examines \c
60 errno. If another task changes \c errno during this time, the result will
61 be incorrect for the interrupted task. */
62 extern int errno;
63
64 #ifdef __cplusplus
65 }
66 #endif
67
68 /** \ingroup avr_errno
69 \def EDOM
70
71 Domain error. */
72 #define EDOM 33
73
74 /** \ingroup avr_errno
75 \def ERANGE
76
77 Range error. */
78 #define ERANGE 34
79
80 #ifndef __DOXYGEN__
81
82 /* (((('E'-64)*26+('N'-64))*26+('O'-64))*26+('S'-64))*26+('Y'-64))*26+'S'-64 */
83 #define ENOSYS ((int)(66081697 & 0x7fff))
84
85 /* (((('E'-64)*26+('I'-64))*26+('N'-64))*26+('T'-64))*26+('R'-64) */
86 #define EINTR ((int)(2453066 & 0x7fff))
87
88 #define E2BIG ENOERR
89 #define EACCES ENOERR
90 #define EADDRINUSE ENOERR
91 #define EADDRNOTAVAIL ENOERR
92 #define EAFNOSUPPORT ENOERR
93 #define EAGAIN ENOERR
94 #define EALREADY ENOERR
95 #define EBADF ENOERR
96 #define EBUSY ENOERR
97 #define ECHILD ENOERR
98 #define ECONNABORTED ENOERR
99 #define ECONNREFUSED ENOERR
100 #define ECONNRESET ENOERR
101 #define EDEADLK ENOERR
102 #define EDESTADDRREQ ENOERR
103 #define EEXIST ENOERR
104 #define EFAULT ENOERR
105 #define EFBIG ENOERR
106 #define EHOSTUNREACH ENOERR
107 #define EILSEQ ENOERR
108 #define EINPROGRESS ENOERR
109 #define EINVAL ENOERR
110 #define EIO ENOERR
111 #define EISCONN ENOERR
112 #define EISDIR ENOERR
113 #define ELOOP ENOERR
114 #define EMFILE ENOERR
115 #define EMLINK ENOERR
116 #define EMSGSIZE ENOERR
117 #define ENAMETOOLONG ENOERR
118 #define ENETDOWN ENOERR
119 #define ENETRESET ENOERR
120 #define ENETUNREACH ENOERR
121 #define ENFILE ENOERR

```

```
122 #define ENOBUFS ENOERR
123 #define ENODEV ENOERR
124 #define ENOENT ENOERR
125 #define ENOEXEC ENOERR
126 #define ENOLCK ENOERR
127 #define ENOMEM ENOERR
128 #define ENOMSG ENOERR
129 #define ENOPROTOPT ENOERR
130 #define ENOSPC ENOERR
131 #define ENOTCONN ENOERR
132 #define ENOTDIR ENOERR
133 #define ENOTEMPTY ENOERR
134 #define ENOTSOCK ENOERR
135 #define ENOTTY ENOERR
136 #define ENXIO ENOERR
137 #define EOPNOTSUPP ENOERR
138 #define EPERM ENOERR
139 #define EPIPE ENOERR
140 #define EPROTONOSUPPORT ENOERR
141 #define EPROTOTYPE ENOERR
142 #define EROFS ENOERR
143 #define ESPIPE ENOERR
144 #define ESRCH ENOERR
145 #define ETIMEDOUT ENOERR
146 #define EWOULDBLOCK ENOERR
147 #define EXDEV ENOERR
148
149 /* (((('E'-64)*26+('N'-64))*26+('O'-64))*26+('E'-64))*26+('R'-64))*26+('R'-64) */
150 #define ENOERR ((int)(66072050 & 0xffff))
151
152 #endif /* !__DOXYGEN__ */
153
154 #endif
```

25.44 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 PRId8 "o"`
- `#define PRIdLEAST8 "o"`
- `#define PRIdFAST8 "o"`
- `#define PRIu8 "u"`
- `#define PRIuLEAST8 "u"`
- `#define PRIuFAST8 "u"`
- `#define PRId8 "x"`
- `#define PRIdLEAST8 "x"`
- `#define PRIdFAST8 "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](#) "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`

Typedefs

Far pointers for memory access >64K

- `typedef int32_t int_farptr_t`
- `typedef uint32_t uint_farptr_t`

25.45 inttypes.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2004,2005,2007,2012 Joerg Wunsch
2 Copyright (c) 2005, Carlos Lamas
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15
16 * Neither the name of the copyright holders nor the names of
17 contributors may be used to endorse or promote products derived
18 from this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE. */
31
32 /* $Id: inttypes.h 2302 2012-11-26 10:52:01Z joerg_wunsch $ */
33
34 #ifndef __INTTYPES_H_
35 #define __INTTYPES_H_
36
37 #include <stdint.h>
38
39 /** \file */
40 /** \defgroup avr_inttypes <inttypes.h>: Integer Type conversions
41 \code #include <inttypes.h> \endcode
42
43 This header file includes the exact-width integer definitions from
44 <stdint.h>, and extends them with additional facilities

```

```

45 provided by the implementation.
46
47 Currently, the extensions include two additional integer types
48 that could hold a "far" pointer (i.e. a code pointer that can
49 address more than 64 KB), as well as standard names for all printf
50 and scanf formatting options that are supported by the \ref avr_stdio.
51 As the library does not support the full range of conversion
52 specifiers from ISO 9899:1999, only those conversions that are
53 actually implemented will be listed here.
54
55 The idea behind these conversion macros is that, for each of the
56 types defined by <stdint.h>, a macro will be supplied that portably
57 allows formatting an object of that type in printf() or scanf()
58 operations. Example:
59
60 \code
61 #include <inttypes.h>
62
63 uint8_t smallval;
64 int32_t longval;
65 ...
66 printf("The hexadecimal value of smallval is %" PRIx8
67 ", the decimal value of longval is %" PRId32 ".\n",
68 smallval, longval);
69 \endcode
70 */
71
72 /** \name Far pointers for memory access >64K */
73
74 /*{*/
75 /** \ingroup avr_inttypes
76 signed integer type that can hold a pointer > 64 KB */
77 typedef int32_t int_farptr_t;
78
79 /** \ingroup avr_inttypes
80 unsigned integer type that can hold a pointer > 64 KB */
81 typedef uint32_t uint_farptr_t;
82 /*}*/
83
84 #if !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS)
85
86
87 /** \name macros for printf and scanf format specifiers
88
89 For C++, these are only included if __STDC_LIMIT_MACROS
90 is defined before including <inttypes.h>.
91 */
92
93 /*{*/
94 /** \ingroup avr_inttypes
95 decimal printf format for int8_t */
96 #define PRId8 "d"
97 /** \ingroup avr_inttypes
98 decimal printf format for int_least8_t */
99 #define PRIdLEAST8 "d"
100 /** \ingroup avr_inttypes
101 decimal printf format for int_fast8_t */
102 #define PRIdFAST8 "d"
103
104 /** \ingroup avr_inttypes
105 integer printf format for int8_t */
106 #define PRIi8 "i"
107 /** \ingroup avr_inttypes
108 integer printf format for int_least8_t */
109 #define PRIiLEAST8 "i"
110 /** \ingroup avr_inttypes
111 integer printf format for int_fast8_t */
112 #define PRIiFAST8 "i"
113
114
115 /** \ingroup avr_inttypes
116 decimal printf format for int16_t */
117 #define PRId16 "d"
118 /** \ingroup avr_inttypes
119 decimal printf format for int_least16_t */
120 #define PRIdLEAST16 "d"
121 /** \ingroup avr_inttypes
122 decimal printf format for int_fast16_t */
123 #define PRIdFAST16 "d"
124
125 /** \ingroup avr_inttypes
126 integer printf format for int16_t */
127 #define PRIi16 "i"
128 /** \ingroup avr_inttypes
129 integer printf format for int_least16_t */
130 #define PRIiLEAST16 "i"
131 /** \ingroup avr_inttypes

```

```
132 integer printf format for int_fast16_t */
133 #define      PRIiFAST16      "i"
134
135
136 /** \ingroup avr_inttypes
137 decimal printf format for int32_t */
138 #define      PRId32          "ld"
139 /** \ingroup avr_inttypes
140 decimal printf format for int_least32_t */
141 #define      PRIdLEAST32     "ld"
142 /** \ingroup avr_inttypes
143 decimal printf format for int_fast32_t */
144 #define      PRIdFAST32      "ld"
145
146 /** \ingroup avr_inttypes
147 integer printf format for int32_t */
148 #define      PRIi32          "li"
149 /** \ingroup avr_inttypes
150 integer printf format for int_least32_t */
151 #define      PRIiLEAST32     "li"
152 /** \ingroup avr_inttypes
153 integer printf format for int_fast32_t */
154 #define      PRIiFAST32      "li"
155
156
157 #ifdef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
158
159 #define      PRId64          "lld"
160 #define      PRIdLEAST64     "lld"
161 #define      PRIdFAST64      "lld"
162
163 #define      PRIi64          "lli"
164 #define      PRIiLEAST64     "lli"
165 #define      PRIiFAST64      "lli"
166
167
168 #define      PRIdMAX         "lld"
169 #define      PRIiMAX         "lli"
170
171 #endif
172
173 /** \ingroup avr_inttypes
174 decimal printf format for intptr_t */
175 #define      PRIdPTR         PRId16
176 /** \ingroup avr_inttypes
177 integer printf format for intptr_t */
178 #define      PRIiPTR         PRIi16
179
180 /** \ingroup avr_inttypes
181 octal printf format for uint8_t */
182 #define      PRIo8           "o"
183 /** \ingroup avr_inttypes
184 octal printf format for uint_least8_t */
185 #define      PRIoLEAST8      "o"
186 /** \ingroup avr_inttypes
187 octal printf format for uint_fast8_t */
188 #define      PRIoFAST8       "o"
189
190 /** \ingroup avr_inttypes
191 decimal printf format for uint8_t */
192 #define      PRIu8           "u"
193 /** \ingroup avr_inttypes
194 decimal printf format for uint_least8_t */
195 #define      PRIuLEAST8      "u"
196 /** \ingroup avr_inttypes
197 decimal printf format for uint_fast8_t */
198 #define      PRIuFAST8       "u"
199
200 /** \ingroup avr_inttypes
201 hexadecimal printf format for uint8_t */
202 #define      PRIx8           "x"
203 /** \ingroup avr_inttypes
204 hexadecimal printf format for uint_least8_t */
205 #define      PRIxLEAST8      "x"
206 /** \ingroup avr_inttypes
207 hexadecimal printf format for uint_fast8_t */
208 #define      PRIxFAST8       "x"
209
210 /** \ingroup avr_inttypes
211 uppercase hexadecimal printf format for uint8_t */
212 #define      PRIX8           "X"
213 /** \ingroup avr_inttypes
214 uppercase hexadecimal printf format for uint_least8_t */
215 #define      PRIXLEAST8      "X"
216 /** \ingroup avr_inttypes
217 uppercase hexadecimal printf format for uint_fast8_t */
218 #define      PRIXFAST8       "X"
```



```
219
220
221 /** \ingroup avr_inttypes
222 octal printf format for uint16_t */
223 #define PRIo16 "o"
224 /** \ingroup avr_inttypes
225 octal printf format for uint_least16_t */
226 #define PRIoLEAST16 "o"
227 /** \ingroup avr_inttypes
228 octal printf format for uint_fast16_t */
229 #define PRIoFAST16 "o"
230
231 /** \ingroup avr_inttypes
232 decimal printf format for uint16_t */
233 #define PRIu16 "u"
234 /** \ingroup avr_inttypes
235 decimal printf format for uint_least16_t */
236 #define PRIuLEAST16 "u"
237 /** \ingroup avr_inttypes
238 decimal printf format for uint_fast16_t */
239 #define PRIuFAST16 "u"
240
241 /** \ingroup avr_inttypes
242 hexadecimal printf format for uint16_t */
243 #define PRIx16 "x"
244 /** \ingroup avr_inttypes
245 hexadecimal printf format for uint_least16_t */
246 #define PRIxLEAST16 "x"
247 /** \ingroup avr_inttypes
248 hexadecimal printf format for uint_fast16_t */
249 #define PRIxFAST16 "x"
250
251 /** \ingroup avr_inttypes
252 uppercase hexadecimal printf format for uint16_t */
253 #define PRIX16 "X"
254 /** \ingroup avr_inttypes
255 uppercase hexadecimal printf format for uint_least16_t */
256 #define PRIXLEAST16 "X"
257 /** \ingroup avr_inttypes
258 uppercase hexadecimal printf format for uint_fast16_t */
259 #define PRIXFAST16 "X"
260
261
262 /** \ingroup avr_inttypes
263 octal printf format for uint32_t */
264 #define PRIo32 "lo"
265 /** \ingroup avr_inttypes
266 octal printf format for uint_least32_t */
267 #define PRIoLEAST32 "lo"
268 /** \ingroup avr_inttypes
269 octal printf format for uint_fast32_t */
270 #define PRIoFAST32 "lo"
271
272 /** \ingroup avr_inttypes
273 decimal printf format for uint32_t */
274 #define PRIu32 "lu"
275 /** \ingroup avr_inttypes
276 decimal printf format for uint_least32_t */
277 #define PRIuLEAST32 "lu"
278 /** \ingroup avr_inttypes
279 decimal printf format for uint_fast32_t */
280 #define PRIuFAST32 "lu"
281
282 /** \ingroup avr_inttypes
283 hexadecimal printf format for uint32_t */
284 #define PRIx32 "lx"
285 /** \ingroup avr_inttypes
286 hexadecimal printf format for uint_least32_t */
287 #define PRIxLEAST32 "lx"
288 /** \ingroup avr_inttypes
289 hexadecimal printf format for uint_fast32_t */
290 #define PRIxFAST32 "lx"
291
292 /** \ingroup avr_inttypes
293 uppercase hexadecimal printf format for uint32_t */
294 #define PRIX32 "lX"
295 /** \ingroup avr_inttypes
296 uppercase hexadecimal printf format for uint_least32_t */
297 #define PRIXLEAST32 "lX"
298 /** \ingroup avr_inttypes
299 uppercase hexadecimal printf format for uint_fast32_t */
300 #define PRIXFAST32 "lX"
301
302
303 #ifdef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
304
305 #define PRIo64 "llo"
```

```

306 #define      PRIoLEAST64      "llo"
307 #define      PRIoFAST64       "llo"
308
309 #define      PRIu64             "llu"
310 #define      PRIuLEAST64       "llu"
311 #define      PRIuFAST64        "llu"
312
313 #define      PRIx64             "llx"
314 #define      PRIxLEAST64       "llx"
315 #define      PRIxFAST64        "llx"
316
317 #define      PRIX64             "llX"
318 #define      PRIXLEAST64       "llX"
319 #define      PRIXFAST64        "llX"
320
321 #define      PRIoMAX            "llo"
322 #define      PRIuMAX            "llu"
323 #define      PRIxMAX            "llx"
324 #define      PRIXMAX            "llX"
325
326 #endif
327
328 /** \ingroup avr_inttypes
329 octal printf format for uintptr_t */
330 #define      PRIoPTR            PRIo16
331 /** \ingroup avr_inttypes
332 decimal printf format for uintptr_t */
333 #define      PRIuPTR            PRIu16
334 /** \ingroup avr_inttypes
335 hexadecimal printf format for uintptr_t */
336 #define      PRIxPTR            PRIx16
337 /** \ingroup avr_inttypes
338 uppercase hexadecimal printf format for uintptr_t */
339 #define      PRIXPTR            PRIX16
340
341
342 /** \ingroup avr_inttypes
343 decimal scanf format for int8_t */
344 #define      SCNd8              "hhd"
345 /** \ingroup avr_inttypes
346 decimal scanf format for int_least8_t */
347 #define      SCNdLEAST8        "hhd"
348 /** \ingroup avr_inttypes
349 decimal scanf format for int_fast8_t */
350 #define      SCNdFAST8         "hhd"
351
352 /** \ingroup avr_inttypes
353 generic-integer scanf format for int8_t */
354 #define      SCNi8              "hhi"
355 /** \ingroup avr_inttypes
356 generic-integer scanf format for int_least8_t */
357 #define      SCNiLEAST8        "hhi"
358 /** \ingroup avr_inttypes
359 generic-integer scanf format for int_fast8_t */
360 #define      SCNiFAST8         "hhi"
361
362
363 /** \ingroup avr_inttypes
364 decimal scanf format for int16_t */
365 #define      SCNd16             "d"
366 /** \ingroup avr_inttypes
367 decimal scanf format for int_least16_t */
368 #define      SCNdLEAST16        "d"
369 /** \ingroup avr_inttypes
370 decimal scanf format for int_fast16_t */
371 #define      SCNdFAST16         "d"
372
373 /** \ingroup avr_inttypes
374 generic-integer scanf format for int16_t */
375 #define      SCNi16             "i"
376 /** \ingroup avr_inttypes
377 generic-integer scanf format for int_least16_t */
378 #define      SCNiLEAST16        "i"
379 /** \ingroup avr_inttypes
380 generic-integer scanf format for int_fast16_t */
381 #define      SCNiFAST16         "i"
382
383
384 /** \ingroup avr_inttypes
385 decimal scanf format for int32_t */
386 #define      SCNd32             "ld"
387 /** \ingroup avr_inttypes
388 decimal scanf format for int_least32_t */
389 #define      SCNdLEAST32        "ld"
390 /** \ingroup avr_inttypes
391 decimal scanf format for int_fast32_t */
392 #define      SCNdFAST32         "ld"

```

```

393
394 /** \ingroup avr_inttypes
395 generic-integer scanf format for int32_t */
396 #define SCNi32 "li"
397 /** \ingroup avr_inttypes
398 generic-integer scanf format for int_least32_t */
399 #define SCNiLEAST32 "li"
400 /** \ingroup avr_inttypes
401 generic-integer scanf format for int_fast32_t */
402 #define SCNiFAST32 "li"
403
404
405 #ifdef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
406
407 #define SCNd64 "lld"
408 #define SCNdLEAST64 "lld"
409 #define SCNdFAST64 "lld"
410
411 #define SCNi64 "lli"
412 #define SCNiLEAST64 "lli"
413 #define SCNiFAST64 "lli"
414
415 #define SCNdMAX "lld"
416 #define SCNiMAX "lli"
417
418 #endif
419
420 /** \ingroup avr_inttypes
421 decimal scanf format for intptr_t */
422 #define SCNdPTR SCNd16
423 /** \ingroup avr_inttypes
424 generic-integer scanf format for intptr_t */
425 #define SCNiPTR SCNi16
426
427 /** \ingroup avr_inttypes
428 octal scanf format for uint8_t */
429 #define SCNo8 "hho"
430 /** \ingroup avr_inttypes
431 octal scanf format for uint_least8_t */
432 #define SCNoLEAST8 "hho"
433 /** \ingroup avr_inttypes
434 octal scanf format for uint_fast8_t */
435 #define SCNoFAST8 "hho"
436
437 /** \ingroup avr_inttypes
438 decimal scanf format for uint8_t */
439 #define SCNu8 "hhu"
440 /** \ingroup avr_inttypes
441 decimal scanf format for uint_least8_t */
442 #define SCNuLEAST8 "hhu"
443 /** \ingroup avr_inttypes
444 decimal scanf format for uint_fast8_t */
445 #define SCNuFAST8 "hhu"
446
447 /** \ingroup avr_inttypes
448 hexadecimal scanf format for uint8_t */
449 #define SCNx8 "hhx"
450 /** \ingroup avr_inttypes
451 hexadecimal scanf format for uint_least8_t */
452 #define SCNxLEAST8 "hhx"
453 /** \ingroup avr_inttypes
454 hexadecimal scanf format for uint_fast8_t */
455 #define SCNxFAST8 "hhx"
456
457 /** \ingroup avr_inttypes
458 octal scanf format for uint16_t */
459 #define SCNo16 "o"
460 /** \ingroup avr_inttypes
461 octal scanf format for uint_least16_t */
462 #define SCNoLEAST16 "o"
463 /** \ingroup avr_inttypes
464 octal scanf format for uint_fast16_t */
465 #define SCNoFAST16 "o"
466
467 /** \ingroup avr_inttypes
468 decimal scanf format for uint16_t */
469 #define SCNu16 "u"
470 /** \ingroup avr_inttypes
471 decimal scanf format for uint_least16_t */
472 #define SCNuLEAST16 "u"
473 /** \ingroup avr_inttypes
474 decimal scanf format for uint_fast16_t */
475 #define SCNuFAST16 "u"
476
477 /** \ingroup avr_inttypes
478 hexadecimal scanf format for uint16_t */
479 #define SCNx16 "x"

```

```

480 /** \ingroup avr_inttypes
481 hexadecimal scanf format for uint_least16_t */
482 #define      SCNxLEAST16      "x"
483 /** \ingroup avr_inttypes
484 hexadecimal scanf format for uint_fast16_t */
485 #define      SCNxFAST16      "x"
486
487
488 /** \ingroup avr_inttypes
489 octal scanf format for uint32_t */
490 #define      SCNo32      "lo"
491 /** \ingroup avr_inttypes
492 octal scanf format for uint_least32_t */
493 #define      SCNoLEAST32      "lo"
494 /** \ingroup avr_inttypes
495 octal scanf format for uint_fast32_t */
496 #define      SCNoFAST32      "lo"
497
498 /** \ingroup avr_inttypes
499 decimal scanf format for uint32_t */
500 #define      SCNu32      "lu"
501 /** \ingroup avr_inttypes
502 decimal scanf format for uint_least32_t */
503 #define      SCNuLEAST32      "lu"
504 /** \ingroup avr_inttypes
505 decimal scanf format for uint_fast32_t */
506 #define      SCNuFAST32      "lu"
507
508 /** \ingroup avr_inttypes
509 hexadecimal scanf format for uint32_t */
510 #define      SCNx32      "lx"
511 /** \ingroup avr_inttypes
512 hexadecimal scanf format for uint_least32_t */
513 #define      SCNxLEAST32      "lx"
514 /** \ingroup avr_inttypes
515 hexadecimal scanf format for uint_fast32_t */
516 #define      SCNxFAST32      "lx"
517
518
519 #ifdef __avr_libc_does_not_implement_long_long_in_printf_or_scanf
520
521 #define      SCNo64      "llo"
522 #define      SCNoLEAST64      "llo"
523 #define      SCNoFAST64      "llo"
524
525 #define      SCNu64      "llu"
526 #define      SCNuLEAST64      "llu"
527 #define      SCNuFAST64      "llu"
528
529 #define      SCNx64      "llx"
530 #define      SCNxLEAST64      "llx"
531 #define      SCNxFAST64      "llx"
532
533 #define      SCNoMAX      "llo"
534 #define      SCNuMAX      "llu"
535 #define      SCNxMAX      "llx"
536
537 #endif
538
539 /** \ingroup avr_inttypes
540 octal scanf format for uintptr_t */
541 #define      SCNoPTR      SCNo16
542 /** \ingroup avr_inttypes
543 decimal scanf format for uintptr_t */
544 #define      SCNuPTR      SCNu16
545 /** \ingroup avr_inttypes
546 hexadecimal scanf format for uintptr_t */
547 #define      SCNxPTR      SCNx16
548
549 /* @} */
550
551
552 #endif /* !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS) */
553
554
555 #endif /* __INTTYPES_H_ */

```

25.46 math.h File Reference

Macros

- #define [M_E](#) 2.7182818284590452354

- #define `M_LOG2E` 1.4426950408889634074 /* log₂ e */
- #define `M_LOG10E` 0.43429448190325182765 /* log₁₀ e */
- #define `M_LN2` 0.69314718055994530942 /* log_e 2 */
- #define `M_LN10` 2.30258509299404568402 /* log_e 10 */
- #define `M_PI` 3.14159265358979323846 /* pi */
- #define `M_PI_2` 1.57079632679489661923 /* pi/2 */
- #define `M_PI_4` 0.78539816339744830962 /* pi/4 */
- #define `M_1_PI` 0.31830988618379067154 /* 1/pi */
- #define `M_2_PI` 0.63661977236758134308 /* 2/pi */
- #define `M_2_SQRTPI` 1.12837916709551257390 /* 2/sqrt(pi) */
- #define `M_SQRT2` 1.41421356237309504880 /* sqrt(2) */
- #define `M_SQRT1_2` 0.70710678118654752440 /* 1/sqrt(2) */
- #define `NAN` __builtin_nan("")
- #define `INFINITY` __builtin_inf()
- #define `__ASM_ALIAS(x)` __asm(#x)

Functions

- float `cosf` (float __x)
- double `cos` (double __x) __ASM_ALIAS(cosf)
- float `sinf` (float __x)
- double `sin` (double __x) __ASM_ALIAS(sinf)
- float `tanf` (float __x)
- double `tan` (double __x) __ASM_ALIAS(tanf)
- static float `fabsf` (float __x)
- static double `fabs` (double __x)
- float `fmodf` (float __x, float __y)
- double `fmod` (double __x, double __y) __ASM_ALIAS(fmodf)
- float `modff` (float __x, float *__iptr)
- double `modf` (double __x, double *__iptr) __ASM_ALIAS(modff)
- float `sqrtf` (float __x)
- double `sqrt` (double __x) __ASM_ALIAS(sqrtf)
- float `cbrtf` (float __x)
- double `cbrt` (double __x) __ASM_ALIAS(cbrtf)
- float `hypotf` (float __x, float __y)
- double `hypot` (double __x, double __y) __ASM_ALIAS(hypotf)
- float `squaref` (float __x)
- double `square` (double __x) __ASM_ALIAS(squaref)
- float `floorf` (float __x)
- double `floor` (double __x) __ASM_ALIAS(floorf)
- float `ceilf` (float __x)
- double `ceil` (double __x) __ASM_ALIAS(ceilf)
- float `frexpf` (float __x, int *__pexp)
- double `frexp` (double __x, int *__pexp) __ASM_ALIAS(frexpf)
- float `ldexpf` (float __x, int __exp)
- double `ldexp` (double __x, int __exp) __ASM_ALIAS(ldexpf)
- float `expf` (float __x)
- double `exp` (double __x) __ASM_ALIAS(expf)
- float `coshf` (float __x)
- double `cosh` (double __x) __ASM_ALIAS(coshf)
- float `sinhf` (float __x)
- double `sinh` (double __x) __ASM_ALIAS(sinhf)
- float `tanhf` (float __x)
- double `tanh` (double __x) __ASM_ALIAS(tanhf)

- float `acosf` (float __x)
- double `acos` (double __x) __ASM_ALIAS(`acosf`)
- float `asinf` (float __x)
- double `asin` (double __x) __ASM_ALIAS(`asinf`)
- float `atanf` (float __x)
- double `atan` (double __x) __ASM_ALIAS(`atanf`)
- float `atan2f` (float __y, float __x)
- double `atan2` (double __y, double __x) __ASM_ALIAS(`atan2f`)
- float `logf` (float __x)
- double `log` (double __x) __ASM_ALIAS(`logf`)
- float `log10f` (float __x)
- double `log10` (double __x) __ASM_ALIAS(`log10f`)
- float `powf` (float __x, float __y)
- double `pow` (double __x, double __y) __ASM_ALIAS(`powf`)
- int `isnanf` (float __x)
- int `isnan` (double __x) __ASM_ALIAS(`isnanf`)
- int `isinf` (float __x)
- int `isinf` (double __x) __ASM_ALIAS(`isinf`)
- static int `isfinitef` (float __x)
- static int `isfinite` (double __x)
- static float `copysignf` (float __x, float __y)
- static double `copysign` (double __x, double __y)
- int `signbitf` (float __x)
- int `signbit` (double __x) __ASM_ALIAS(`signbitf`)
- float `fdimf` (float __x, float __y)
- double `fdim` (double __x, double __y) __ASM_ALIAS(`fdimf`)
- float `fmaf` (float __x, float __y, float __z)
- double `fma` (double __x, double __y, double __z) __ASM_ALIAS(`fmaf`)
- float `fmaxf` (float __x, float __y)
- double `fmax` (double __x, double __y) __ASM_ALIAS(`fmaxf`)
- float `fminf` (float __x, float __y)
- double `fmin` (double __x, double __y) __ASM_ALIAS(`fminf`)
- float `truncf` (float __x)
- double `trunc` (double __x) __ASM_ALIAS(`truncf`)
- float `roundf` (float __x)
- double `round` (double __x) __ASM_ALIAS(`roundf`)
- long `lroundf` (float __x)
- long `lround` (double __x) __ASM_ALIAS(`lroundf`)
- long `lrintf` (float __x)
- long `lrint` (double __x) __ASM_ALIAS(`lrintf`)

25.46.1 Macro Definition Documentation

25.46.1.1 INFINITY `#define INFINITY __builtin_inf()`

INFINITY constant.

25.46.1.2 M_1_PI `#define M_1_PI 0.31830988618379067154 /* 1/pi */`

The constant $1/\pi$.

25.46.1.3 M_2_PI `#define M_2_PI 0.63661977236758134308 /* 2/pi */`

The constant $2/\pi$.

25.46.1.4 M_2_SQRTPI `#define M_2_SQRTPI 1.12837916709551257390 /* 2/sqrt(pi) */`

The constant $2/\sqrt{\pi}$.

25.46.1.5 M_LN10 `#define M_LN10 2.30258509299404568402 /* log_e 10 */`

The natural logarithm of the 10.

25.46.1.6 M_LN2 `#define M_LN2 0.69314718055994530942 /* log_e 2 */`

The natural logarithm of the 2.

25.46.1.7 M_LOG10E `#define M_LOG10E 0.43429448190325182765 /* log_10 e */`

The logarithm of the e to base 10.

25.46.1.8 M_LOG2E `#define M_LOG2E 1.4426950408889634074 /* log_2 e */`

The logarithm of the e to base 2.

25.46.1.9 M_PI `#define M_PI 3.14159265358979323846 /* pi */`

The constant π .

25.46.1.10 M_PI_2 `#define M_PI_2 1.57079632679489661923 /* pi/2 */`

The constant $\pi/2$.

25.46.1.11 M_PI_4 `#define M_PI_4 0.78539816339744830962 /* pi/4 */`

The constant $\pi/4$.

25.46.1.12 M_SQRT1_2 `#define M_SQRT1_2 0.70710678118654752440 /* 1/sqrt(2) */`

The constant $1/\sqrt{2}$.

25.46.1.13 M_SQRT2 `#define M_SQRT2 1.41421356237309504880 /* sqrt(2) */`

The square root of 2.

25.46.1.14 NAN `#define NAN __builtin_nan("")`

NAN constant.

25.46.2 Function Documentation

25.46.2.1 acos() `double acos (
double __x)`

The alias for [acosf\(\)](#).

25.46.2.2 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 $[-1, +1]$.

25.46.2.3 asin() `double asin (
double __x)`

The alias for [asinf\(\)](#).

25.46.2.4 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 $[-1, +1]$.

25.46.2.5 atan() `double atan (
double __x)`

The alias for [atanf\(\)](#).

25.46.2.6 atan2() `double atan2 (
double __y,
double __x)`

The alias for [atan2f\(\)](#).

25.46.2.7 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.

25.46.2.8 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.

25.46.2.9 cbrt() `double cbrt (`
`double __x)`

The alias for [cbrtf\(\)](#).

25.46.2.10 cbrtf() `float cbrtf (`
`float __x)`

The [cbrt\(\)](#) function returns the cube root of `__x`.

25.46.2.11 ceil() `double ceil (`
`double __x)`

The alias for [ceilf\(\)](#).

25.46.2.12 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.

25.46.2.13 copysignf() `static float copysignf (`
`float __x,`
`float __y) [static]`

The [copysignf\(\)](#) function returns `__x` but with the sign of `__y`. They work even if `__x` or `__y` are NaN or zero.

25.46.2.14 cos() `double cos (`
`double __x)`

The alias for [cosf\(\)](#).

25.46.2.15 cosf() `float cosf (`
`float __x)`

The [cosf\(\)](#) function returns the cosine of `__x`, measured in radians.

25.46.2.16 cosh() `double cosh (`
`double __x)`

The alias for [coshf\(\)](#).

25.46.2.17 coshf() `float coshf (`
 `float __x)`

The [coshf\(\)](#) function returns the hyperbolic cosine of `__x`.

25.46.2.18 exp() `double exp (`
 `double __x)`

The alias for [expf\(\)](#).

25.46.2.19 expf() `float expf (`
 `float __x)`

The [expf\(\)](#) function returns the exponential value of `__x`.

25.46.2.20 fabsf() `static float fabsf (`
 `float __x) [static]`

The [fabsf\(\)](#) function computes the absolute value of a floating-point number `__x`.

25.46.2.21 fdim() `double fdim (`
 `double __x,`
 `double __y)`

The alias for [fdimf\(\)](#).

25.46.2.22 fdimf() `float fdimf (`
 `float __x,`
 `float __y)`

The [fdimf\(\)](#) function returns $\maxf(__x - __y, 0)$. If `__x` or `__y` or both are NaN, NaN is returned.

25.46.2.23 floor() `double floor (`
 `double __x)`

The alias for [floorf\(\)](#).

25.46.2.24 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.

25.46.2.25 fma() `double fma (`
 `double __x,`
 `double __y,`
 `double __z)`

The alias for [fmaf\(\)](#).

25.46.2.26 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.

25.46.2.27 fmax() `double fmax (`
 `double __x,`
 `double __y)`

The alias for `fmaxf()`.

25.46.2.28 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.

25.46.2.29 fmin() `double fmin (`
 `double __x,`
 `double __y)`

The alias for `fminf()`.

25.46.2.30 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.

25.46.2.31 fmod() `double fmod (`
 `double __x,`
 `double __y)`

The alias for `fmodf()`.

25.46.2.32 fmodf() `float fmodf (`
 `float __x,`
 `float __y)`

The function `fmodf()` returns the floating-point remainder of `__x / __y`.

25.46.2.33 frexp() `double frexp (`
 `double __x,`
 `int * __pexp)`

The alias for `frexpf()`.

25.46.2.34 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 storing the exponent.

25.46.2.35 hypot() `double hypot (`
`double __x,`
`double __y)`

The alias for `hypotf()`.

25.46.2.36 hypotf() `float hypotf (`
`float __x,`
`float __y)`

The `hypotf()` function returns $\sqrt{\text{__x}*\text{__x} + \text{__y}*\text{__y}}$. This is the length of the hypotenuse of a right triangle with sides of length `__x` and `__y`, or the distance of the point $(\text{__x}, \text{__y})$ 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.

25.46.2.37 isfinite() `static int isfinite (`
`double __x) [static]`

Parameters

<code>__x</code>	The alias for <code>isfinitef()</code> .
------------------	--

25.46.2.38 isfinitef() `static int isfinitef (`
`float __x) [static]`

The `isfinitef()` function returns a nonzero value if `__x` is finite: not plus or minus infinity, and not NaN.

25.46.2.39 isinf() `int isinf (`
`double __x)`

The alias for `isinf()`.

25.46.2.40 `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.

Note

The GCC 4.3 can replace this function with inline code that returns the 1 value for both infinities (gcc bug #35509).

25.46.2.41 `isnan()` `int isnan (`
 `double __x)`

The alias for `isnanf()`.

25.46.2.42 `isnanf()` `int isnanf (`
 `float __x)`

The function `isnanf()` returns 1 if the argument `__x` represents a "not-a-number" (NaN) object, otherwise 0.

25.46.2.43 `ldexp()` `double ldexp (`
 `double __x,`
 `int __exp)`

The alias for `ldexpf()`.

25.46.2.44 `ldexpf()` `float ldexpf (`
 `float __x,`
 `int __exp)`

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 `__exp`.

25.46.2.45 `log()` `double log (`
 `double __x)`

The alias for `logf()`.

25.46.2.46 `log10()` `double log10 (`
 `double __x)`

The alias for `log10f()`.

25.46.2.47 `log10f()` `float log10f (`
 `float __x)`

The `log10f()` function returns the logarithm of argument `__x` to base 10.

25.46.2.48 `logf()` `float logf (`
`float __x)`

The `logf()` function returns the natural logarithm of argument `__x`.

25.46.2.49 `lrint()` `long lrint (`
`double __x)`

The alias for `lrintf()`.

25.46.2.50 `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).

25.46.2.51 `lround()` `long lround (`
`double __x)`

The alias for `lroundf()`.

25.46.2.52 `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 `roundf()` 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).

25.46.2.53 `modf()` `double modf (`
`double __x,`
`double * __iptr)`

An alias for `modff()`.

25.46.2.54 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 double 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.

25.46.2.55 pow() `double pow (`
 `double __x,`
 `double __y)`

The alias for `powf()`.

25.46.2.56 powf() `float powf (`
 `float __x,`
 `float __y)`

The function `powf()` returns the value of `__x` to the exponent `__y`.

25.46.2.57 round() `double round (`
 `double __x)`

The alias for `roundf()`.

25.46.2.58 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.

25.46.2.59 signbit() `int signbit (`
 `double __x)`

The alias for `signbitf()`.

25.46.2.60 **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.

25.46.2.61 **sin()** `double sin (`
`double __x)`

The alias for **sinf()**.

25.46.2.62 **sinf()** `float sinf (`
`float __x)`

The **sinf()** function returns the sine of `__x`, measured in radians.

25.46.2.63 **sinh()** `double sinh (`
`double __x)`

The alias for **sinhf()**.

25.46.2.64 **sinhf()** `float sinhf (`
`float __x)`

The **sinhf()** function returns the hyperbolic sine of `__x`.

25.46.2.65 **sqrt()** `double sqrt (`
`double __x)`

An alias for **sqrtf()**.

25.46.2.66 **sqrtf()** `float sqrtf (`
`float __x)`

The **sqrtf()** function returns the non-negative square root of `__x`.

25.46.2.67 **square()** `double square (`
`double __x)`

The alias for **squaref()**.

25.46.2.68 **squaref()** `float squaref (`
`float __x)`

The function **squaref()** returns `__x * __x`.

Note

This function does not belong to the C standard definition.

25.46.2.69 tan() double tan (
double __x)

The alias for [tanf\(\)](#).

25.46.2.70 tanf() float tanf (
float __x)

The [tanf\(\)](#) function returns the tangent of __x, measured in radians.

25.46.2.71 tanh() double tanh (
double __x)

The alias for [tanhf\(\)](#).

25.46.2.72 tanhf() float tanhf (
float __x)

The [tanhf\(\)](#) function returns the hyperbolic tangent of __x.

25.46.2.73 trunc() double trunc (
double __x)

The alias for [truncf\(\)](#).

25.46.2.74 truncf() float truncf (
float __x)

The [truncf\(\)](#) function rounds __x to the nearest integer not larger in absolute value.

25.47 math.h

[Go to the documentation of this file.](#)

```
1 /* Copyright (c) 2002,2007-2009 Michael Stumpf
2
3 Portions of documentation Copyright (c) 1990 - 1994
4 The Regents of the University of California.
5
6 All rights reserved.
7
8 Redistribution and use in source and binary forms, with or without
9 modification, are permitted provided that the following conditions are met:
10
11 * Redistributions of source code must retain the above copyright
12 notice, this list of conditions and the following disclaimer.
13
14 * Redistributions in binary form must reproduce the above copyright
15 notice, this list of conditions and the following disclaimer in
16 the documentation and/or other materials provided with the
17 distribution.
18
19 * Neither the name of the copyright holders nor the names of
20 contributors may be used to endorse or promote products derived
21 from this software without specific prior written permission.
22
23 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
24 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
25 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
26 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
27 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

28 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
29 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
30 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
31 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
32 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
33 POSSIBILITY OF SUCH DAMAGE. */
34
35 /* $Id: math.h 2554 2021-05-20 22:22:24Z joerg_wunsch $ */
36
37 /*
38 math.h - mathematical functions
39
40 Author : Michael Stumpf
41 Michael.Stumpf@t-online.de
42
43 __ATTR_CONST__ added by marekm@linux.org.pl for functions
44 that "do not examine any values except their arguments, and have
45 no effects except the return value", for better optimization by gcc.
46 */
47
48 #ifndef __MATH_H
49 #define __MATH_H
50
51 /** \file */
52 /** \defgroup avr_math <math.h>: Mathematics
53 \code #include <math.h> \endcode
54
55 This header file declares basic mathematics constants and
56 functions.
57
58 \par Notes:
59 - In order to access the functions declared herein, it is usually
60 also required to additionally link against the library \c libm.a.
61 See also the related \ref faq_libm "FAQ entry".
62 - Math functions do not raise exceptions and do not change the
63 \c errno variable. Therefore the majority of them are declared
64 with const attribute, for better optimization by GCC. */
65
66
67 /** \ingroup avr_math */
68 /*{*}
69
70 /** The constant \a e. */
71 #define M_E 2.7182818284590452354
72
73 /** The logarithm of the \a e to base 2. */
74 #define M_LOG2E 1.4426950408889634074 /* log_2 e */
75
76 /** The logarithm of the \a e to base 10. */
77 #define M_LOG10E 0.43429448190325182765 /* log_10 e */
78
79 /** The natural logarithm of the 2. */
80 #define M_LN2 0.69314718055994530942 /* log_e 2 */
81
82 /** The natural logarithm of the 10. */
83 #define M_LN10 2.30258509299404568402 /* log_e 10 */
84
85 /** The constant \a pi. */
86 #define M_PI 3.14159265358979323846 /* pi */
87
88 /** The constant \a pi/2. */
89 #define M_PI_2 1.57079632679489661923 /* pi/2 */
90
91 /** The constant \a pi/4. */
92 #define M_PI_4 0.78539816339744830962 /* pi/4 */
93
94 /** The constant \a 1/pi. */
95 #define M_1_PI 0.31830988618379067154 /* 1/pi */
96
97 /** The constant \a 2/pi. */
98 #define M_2_PI 0.63661977236758134308 /* 2/pi */
99
100 /** The constant \a 2/sqrt(pi). */
101 #define M_2_SQRTPI 1.12837916709551257390 /* 2/sqrt(pi) */
102
103 /** The square root of 2. */
104 #define M_SQRT2 1.41421356237309504880 /* sqrt(2) */
105
106 /** The constant \a 1/sqrt(2). */
107 #define M_SQRT1_2 0.70710678118654752440 /* 1/sqrt(2) */
108
109 /** NAN constant. */
110 #define NAN __builtin_nan("")
111
112 /** INFINITY constant. */
113 #define INFINITY __builtin_inf()
114

```

```

115
116 #ifndef __ATTR_CONST__
117 # define __ATTR_CONST__ __attribute__((__const__))
118 #endif
119
120 #if __SIZEOF_DOUBLE__ == __SIZEOF_FLOAT__
121 /* In order to provide aliases for double functions in the case where
122 double = float, use declarations with according assembler name.
123 That way:
124 1) We do *NOT* use macros like
125 #define sin sinf
126 because that would interfere with C++. For example, if some class
127 would implement a method 'sin', or if there was polymorphism for
128 a function 'sin', then we would silently rename these to 'sinf'.
129 2) We have proper prototypes, both for 'sin' and for 'sinf'.
130 3) It is zero-overhead. */
131 #define __ASM_ALIAS(x) __asm(#x)
132 #else
133 /* double != float: Provide double prototypes. */
134 #define __ASM_ALIAS(x) /* empty */
135 #endif
136
137 #ifdef __cplusplus
138 extern "C" {
139 #endif
140
141 /**
142 The cosf() function returns the cosine of \a __x, measured in radians.
143 */
144 __ATTR_CONST__ extern float cosf (float __x);
145 __ATTR_CONST__ extern double cos (double __x) __ASM_ALIAS(cosf);    /**< The alias for cosf(). */
146
147 /**
148 The sinf() function returns the sine of \a __x, measured in radians.
149 */
150 __ATTR_CONST__ extern float sinf (float __x);
151 __ATTR_CONST__ extern double sin (double __x) __ASM_ALIAS(sinf);    /**< The alias for sinf(). */
152
153 /**
154 The tanf() function returns the tangent of \a __x, measured in radians.
155 */
156 __ATTR_CONST__ extern float tanf (float __x);
157 __ATTR_CONST__ extern double tan (double __x) __ASM_ALIAS(tanf);    /**< The alias for tanf(). */
158
159 /**
160 The fabsf() function computes the absolute value of a floating-point
161 number \a __x.
162 */
163 static inline float fabsf (float __x)
164 {
165     return __builtin_fabsf (__x);
166 }
167
168 static inline double fabs (double __x)
169 {
170     return __builtin_fabs (__x);
171 }
172
173 /**
174 The function fmodf() returns the floating-point remainder of <em>__x /
175 __y</em>.
176 */
177 __ATTR_CONST__ extern float fmodf (float __x, float __y);
178 __ATTR_CONST__ extern double fmod (double __x, double __y) __ASM_ALIAS(fmodf);    /**< The alias for
179 fmodf(). */
180
181 /**
182 The modff() function breaks the argument \a __x into integral and
183 fractional parts, each of which has the same sign as the argument.
184 It stores the integral part as a double in the object pointed to by
185 \a __iptr.
186
187 The modff() function returns the signed fractional part of \a __x.
188
189 \note This implementation skips writing by zero pointer. However,
190 the GCC 4.3 can replace this function with inline code that does not
191 permit to use NULL address for the avoiding of storing.
192 */
193 extern float modff (float __x, float *__iptr);
194
195 /** An alias for modff(). */
196 extern double modf (double __x, double *__iptr) __ASM_ALIAS(modff);
197
198 /**
199 The sqrtf() function returns the non-negative square root of \a __x.
200 */
201 __ATTR_CONST__ extern float sqrtf (float __x);

```

```

201
202 /** An alias for sqrtf(). */
203 __ATTR_CONST__ extern double sqrt (double __x) __ASM_ALIAS(sqrtf);
204
205 /**
206 The cbrt() function returns the cube root of \a __x.
207 */
208 __ATTR_CONST__ extern float cbrtf (float __x);
209 __ATTR_CONST__ extern double cbrt (double __x) __ASM_ALIAS(cbrtf);    /**< The alias for cbrtf(). */
210
211 /**
212 The hypotf() function returns <em>sqrtf(__x*__x + __y*__y)</em>. This
213 is the length of the hypotenuse of a right triangle with sides of
214 length \a __x and \a __y, or the distance of the point (\a __x, \a
215 __y) from the origin. Using this function instead of the direct
216 formula is wise, since the error is much smaller. No underflow with
217 small \a __x and \a __y. No overflow if result is in range.
218 */
219 __ATTR_CONST__ extern float hypotf (float __x, float __y);
220 __ATTR_CONST__ extern double hypot (double __x, double __y) __ASM_ALIAS(hypotf);    /**< The alias for
hypotf(). */
221
222 /**
223 The function squaref() returns <em>__x * __x</em>.
224
225 \note This function does not belong to the C standard definition.
226 */
227 __ATTR_CONST__ extern float squaref (float __x);
228 __ATTR_CONST__ extern double square (double __x) __ASM_ALIAS(squaref);    /**< The alias for squaref().
*/
229
230 /**
231 The floorf() function returns the largest integral value less than or
232 equal to \a __x, expressed as a floating-point number.
233 */
234 __ATTR_CONST__ extern float floorf (float __x);
235 __ATTR_CONST__ extern double floor (double __x) __ASM_ALIAS(floorf);    /**< The alias for floorf().
*/
236
237 /**
238 The ceilf() function returns the smallest integral value greater than
239 or equal to \a __x, expressed as a floating-point number.
240 */
241 __ATTR_CONST__ extern float ceilf (float __x);
242 __ATTR_CONST__ extern double ceil (double __x) __ASM_ALIAS(ceilf);    /**< The alias for ceilf(). */
243
244 /**
245 The frexpf() function breaks a floating-point number into a normalized
246 fraction and an integral power of 2. It stores the integer in the \c
247 int object pointed to by \a __pexp.
248
249 If \a __x is a normal float point number, the frexpf() function
250 returns the value \c v, such that \c v has a magnitude in the
251 interval [1/2, 1) or zero, and \a __x equals \c v times 2 raised to
252 the power \a __pexp. If \a __x is zero, both parts of the result are
253 zero. If \a __x is not a finite number, the frexpf() returns \a __x as
254 is and stores 0 by \a __pexp.
255
256 \note This implementation permits a zero pointer as a directive to
257 skip a storing the exponent.
258 */
259 __ATTR_CONST__ extern float frexpf (float __x, int *__pexp);
260 __ATTR_CONST__ extern double frexp (double __x, int *__pexp) __ASM_ALIAS(frexpf);    /**< The alias
for frexpf(). */
261
262 /**
263 The ldexpf() function multiplies a floating-point number by an integral
264 power of 2. It returns the value of \a __x times 2 raised to the power
265 \a __exp.
266 */
267 __ATTR_CONST__ extern float ldexpf (float __x, int __exp);
268 __ATTR_CONST__ extern double ldexp (double __x, int __exp) __ASM_ALIAS(ldexpf);    /**< The alias for
ldexpf(). */
269
270 /**
271 The expf() function returns the exponential value of \a __x.
272 */
273 __ATTR_CONST__ extern float expf (float __x);
274 __ATTR_CONST__ extern double exp (double __x) __ASM_ALIAS(expf);    /**< The alias for expf(). */
275
276 /**
277 The coshf() function returns the hyperbolic cosine of \a __x.
278 */
279 __ATTR_CONST__ extern float coshf (float __x);
280 __ATTR_CONST__ extern double cosh (double __x) __ASM_ALIAS(coshf);    /**< The alias for coshf(). */
281
282 /**

```

```

283 The sinh() function returns the hyperbolic sine of \a __x.
284 */
285 __ATTR_CONST__ extern float sinh (float __x);
286 __ATTR_CONST__ extern double sinh (double __x) __ASM_ALIAS(sinh);    /**< The alias for sinh(). */
287
288 /**
289 The tanh() function returns the hyperbolic tangent of \a __x.
290 */
291 __ATTR_CONST__ extern float tanh (float __x);
292 __ATTR_CONST__ extern double tanh (double __x) __ASM_ALIAS(tanh);    /**< The alias for tanh(). */
293
294 /**
295 The acosf() function computes the principal value of the arc cosine of
296 \a __x. The returned value is in the range [0, pi] radians. A domain
297 error occurs for arguments not in the range [-1, +1].
298 */
299 __ATTR_CONST__ extern float acosf (float __x);
300 __ATTR_CONST__ extern double acos (double __x) __ASM_ALIAS(acosf);    /**< The alias for acosf(). */
301
302 /**
303 The asinf() function computes the principal value of the arc sine of
304 \a __x. The returned value is in the range [-pi/2, pi/2] radians. A
305 domain error occurs for arguments not in the range [-1, +1].
306 */
307 __ATTR_CONST__ extern float asinf (float __x);
308 __ATTR_CONST__ extern double asin (double __x) __ASM_ALIAS(asinf);    /**< The alias for asinf(). */
309
310 /**
311 The atanf() function computes the principal value of the arc tangent
312 of \a __x. The returned value is in the range [-pi/2, pi/2] radians.
313 */
314 __ATTR_CONST__ extern float atanf (float __x);
315 __ATTR_CONST__ extern double atan (double __x) __ASM_ALIAS(atanf);    /**< The alias for atanf(). */
316
317 /**
318 The atan2f() function computes the principal value of the arc tangent
319 of <em>__y / __x</em>, using the signs of both arguments to determine
320 the quadrant of the return value. The returned value is in the range
321 [-pi, +pi] radians.
322 */
323 __ATTR_CONST__ extern float atan2f (float __y, float __x);
324 __ATTR_CONST__ extern double atan2 (double __y, double __x) __ASM_ALIAS(atan2f);    /**< The alias for
    atan2f(). */
325
326 /**
327 The logf() function returns the natural logarithm of argument \a __x.
328 */
329 __ATTR_CONST__ extern float logf (float __x);
330 __ATTR_CONST__ extern double log (double __x) __ASM_ALIAS(logf);    /**< The alias for logf(). */
331
332 /**
333 The log10f() function returns the logarithm of argument \a __x to base 10.
334 */
335 __ATTR_CONST__ extern float log10f (float __x);
336 __ATTR_CONST__ extern double log10 (double __x) __ASM_ALIAS(log10f);    /**< The alias for log10f().
    */
337
338 /**
339 The function powf() returns the value of \a __x to the exponent \a __y.
340 */
341 __ATTR_CONST__ extern float powf (float __x, float __y);
342 __ATTR_CONST__ extern double pow (double __x, double __y) __ASM_ALIAS(powf);    /**< The alias for
    powf(). */
343
344 /**
345 The function isnanf() returns 1 if the argument \a __x represents a
346 "not-a-number" (NaN) object, otherwise 0.
347 */
348 __ATTR_CONST__ extern int isnanf (float __x);
349 __ATTR_CONST__ extern int isnan (double __x) __ASM_ALIAS(isnanf);    /**< The alias for isnanf().
    */
350
351 /**
352 The function isinff() returns 1 if the argument \a __x is positive
353 infinity, -1 if \a __x is negative infinity, and 0 otherwise.
354
355 \note The GCC 4.3 can replace this function with inline code that
356 returns the 1 value for both infinities (gcc bug #35509).
357 */
358 __ATTR_CONST__ extern int isinff (float __x);
359 __ATTR_CONST__ extern int isinf (double __x) __ASM_ALIAS(isinff);    /**< The alias for isinff().
    */
360
361 /**
362 The isfinitef() function returns a nonzero value if \a __x is finite:
363 not plus or minus infinity, and not NaN.
364 */

```

```

365 __ATTR_CONST__ static inline int isfinitef (float __x)
366 {
367     unsigned char __exp;
368     __asm__ (
369         "mov    %0, %C1    \n\t"
370         "lsl    %0        \n\t"
371         "mov    %0, %D1    \n\t"
372         "rol    %0        "
373         : "=r" (__exp)
374         : "r" (__x) );
375     return __exp != 0xff;
376 }
377
378 #if __SIZEOF_DOUBLE__ == __SIZEOF_FLOAT__
379 static inline int isfinite (double __x) /**< The alias for isfinitef(). */
380 {
381     return isfinitef (__x);
382 }
383 #endif /* double = float */
384
385 /**
386 The copysignf() function returns \a __x but with the sign of \a __y.
387 They work even if \a __x or \a __y are NaN or zero.
388 */
389 __ATTR_CONST__ static inline float copysignf (float __x, float __y)
390 {
391     __asm__ (
392         "bst    %D2, 7    \n\t"
393         "bld    %D0, 7    "
394         : "=r" (__x)
395         : "0" (__x), "r" (__y) );
396     return __x;
397 }
398
399 __ATTR_CONST__ static inline double copysign (double __x, double __y)
400 {
401     __asm__ (
402         "bst    %r1+%2-1, 7" "\n\t"
403         "bld    %r0+%2-1, 7"
404         : "+r" (__x)
405         : "r" (__y), "n" (__SIZEOF_DOUBLE__);
406     return __x;
407 }
408
409 /**
410 The signbitf() function returns a nonzero value if the value of \a __x
411 has its sign bit set. This is not the same as '\a __x < 0.0',
412 because IEEE 754 floating point allows zero to be signed. The
413 comparison '-0.0 < 0.0' is false, but 'signbit (-0.0)' will return a
414 nonzero value.
415 */
416 __ATTR_CONST__ extern int signbitf (float __x);
417 __ATTR_CONST__ extern int signbit (double __x) __ASM_ALIAS(signbitf); /**< The alias for signbitf().
418 */
419 /**
420 The fdimf() function returns <em>maxf(__x - __y, 0)</em>. If \a __x or
421 \a __y or both are NaN, NaN is returned.
422 */
423 __ATTR_CONST__ extern float fdimf (float __x, float __y);
424 __ATTR_CONST__ extern double fdim (double __x, double __y) __ASM_ALIAS(fdimf); /**< The alias for
425 fdimf(). */
426 /**
427 The fmaf() function performs floating-point multiply-add. This is the
428 operation <em>(__x * __y) + __z</em>, but the intermediate result is
429 not rounded to the destination type. This can sometimes improve the
430 precision of a calculation.
431 */
432 __ATTR_CONST__ extern float fmaf (float __x, float __y, float __z);
433 __ATTR_CONST__ extern double fma (double __x, double __y, double __z) __ASM_ALIAS(fmaf); /**< The
434 alias for fmaf(). */
435 /**
436 The fmaxf() function returns the greater of the two values \a __x and
437 \a __y. If an argument is NaN, the other argument is returned. If
438 both arguments are NaN, NaN is returned.
439 */
440 __ATTR_CONST__ extern float fmaxf (float __x, float __y);
441 __ATTR_CONST__ extern double fmax (double __x, double __y) __ASM_ALIAS(fmaxf); /**< The alias for
442 fmaxf(). */
443 /**
444 The fminf() function returns the lesser of the two values \a __x and
445 \a __y. If an argument is NaN, the other argument is returned. If
446 both arguments are NaN, NaN is returned.
447 */

```

```

448 __ATTR_CONST__ extern float fminf (float __x, float __y);
449 __ATTR_CONST__ extern double fmin (double __x, double __y) __ASM_ALIAS(fminf);    /**< The alias for
    fminf(). */
450
451 /**
452 The truncf() function rounds \a __x to the nearest integer not larger
453 in absolute value.
454 */
455 __ATTR_CONST__ extern float truncf (float __x);
456 __ATTR_CONST__ extern double trunc (double __x) __ASM_ALIAS(truncf);    /**< The alias for truncf().
    */
457
458 /**
459 The roundf() function rounds \a __x to the nearest integer, but rounds
460 halfway cases away from zero (instead of to the nearest even integer).
461 Overflow is impossible.
462
463 \return The rounded value. If \a __x is an integral or infinite, \a
464 __x itself is returned. If \a __x is \c NaN, then \c NaN is returned.
465 */
466 __ATTR_CONST__ extern float roundf (float __x);
467 __ATTR_CONST__ extern double round (double __x) __ASM_ALIAS(roundf);    /**< The alias for roundf().
    */
468
469 /**
470 The lroundf() function rounds \a __x to the nearest integer, but rounds
471 halfway cases away from zero (instead of to the nearest even integer).
472 This function is similar to roundf() function, but it differs in type of
473 return value and in that an overflow is possible.
474
475 \return The rounded long integer value. If \a __x is not a finite number
476 or an overflow was, this realization returns the \c LONG_MIN value
477 (0x80000000).
478 */
479 __ATTR_CONST__ extern long lroundf (float __x);
480 __ATTR_CONST__ extern long lround (double __x) __ASM_ALIAS(lroundf);    /**< The alias for lroundf().
    */
481
482 /**
483 The lrintf() function rounds \a __x to the nearest integer, rounding the
484 halfway cases to the even integer direction. (That is both 1.5 and 2.5
485 values are rounded to 2). This function is similar to rintf() function,
486 but it differs in type of return value and in that an overflow is
487 possible.
488
489 \return The rounded long integer value. If \a __x is not a finite
490 number or an overflow was, this realization returns the \c LONG_MIN
491 value (0x80000000).
492 */
493 __ATTR_CONST__ extern long lrintf (float __x);
494 __ATTR_CONST__ extern long lrint (double __x) __ASM_ALIAS(lrintf);    /**< The alias for lrintf().
    */
495
496 #ifdef __cplusplus
497 }
498 #endif
499
500 /* @} */
501 #endif /* !__MATH_H */

```

25.48 setjmp.h File Reference

Functions

- int [setjmp](#) (jmp_buf __jmpb)
- void [longjmp](#) (jmp_buf __jmpb, int __ret) __ATTR_NORETURN__

25.49 setjmp.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002,2007 Marek Michalkiewicz
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6

```

```

7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: setjmp.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
32
33 #ifndef __SETJMP_H_
34 #define __SETJMP_H_ 1
35
36 #ifdef __cplusplus
37 extern "C" {
38 #endif
39
40 /*
41 jmp_buf:
42 offset size description
43 0 16/2 call-saved registers (r2-r17)
44 (AVR_TINY arch has only 2 call saved registers (r18,r19))
45 16/2 2 frame pointer (r29:r28)
46 18/4 2 stack pointer (SPH:SPL)
47 20/6 1 status register (SREG)
48 21/7 2/3 return address (PC) (2 bytes used for <=128Kw flash)
49 23/24/9 = total size (AVR_TINY arch always has 2 bytes PC)
50 */
51
52 #if !defined(__DOXYGEN__)
53
54 #if defined(__AVR_TINY__)
55 # define _JBLEN 9
56 #elif defined(__AVR_3_BYTE_PC__) && __AVR_3_BYTE_PC__
57 # define _JBLEN 24
58 #else
59 # define _JBLEN 23
60 #endif
61 typedef struct _jmp_buf { unsigned char _jb[_JBLEN]; } jmp_buf[1];
62
63 #endif /* not __DOXYGEN__ */
64
65 /** \file */
66 /** \defgroup setjmp <setjmp.h>: Non-local goto
67
68 While the C language has the dreaded \c goto statement, it can only be
69 used to jump to a label in the same (local) function. In order to jump
70 directly to another (non-local) function, the C library provides the
71 setjmp() and longjmp() functions. setjmp() and longjmp() are useful for
72 dealing with errors and interrupts encountered in a low-level subroutine
73 of a program.
74
75 \note setjmp() and longjmp() make programs hard to understand and maintain.
76 If possible, an alternative should be used.
77
78 \note longjmp() can destroy changes made to global register
79 variables (see \ref faq_regbind).
80
81 For a very detailed discussion of setjmp()/longjmp(), see Chapter 7 of
82 <em>Advanced Programming in the UNIX Environment</em>, by W. Richard
83 Stevens.
84
85 Example:
86
87 \code
88 #include <setjmp.h>
89
90 jmp_buf env;
91
92 int main (void)
93 {

```



```

94 if (setjmp (env))
95 {
96 ...   handle error ...
97 }
98
99 while (1)
100 {
101 ...   main processing loop which calls foo() some where ...
102 }
103 }
104
105 ...
106
107 void foo (void)
108 {
109 ...   blah, blah, blah ...
110
111 if (err)
112 {
113 longjmp (env, 1);
114 }
115 }
116 \endcode */
117
118 #if !(defined(__ATTR_NORETURN__) || defined(__DOXYGEN__))
119 #define __ATTR_NORETURN__ __attribute__((__noreturn__))
120 #endif
121
122 /** \ingroup setjmp
123 \brief Save stack context for non-local goto.
124
125 \code #include <setjmp.h>\endcode
126
127 setjmp() saves the stack context/environment in \e __jmpb for later use by
128 longjmp(). The stack context will be invalidated if the function which
129 called setjmp() returns.
130
131 \param __jmpb Variable of type \c jmp_buf which holds the stack
132 information such that the environment can be restored.
133
134 \returns setjmp() returns 0 if returning directly, and
135 non-zero when returning from longjmp() using the saved context. */
136
137 extern int setjmp(jmp_buf __jmpb);
138
139 /** \ingroup setjmp
140 \brief Non-local jump to a saved stack context.
141
142 \code #include <setjmp.h>\endcode
143
144 longjmp() restores the environment saved by the last call of setjmp() with
145 the corresponding \e __jmpb argument. After longjmp() is completed,
146 program execution continues as if the corresponding call of setjmp() had
147 just returned the value \e __ret.
148
149 \note longjmp() cannot cause 0 to be returned. If longjmp() is invoked
150 with a second argument of 0, 1 will be returned instead.
151
152 \param __jmpb Information saved by a previous call to setjmp().
153 \param __ret Value to return to the caller of setjmp().
154
155 \returns This function never returns. */
156
157 extern void longjmp(jmp_buf __jmpb, int __ret) __ATTR_NORETURN__;
158
159 #ifdef __cplusplus
160 }
161 #endif
162
163 #endif /* !__SETJMP_H_ */

```

25.50 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`

25.51 stdint.h

[Go to the documentation of this file.](#)

```
1 /* Copyright (c) 2002,2004,2005 Marek Michalkiewicz
2 Copyright (c) 2005, Carlos Lamas
3 Copyright (c) 2005,2007 Joerg Wunsch
4 Copyright (c) 2013 Embecosm
5 All rights reserved.
6
7 Redistribution and use in source and binary forms, with or without
8 modification, are permitted provided that the following conditions are met:
9
10 * Redistributions of source code must retain the above copyright
11 notice, this list of conditions and the following disclaimer.
12
13 * Redistributions in binary form must reproduce the above copyright
14 notice, this list of conditions and the following disclaimer in
15 the documentation and/or other materials provided with the
16 distribution.
17
18 * Neither the name of the copyright holders nor the names of
19 contributors may be used to endorse or promote products derived
20 from this software without specific prior written permission.
21
22 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
23 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
24 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
25 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
26 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
27 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
28 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
29 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
30 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
31 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
32 POSSIBILITY OF SUCH DAMAGE. */
33
34 /* $Id: stdint.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
35
36 /*
37 * ISO/IEC 9899:1999 7.18 Integer types <stdint.h>
38 */
39
40 #ifndef __STDINT_H_
41 #define __STDINT_H_
42
43 /** \file */
44 /** \defgroup avr_stdint <stdint.h>: Standard Integer Types
45 \code #include <stdint.h> \endcode
46
47 Use [u]intN_t if you need exactly N bits.
48
49 Since these typedefs are mandated by the C99 standard, they are preferred
50 over rolling your own typedefs. */
51
52 #ifndef __DOXYGEN__
```

```

53 /*
54 * __USING_MINT8 is defined to 1 if the -mint8 option is in effect.
55 */
56 #if __INT_MAX__ == 127
57 # define __USING_MINT8 1
58 #else
59 # define __USING_MINT8 0
60 #endif
61
62 #endif /* !__DOXYGEN__ */
63
64 /* Integer types */
65
66 #if defined(__DOXYGEN__)
67
68 /* doxygen gets confused by the __attribute__ stuff */
69
70 /** \name Exact-width integer types
71 Integer types having exactly the specified width */
72
73 /*@{*/
74
75 /** \ingroup avr_stdint
76 8-bit signed type. */
77
78 typedef signed char int8_t;
79
80 /** \ingroup avr_stdint
81 8-bit unsigned type. */
82
83 typedef unsigned char uint8_t;
84
85 /** \ingroup avr_stdint
86 16-bit signed type. */
87
88 typedef signed int int16_t;
89
90 /** \ingroup avr_stdint
91 16-bit unsigned type. */
92
93 typedef unsigned int uint16_t;
94
95 /** \ingroup avr_stdint
96 32-bit signed type. */
97
98 typedef signed long int int32_t;
99
100 /** \ingroup avr_stdint
101 32-bit unsigned type. */
102
103 typedef unsigned long int uint32_t;
104
105 /** \ingroup avr_stdint
106 64-bit signed type.
107 \note This type is not available when the compiler
108 option -mint8 is in effect. */
109
110 typedef signed long long int int64_t;
111
112 /** \ingroup avr_stdint
113 64-bit unsigned type.
114 \note This type is not available when the compiler
115 option -mint8 is in effect. */
116
117 typedef unsigned long long int uint64_t;
118
119 /*@}*/
120
121 #else /* !defined(__DOXYGEN__) */
122
123 /* actual implementation goes here */
124
125 typedef signed int int8_t __attribute__((__mode__(__QI__)));
126 typedef unsigned int uint8_t __attribute__((__mode__(__QI__)));
127 typedef signed int int16_t __attribute__((__mode__(__HI__)));
128 typedef unsigned int uint16_t __attribute__((__mode__(__HI__)));
129 typedef signed int int32_t __attribute__((__mode__(__SI__)));
130 typedef unsigned int uint32_t __attribute__((__mode__(__SI__)));
131 #if !__USING_MINT8
132 typedef signed int int64_t __attribute__((__mode__(__DI__)));
133 typedef unsigned int uint64_t __attribute__((__mode__(__DI__)));
134 #endif
135
136 #endif /* defined(__DOXYGEN__) */
137
138 /** \name Integer types capable of holding object pointers
139 These allow you to declare variables of the same size as a pointer. */

```

```
140
141 /*@{*/
142
143 /** \ingroup avr_stdint
144 Signed pointer compatible type. */
145
146 typedef int16_t intptr_t;
147
148 /** \ingroup avr_stdint
149 Unsigned pointer compatible type. */
150
151 typedef uint16_t uintptr_t;
152
153 /*@}*/
154
155 /** \name Minimum-width integer types
156 Integer types having at least the specified width */
157
158 /*@{*/
159
160 /** \ingroup avr_stdint
161 signed int with at least 8 bits. */
162
163 typedef int8_t int_least8_t;
164
165 /** \ingroup avr_stdint
166 unsigned int with at least 8 bits. */
167
168 typedef uint8_t uint_least8_t;
169
170 /** \ingroup avr_stdint
171 signed int with at least 16 bits. */
172
173 typedef int16_t int_least16_t;
174
175 /** \ingroup avr_stdint
176 unsigned int with at least 16 bits. */
177
178 typedef uint16_t uint_least16_t;
179
180 /** \ingroup avr_stdint
181 signed int with at least 32 bits. */
182
183 typedef int32_t int_least32_t;
184
185 /** \ingroup avr_stdint
186 unsigned int with at least 32 bits. */
187
188 typedef uint32_t uint_least32_t;
189
190 #if !__USING_MINT8 || defined(__DOXYGEN__)
191 /** \ingroup avr_stdint
192 signed int with at least 64 bits.
193 \note This type is not available when the compiler
194 option -mint8 is in effect. */
195
196 typedef int64_t int_least64_t;
197
198 /** \ingroup avr_stdint
199 unsigned int with at least 64 bits.
200 \note This type is not available when the compiler
201 option -mint8 is in effect. */
202
203 typedef uint64_t uint_least64_t;
204 #endif
205
206 /*@}*/
207
208
209 /** \name Fastest minimum-width integer types
210 Integer types being usually fastest having at least the specified width */
211
212 /*@{*/
213
214 /** \ingroup avr_stdint
215 fastest signed int with at least 8 bits. */
216
217 typedef int8_t int_fast8_t;
218
219 /** \ingroup avr_stdint
220 fastest unsigned int with at least 8 bits. */
221
222 typedef uint8_t uint_fast8_t;
223
224 /** \ingroup avr_stdint
225 fastest signed int with at least 16 bits. */
226
```

```

227 typedef int16_t int_fast16_t;
228
229 /** \ingroup avr_stdint
230 fastest unsigned int with at least 16 bits. */
231
232 typedef uint16_t uint_fast16_t;
233
234 /** \ingroup avr_stdint
235 fastest signed int with at least 32 bits. */
236
237 typedef int32_t int_fast32_t;
238
239 /** \ingroup avr_stdint
240 fastest unsigned int with at least 32 bits. */
241
242 typedef uint32_t uint_fast32_t;
243
244 #if !__USING_MINT8 || defined(__DOXYGEN__)
245 /** \ingroup avr_stdint
246 fastest signed int with at least 64 bits.
247 \note This type is not available when the compiler
248 option -mint8 is in effect. */
249
250 typedef int64_t int_fast64_t;
251
252 /** \ingroup avr_stdint
253 fastest unsigned int with at least 64 bits.
254 \note This type is not available when the compiler
255 option -mint8 is in effect. */
256
257 typedef uint64_t uint_fast64_t;
258 #endif
259
260 /* @} */
261
262
263 /** \name Greatest-width integer types
264 Types designating integer data capable of representing any value of
265 any integer type in the corresponding signed or unsigned category */
266
267 /* @{ */
268
269 #if __USING_MINT8
270 typedef int32_t intmax_t;
271
272 typedef uint32_t uintmax_t;
273 #else /* !__USING_MINT8 */
274 /** \ingroup avr_stdint
275 largest signed int available. */
276
277 typedef int64_t intmax_t;
278
279 /** \ingroup avr_stdint
280 largest unsigned int available. */
281
282 typedef uint64_t uintmax_t;
283 #endif /* __USING_MINT8 */
284
285 /* @} */
286
287 #ifndef __DOXYGEN__
288 /* Helping macro */
289 #ifndef __CONCAT
290 #define __CONCATenate(left, right) left ## right
291 #define __CONCAT(left, right) __CONCATenate(left, right)
292 #endif
293
294 #endif /* !__DOXYGEN__ */
295
296 #if !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS)
297
298 /** \name Limits of specified-width integer types
299 C++ implementations should define these macros only when
300 __STDC_LIMIT_MACROS is defined before <stdint.h> is included */
301
302 /* @{ */
303
304 /** \ingroup avr_stdint
305 largest positive value an int8_t can hold. */
306
307 #define INT8_MAX 0x7f
308
309 /** \ingroup avr_stdint
310 smallest negative value an int8_t can hold. */
311
312 #define INT8_MIN (-INT8_MAX - 1)
313

```

```

314 #if __USING_MINT8
315
316 #define UINT8_MAX (__CONCAT(INT8_MAX, U) * 2U + 1U)
317
318 #define INT16_MAX 0x7ffffL
319 #define INT16_MIN (-INT16_MAX - 1L)
320 #define UINT16_MAX (__CONCAT(INT16_MAX, U) * 2UL + 1UL)
321
322 #define INT32_MAX 0x7fffffffL
323 #define INT32_MIN (-INT32_MAX - 1LL)
324 #define UINT32_MAX (__CONCAT(INT32_MAX, U) * 2ULL + 1ULL)
325
326 #else /* !__USING_MINT8 */
327
328 /** \ingroup avr_stdint
329 largest value an uint8_t can hold. */
330
331 #define UINT8_MAX (INT8_MAX * 2 + 1)
332
333 /** \ingroup avr_stdint
334 largest positive value an int16_t can hold. */
335
336 #define INT16_MAX 0x7fff
337
338 /** \ingroup avr_stdint
339 smallest negative value an int16_t can hold. */
340
341 #define INT16_MIN (-INT16_MAX - 1)
342
343 /** \ingroup avr_stdint
344 largest value an uint16_t can hold. */
345
346 #define UINT16_MAX (__CONCAT(INT16_MAX, U) * 2U + 1U)
347
348 /** \ingroup avr_stdint
349 largest positive value an int32_t can hold. */
350
351 #define INT32_MAX 0x7fffffffL
352
353 /** \ingroup avr_stdint
354 smallest negative value an int32_t can hold. */
355
356 #define INT32_MIN (-INT32_MAX - 1L)
357
358 /** \ingroup avr_stdint
359 largest value an uint32_t can hold. */
360
361 #define UINT32_MAX (__CONCAT(INT32_MAX, U) * 2UL + 1UL)
362
363 #endif /* __USING_MINT8 */
364
365 /** \ingroup avr_stdint
366 largest positive value an int64_t can hold. */
367
368 #define INT64_MAX 0x7fffffffffffffffLL
369
370 /** \ingroup avr_stdint
371 smallest negative value an int64_t can hold. */
372
373 #define INT64_MIN (-INT64_MAX - 1LL)
374
375 /** \ingroup avr_stdint
376 largest value an uint64_t can hold. */
377
378 #define UINT64_MAX (__CONCAT(INT64_MAX, U) * 2ULL + 1ULL)
379
380 /* @} */
381
382 /** \name Limits of minimum-width integer types */
383 /* @{ */
384
385 /** \ingroup avr_stdint
386 largest positive value an int_least8_t can hold. */
387
388 #define INT_LEAST8_MAX INT8_MAX
389
390 /** \ingroup avr_stdint
391 smallest negative value an int_least8_t can hold. */
392
393 #define INT_LEAST8_MIN INT8_MIN
394
395 /** \ingroup avr_stdint
396 largest value an uint_least8_t can hold. */
397
398 #define UINT_LEAST8_MAX UINT8_MAX
399
400 /** \ingroup avr_stdint

```



```
401 largest positive value an int_least16_t can hold. */
402
403 #define INT_LEAST16_MAX INT16_MAX
404
405 /** \ingroup avr_stdint
406 smallest negative value an int_least16_t can hold. */
407
408 #define INT_LEAST16_MIN INT16_MIN
409
410 /** \ingroup avr_stdint
411 largest value an uint_least16_t can hold. */
412
413 #define UINT_LEAST16_MAX UINT16_MAX
414
415 /** \ingroup avr_stdint
416 largest positive value an int_least32_t can hold. */
417
418 #define INT_LEAST32_MAX INT32_MAX
419
420 /** \ingroup avr_stdint
421 smallest negative value an int_least32_t can hold. */
422
423 #define INT_LEAST32_MIN INT32_MIN
424
425 /** \ingroup avr_stdint
426 largest value an uint_least32_t can hold. */
427
428 #define UINT_LEAST32_MAX UINT32_MAX
429
430 /** \ingroup avr_stdint
431 largest positive value an int_least64_t can hold. */
432
433 #define INT_LEAST64_MAX INT64_MAX
434
435 /** \ingroup avr_stdint
436 smallest negative value an int_least64_t can hold. */
437
438 #define INT_LEAST64_MIN INT64_MIN
439
440 /** \ingroup avr_stdint
441 largest value an uint_least64_t can hold. */
442
443 #define UINT_LEAST64_MAX UINT64_MAX
444
445 /* @} */
446
447 /** \name Limits of fastest minimum-width integer types */
448
449 /* @{ */
450
451 /** \ingroup avr_stdint
452 largest positive value an int_fast8_t can hold. */
453
454 #define INT_FAST8_MAX INT8_MAX
455
456 /** \ingroup avr_stdint
457 smallest negative value an int_fast8_t can hold. */
458
459 #define INT_FAST8_MIN INT8_MIN
460
461 /** \ingroup avr_stdint
462 largest value an uint_fast8_t can hold. */
463
464 #define UINT_FAST8_MAX UINT8_MAX
465
466 /** \ingroup avr_stdint
467 largest positive value an int_fast16_t can hold. */
468
469 #define INT_FAST16_MAX INT16_MAX
470
471 /** \ingroup avr_stdint
472 smallest negative value an int_fast16_t can hold. */
473
474 #define INT_FAST16_MIN INT16_MIN
475
476 /** \ingroup avr_stdint
477 largest value an uint_fast16_t can hold. */
478
479 #define UINT_FAST16_MAX UINT16_MAX
480
481 /** \ingroup avr_stdint
482 largest positive value an int_fast32_t can hold. */
483
484 #define INT_FAST32_MAX INT32_MAX
485
486 /** \ingroup avr_stdint
487 smallest negative value an int_fast32_t can hold. */
```

```

488
489 #define INT_FAST32_MIN INT32_MIN
490
491 /** \ingroup avr_stdint
492 largest value an uint_fast32_t can hold. */
493
494 #define UINT_FAST32_MAX UINT32_MAX
495
496 /** \ingroup avr_stdint
497 largest positive value an int_fast64_t can hold. */
498
499 #define INT_FAST64_MAX INT64_MAX
500
501 /** \ingroup avr_stdint
502 smallest negative value an int_fast64_t can hold. */
503
504 #define INT_FAST64_MIN INT64_MIN
505
506 /** \ingroup avr_stdint
507 largest value an uint_fast64_t can hold. */
508
509 #define UINT_FAST64_MAX UINT64_MAX
510
511 /*}*/
512
513 /** \name Limits of integer types capable of holding object pointers */
514
515 /*{*/
516
517 /** \ingroup avr_stdint
518 largest positive value an intptr_t can hold. */
519
520 #define INTPTR_MAX INT16_MAX
521
522 /** \ingroup avr_stdint
523 smallest negative value an intptr_t can hold. */
524
525 #define INTPTR_MIN INT16_MIN
526
527 /** \ingroup avr_stdint
528 largest value an uintptr_t can hold. */
529
530 #define UINTPTR_MAX UINT16_MAX
531
532 /*}*/
533
534 /** \name Limits of greatest-width integer types */
535
536 /*{*/
537
538 /** \ingroup avr_stdint
539 largest positive value an intmax_t can hold. */
540
541 #define INTMAX_MAX INT64_MAX
542
543 /** \ingroup avr_stdint
544 smallest negative value an intmax_t can hold. */
545
546 #define INTMAX_MIN INT64_MIN
547
548 /** \ingroup avr_stdint
549 largest value an uintmax_t can hold. */
550
551 #define UINTMAX_MAX UINT64_MAX
552
553 /*}*/
554
555 /** \name Limits of other integer types
556 C++ implementations should define these macros only when
557 __STDC_LIMIT_MACROS is defined before <stdint.h> is included */
558
559 /*{*/
560
561 /** \ingroup avr_stdint
562 largest positive value a ptrdiff_t can hold. */
563
564 #define PTRDIFF_MAX INT16_MAX
565
566 /** \ingroup avr_stdint
567 smallest negative value a ptrdiff_t can hold. */
568
569 #define PTRDIFF_MIN INT16_MIN
570
571
572 /* Limits of sig_atomic_t */
573 /* signal.h is currently not implemented (not avr/signal.h) */
574

```

```

575 /** \ingroup avr_stdint
576 largest positive value a sig_atomic_t can hold. */
577
578 #define SIG_ATOMIC_MAX INT8_MAX
579
580 /** \ingroup avr_stdint
581 smallest negative value a sig_atomic_t can hold. */
582
583 #define SIG_ATOMIC_MIN INT8_MIN
584
585
586 /** \ingroup avr_stdint
587 largest value a size_t can hold. */
588
589 #define SIZE_MAX UINT16_MAX
590
591
592 /* Limits of wchar_t */
593 /* wchar.h is currently not implemented */
594 /* #define WCHAR_MAX */
595 /* #define WCHAR_MIN */
596
597
598 /* Limits of wint_t */
599 /* wchar.h is currently not implemented */
600 #ifndef WCHAR_MAX
601 #define WCHAR_MAX __WCHAR_MAX__
602 #define WCHAR_MIN __WCHAR_MIN__
603 #endif
604 #ifndef WINT_MAX
605 #define WINT_MAX __WINT_MAX__
606 #define WINT_MIN __WINT_MIN__
607 #endif
608
609
610 #endif /* !defined(__cplusplus) || defined(__STDC_LIMIT_MACROS) */
611
612 #if (!defined __cplusplus || __cplusplus >= 201103L \
613 || defined __STDC_CONSTANT_MACROS)
614
615 /** \name Macros for integer constants
616 C++ implementations should define these macros only when
617 __STDC_CONSTANT_MACROS is defined before <stdint.h> is included.
618
619 These definitions are valid for integer constants without suffix and
620 for macros defined as integer constant without suffix */
621
622 /* The GNU C preprocessor defines special macros in the implementation
623 namespace to allow a definition that works in #if expressions. */
624 #ifdef __INT8_C
625 #define INT8_C(c) __INT8_C(c)
626 #define INT16_C(c) __INT16_C(c)
627 #define INT32_C(c) __INT32_C(c)
628 #define INT64_C(c) __INT64_C(c)
629 #define UINT8_C(c) __UINT8_C(c)
630 #define UINT16_C(c) __UINT16_C(c)
631 #define UINT32_C(c) __UINT32_C(c)
632 #define UINT64_C(c) __UINT64_C(c)
633 #define INTMAX_C(c) __INTMAX_C(c)
634 #define UINTMAX_C(c) __UINTMAX_C(c)
635 #else
636 /** \ingroup avr_stdint
637 define a constant of type int8_t */
638
639 #define INT8_C(value) ((int8_t) value)
640
641 /** \ingroup avr_stdint
642 define a constant of type uint8_t */
643
644 #define UINT8_C(value) ((uint8_t) __CONCAT(value, U))
645
646 #if __USING_MINT8
647
648 #define INT16_C(value) __CONCAT(value, L)
649 #define UINT16_C(value) __CONCAT(value, UL)
650
651 #define INT32_C(value) ((int32_t) __CONCAT(value, LL))
652 #define UINT32_C(value) ((uint32_t) __CONCAT(value, ULL))
653
654 #else /* !__USING_MINT8 */
655
656 /** \ingroup avr_stdint
657 define a constant of type int16_t */
658
659 #define INT16_C(value) value
660
661 /** \ingroup avr_stdint

```

```

662 define a constant of type uint16_t */
663
664 #define UINT16_C(value) __CONCAT(value, U)
665
666 /** \ingroup avr_stdint
667 define a constant of type int32_t */
668
669 #define INT32_C(value) __CONCAT(value, L)
670
671 /** \ingroup avr_stdint
672 define a constant of type uint32_t */
673
674 #define UINT32_C(value) __CONCAT(value, UL)
675
676 #endif /* __USING_MINT8 */
677
678 /** \ingroup avr_stdint
679 define a constant of type int64_t */
680
681 #define INT64_C(value) __CONCAT(value, LL)
682
683 /** \ingroup avr_stdint
684 define a constant of type uint64_t */
685
686 #define UINT64_C(value) __CONCAT(value, ULL)
687
688 /** \ingroup avr_stdint
689 define a constant of type intmax_t */
690
691 #define INTMAX_C(value) __CONCAT(value, LL)
692
693 /** \ingroup avr_stdint
694 define a constant of type uintmax_t */
695
696 #define UINTMAX_C(value) __CONCAT(value, ULL)
697
698 #endif /* !__INT8_C */
699
700 /* @} */
701
702 #endif /* (!defined __cplusplus || __cplusplus >= 201103L \
703 || defined __STDC_CONSTANT_MACROS) */
704
705
706 #endif /* _STDINT_H_ */

```

25.52 stdio.h File Reference

Macros

- #define [stdin](#) ([__job\[0\]](#))
- #define [stdout](#) ([__job\[1\]](#))
- #define [stderr](#) ([__job\[2\]](#))
- #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)

25.52.1 Macro Definition Documentation

25.52.1.1 _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\(\)](#).

25.52.1.2 _FDEV_ERR `#define _FDEV_ERR (-1)`

Return code for an error condition during device read.

To be used in the get function of [fdevopen\(\)](#).

25.52.1.3 _FDEV_SETUP_READ `#define _FDEV_SETUP_READ __SRD`

[fdev_setup_stream\(\)](#) with read intent

25.52.1.4 _FDEV_SETUP_RW `#define _FDEV_SETUP_RW (__SRD|__SWR)`

[fdev_setup_stream\(\)](#) with read/write intent

25.52.1.5 _FDEV_SETUP_WRITE `#define _FDEV_SETUP_WRITE __SWR`

[fdev_setup_stream\(\)](#) with write intent

25.52.1.6 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.

25.52.1.7 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.)

25.52.1.8 fdev_get_udata `#define fdev_get_udata(
stream) ((stream)->udata)`

This macro retrieves a pointer to user defined data from a FILE stream object.

25.52.1.9 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.

25.52.1.10 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\(\)](#).

25.52.1.11 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\(\)](#).

25.52.1.12 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\(\)](#).

25.52.1.13 getchar `#define getchar(
 void) fgetc(stdin)`

The macro `getchar` reads a character from `stdin`. Return values and error handling is identical to [fgetc\(\)](#).

25.52.1.14 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\(\)](#).

25.52.1.15 putchar `#define putchar(
 __c) fputc(__c, stdout)`

The macro `putchar` sends character `c` to `stdout`.

25.52.1.16 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`).

25.52.1.17 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`.

25.52.1.18 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`.

25.52.2 Typedef Documentation

25.52.2.1 FILE `typedef struct __file FILE`

`FILE` is the opaque structure that is passed around between the various standard IO functions.

25.52.3 Function Documentation

25.52.3.1 clearerr() `void clearerr (
 FILE * __stream)`

Clear the error and end-of-file flags of `stream`.

25.52.3.2 fclose() `int fclose (
 FILE * __stream)`

This function closes `stream`, and disallows and 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).

25.52.3.3 `feof()` `int feof (`
 `FILE * __stream)`

Test the end-of-file flag of `stream`. This flag can only be cleared by a call to `clearerr()`.

25.52.3.4 `ferror()` `int ferror (`
 `FILE * __stream)`

Test the error flag of `stream`. This flag can only be cleared by a call to `clearerr()`.

25.52.3.5 `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.

25.52.3.6 `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.

25.52.3.7 `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.

25.52.3.8 `fprintf()` `int fprintf (`
 `FILE * __stream,`
 `const char * __fmt,`
 `...)`

The function `fprintf` performs formatted output to `stream`. See `vfprintf()` for details.

25.52.3.9 `fprintf_P()` `int fprintf_P (`
 `FILE * __stream,`
 `const char * __fmt,`
 `...)`

Variant of `fprintf()` that uses a `fmt` string that resides in program memory.

25.52.3.10 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.

25.52.3.11 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.

25.52.3.12 fputs_P() `int fputs_P (`
 `const char * __str,`
 `FILE * __stream)`

Variant of `fputs()` where `str` resides in program memory.

25.52.3.13 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.

25.52.3.14 fscanf() `int fscanf (`
 `FILE * __stream,`
 `const char * __fmt,`
 `...)`

The function `fscanf` performs formatted input, reading the input data from `stream`.

See `vfscanf()` for details.

25.52.3.15 fscanf_P() `int fscanf_P (`
 `FILE * __stream,`
 `const char * __fmt,`
 `...)`

Variant of `fscanf()` using a `fmt` string in program memory.

25.52.3.16 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.

25.52.3.17 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.

25.52.3.18 printf() `int printf (`
 `const char * __fmt,`
 `...)`

The function `printf` performs formatted output to stream `stdout`. See [vfprintf\(\)](#) for details.

25.52.3.19 printf_P() `int printf_P (`
 `const char * __fmt,`
 `...)`

Variant of [printf\(\)](#) that uses a `fmt` string that resides in program memory.

25.52.3.20 puts() `int puts (`
 `const char * __str)`

Write the string pointed to by `str`, and a trailing newline character, to `stdout`.

25.52.3.21 puts_P() `int puts_P (`
 `const char * __str)`

Variant of [puts\(\)](#) where `str` resides in program memory.

25.52.3.22 scanf() `int scanf (`
 `const char * __fmt,`
 `...)`

The function `scanf` performs formatted input from stream `stdin`.

See [vfscanf\(\)](#) for details.

25.52.3.23 scanf_P() `int scanf_P (`
 `const char * __fmt,`
 `...)`

Variant of [scanf\(\)](#) where `fmt` resides in program memory.

25.52.3.24 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.

25.52.3.25 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.

25.52.3.26 sprintf() `int sprintf (`
 `char * __s,`
 `const char * __fmt,`
 `...)`

Variant of [printf\(\)](#) that sends the formatted characters to string `s`.

25.52.3.27 sprintf_P() `int sprintf_P (`
 `char * __s,`
 `const char * __fmt,`
 `...)`

Variant of [sprintf\(\)](#) that uses a `fmt` string that resides in program memory.

25.52.3.28 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.

25.52.3.29 sscanf_P() `int sscanf_P (`
 `const char * __buf,`
 `const char * __fmt,`
 `...)`

Variant of [sscanf\(\)](#) using a `fmt` string in program memory.

25.52.3.30 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.

25.52.3.31 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_flt -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.

25.52.3.32 `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.

25.52.3.33 `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
```


25.52.3.34 `vfscanf_P()` `int vfscanf_P (`
 `FILE * __stream,`
 `const char * __fmt,`
 `va_list __ap)`

Variant of `vfscanf()` using a `fmt` string in program memory.

25.52.3.35 `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 `vfprintf()`.

See `vfprintf()` for details.

25.52.3.36 `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.

25.52.3.37 `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.

25.52.3.38 `vsnprintf_P()` `int vsnprintf_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.

25.52.3.39 `vsprintf()` `int vsprintf (`
 `char * __s,`
 `const char * __fmt,`
 `va_list ap)`

Like `sprintf()` but takes a variable argument list for the arguments.

```

25.52.3.40 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.

25.53 stdio.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, 2005, 2007 Joerg Wunsch
2 All rights reserved.
3
4 Portions of documentation Copyright (c) 1990, 1991, 1993
5 The Regents of the University of California.
6
7 All rights reserved.
8
9 Redistribution and use in source and binary forms, with or without
10 modification, are permitted provided that the following conditions are met:
11
12 * Redistributions of source code must retain the above copyright
13 notice, this list of conditions and the following disclaimer.
14
15 * Redistributions in binary form must reproduce the above copyright
16 notice, this list of conditions and the following disclaimer in
17 the documentation and/or other materials provided with the
18 distribution.
19
20 * Neither the name of the copyright holders nor the names of
21 contributors may be used to endorse or promote products derived
22 from this software without specific prior written permission.
23
24 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
25 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
26 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
27 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
28 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
29 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
30 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
31 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
32 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
33 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
34 POSSIBILITY OF SUCH DAMAGE.
35
36 $Id: stdio.h 2539 2017-06-11 15:25:02Z joerg_wunsch $
37 */
38
39 #ifndef _STDIO_H_
40 #define _STDIO_H_ 1
41
42 #ifndef __ASSEMBLER__
43
44 #include <inttypes.h>
45 #include <stdarg.h>
46
47 #ifndef __DOXYGEN__
48 #define __need_NULL
49 #define __need_size_t
50 #include <stddef.h>
51 #endif /* !__DOXYGEN__ */
52
53 /** \file */
54 /** \defgroup avr_stdio <stdio.h>: Standard IO facilities
55 \code #include <stdio.h> \endcode
56
57 <h3>Introduction to the Standard IO facilities</h3>
58
59 This file declares the standard IO facilities that are implemented
60 in \c avr-libc. Due to the nature of the underlying hardware,
61 only a limited subset of standard IO is implemented. There is no
62 actual file implementation available, so only device IO can be
63 performed. Since there's no operating system, the application
64 needs to provide enough details about their devices in order to
65 make them usable by the standard IO facilities.
66
67 Due to space constraints, some functionality has not been
68 implemented at all (like some of the \c printf conversions that
69 have been left out). Nevertheless, potential users of this
70 implementation should be warned: the \c printf and \c scanf families of functions, although

```

```

71 usually associated with presumably simple things like the
72 famous "Hello, world!" program, are actually fairly complex
73 which causes their inclusion to eat up a fair amount of code space.
74 Also, they are not fast due to the nature of interpreting the
75 format string at run-time. Whenever possible, resorting to the
76 (sometimes non-standard) predetermined conversion facilities that are
77 offered by avr-libc will usually cost much less in terms of speed
78 and code size.
79
80 <h3>Tunable options for code size vs. feature set</h3>
81
82 In order to allow programmers a code size vs. functionality tradeoff,
83 the function vfprintf() which is the heart of the printf family can be
84 selected in different flavours using linker options. See the
85 documentation of vfprintf() for a detailed description. The same
86 applies to vfscanf() and the \c scanf family of functions.
87
88 <h3>Outline of the chosen API</h3>
89
90 The standard streams \c stdin, \c stdout, and \c stderr are
91 provided, but contrary to the C standard, since avr-libc has no
92 knowledge about applicable devices, these streams are not already
93 pre-initialized at application startup. Also, since there is no
94 notion of "file" whatsoever to avr-libc, there is no function
95 \c fopen() that could be used to associate a stream to some device.
96 (See \ref stdio_note1 "note 1".) Instead, the function \c fdevopen()
97 is provided to associate a stream to a device, where the device
98 needs to provide a function to send a character, to receive a
99 character, or both. There is no differentiation between "text" and
100 "binary" streams inside avr-libc. Character \c \n is sent
101 literally down to the device's \c put() function. If the device
102 requires a carriage return (\c \r) character to be sent before
103 the linefeed, its \c put() routine must implement this (see
104 \ref stdio_note2 "note 2").
105
106 As an alternative method to fdevopen(), the macro
107 fdev_setup_stream() might be used to setup a user-supplied FILE
108 structure.
109
110 It should be noted that the automatic conversion of a newline
111 character into a carriage return - newline sequence breaks binary
112 transfers. If binary transfers are desired, no automatic
113 conversion should be performed, but instead any string that aims
114 to issue a CR-LF sequence must use <tt>"r\n"</tt> explicitly.
115
116 For convenience, the first call to \c fdevopen() that opens a
117 stream for reading will cause the resulting stream to be aliased
118 to \c stdin. Likewise, the first call to \c fdevopen() that opens
119 a stream for writing will cause the resulting stream to be aliased
120 to both, \c stdout, and \c stderr. Thus, if the open was done
121 with both, read and write intent, all three standard streams will
122 be identical. Note that these aliases are indistinguishable from
123 each other, thus calling \c fclose() on such a stream will also
124 effectively close all of its aliases (\ref stdio_note3 "note 3").
125
126 It is possible to tie additional user data to a stream, using
127 fdev_set_userdata(). The backend put and get functions can then
128 extract this user data using fdev_get_userdata(), and act
129 appropriately. For example, a single put function could be used
130 to talk to two different UARTs that way, or the put and get
131 functions could keep internal state between calls there.
132
133 <h3>Format strings in flash ROM</h3>
134
135 All the \c printf and \c scanf family functions come in two flavours: the
136 standard name, where the format string is expected to be in
137 SRAM, as well as a version with the suffix "_P" where the format
138 string is expected to reside in the flash ROM. The macro
139 \c PSTR (explained in \ref avr_pgmspace) becomes very handy
140 for declaring these format strings.
141
142 \anchor stdio_without_malloc
143 <h3>Running stdio without malloc</h3>
144
145 By default, fdevopen() requires malloc(). As this is often
146 not desired in the limited environment of a microcontroller, an
147 alternative option is provided to run completely without malloc().
148
149 The macro fdev_setup_stream() is provided to prepare a
150 user-supplied FILE buffer for operation with stdio.
151
152 <h4>Example</h4>
153
154 \code
155 #include <stdio.h>
156
157 static int uart_putchar(char c, FILE *stream);

```

```

158
159 static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,
160 _FDEV_SETUP_WRITE);
161
162 static int
163 uart_putchar(char c, FILE *stream)
164 {
165
166 if (c == '\n')
167 uart_putchar('\r', stream);
168 loop_until_bit_is_set(UCSRA, UDRE);
169 UDR = c;
170 return 0;
171 }
172
173 int
174 main(void)
175 {
176 init_uart();
177 stdout = &mystdout;
178 printf("Hello, world!\n");
179
180 return 0;
181 }
182 \endcode
183
184 This example uses the initializer form FDEV_SETUP_STREAM() rather
185 than the function-like fdev_setup_stream(), so all data
186 initialization happens during C start-up.
187
188 If streams initialized that way are no longer needed, they can be
189 destroyed by first calling the macro fdev_close(), and then
190 destroying the object itself. No call to fclose() should be
191 issued for these streams. While calling fclose() itself is
192 harmless, it will cause an undefined reference to free() and thus
193 cause the linker to link the malloc module into the application.
194
195 <h3>Notes</h3>
196
197 \anchor stdio_note1 \par Note 1:
198 It might have been possible to implement a device abstraction that
199 is compatible with \c fopen() but since this would have required
200 to parse a string, and to take all the information needed either
201 out of this string, or out of an additional table that would need to be
202 provided by the application, this approach was not taken.
203
204 \anchor stdio_note2 \par Note 2:
205 This basically follows the Unix approach: if a device such as a
206 terminal needs special handling, it is in the domain of the
207 terminal device driver to provide this functionality. Thus, a
208 simple function suitable as \c put() for \c fdevopen() that talks
209 to a UART interface might look like this:
210
211 \code
212 int
213 uart_putchar(char c, FILE *stream)
214 {
215
216 if (c == '\n')
217 uart_putchar('\r', stream);
218 loop_until_bit_is_set(UCSRA, UDRE);
219 UDR = c;
220 return 0;
221 }
222 \endcode
223
224 \anchor stdio_note3 \par Note 3:
225 This implementation has been chosen because the cost of maintaining
226 an alias is considerably smaller than the cost of maintaining full
227 copies of each stream. Yet, providing an implementation that offers
228 the complete set of standard streams was deemed to be useful. Not
229 only that writing \c printf() instead of <tt>fprintf(mystream, ...)</tt>
230 saves typing work, but since avr-gcc needs to resort to pass all
231 arguments of variadic functions on the stack (as opposed to passing
232 them in registers for functions that take a fixed number of
233 parameters), the ability to pass one parameter less by implying
234 \c stdin or stdout will also save some execution time.
235 */
236
237 #if !defined(__DOXYGEN__)
238
239 /*
240 * This is an internal structure of the library that is subject to be
241 * changed without warnings at any time. Please do *never* reference
242 * elements of it beyond by using the official interfaces provided.
243 */
244 struct __file {

```

```

245     char      *buf;          /* buffer pointer */
246     unsigned char ungetc;    /* ungetc() buffer */
247     uint8_t flags;          /* flags, see below */
248 #define __SRD    0x0001      /* OK to read */
249 #define __SWR    0x0002      /* OK to write */
250 #define __SSTR    0x0004      /* this is an sprintf/snprintf string */
251 #define __SPGM    0x0008      /* fmt string is in progmem */
252 #define __SERR    0x0010      /* found error */
253 #define __SEOF    0x0020      /* found EOF */
254 #define __SUNGET 0x0040      /* ungetc() happened */
255 #define __SMALLOC 0x0080      /* handle is malloc()ed */
256 #if 0
257 /* possible future extensions, will require uint16_t flags */
258 #define __SRW    0x0100      /* open for reading & writing */
259 #define __SLBF    0x0200      /* line buffered */
260 #define __SNBF    0x0400      /* unbuffered */
261 #define __SMBF    0x0800      /* buf is from malloc */
262 #endif
263     int size;                /* size of buffer */
264     int len;                 /* characters read or written so far */
265     int (*put)(char, struct __file *); /* function to write one char to device */
266     int (*get)(struct __file *); /* function to read one char from device */
267     void *udata;             /* User defined and accessible data. */
268 };
269
270 #endif /* not __DOXYGEN__ */
271
272 /*@{*/
273 /**
274  \c FILE is the opaque structure that is passed around between the
275  various standard IO functions.
276  */
277 typedef struct __file FILE;
278
279 /**
280  Stream that will be used as an input stream by the simplified
281  functions that don't take a \c stream argument.
282
283  The first stream opened with read intent using \c fdevopen()
284  will be assigned to \c stdin.
285  */
286 #define stdin (__iob[0])
287
288 /**
289  Stream that will be used as an output stream by the simplified
290  functions that don't take a \c stream argument.
291
292  The first stream opened with write intent using \c fdevopen()
293  will be assigned to both, \c stdin, and \c stderr.
294  */
295 #define stdout (__iob[1])
296
297 /**
298  Stream destined for error output. Unless specifically assigned,
299  identical to \c stdout.
300
301  If \c stderr should point to another stream, the result of
302  another \c fdevopen() must be explicitly assigned to it without
303  closing the previous \c stderr (since this would also close
304  \c stdout).
305  */
306 #define stderr (__iob[2])
307
308 /**
309  \c EOF declares the value that is returned by various standard IO
310  functions in case of an error. Since the AVR platform (currently)
311  doesn't contain an abstraction for actual files, its origin as
312  "end of file" is somewhat meaningless here.
313  */
314 #define EOF (-1)
315
316 /** This macro inserts a pointer to user defined data into a FILE
317  stream object.
318
319  The user data can be useful for tracking state in the put and get
320  functions supplied to the fdevopen() function. */
321 #define fdev_set_udata(stream, u) do { (stream)->udata = u; } while(0)
322
323 /** This macro retrieves a pointer to user defined data from a FILE
324  stream object. */
325 #define fdev_get_udata(stream) ((stream)->udata)
326
327 #if defined(__DOXYGEN__)
328 /**
329  \brief Setup a user-supplied buffer as an stdio stream
330
331  This macro takes a user-supplied buffer \c stream, and sets it up

```

```

332 as a stream that is valid for stdio operations, similar to one that
333 has been obtained dynamically from fdevopen(). The buffer to setup
334 must be of type FILE.
335
336 The arguments \c put and \c get are identical to those that need to
337 be passed to fdevopen().
338
339 The \c rwflag argument can take one of the values _FDEV_SETUP_READ,
340 _FDEV_SETUP_WRITE, or _FDEV_SETUP_RW, for read, write, or read/write
341 intent, respectively.
342
343 \note No assignments to the standard streams will be performed by
344 fdev_setup_stream(). If standard streams are to be used, these
345 need to be assigned by the user. See also under
346 \ref stdio_without_malloc "Running stdio without malloc()".
347 */
348 #define fdev_setup_stream(stream, put, get, rwflag)
349 #else /* !DOXYGEN */
350 #define fdev_setup_stream(stream, p, g, f) \
351 do { \
352 (stream)->put = p; \
353 (stream)->get = g; \
354 (stream)->flags = f; \
355 (stream)->udata = 0; \
356 } while(0)
357 #endif /* DOXYGEN */
358
359 #define _FDEV_SETUP_READ __SRD /**< fdev_setup_stream() with read intent */
360 #define _FDEV_SETUP_WRITE __SWR /**< fdev_setup_stream() with write intent */
361 #define _FDEV_SETUP_RW (__SRD|__SWR) /**< fdev_setup_stream() with read/write intent */
362
363 /**
364 * Return code for an error condition during device read.
365 *
366 * To be used in the get function of fdevopen().
367 */
368 #define _FDEV_ERR (-1)
369
370 /**
371 * Return code for an end-of-file condition during device read.
372 *
373 * To be used in the get function of fdevopen().
374 */
375 #define _FDEV_EOF (-2)
376
377 #if defined(__DOXYGEN__)
378 /**
379 \brief Initializer for a user-supplied stdio stream
380
381 This macro acts similar to fdev_setup_stream(), but it is to be
382 used as the initializer of a variable of type FILE.
383
384 The remaining arguments are to be used as explained in
385 fdev_setup_stream().
386 */
387 #define FDEV_SETUP_STREAM(put, get, rwflag)
388 #else /* !DOXYGEN */
389 #define FDEV_SETUP_STREAM(p, g, f) \
390 { \
391 .put = p, \
392 .get = g, \
393 .flags = f, \
394 .udata = 0, \
395 }
396 #endif /* DOXYGEN */
397
398 #ifdef __cplusplus
399 extern "C" {
400 #endif
401
402 #if !defined(__DOXYGEN__)
403 /**
404 * Doxygen documentation can be found in fdevopen.c.
405 */
406
407 extern struct __file *__iob[];
408
409 #if defined(__STDIO_FDEVOPEN_COMPAT_12)
410 /**
411 * Declare prototype for the discontinued version of fdevopen() that
412 * has been in use up to avr-libc 1.2.x. The new implementation has
413 * some backwards compatibility with the old version.
414 */
415 extern FILE *fdevopen(int (*__put)(char), int (*__get)(void),
416 int __opts __attribute__((unused)));
417 #else /* !defined(__STDIO_FDEVOPEN_COMPAT_12) */
418 /** New prototype for avr-libc 1.4 and above. */

```

```

419 extern FILE *fdevopen(int (*__put)(char, FILE*), int (*__get)(FILE*));
420 #endif /* defined(__STDIO_FDEVOPEN_COMPAT_12) */
421
422 #endif /* not __DOXYGEN__ */
423
424 /**
425 This function closes \c stream, and disallows and further
426 IO to and from it.
427
428 When using fdevopen() to setup the stream, a call to fclose() is
429 needed in order to free the internal resources allocated.
430
431 If the stream has been set up using fdev_setup_stream() or
432 FDEV_SETUP_STREAM(), use fdev_close() instead.
433
434 It currently always returns 0 (for success).
435 */
436 extern int fclose(FILE *__stream);
437
438 /**
439 This macro frees up any library resources that might be associated
440 with \c stream. It should be called if \c stream is no longer
441 needed, right before the application is going to destroy the
442 \c stream object itself.
443
444 (Currently, this macro evaluates to nothing, but this might change
445 in future versions of the library.)
446 */
447 #if defined(__DOXYGEN__)
448 # define fdev_close()
449 #else
450 # define fdev_close() ((void)0)
451 #endif
452
453 /**
454 \c vfprintf is the central facility of the \c printf family of
455 functions. It outputs values to \c stream under control of a
456 format string passed in \c fmt. The actual values to print are
457 passed as a variable argument list \c ap.
458
459 \c vfprintf returns the number of characters written to \c stream,
460 or \c EOF in case of an error. Currently, this will only happen
461 if \c stream has not been opened with write intent.
462
463 The format string is composed of zero or more directives: ordinary
464 characters (not \c %), which are copied unchanged to the output
465 stream; and conversion specifications, each of which results in
466 fetching zero or more subsequent arguments. Each conversion
467 specification is introduced by the \c % character. The arguments must
468 properly correspond (after type promotion) with the conversion
469 specifier. After the \c %, the following appear in sequence:
470
471 - Zero or more of the following flags:
472 <ul>
473 <li> \c # The value should be converted to an "alternate form". For
474 c, d, i, s, and u conversions, this option has no effect.
475 For o conversions, the precision of the number is
476 increased to force the first character of the output
477 string to a zero (except if a zero value is printed with
478 an explicit precision of zero). For x and X conversions,
479 a non-zero result has the string '0x' (or '0X' for X
480 conversions) prepended to it.</li>
481 <li> \c 0 (zero) Zero padding. For all conversions, the converted
482 value is padded on the left with zeros rather than blanks.
483 If a precision is given with a numeric conversion (d, i,
484 o, u, i, x, and X), the 0 flag is ignored.</li>
485 <li> \c - A negative field width flag; the converted value is to be
486 left adjusted on the field boundary. The converted value
487 is padded on the right with blanks, rather than on the
488 left with blanks or zeros. A - overrides a 0 if both are
489 given.</li>
490 <li> ' ' (space) A blank should be left before a positive number
491 produced by a signed conversion (d, or i).</li>
492 <li> \c + A sign must always be placed before a number produced by a
493 signed conversion. A + overrides a space if both are
494 used.</li>
495 </ul>
496
497 - An optional decimal digit string specifying a minimum field width.
498 If the converted value has fewer characters than the field width, it
499 will be padded with spaces on the left (or right, if the left-adjustment
500 flag has been given) to fill out the field width.
501 - An optional precision, in the form of a period . followed by an
502 optional digit string. If the digit string is omitted, the
503 precision is taken as zero. This gives the minimum number of
504 digits to appear for d, i, o, u, x, and X conversions, or the
505 maximum number of characters to be printed from a string for \c s

```

```

506 conversions.
507 - An optional \c l or \c h length modifier, that specifies that the
508 argument for the d, i, o, u, x, or X conversion is a \c "long int"
509 rather than \c int. The \c h is ignored, as \c "short int" is
510 equivalent to \c int.
511 - A character that specifies the type of conversion to be applied.
512
513 The conversion specifiers and their meanings are:
514
515 - \c diouxX The int (or appropriate variant) argument is converted
516 to signed decimal (d and i), unsigned octal (o), unsigned
517 decimal (u), or unsigned hexadecimal (x and X) notation.
518 The letters "abcdef" are used for x conversions; the
519 letters "ABCDEF" are used for X conversions. The
520 precision, if any, gives the minimum number of digits that
521 must appear; if the converted value requires fewer digits,
522 it is padded on the left with zeros.
523 - \c p The <tt>void *</tt> argument is taken as an unsigned integer,
524 and converted similarly as a <tt>%#x</tt> command would do.
525 - \c c The \c int argument is converted to an \c "unsigned char", and the
526 resulting character is written.
527 - \c s The \c "char *" argument is expected to be a pointer to an array
528 of character type (pointer to a string). Characters from
529 the array are written up to (but not including) a
530 terminating NUL character; if a precision is specified, no
531 more than the number specified are written. If a precision
532 is given, no null character need be present; if the
533 precision is not specified, or is greater than the size of
534 the array, the array must contain a terminating NUL
535 character.
536 - \c % A \c % is written. No argument is converted. The complete
537 conversion specification is "%%".
538 - \c eE The double argument is rounded and converted in the format
539 \c "[-]d.ddde±dd" where there is one digit before the
540 decimal-point character and the number of digits after it
541 is equal to the precision; if the precision is missing, it
542 is taken as 6; if the precision is zero, no decimal-point
543 character appears. An \c E conversion uses the letter \c 'E'
544 (rather than \c 'e') to introduce the exponent. The exponent
545 always contains two digits; if the value is zero,
546 the exponent is 00.
547 - \c fF The double argument is rounded and converted to decimal notation
548 in the format \c "[-]ddd.ddd", where the number of digits after the
549 decimal-point character is equal to the precision specification.
550 If the precision is missing, it is taken as 6; if the precision
551 is explicitly zero, no decimal-point character appears. If a
552 decimal point appears, at least one digit appears before it.
553 - \c gG The double argument is converted in style \c f or \c e (or
554 \c F or \c E for \c G conversions). The precision
555 specifies the number of significant digits. If the
556 precision is missing, 6 digits are given; if the precision
557 is zero, it is treated as 1. Style \c e is used if the
558 exponent from its conversion is less than -4 or greater
559 than or equal to the precision. Trailing zeros are removed
560 from the fractional part of the result; a decimal point
561 appears only if it is followed by at least one digit.
562 - \c S Similar to the \c s format, except the pointer is expected to
563 point to a program-memory (ROM) string instead of a RAM string.
564
565 In no case does a non-existent or small field width cause truncation of a
566 numeric field; if the result of a conversion is wider than the field
567 width, the field is expanded to contain the conversion result.
568
569 Since the full implementation of all the mentioned features becomes
570 fairly large, three different flavours of fprintf() can be
571 selected using linker options. The default fprintf() implements
572 all the mentioned functionality except floating point conversions.
573 A minimized version of fprintf() is available that only implements
574 the very basic integer and string conversion facilities, but only
575 the \c # additional option can be specified using conversion
576 flags (these flags are parsed correctly from the format
577 specification, but then simply ignored). This version can be
578 requested using the following \ref gcc_minusW "compiler options":
579
580 \code
581 -Wl,-u,fprintf -lprintf_min
582 \endcode
583
584 If the full functionality including the floating point conversions
585 is required, the following options should be used:
586
587 \code
588 -Wl,-u,fprintf -lprintf_float -lm
589 \endcode
590
591 \par Limitations:
592 - The specified width and precision can be at most 255.

```



```

593
594 \par Notes:
595 - For floating-point conversions, if you link default or minimized
596 version of vfprintf(), the symbol \c ? will be output and double
597 argument will be skipped. So you output below will not be crashed.
598 For default version the width field and the "pad to left" ( symbol
599 minus ) option will work in this case.
600 - The \c hh length modifier is ignored (\c char argument is
601 promoted to \c int). More exactly, this realization does not check
602 the number of \c h symbols.
603 - But the \c ll length modifier will to abort the output, as this
604 realization does not operate \c long \c long arguments.
605 - The variable width or precision field (an asterisk \c * symbol)
606 is not realized and will to abort the output.
607
608 */
609
610 extern int vfprintf(FILE *__stream, const char *__fmt, va_list __ap);
611
612 /**
613 Variant of \c vfprintf() that uses a \c fmt string that resides
614 in program memory.
615 */
616 extern int vfprintf_P(FILE *__stream, const char *__fmt, va_list __ap);
617
618 /**
619 The function \c fputc sends the character \c c (though given as type
620 \c int) to \c stream. It returns the character, or \c EOF in case
621 an error occurred.
622 */
623 extern int fputc(int __c, FILE *__stream);
624
625 #if !defined(__DOXYGEN__)
626
627 /* putc() function implementation, required by standard */
628 extern int putc(int __c, FILE *__stream);
629
630 /* putchar() function implementation, required by standard */
631 extern int putchar(int __c);
632
633 #endif /* not __DOXYGEN__ */
634
635 /**
636 The macro \c putc used to be a "fast" macro implementation with a
637 functionality identical to fputc(). For space constraints, in
638 \c avr-libc, it is just an alias for \c fputc.
639 */
640 #define putc(__c, __stream) fputc(__c, __stream)
641
642 /**
643 The macro \c putchar sends character \c c to \c stdout.
644 */
645 #define putchar(__c) fputc(__c, stdout)
646
647 /**
648 The function \c printf performs formatted output to stream
649 \c stdout. See \c vfprintf() for details.
650 */
651 extern int printf(const char *__fmt, ...);
652
653 /**
654 Variant of \c printf() that uses a \c fmt string that resides
655 in program memory.
656 */
657 extern int printf_P(const char *__fmt, ...);
658
659 /**
660 The function \c vprintf performs formatted output to stream
661 \c stdout, taking a variable argument list as in vfprintf().
662
663 See vfprintf() for details.
664 */
665 extern int vprintf(const char *__fmt, va_list __ap);
666
667 /**
668 Variant of \c printf() that sends the formatted characters
669 to string \c s.
670 */
671 extern int sprintf(char *__s, const char *__fmt, ...);
672
673 /**
674 Variant of \c sprintf() that uses a \c fmt string that resides
675 in program memory.
676 */
677 extern int sprintf_P(char *__s, const char *__fmt, ...);
678
679 /**

```

```

680 Like \c sprintf(), but instead of assuming \c s to be of infinite
681 size, no more than \c n characters (including the trailing NUL
682 character) will be converted to \c s.
683
684 Returns the number of characters that would have been written to
685 \c s if there were enough space.
686 */
687 extern int  snprintf(char *__s, size_t __n, const char *__fmt, ...);
688
689 /**
690 Variant of \c snprintf() that uses a \c fmt string that resides
691 in program memory.
692 */
693 extern int  snprintf_P(char *__s, size_t __n, const char *__fmt, ...);
694
695 /**
696 Like \c sprintf() but takes a variable argument list for the
697 arguments.
698 */
699 extern int  vsprintf(char *__s, const char *__fmt, va_list ap);
700
701 /**
702 Variant of \c vsprintf() that uses a \c fmt string that resides
703 in program memory.
704 */
705 extern int  vsprintf_P(char *__s, const char *__fmt, va_list ap);
706
707 /**
708 Like \c vsprintf(), but instead of assuming \c s to be of infinite
709 size, no more than \c n characters (including the trailing NUL
710 character) will be converted to \c s.
711
712 Returns the number of characters that would have been written to
713 \c s if there were enough space.
714 */
715 extern int  vsnprintf(char *__s, size_t __n, const char *__fmt, va_list ap);
716
717 /**
718 Variant of \c vsnprintf() that uses a \c fmt string that resides
719 in program memory.
720 */
721 extern int  vsnprintf_P(char *__s, size_t __n, const char *__fmt, va_list ap);
722
723 /**
724 The function \c fprintf performs formatted output to \c stream.
725 See \c vfprintf() for details.
726 */
727 extern int  fprintf(FILE *__stream, const char *__fmt, ...);
728
729 /**
730 Variant of \c fprintf() that uses a \c fmt string that resides
731 in program memory.
732 */
733 extern int  fprintf_P(FILE *__stream, const char *__fmt, ...);
734
735 /**
736 Write the string pointed to by \c str to stream \c stream.
737 Returns 0 on success and EOF on error.
738 */
739 extern int  fputc(const char *__str, FILE *__stream);
740
741 /**
742 Variant of fputc() where \c str resides in program memory.
743 */
744 extern int  fputc_P(const char *__str, FILE *__stream);
745
746 /**
747 Write the string pointed to by \c str, and a trailing newline
748 character, to \c stdout.
749 */
750 extern int  puts(const char *__str);
751
752 /**
753 Variant of puts() where \c str resides in program memory.
754 */
755 extern int  puts_P(const char *__str);
756
757 /**
758 Write \c nmemb objects, \c size bytes each, to \c stream.
759 The first byte of the first object is referenced by \c ptr.
760
761 Returns the number of objects successfully written, i. e.
762 \c nmemb unless an output error occurred.
763 */
764 extern size_t fwrite(const void *__ptr, size_t __size, size_t __nmemb,
765 FILE *__stream);
766

```

```

767 /**
768 The function \c fgetc reads a character from \c stream. It returns
769 the character, or \c EOF in case end-of-file was encountered or an
770 error occurred. The routines feof() or ferror() must be used to
771 distinguish between both situations.
772 */
773 extern int fgetc(FILE *__stream);
774
775 #if !defined(__DOXYGEN__)
776
777 /* getc() function implementation, required by standard */
778 extern int getc(FILE *__stream);
779
780 /* getchar() function implementation, required by standard */
781 extern int getchar(void);
782
783 #endif /* not __DOXYGEN__ */
784
785 /**
786 The macro \c getc used to be a "fast" macro implementation with a
787 functionality identical to fgetc(). For space constraints, in
788 \c avr-libc, it is just an alias for \c fgetc.
789 */
790 #define getc(__stream) fgetc(__stream)
791
792 /**
793 The macro \c getchar reads a character from \c stdin. Return
794 values and error handling is identical to fgetc().
795 */
796 #define getchar() fgetc(stdin)
797
798 /**
799 The ungetc() function pushes the character \c c (converted to an
800 unsigned char) back onto the input stream pointed to by \c stream.
801 The pushed-back character will be returned by a subsequent read on
802 the stream.
803
804 Currently, only a single character can be pushed back onto the
805 stream.
806
807 The ungetc() function returns the character pushed back after the
808 conversion, or \c EOF if the operation fails. If the value of the
809 argument \c c character equals \c EOF, the operation will fail and
810 the stream will remain unchanged.
811 */
812 extern int ungetc(int __c, FILE *__stream);
813
814 /**
815 Read at most <tt>size - 1</tt> bytes from \c stream, until a
816 newline character was encountered, and store the characters in the
817 buffer pointed to by \c str. Unless an error was encountered while
818 reading, the string will then be terminated with a \c NUL
819 character.
820
821 If an error was encountered, the function returns NULL and sets the
822 error flag of \c stream, which can be tested using ferror().
823 Otherwise, a pointer to the string will be returned. */
824 extern char *fgets(char *__str, int __size, FILE *__stream);
825
826 /**
827 Similar to fgets() except that it will operate on stream \c stdin,
828 and the trailing newline (if any) will not be stored in the string.
829 It is the caller's responsibility to provide enough storage to hold
830 the characters read. */
831 extern char *gets(char *__str);
832
833 /**
834 Read \c nmemb objects, \c size bytes each, from \c stream,
835 to the buffer pointed to by \c ptr.
836
837 Returns the number of objects successfully read, i. e.
838 \c nmemb unless an input error occurred or end-of-file was
839 encountered. feof() and ferror() must be used to distinguish
840 between these two conditions.
841 */
842 extern size_t fread(void *__ptr, size_t __size, size_t __nmemb,
843 FILE *__stream);
844
845 /**
846 Clear the error and end-of-file flags of \c stream.
847 */
848 extern void clearerr(FILE *__stream);
849
850 #if !defined(__DOXYGEN__)
851 /* fast inlined version of clearerr() */
852 #define clearerror(s) do { (s)->flags &= ~(__SERR | __SEOF); } while(0)
853 #endif /* !defined(__DOXYGEN__) */

```

```

854
855 /**
856 Test the end-of-file flag of \c stream.    This flag can only be cleared
857 by a call to clearerr().
858 */
859 extern int  feof(FILE *__stream);
860
861 #if !defined(__DOXYGEN__)
862 /* fast inlined version of feof() */
863 #define feof(s) ((s)->flags & __SEOF)
864 #endif /* !defined(__DOXYGEN__) */
865
866 /**
867 Test the error flag of \c stream.    This flag can only be cleared
868 by a call to clearerr().
869 */
870 extern int  ferror(FILE *__stream);
871
872 #if !defined(__DOXYGEN__)
873 /* fast inlined version of ferror() */
874 #define ferror(s) ((s)->flags & __SERR)
875 #endif /* !defined(__DOXYGEN__) */
876
877 extern int  vfscanf(FILE *__stream, const char *__fmt, va_list __ap);
878
879 /**
880 Variant of vfscanf() using a \c fmt string in program memory.
881 */
882 extern int  vfscanf_P(FILE *__stream, const char *__fmt, va_list __ap);
883
884 /**
885 The function \c fscanf performs formatted input, reading the
886 input data from \c stream.
887
888 See vfscanf() for details.
889 */
890 extern int  fscanf(FILE *__stream, const char *__fmt, ...);
891
892 /**
893 Variant of fscanf() using a \c fmt string in program memory.
894 */
895 extern int  fscanf_P(FILE *__stream, const char *__fmt, ...);
896
897 /**
898 The function \c scanf performs formatted input from stream \c stdin.
899
900 See vfscanf() for details.
901 */
902 extern int  scanf(const char *__fmt, ...);
903
904 /**
905 Variant of scanf() where \c fmt resides in program memory.
906 */
907 extern int  scanf_P(const char *__fmt, ...);
908
909 /**
910 The function \c vscanf performs formatted input from stream
911 \c stdin, taking a variable argument list as in vfscanf().
912
913 See vfscanf() for details.
914 */
915 extern int  vscanf(const char *__fmt, va_list __ap);
916
917 /**
918 The function \c sscanf performs formatted input, reading the
919 input data from the buffer pointed to by \c buf.
920
921 See vfscanf() for details.
922 */
923 extern int  sscanf(const char *__buf, const char *__fmt, ...);
924
925 /**
926 Variant of sscanf() using a \c fmt string in program memory.
927 */
928 extern int  sscanf_P(const char *__buf, const char *__fmt, ...);
929
930 #if defined(__DOXYGEN__)
931 /**
932 Flush \c stream.
933
934 This is a null operation provided for source-code compatibility
935 only, as the standard IO implementation currently does not perform
936 any buffering.
937 */
938 extern int  fflush(FILE *stream);
939 #else
940 static __inline__ int fflush(FILE *stream __attribute__((unused)))

```

```

941 {
942     return 0;
943 }
944 #endif
945
946 #ifndef __DOXYGEN__
947 /* only mentioned for libstdc++ support, not implemented in library */
948 #define BUFSIZ 1024
949 #define _IONBF 0
950 __extension__ typedef long long fpos_t;
951 extern int fgetpos(FILE *stream, fpos_t *pos);
952 extern FILE *fopen(const char *path, const char *mode);
953 extern FILE *freopen(const char *path, const char *mode, FILE *stream);
954 extern FILE *fdopen(int, const char *);
955 extern int fseek(FILE *stream, long offset, int whence);
956 extern int fsetpos(FILE *stream, fpos_t *pos);
957 extern long ftell(FILE *stream);
958 extern int fileno(FILE *);
959 extern void perror(const char *s);
960 extern int remove(const char *pathname);
961 extern int rename(const char *oldpath, const char *newpath);
962 extern void rewind(FILE *stream);
963 extern void setbuf(FILE *stream, char *buf);
964 extern int setvbuf(FILE *stream, char *buf, int mode, size_t size);
965 extern FILE *tmpfile(void);
966 extern char *tmpnam(char *s);
967 #endif /* !__DOXYGEN__ */
968
969 #ifdef __cplusplus
970 }
971 #endif
972
973 /* @} */
974
975 #ifndef __DOXYGEN__
976 /*
977 * The following constants are currently not used by avr-libc's
978 * stdio subsystem. They are defined here since the gcc build
979 * environment expects them to be here.
980 */
981 #define SEEK_SET 0
982 #define SEEK_CUR 1
983 #define SEEK_END 2
984
985 #endif
986
987 #endif /* __ASSEMBLER__ */
988
989 #endif /* _STDLIB_H_ */

```

25.54 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) `__ATTR_NORETURN__`
- 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) `__ATTR_PURE__`
- int `atoi` (const char *__s) `__ATTR_PURE__`
- void `exit` (int __status) `__ATTR_NORETURN__`
- void * `malloc` (size_t __size) `__ATTR_MALLOC__`
- void `free` (void *__ptr)
- void * `calloc` (size_t __nele, size_t __size) `__ATTR_MALLOC__`
- void * `realloc` (void *__ptr, size_t __size) `__ATTR_MALLOC__`
- double `strtod` (const char *__nptr, char **__endptr)
- double `atof` (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.

Note that these functions are not located in the default library, `libc.a`, but in the mathematical library, `libm.a`. So when linking the application, the `-lm` option needs to be specified.

- `#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 * `dtostre` (double __val, char *__s, unsigned char __prec, unsigned char __flags)
- char * `dststrf` (double __val, signed char __width, unsigned char __prec, char *__s)

25.54.1 Macro Definition Documentation

25.54.1.1 `RAND_MAX` `#define RAND_MAX 0x7FFF`

Highest number that can be generated by `rand()`.

25.54.2 Typedef Documentation

25.54.2.1 `__compar_fn_t` `typedef int (* __compar_fn_t) (const void *, const void *)`

Comparison function type for `qsort()`, just for convenience.

25.54.3 Function Documentation

25.54.3.1 `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.

25.54.3.2 `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.

25.54.3.3 `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.

25.54.3.4 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.

25.54.3.5 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.

25.54.3.6 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.

25.54.3.7 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`.

25.54.3.8 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.

In a C++ context, global destructors will be called before halting execution.

25.54.3.9 `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.

25.54.3.10 `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.

25.54.3.11 `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`.

25.54.3.12 `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.

25.54.3.13 `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.

25.54.3.14 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.

25.54.3.15 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.

25.54.3.16 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.

25.54.3.17 srand() `void srand (`
`unsigned int __seed)`

Pseudo-random number generator seeding; see [rand\(\)](#).

25.54.3.18 strtod() `double strtod (`
`const char * nptr,`
`char ** endptr)`

The [strtod\(\)](#) function converts the initial portion of the string pointed to by `nptr` to double 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 [strtod\(\)](#) 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`.

25.54.3.19 `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.

25.54.3.20 `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.

25.54.4 Variable Documentation

25.54.4.1 `__malloc_heap_end` `char* __malloc_heap_end` [extern]

`malloc()` tunable.

25.54.4.2 `__malloc_heap_start` `char* __malloc_heap_start` [extern]

`malloc()` tunable.

25.54.4.3 `__malloc_margin` `size_t __malloc_margin` [extern]

`malloc()` tunable.

25.55 stdlib.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, Marek Michalkiewicz
2 Copyright (c) 2004,2007 Joerg Wunsch
3
4 Portions of documentation Copyright (c) 1990, 1991, 1993, 1994
5 The Regents of the University of California.
6
7 All rights reserved.
8
9 Redistribution and use in source and binary forms, with or without
10 modification, are permitted provided that the following conditions are met:
11
12 * Redistributions of source code must retain the above copyright
13 notice, this list of conditions and the following disclaimer.
14
15 * Redistributions in binary form must reproduce the above copyright
16 notice, this list of conditions and the following disclaimer in
17 the documentation and/or other materials provided with the
18 distribution.
19
20 * Neither the name of the copyright holders nor the names of
21 contributors may be used to endorse or promote products derived
22 from this software without specific prior written permission.
23
24 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
25 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
26 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
27 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
28 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
29 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
30 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
31 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
32 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
33 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
34 POSSIBILITY OF SUCH DAMAGE.
35
36 $Id: stdlib.h 2524 2016-09-07 12:08:25Z joerg_wunsch $
37 */
38
39 #ifndef _STDLIB_H_
40 #define _STDLIB_H_ 1
41
42 #ifndef __ASSEMBLER__
43
44 #ifndef __DOXYGEN__
45 #define __need_NULL
46 #define __need_size_t
47 #define __need_wchar_t
48 #include <stddef.h>
49
50 #ifndef __ptr_t
51 #define __ptr_t void *
52 #endif
53 #endif /* !__DOXYGEN__ */
54
55 #ifdef __cplusplus
56 extern "C" {
57 #endif
58
59 /** \file */

```

```

60
61 /** \defgroup avr_stdlib <stdlib.h>: General utilities
62 \code #include <stdlib.h> \endcode
63
64 This file declares some basic C macros and functions as
65 defined by the ISO standard, plus some AVR-specific extensions.
66 */
67
68 /*{*/
69 /** Result type for function div(). */
70 typedef struct {
71     int quot;           /**< The Quotient. */
72     int rem;            /**< The Remainder. */
73 } div_t;
74
75 /** Result type for function ldiv(). */
76 typedef struct {
77     long quot;          /**< The Quotient. */
78     long rem;           /**< The Remainder. */
79 } ldiv_t;
80
81 /** Comparison function type for qsort(), just for convenience. */
82 typedef int (*__compar_fn_t)(const void *, const void *);
83
84 #ifndef __DOXYGEN__
85
86 #ifndef __ATTR_CONST__
87 # define __ATTR_CONST__ __attribute__((__const__))
88 #endif
89
90 #ifndef __ATTR_MALLOC__
91 # define __ATTR_MALLOC__ __attribute__((__malloc__))
92 #endif
93
94 #ifndef __ATTR_NORETURN__
95 # define __ATTR_NORETURN__ __attribute__((__noreturn__))
96 #endif
97
98 #ifndef __ATTR_PURE__
99 # define __ATTR_PURE__ __attribute__((__pure__))
100 #endif
101
102 #ifndef __ATTR_GNU_INLINE__
103 # ifdef __GNUC_STDC_INLINE__
104 #   define __ATTR_GNU_INLINE__ __attribute__((__gnu_inline__))
105 # else
106 #   define __ATTR_GNU_INLINE__
107 # endif
108 #endif
109
110 #endif
111
112 /** The abort() function causes abnormal program termination to occur.
113 This realization disables interrupts and jumps to _exit() function
114 with argument equal to 1. In the limited AVR environment, execution is
115 effectively halted by entering an infinite loop. */
116 extern void abort(void) __ATTR_NORETURN__;
117
118 /** The abs() function computes the absolute value of the integer \c i.
119 \note The abs() and labs() functions are builtins of gcc.
120 */
121 extern int abs(int __i) __ATTR_CONST__;
122 #ifndef __DOXYGEN__
123 #define abs(__i) __builtin_abs(__i)
124 #endif
125
126 /** The labs() function computes the absolute value of the long integer
127 \c i.
128 \note The abs() and labs() functions are builtins of gcc.
129 */
130 extern long labs(long __i) __ATTR_CONST__;
131 #ifndef __DOXYGEN__
132 #define labs(__i) __builtin_labs(__i)
133 #endif
134
135 /**
136 The bsearch() function searches an array of \c nmemb objects, the
137 initial member of which is pointed to by \c base, for a member
138 that matches the object pointed to by \c key. The size of each
139 member of the array is specified by \c size.
140
141 The contents of the array should be in ascending sorted order
142 according to the comparison function referenced by \c compar.
143 The \c compar routine is expected to have two arguments which
144 point to the key object and to an array member, in that order,
145 and should return an integer less than, equal to, or greater than
146 zero if the key object is found, respectively, to be less than,

```

```

147 to match, or be greater than the array member.
148
149 The bsearch() function returns a pointer to a matching member of
150 the array, or a null pointer if no match is found. If two
151 members compare as equal, which member is matched is unspecified.
152 */
153 extern void *bsearch(const void *__key, const void *__base, size_t __nmem,
154                      size_t __size, int (*__compar)(const void *, const void *));
155
156 /* __divmodhi4 and __divmodsi4 from libgcc.a */
157 /**
158 The div() function computes the value \c num/denom and returns
159 the quotient and remainder in a structure named \c div_t that
160 contains two int members named \c quot and \c rem.
161 */
162 extern div_t div(int __num, int __denom) __asm__("__divmodhi4") __ATTR_CONST__;
163 /**
164 The ldiv() function computes the value \c num/denom and returns
165 the quotient and remainder in a structure named \c ldiv_t that
166 contains two long integer members named \c quot and \c rem.
167 */
168 extern ldiv_t ldiv(long __num, long __denom) __asm__("__divmodsi4") __ATTR_CONST__;
169
170 /**
171 The qsort() function is a modified partition-exchange sort, or
172 quicksort.
173
174 The qsort() function sorts an array of \c nmem objects, the
175 initial member of which is pointed to by \c base. The size of
176 each object is specified by \c size. The contents of the array
177 base are sorted in ascending order according to a comparison
178 function pointed to by \c compar, which requires two arguments
179 pointing to the objects being compared.
180
181 The comparison function must return an integer less than, equal
182 to, or greater than zero if the first argument is considered to
183 be respectively less than, equal to, or greater than the second.
184 */
185 extern void qsort(void *__base, size_t __nmem, size_t __size,
186                  __compar_fn_t __compar);
187
188 /**
189 The strtol() function converts the string in \c nptr to a long
190 value. The conversion is done according to the given base, which
191 must be between 2 and 36 inclusive, or be the special value 0.
192
193 The string may begin with an arbitrary amount of white space (as
194 determined by isspace()) followed by a single optional \c '+' or \c '-'
195 sign. If \c base is zero or 16, the string may then include a
196 \c "0x" prefix, and the number will be read in base 16; otherwise,
197 a zero base is taken as 10 (decimal) unless the next character is
198 \c '0', in which case it is taken as 8 (octal).
199
200 The remainder of the string is converted to a long value in the
201 obvious manner, stopping at the first character which is not a
202 valid digit in the given base. (In bases above 10, the letter \c 'A'
203 in either upper or lower case represents 10, \c 'B' represents 11,
204 and so forth, with \c 'Z' representing 35.)
205
206 If \c endptr is not NULL, strtol() stores the address of the first
207 invalid character in \c *endptr. If there were no digits at all,
208 however, strtol() stores the original value of \c nptr in \c
209 *endptr. (Thus, if \c *nptr is not \c '\\0' but \c **endptr is \c '\\0'
210 on return, the entire string was valid.)
211
212 The strtol() function returns the result of the conversion, unless
213 the value would underflow or overflow. If no conversion could be
214 performed, 0 is returned. If an overflow or underflow occurs, \c
215 errno is set to \ref avr_errno "ERANGE" and the function return value
216 is clamped to \c LONG_MIN or \c LONG_MAX, respectively.
217 */
218 extern long strtol(const char *__nptr, char **__endptr, int __base);
219
220 /**
221 The strtoul() function converts the string in \c nptr to an
222 unsigned long value. The conversion is done according to the
223 given base, which must be between 2 and 36 inclusive, or be the
224 special value 0.
225
226 The string may begin with an arbitrary amount of white space (as
227 determined by isspace()) followed by a single optional \c '+' or \c '-'
228 sign. If \c base is zero or 16, the string may then include a
229 \c "0x" prefix, and the number will be read in base 16; otherwise,
230 a zero base is taken as 10 (decimal) unless the next character is
231 \c '0', in which case it is taken as 8 (octal).
232
233 The remainder of the string is converted to an unsigned long value

```

```

234 in the obvious manner, stopping at the first character which is
235 not a valid digit in the given base. (In bases above 10, the
236 letter \c 'A' in either upper or lower case represents 10, \c 'B'
237 represents 11, and so forth, with \c 'Z' representing 35.)
238
239 If \c endptr is not NULL, strtoul() stores the address of the first
240 invalid character in \c *endptr. If there were no digits at all,
241 however, strtoul() stores the original value of \c nptr in \c
242 *endptr. (Thus, if \c *nptr is not \c '\\0' but \c **endptr is \c '\\0'
243 on return, the entire string was valid.)
244
245 The strtoul() function return either the result of the conversion
246 or, if there was a leading minus sign, the negation of the result
247 of the conversion, unless the original (non-negated) value would
248 overflow; in the latter case, strtoul() returns ULONG_MAX, and \c
249 errno is set to \ref avr_errno "ERANGE". If no conversion could
250 be performed, 0 is returned.
251 */
252 extern unsigned long strtoul(const char *__nptr, char **__endptr, int __base);
253
254 /**
255 The atol() function converts the initial portion of the string
256 pointed to by \p s to long integer representation. In contrast to
257
258 \code strtol(s, (char **)NULL, 10); \endcode
259
260 this function does not detect overflow (\c errno is not changed and
261 the result value is not predictable), uses smaller memory (flash and
262 stack) and works more quickly.
263 */
264 extern long atol(const char *__s) __ATTR_PURE__;
265
266 /**
267 The atoi() function converts the initial portion of the string
268 pointed to by \p s to integer representation. In contrast to
269
270 \code (int)strtol(s, (char **)NULL, 10); \endcode
271
272 this function does not detect overflow (\c errno is not changed and
273 the result value is not predictable), uses smaller memory (flash and
274 stack) and works more quickly.
275 */
276 extern int atoi(const char *__s) __ATTR_PURE__;
277
278 /**
279 The exit() function terminates the application. Since there is no
280 environment to return to, \c status is ignored, and code execution
281 will eventually reach an infinite loop, thereby effectively halting
282 all code processing. Before entering the infinite loop, interrupts
283 are globally disabled.
284
285 In a C++ context, global destructors will be called before halting
286 execution.
287 */
288 extern void exit(int __status) __ATTR_NORETURN__;
289
290 /**
291 The malloc() function allocates \c size bytes of memory.
292 If malloc() fails, a NULL pointer is returned.
293
294 Note that malloc() does \e not initialize the returned memory to
295 zero bytes.
296
297 See the chapter about \ref malloc "malloc() usage" for implementation
298 details.
299 */
300 extern void *malloc(size_t __size) __ATTR_MALLOC__;
301
302 /**
303 The free() function causes the allocated memory referenced by \c
304 ptr to be made available for future allocations. If \c ptr is
305 NULL, no action occurs.
306 */
307 extern void free(void *__ptr);
308
309 /**
310 \c malloc() \ref malloc_tunables "tunable".
311 */
312 extern size_t __malloc_margin;
313
314 /**
315 \c malloc() \ref malloc_tunables "tunable".
316 */
317 extern char *__malloc_heap_start;
318
319 /**
320 \c malloc() \ref malloc_tunables "tunable".

```

```

321 */
322 extern char *__malloc_heap_end;
323
324 /**
325 Allocate \c nele elements of \c size each. Identical to calling
326 \c malloc() using <tt>nele * size</tt> as argument, except the
327 allocated memory will be cleared to zero.
328 */
329 extern void *calloc(size_t __nele, size_t __size) __ATTR_MALLOC__;
330
331 /**
332 The realloc() function tries to change the size of the region
333 allocated at \c ptr to the new \c size value. It returns a
334 pointer to the new region. The returned pointer might be the
335 same as the old pointer, or a pointer to a completely different
336 region.
337
338 The contents of the returned region up to either the old or the new
339 size value (whatever is less) will be identical to the contents of
340 the old region, even in case a new region had to be allocated.
341
342 It is acceptable to pass \c ptr as NULL, in which case realloc()
343 will behave identical to malloc().
344
345 If the new memory cannot be allocated, realloc() returns NULL, and
346 the region at \c ptr will not be changed.
347 */
348 extern void *realloc(void *__ptr, size_t __size) __ATTR_MALLOC__;
349
350 extern double strtod(const char *__nptr, char **__endptr);
351
352 /** \ingroup avr_stdlib
353 \fn double atof (const char *nptr)
354
355 The atof() function converts the initial portion of the string pointed
356 to by \a nptr to double representation.
357
358 It is equivalent to calling
359 \code strtod(nptr, (char **)0); \endcode
360 */
361 extern double atof(const char *__nptr);
362
363 /** Highest number that can be generated by rand(). */
364 #define RAND_MAX 0x7FFF
365
366 /**
367 The rand() function computes a sequence of pseudo-random integers in the
368 range of 0 to \c RAND_MAX (as defined by the header file <stdlib.h>).
369
370 The srand() function sets its argument \c seed as the seed for a new
371 sequence of pseudo-random numbers to be returned by rand(). These
372 sequences are repeatable by calling srand() with the same seed value.
373
374 If no seed value is provided, the functions are automatically seeded with
375 a value of 1.
376
377 In compliance with the C standard, these functions operate on
378 \c int arguments. Since the underlying algorithm already uses
379 32-bit calculations, this causes a loss of precision. See
380 \c random() for an alternate set of functions that retains full
381 32-bit precision.
382 */
383 extern int rand(void);
384 /**
385 Pseudo-random number generator seeding; see rand().
386 */
387 extern void srand(unsigned int __seed);
388
389 /**
390 Variant of rand() that stores the context in the user-supplied
391 variable located at \c ctx instead of a static library variable
392 so the function becomes re-entrant.
393 */
394 extern int rand_r(unsigned long *__ctx);
395 /** @} */
396
397 /** @{ */
398 /** \name Non-standard (i.e. non-ISO C) functions.
399 \ingroup avr_stdlib
400 */
401 /**
402 \brief Convert an integer to a string.
403
404 The function itoa() converts the integer value from \c val into an
405 ASCII representation that will be stored under \c s. The caller
406 is responsible for providing sufficient storage in \c s.
407

```



```

408 \note The minimal size of the buffer \c s depends on the choice of
409 radix. For example, if the radix is 2 (binary), you need to supply a buffer
410 with a minimal length of 8 * sizeof (int) + 1 characters, i.e. one
411 character for each bit plus one for the string terminator. Using a larger
412 radix will require a smaller minimal buffer size.
413
414 \warning If the buffer is too small, you risk a buffer overflow.
415
416 Conversion is done using the \c radix as base, which may be a
417 number between 2 (binary conversion) and up to 36. If \c radix
418 is greater than 10, the next digit after \c '9' will be the letter
419 \c 'a'.
420
421 If radix is 10 and val is negative, a minus sign will be prepended.
422
423 The itoa() function returns the pointer passed as \c s.
424 */
425 #ifdef __DOXYGEN__
426 extern char *itoa(int val, char *s, int radix);
427 #else
428 extern __inline__ __ATTR_GNU_INLINE__
429 char *itoa (int __val, char *__s, int __radix)
430 {
431     if (!__builtin_constant_p (__radix)) {
432         extern char *__itoa (int, char *, int);
433         return __itoa (__val, __s, __radix);
434     } else if (__radix < 2 || __radix > 36) {
435         *__s = 0;
436         return __s;
437     } else {
438         extern char *__itoa_ncheck (int, char *, unsigned char);
439         return __itoa_ncheck (__val, __s, __radix);
440     }
441 }
442 #endif
443
444 /**
445 \ingroup avr_stdlib
446
447 \brief Convert a long integer to a string.
448
449 The function ltoa() converts the long integer value from \c val into an
450 ASCII representation that will be stored under \c s. The caller
451 is responsible for providing sufficient storage in \c s.
452
453 \note The minimal size of the buffer \c s depends on the choice of
454 radix. For example, if the radix is 2 (binary), you need to supply a buffer
455 with a minimal length of 8 * sizeof (long int) + 1 characters, i.e. one
456 character for each bit plus one for the string terminator. Using a larger
457 radix will require a smaller minimal buffer size.
458
459 \warning If the buffer is too small, you risk a buffer overflow.
460
461 Conversion is done using the \c radix as base, which may be a
462 number between 2 (binary conversion) and up to 36. If \c radix
463 is greater than 10, the next digit after \c '9' will be the letter
464 \c 'a'.
465
466 If radix is 10 and val is negative, a minus sign will be prepended.
467
468 The ltoa() function returns the pointer passed as \c s.
469 */
470 #ifdef __DOXYGEN__
471 extern char *ltoa(long val, char *s, int radix);
472 #else
473 extern __inline__ __ATTR_GNU_INLINE__
474 char *ltoa (long __val, char *__s, int __radix)
475 {
476     if (!__builtin_constant_p (__radix)) {
477         extern char *__ltoa (long, char *, int);
478         return __ltoa (__val, __s, __radix);
479     } else if (__radix < 2 || __radix > 36) {
480         *__s = 0;
481         return __s;
482     } else {
483         extern char *__ltoa_ncheck (long, char *, unsigned char);
484         return __ltoa_ncheck (__val, __s, __radix);
485     }
486 }
487 #endif
488
489 /**
490 \ingroup avr_stdlib
491
492 \brief Convert an unsigned integer to a string.
493
494 The function utoa() converts the unsigned integer value from \c val into an

```

```

495 ASCII representation that will be stored under \c s.    The caller
496 is responsible for providing sufficient storage in \c s.
497
498 \note The minimal size of the buffer \c s depends on the choice of
499 radix.  For example, if the radix is 2 (binary), you need to supply a buffer
500 with a minimal length of 8 * sizeof (unsigned int) + 1 characters, i.e. one
501 character for each bit plus one for the string terminator.  Using a larger
502 radix will require a smaller minimal buffer size.
503
504 \warning If the buffer is too small, you risk a buffer overflow.
505
506 Conversion is done using the \c radix as base, which may be a
507 number between 2 (binary conversion) and up to 36.    If \c radix
508 is greater than 10, the next digit after \c '9' will be the letter
509 \c 'a'.
510
511 The utoa() function returns the pointer passed as \c s.
512 */
513 #ifdef __DOXYGEN__
514 extern char *utoa(unsigned int val, char *s, int radix);
515 #else
516 extern __inline__ __ATTR_GNU_INLINE__
517 char *utoa (unsigned int __val, char *__s, int __radix)
518 {
519     if (!__builtin_constant_p (__radix)) {
520         extern char *__utoa (unsigned int, char *, int);
521         return __utoa (__val, __s, __radix);
522     } else if (__radix < 2 || __radix > 36) {
523         *__s = 0;
524         return __s;
525     } else {
526         extern char *__utoa_ncheck (unsigned int, char *, unsigned char);
527         return __utoa_ncheck (__val, __s, __radix);
528     }
529 }
530 #endif
531
532 /**
533 \ingroup avr_stdlib
534 \brief Convert an unsigned long integer to a string.
535
536 The function ultoa() converts the unsigned long integer value from
537 \c val into an ASCII representation that will be stored under \c s.
538 The caller is responsible for providing sufficient storage in \c s.
539
540 \note The minimal size of the buffer \c s depends on the choice of
541 radix.  For example, if the radix is 2 (binary), you need to supply a buffer
542 with a minimal length of 8 * sizeof (unsigned long int) + 1 characters,
543 i.e. one character for each bit plus one for the string terminator.  Using a
544 larger radix will require a smaller minimal buffer size.
545
546 \warning If the buffer is too small, you risk a buffer overflow.
547
548 Conversion is done using the \c radix as base, which may be a
549 number between 2 (binary conversion) and up to 36.    If \c radix
550 is greater than 10, the next digit after \c '9' will be the letter
551 \c 'a'.
552
553 The ultoa() function returns the pointer passed as \c s.
554 */
555 #ifdef __DOXYGEN__
556 extern char *ultoa(unsigned long val, char *s, int radix);
557 #else
558 extern __inline__ __ATTR_GNU_INLINE__
559 char *ultoa (unsigned long __val, char *__s, int __radix)
560 {
561     if (!__builtin_constant_p (__radix)) {
562         extern char *__ultoa (unsigned long, char *, int);
563         return __ultoa (__val, __s, __radix);
564     } else if (__radix < 2 || __radix > 36) {
565         *__s = 0;
566         return __s;
567     } else {
568         extern char *__ultoa_ncheck (unsigned long, char *, unsigned char);
569         return __ultoa_ncheck (__val, __s, __radix);
570     }
571 }
572 #endif
573
574 /** \ingroup avr_stdlib
575 Highest number that can be generated by random(). */
576 #define RANDOM_MAX 0x7FFFFFFF
577
578 /**
579 \ingroup avr_stdlib
580 The random() function computes a sequence of pseudo-random integers in the
581 range of 0 to \c RANDOM_MAX (as defined by the header file <stdlib.h>).

```

```

582
583 The random() function sets its argument \c seed as the seed for a new
584 sequence of pseudo-random numbers to be returned by rand(). These
585 sequences are repeatable by calling random() with the same seed value.
586
587 If no seed value is provided, the functions are automatically seeded with
588 a value of 1.
589 */
590 extern long random(void);
591 /**
592 \ingroup avr_stdlib
593 Pseudo-random number generator seeding; see random().
594 */
595 extern void srand(unsigned long __seed);
596
597 /**
598 \ingroup avr_stdlib
599 Variant of random() that stores the context in the user-supplied
600 variable located at \c ctx instead of a static library variable
601 so the function becomes re-entrant.
602 */
603 extern long random_r(unsigned long *__ctx);
604 #endif /* __ASSEMBLER */
605 /* @} */
606
607 /* @*/
608 /** \name Conversion functions for double arguments.
609 \ingroup avr_stdlib
610 Note that these functions are not located in the default library,
611 <tt>libc.a</tt>, but in the mathematical library, <tt>libm.a</tt>.
612 So when linking the application, the \c -lm option needs to be
613 specified.
614 */
615 /** \ingroup avr_stdlib
616 Bit value that can be passed in \c flags to dtostre(). */
617 #define DTOSTR_ALWAYS_SIGN 0x01 /* put '+' or '-' for positives */
618 /** \ingroup avr_stdlib
619 Bit value that can be passed in \c flags to dtostre(). */
620 #define DTOSTR_PLUS_SIGN 0x02 /* put '+' rather than '-' */
621 /** \ingroup avr_stdlib
622 Bit value that can be passed in \c flags to dtostre(). */
623 #define DTOSTR_UPPERCASE 0x04 /* put 'E' rather than 'e' */
624
625 #ifndef __ASSEMBLER__
626
627 /**
628 \ingroup avr_stdlib
629 The dtostre() function converts the double value passed in \c val into
630 an ASCII representation that will be stored under \c s. The caller
631 is responsible for providing sufficient storage in \c s.
632
633 Conversion is done in the format \c "[-]d.ddde+dd" where there is
634 one digit before the decimal-point character and the number of
635 digits after it is equal to the precision \c prec; if the precision
636 is zero, no decimal-point character appears. If \c flags has the
637 DTOSTR_UPPERCASE bit set, the letter \c 'E' (rather than \c 'e') will be
638 used to introduce the exponent. The exponent always contains two
639 digits; if the value is zero, the exponent is \c "00".
640
641 If \c flags has the DTOSTR_ALWAYS_SIGN bit set, a space character
642 will be placed into the leading position for positive numbers.
643
644 If \c flags has the DTOSTR_PLUS_SIGN bit set, a plus sign will be
645 used instead of a space character in this case.
646
647 The dtostre() function returns the pointer to the converted string \c s.
648 */
649 extern char *dtostre(double __val, char *__s, unsigned char __prec,
650 unsigned char __flags);
651
652 /**
653 \ingroup avr_stdlib
654 The dtostrf() function converts the double value passed in \c val into
655 an ASCII representation that will be stored under \c s. The caller
656 is responsible for providing sufficient storage in \c s.
657
658 Conversion is done in the format \c "[-]d.ddd". The minimum field
659 width of the output string (including the possible \c '.' and the possible
660 sign for negative values) is given in \c width, and \c prec determines
661 the number of digits after the decimal sign. \c width is signed value,
662 negative for left adjustment.
663
664 The dtostrf() function returns the pointer to the converted string \c s.
665 */
666 extern char *dtostrf(double __val, signed char __width,
667 unsigned char __prec, char *__s);
668

```

```

669 /**
670 \ingroup avr_stdlib
671 Successful termination for exit(); evaluates to 0.
672 */
673 #define EXIT_SUCCESS 0
674
675 /**
676 \ingroup avr_stdlib
677 Unsuccessful termination for exit(); evaluates to a non-zero value.
678 */
679 #define EXIT_FAILURE 1
680
681 /* @} */
682
683 #ifndef __DOXYGEN__
684 /* dummy declarations for libstdc++ compatibility */
685 extern int atexit(void (*)(void));
686 extern int system (const char *);
687 extern char *getenv (const char *);
688 #endif /* __DOXYGEN__ */
689
690 #ifdef __cplusplus
691 }
692 #endif
693
694 #endif /* __ASSEMBLER__ */
695
696 #endif /* _STDLIB_H_ */

```

25.56 string.h File Reference

Macros

- `#define _FFS(x)`

Functions

- int `ffs` (int __val)
- int `ffsl` (long __val)
- __extension__ int `ffsll` (long long __val)
- void * `memcpy` (void *, const void *, int, size_t)
- void * `memchr` (const void *, int, size_t) `__ATTR_PURE__`
- int `memcmp` (const void *, const void *, size_t) `__ATTR_PURE__`
- void * `memcpy` (void *, const void *, size_t)
- void * `memmem` (const void *, size_t, const void *, size_t) `__ATTR_PURE__`
- void * `memmove` (void *, const void *, size_t)
- void * `memrchr` (const void *, int, size_t) `__ATTR_PURE__`
- void * `memset` (void *, int, size_t)
- char * `strcat` (char *, const char *)
- char * `strchr` (const char *, int) `__ATTR_PURE__`
- char * `strchrnul` (const char *, int) `__ATTR_PURE__`
- int `strcmp` (const char *, const char *) `__ATTR_PURE__`
- char * `strcpy` (char *, const char *)
- int `strcasecmp` (const char *, const char *) `__ATTR_PURE__`
- char * `strcasestr` (const char *, const char *) `__ATTR_PURE__`
- size_t `strcspn` (const char *__s, const char *__reject) `__ATTR_PURE__`
- char * `strdup` (const char *s1)
- size_t `strlcat` (char *, const char *, size_t)
- size_t `strncpy` (char *, const char *, size_t)
- size_t `strlen` (const char *) `__ATTR_PURE__`
- char * `strlwr` (char *)
- char * `strncat` (char *, const char *, size_t)

- int `strncmp` (const char *, const char *, size_t) `__ATTR_PURE__`
- char * `strncpy` (char *, const char *, size_t)
- int `strncasecmp` (const char *, const char *, size_t) `__ATTR_PURE__`
- size_t `strlen` (const char *, size_t) `__ATTR_PURE__`
- char * `strpbrk` (const char *__s, const char *__accept) `__ATTR_PURE__`
- char * `strchr` (const char *, int) `__ATTR_PURE__`
- char * `strev` (char *)
- char * `strsep` (char **, const char *)
- size_t `strspn` (const char *__s, const char *__accept) `__ATTR_PURE__`
- char * `strstr` (const char *, const char *) `__ATTR_PURE__`
- char * `strtok` (char *, const char *)
- char * `strtok_r` (char *, const char *, char **)
- char * `strupr` (char *)

25.57 string.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002,2007 Marek Michalkiewicz
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30
31 /* $Id: string.h 2515 2016-02-08 22:46:42Z joerg_wunsch $ */
32
33 /*
34 string.h
35
36 Contributors:
37 Created by Marek Michalkiewicz <marekm@linux.org.pl>
38 */
39
40 #ifndef _STRING_H_
41 #define _STRING_H_ 1
42
43 #ifndef __DOXYGEN__
44 #define __need_NULL
45 #define __need_size_t
46 #include <stddef.h>
47
48 #ifndef __ATTR_PURE__
49 #define __ATTR_PURE__ __attribute__((__pure__))
50 #endif
51
52 #ifndef __ATTR_CONST__
53 #define __ATTR_CONST__ __attribute__((__const__))
54 #endif
55 #endif /* !__DOXYGEN__ */
56
57 #ifdef __cplusplus
58 extern "C" {

```

```

59 #endif
60
61 /** \file */
62 /** \defgroup avr_string <string.h>: Strings
63 \code #include <string.h> \endcode
64
65 The string functions perform string operations on NULL terminated
66 strings.
67
68 \note If the strings you are working on resident in program space (flash),
69 you will need to use the string functions described in \ref avr_pgmSPACE. */
70
71
72 /** \ingroup avr_string
73
74 This macro finds the first (least significant) bit set in the
75 input value.
76
77 This macro is very similar to the function ffs() except that
78 it evaluates its argument at compile-time, so it should only
79 be applied to compile-time constant expressions where it will
80 reduce to a constant itself.
81 Application of this macro to expressions that are not constant
82 at compile-time is not recommended, and might result in a huge
83 amount of code generated.
84
85 \returns The _FFS() macro returns the position of the first
86 (least significant) bit set in the word val, or 0 if no bits are set.
87 The least significant bit is position 1. Only 16 bits of argument
88 are evaluated.
89 */
90 #if defined(__DOXYGEN__)
91 #define _FFS(x)
92 #else /* !DOXYGEN */
93 #define _FFS(x) \
94 (1 \
95 + (((x) & 1) == 0) \
96 + (((x) & 3) == 0) \
97 + (((x) & 7) == 0) \
98 + (((x) & 017) == 0) \
99 + (((x) & 037) == 0) \
100 + (((x) & 077) == 0) \
101 + (((x) & 0177) == 0) \
102 + (((x) & 0377) == 0) \
103 + (((x) & 0777) == 0) \
104 + (((x) & 01777) == 0) \
105 + (((x) & 03777) == 0) \
106 + (((x) & 07777) == 0) \
107 + (((x) & 017777) == 0) \
108 + (((x) & 037777) == 0) \
109 + (((x) & 077777) == 0) \
110 - (((x) & 0177777) == 0) * 16)
111 #endif /* DOXYGEN */
112
113 /** \ingroup avr_string
114 \fn int ffs(int val);
115
116 \brief This function finds the first (least significant) bit set in the input value.
117
118 \returns The ffs() function returns the position of the first
119 (least significant) bit set in the word val, or 0 if no bits are set.
120 The least significant bit is position 1.
121
122 \note For expressions that are constant at compile time, consider
123 using the \ref _FFS macro instead.
124 */
125 extern int ffs(int __val) __ATTR_CONST__;
126
127 /** \ingroup avr_string
128 \fn int ffs1(long val);
129
130 \brief Same as ffs(), for an argument of type long. */
131 extern int ffs1(long __val) __ATTR_CONST__;
132
133 /** \ingroup avr_string
134 \fn int ffs11(long long val);
135
136 \brief Same as ffs(), for an argument of type long long. */
137 __extension__ extern int ffs11(long long __val) __ATTR_CONST__;
138
139 /** \ingroup avr_string
140 \fn void *memcpy(void *dest, const void *src, int val, size_t len)
141 \brief Copy memory area.
142
143 The memcpy() function copies no more than \p len bytes from memory
144 area \p src to memory area \p dest, stopping when the character \p val
145 is found.

```

```

146
147 \returns The memccpy() function returns a pointer to the next character
148 in \p dest after \p val, or NULL if \p val was not found in the first
149 \p len characters of \p src. */
150 extern void *memccpy(void *, const void *, int, size_t);
151
152 /** \ingroup avr_string
153 \fn void *memchr(const void *src, int val, size_t len)
154 \brief Scan memory for a character.
155
156 The memchr() function scans the first len bytes of the memory area pointed
157 to by src for the character val. The first byte to match val (interpreted
158 as an unsigned character) stops the operation.
159
160 \returns The memchr() function returns a pointer to the matching byte or
161 NULL if the character does not occur in the given memory area. */
162 extern void *memchr(const void *, int, size_t) __ATTR_PURE__;
163
164 /** \ingroup avr_string
165 \fn int memcmp(const void *s1, const void *s2, size_t len)
166 \brief Compare memory areas
167
168 The memcmp() function compares the first len bytes of the memory areas s1
169 and s2. The comparison is performed using unsigned char operations.
170
171 \returns The memcmp() function returns an integer less than, equal to, or
172 greater than zero if the first len bytes of s1 is found, respectively, to be
173 less than, to match, or be greater than the first len bytes of s2.
174
175 \note Be sure to store the result in a 16 bit variable since you may get
176 incorrect results if you use an unsigned char or char due to truncation.
177
178 \warning This function is not -mint8 compatible, although if you only care
179 about testing for equality, this function should be safe to use. */
180 extern int memcmp(const void *, const void *, size_t) __ATTR_PURE__;
181
182 /** \ingroup avr_string
183 \fn void *memcpy(void *dest, const void *src, size_t len)
184 \brief Copy a memory area.
185
186 The memcpy() function copies len bytes from memory area src to memory area
187 dest. The memory areas may not overlap. Use memmove() if the memory
188 areas do overlap.
189
190 \returns The memcpy() function returns a pointer to dest. */
191 extern void *memcpy(void *, const void *, size_t);
192
193 /** \ingroup avr_string
194 \fn void *memmem(const void *s1, size_t len1, const void *s2, size_t len2)
195
196 The memmem() function finds the start of the first occurrence of the
197 substring \p s2 of length \p len2 in the memory area \p s1 of length
198 \p len1.
199
200 \return The memmem() function returns a pointer to the beginning of
201 the substring, or \c NULL if the substring is not found. If \p len2
202 is zero, the function returns \p s1. */
203 extern void *memmem(const void *, size_t, const void *, size_t) __ATTR_PURE__;
204
205 /** \ingroup avr_string
206 \fn void *memmove(void *dest, const void *src, size_t len)
207 \brief Copy memory area.
208
209 The memmove() function copies len bytes from memory area src to memory area
210 dest. The memory areas may overlap.
211
212 \returns The memmove() function returns a pointer to dest. */
213 extern void *memmove(void *, const void *, size_t);
214
215 /** \ingroup avr_string
216 \fn void *memrchr(const void *src, int val, size_t len)
217
218 The memrchr() function is like the memchr() function, except that it
219 searches backwards from the end of the \p len bytes pointed to by \p
220 src instead of forwards from the front. (Glibc, GNU extension.)
221
222 \return The memrchr() function returns a pointer to the matching
223 byte or \c NULL if the character does not occur in the given memory
224 area. */
225 extern void *memrchr(const void *, int, size_t) __ATTR_PURE__;
226
227 /** \ingroup avr_string
228 \fn void *memset(void *dest, int val, size_t len)
229 \brief Fill memory with a constant byte.
230
231 The memset() function fills the first len bytes of the memory area pointed
232 to by dest with the constant byte val.

```

```

233
234 \returns The memset() function returns a pointer to the memory area dest. */
235 extern void *memset(void *, int, size_t);
236
237 /** \ingroup avr_string
238 \fn char *strcat(char *dest, const char *src)
239 \brief Concatenate two strings.
240
241 The strcat() function appends the src string to the dest string
242 overwriting the '\\0' character at the end of dest, and then adds a
243 terminating '\\0' character. The strings may not overlap, and the dest
244 string must have enough space for the result.
245
246 \returns The strcat() function returns a pointer to the resulting string
247 dest. */
248 extern char *strcat(char *, const char *);
249
250 /** \ingroup avr_string
251 \fn char *strchr(const char *src, int val)
252 \brief Locate character in string.
253
254 The strchr() function returns a pointer to the first occurrence of
255 the character \p val in the string \p src.
256
257 Here "character" means "byte" - these functions do not work with
258 wide or multi-byte characters.
259
260 \returns The strchr() function returns a pointer to the matched
261 character or \c NULL if the character is not found. */
262 extern char *strchr(const char *, int) __ATTR_PURE__;
263
264 /** \ingroup avr_string
265 \fn char *strchrnul(const char *s, int c)
266
267 The strchrnul() function is like strchr() except that if \p c is not
268 found in \p s, then it returns a pointer to the null byte at the end
269 of \p s, rather than \c NULL. (Glibc, GNU extension.)
270
271 \return The strchrnul() function returns a pointer to the matched
272 character, or a pointer to the null byte at the end of \p s (i.e.,
273 \c s[strlen(s)) if the character is not found. */
274 extern char *strchrnul(const char *, int) __ATTR_PURE__;
275
276 /** \ingroup avr_string
277 \fn int strcmp(const char *s1, const char *s2)
278 \brief Compare two strings.
279
280 The strcmp() function compares the two strings \p s1 and \p s2.
281
282 \returns The strcmp() function returns an integer less than, equal
283 to, or greater than zero if \p s1 is found, respectively, to be less
284 than, to match, or be greater than \p s2. A consequence of the
285 ordering used by strcmp() is that if \p s1 is an initial substring
286 of \p s2, then \p s1 is considered to be "less than" \p s2. */
287 extern int strcmp(const char *, const char *) __ATTR_PURE__;
288
289 /** \ingroup avr_string
290 \fn char *strcpy(char *dest, const char *src)
291 \brief Copy a string.
292
293 The strcpy() function copies the string pointed to by src (including the
294 terminating '\\0' character) to the array pointed to by dest. The strings
295 may not overlap, and the destination string dest must be large enough to
296 receive the copy.
297
298 \returns The strcpy() function returns a pointer to the destination
299 string dest.
300
301 \note If the destination string of a strcpy() is not large enough (that
302 is, if the programmer was stupid/lazy, and failed to check the size before
303 copying) then anything might happen. Overflowing fixed length strings is
304 a favourite cracker technique. */
305 extern char *strcpy(char *, const char *);
306
307 /** \ingroup avr_string
308 \fn int strcasecmp(const char *s1, const char *s2)
309 \brief Compare two strings ignoring case.
310
311 The strcasecmp() function compares the two strings \p s1 and \p s2,
312 ignoring the case of the characters.
313
314 \returns The strcasecmp() function returns an integer less than,
315 equal to, or greater than zero if \p s1 is found, respectively, to
316 be less than, to match, or be greater than \p s2. A consequence of
317 the ordering used by strcasecmp() is that if \p s1 is an initial
318 substring of \p s2, then \p s1 is considered to be "less than"
319 \p s2. */

```



```

320 extern int strcasecmp(const char *, const char *) __ATTR_PURE__;
321
322 /** \ingroup avr_string
323 \fn char *strcasestr(const char *s1, const char *s2)
324
325 The strcasestr() function finds the first occurrence of the
326 substring \p s2 in the string \p s1. This is like strstr(), except
327 that it ignores case of alphabetic symbols in searching for the
328 substring. (Glibc, GNU extension.)
329
330 \return The strcasestr() function returns a pointer to the beginning
331 of the substring, or \c NULL if the substring is not found. If \p s2
332 points to a string of zero length, the function returns \p s1. */
333 extern char *strcasestr(const char *, const char *) __ATTR_PURE__;
334
335 /** \ingroup avr_string
336 \fn size_t strcspn(const char *s, const char *reject)
337
338 The strcspn() function calculates the length of the initial segment
339 of \p s which consists entirely of characters not in \p reject.
340
341 \return The strcspn() function returns the number of characters in
342 the initial segment of \p s which are not in the string \p reject.
343 The terminating zero is not considered as a part of string. */
344 extern size_t strcspn(const char *__s, const char *__reject) __ATTR_PURE__;
345
346 /** \ingroup avr_string
347 \fn char *strdup(const char *s1)
348 \brief Duplicate a string.
349
350 The strdup() function allocates memory and copies into it the string
351 addressed by s1, including the terminating null character.
352
353 \warning The strdup() function calls malloc() to allocate the memory
354 for the duplicated string! The user is responsible for freeing the
355 memory by calling free().
356
357 \returns The strdup() function returns a pointer to the resulting string
358 dest. If malloc() cannot allocate enough storage for the string, strdup()
359 will return NULL.
360
361 \warning Be sure to check the return value of the strdup() function to
362 make sure that the function has succeeded in allocating the memory!
363 */
364 extern char *strdup(const char *s1);
365
366 /** \ingroup avr_string
367 \fn size_t strlcat(char *dst, const char *src, size_t siz)
368 \brief Concatenate two strings.
369
370 Appends \p src to string \p dst of size \p siz (unlike strncat(),
371 \p siz is the full size of \p dst, not space left). At most \p siz-1
372 characters will be copied. Always NULL terminates (unless \p siz <=
373 \p strlen(dst)).
374
375 \returns The strlcat() function returns strlen(src) + MIN(siz,
376 strlen(initial dst)). If retval >= siz, truncation occurred. */
377 extern size_t strlcat(char *, const char *, size_t);
378
379 /** \ingroup avr_string
380 \fn size_t strlcpy(char *dst, const char *src, size_t siz)
381 \brief Copy a string.
382
383 Copy \p src to string \p dst of size \p siz. At most \p siz-1
384 characters will be copied. Always NULL terminates (unless \p siz == 0).
385
386 \returns The strlcpy() function returns strlen(src). If retval >= siz,
387 truncation occurred. */
388 extern size_t strlcpy(char *, const char *, size_t);
389
390 /** \ingroup avr_string
391 \fn size_t strlen(const char *src)
392 \brief Calculate the length of a string.
393
394 The strlen() function calculates the length of the string src, not
395 including the terminating '\\0' character.
396
397 \returns The strlen() function returns the number of characters in
398 src. */
399 extern size_t strlen(const char *) __ATTR_PURE__;
400
401 /** \ingroup avr_string
402 \fn char *strlwr(char *s)
403 \brief Convert a string to lower case.
404
405 The strlwr() function will convert a string to lower case. Only the upper
406 case alphabetic characters [A .. Z] are converted. Non-alphabetic

```

```

407 characters will not be changed.
408
409 \returns The strlwr() function returns a pointer to the converted
410 string. */
411 extern char *strlwr(char *);
412
413 /** \ingroup avr_string
414 \fn char *strncat(char *dest, const char *src, size_t len)
415 \brief Concatenate two strings.
416
417 The strncat() function is similar to strcat(), except that only the first
418 n characters of src are appended to dest.
419
420 \returns The strncat() function returns a pointer to the resulting string
421 dest. */
422 extern char *strncat(char *, const char *, size_t);
423
424 /** \ingroup avr_string
425 \fn int strncmp(const char *s1, const char *s2, size_t len)
426 \brief Compare two strings.
427
428 The strncmp() function is similar to strcmp(), except it only compares the
429 first (at most) n characters of s1 and s2.
430
431 \returns The strncmp() function returns an integer less than, equal to, or
432 greater than zero if s1 (or the first n bytes thereof) is found,
433 respectively, to be less than, to match, or be greater than s2. */
434 extern int strncmp(const char *, const char *, size_t) __ATTR_PURE__;
435
436 /** \ingroup avr_string
437 \fn char *strncpy(char *dest, const char *src, size_t len)
438 \brief Copy a string.
439
440 The strncpy() function is similar to strcpy(), except that not more than n
441 bytes of src are copied. Thus, if there is no null byte among the first n
442 bytes of src, the result will not be null-terminated.
443
444 In the case where the length of src is less than that of n, the remainder
445 of dest will be padded with nulls.
446
447 \returns The strncpy() function returns a pointer to the destination
448 string dest. */
449 extern char *strncpy(char *, const char *, size_t);
450
451 /** \ingroup avr_string
452 \fn int strncasecmp(const char *s1, const char *s2, size_t len)
453 \brief Compare two strings ignoring case.
454
455 The strncasecmp() function is similar to strcasecmp(), except it
456 only compares the first \p len characters of \p s1.
457
458 \returns The strncasecmp() function returns an integer less than,
459 equal to, or greater than zero if \p s1 (or the first \p len bytes
460 thereof) is found, respectively, to be less than, to match, or be
461 greater than \p s2. A consequence of the ordering used by
462 strncasecmp() is that if \p s1 is an initial substring of \p s2,
463 then \p s1 is considered to be "less than" \p s2. */
464 extern int strncasecmp(const char *, const char *, size_t) __ATTR_PURE__;
465
466 /** \ingroup avr_string
467 \fn size_t strlen(const char *src, size_t len)
468 \brief Determine the length of a fixed-size string.
469
470 The strlen function returns the number of characters in the string
471 pointed to by src, not including the terminating '\\0' character, but at
472 most len. In doing this, strlen looks only at the first len characters at
473 src and never beyond src+len.
474
475 \returns The strlen function returns strlen(src), if that is less than
476 len, or len if there is no '\\0' character among the first len
477 characters pointed to by src. */
478 extern size_t strlen(const char *, size_t) __ATTR_PURE__;
479
480 /** \ingroup avr_string
481 \fn char *strpbrk(const char *s, const char *accept)
482
483 The strpbrk() function locates the first occurrence in the string
484 \p s of any of the characters in the string \p accept.
485
486 \return The strpbrk() function returns a pointer to the character
487 in \p s that matches one of the characters in \p accept, or \c NULL
488 if no such character is found. The terminating zero is not
489 considered as a part of string: if one or both args are empty, the
490 result will be \c NULL. */
491 extern char *strpbrk(const char *__s, const char *__accept) __ATTR_PURE__;
492
493 /** \ingroup avr_string

```

```

494 \fn char *strrchr(const char *src, int val)
495 \brief Locate character in string.
496
497 The strrchr() function returns a pointer to the last occurrence of the
498 character val in the string src.
499
500 Here "character" means "byte" - these functions do not work with wide or
501 multi-byte characters.
502
503 \returns The strrchr() function returns a pointer to the matched character
504 or NULL if the character is not found. */
505 extern char *strrchr(const char *, int) __ATTR_PURE__;
506
507 /** \ingroup avr_string
508 \fn char *strrev(char *s)
509 \brief Reverse a string.
510
511 The strrev() function reverses the order of the string.
512
513 \returns The strrev() function returns a pointer to the beginning of the
514 reversed string. */
515 extern char *strrev(char *);
516
517 /** \ingroup avr_string
518 \fn char *strsep(char **sp, const char *delim)
519 \brief Parse a string into tokens.
520
521 The strsep() function locates, in the string referenced by \p *sp,
522 the first occurrence of any character in the string \p delim (or the
523 terminating '\\0' character) and replaces it with a '\\0'. The
524 location of the next character after the delimiter character (or \c
525 NULL, if the end of the string was reached) is stored in \p *sp. An
526 "empty" field, i.e. one caused by two adjacent delimiter
527 characters, can be detected by comparing the location referenced by
528 the pointer returned in \p *sp to '\\0'.
529
530 \return The strsep() function returns a pointer to the original
531 value of \p *sp. If \p *sp is initially \c NULL, strsep() returns
532 \c NULL. */
533 extern char *strsep(char **, const char *);
534
535 /** \ingroup avr_string
536 \fn size_t strspn(const char *s, const char *accept)
537
538 The strspn() function calculates the length of the initial segment
539 of \p s which consists entirely of characters in \p accept.
540
541 \return The strspn() function returns the number of characters in
542 the initial segment of \p s which consist only of characters from \p
543 accept. The terminating zero is not considered as a part of string. */
544 extern size_t strspn(const char *__s, const char *__accept) __ATTR_PURE__;
545
546 /** \ingroup avr_string
547 \fn char *strstr(const char *s1, const char *s2)
548 \brief Locate a substring.
549
550 The strstr() function finds the first occurrence of the substring \p
551 s2 in the string \p s1. The terminating '\\0' characters are not
552 compared.
553
554 \returns The strstr() function returns a pointer to the beginning of
555 the substring, or \c NULL if the substring is not found. If \p s2
556 points to a string of zero length, the function returns \p s1. */
557 extern char *strstr(const char *, const char *) __ATTR_PURE__;
558
559 /** \ingroup avr_string
560 \fn char *strtok(char *s, const char *delim)
561 \brief Parses the string s into tokens.
562
563 strtok parses the string s into tokens. The first call to strtok
564 should have s as its first argument. Subsequent calls should have
565 the first argument set to NULL. If a token ends with a delimiter, this
566 delimiting character is overwritten with a '\\0' and a pointer to the next
567 character is saved for the next call to strtok. The delimiter string
568 delim may be different for each call.
569
570 \returns The strtok() function returns a pointer to the next token or
571 NULL when no more tokens are found.
572
573 \note strtok() is NOT reentrant. For a reentrant version of this function
574 see \c strtok_r().
575 */
576 extern char *strtok(char *, const char *);
577
578 /** \ingroup avr_string
579 \fn char *strtok_r(char *string, const char *delim, char **last)
580 \brief Parses string into tokens.

```

```

581
582 strtok_r parses string into tokens. The first call to strtok_r
583 should have string as its first argument. Subsequent calls should have
584 the first argument set to NULL. If a token ends with a delimiter, this
585 delimiting character is overwritten with a '\\0' and a pointer to the next
586 character is saved for the next call to strtok_r. The delimiter string
587 \p delim may be different for each call. \p last is a user allocated char*
588 pointer. It must be the same while parsing the same string. strtok_r is
589 a reentrant version of strtok().
590
591 \returns The strtok_r() function returns a pointer to the next token or
592 NULL when no more tokens are found. */
593 extern char *strtok_r(char *, const char *, char **);
594
595 /** \ingroup avr_string
596 \fn char *strupr(char *s)
597 \brief Convert a string to upper case.
598
599 The strupr() function will convert a string to upper case. Only the lower
600 case alphabetic characters [a .. z] are converted. Non-alphabetic
601 characters will not be changed.
602
603 \returns The strupr() function returns a pointer to the converted
604 string. The pointer is the same as that passed in since the operation is
605 perform in place. */
606 extern char *strupr(char *);
607
608 #ifndef __DOXYGEN__
609 /* libstdc++ compatibility, dummy declarations */
610 extern int strcoll(const char *s1, const char *s2);
611 extern char *strerror(int errnum);
612 extern size_t strxfrm(char *dest, const char *src, size_t n);
613 #endif /* !__DOXYGEN__ */
614
615 #ifdef __cplusplus
616 }
617 #endif
618
619 #endif /* _STRING_H_ */
620

```

25.58 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](#) *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)
- double [solar_declination](#) (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)

25.58.1 Macro Definition Documentation

25.58.1.1 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;
```

25.58.1.2 ONE_DAY `#define ONE_DAY 86400`

One day, expressed in seconds

25.58.1.3 ONE_DEGREE `#define ONE_DEGREE 3600`

Angular degree, expressed in arc seconds

25.58.1.4 ONE_HOUR `#define ONE_HOUR 3600`

One hour, expressed in seconds

25.58.1.5 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;
```

25.58.2 Enumeration Type Documentation**25.58.2.1 _MONTHS_** `enum _MONTHS_`

Enumerated labels for the months.

25.58.2.2 _WEEK_DAYS_ `enum _WEEK_DAYS_`

Enumerated labels for the days of the week.

25.58.3 Function Documentation**25.58.3.1 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

25.58.3.2 asctime_r() `void asctime_r (
const struct tm * timeptr,
char * buf)`

Re entrant version of [asctime\(\)](#).

25.58.3.3 ctime() `char * ctime (`
 `const time_t * timer)`

The ctime function is equivalent to `asctime(localtime(timer))`

25.58.3.4 ctime_r() `void ctime_r (`
 `const time_t * timer,`
 `char * buf)`

Re entrant version of `ctime()`.

25.58.3.5 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.

25.58.3.6 difftime() `int32_t difftime (`
 `time_t time1,`
 `time_t time0)`

The difftime function returns the difference between two binary time stamps, `time1 - time0`.

25.58.3.7 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.

25.58.3.8 fatfs_time() `uint32_t fatfs_time (`
 `const struct tm * timeptr)`

Convert a Y2K time stamp into a FAT file system time stamp.

25.58.3.9 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.

25.58.3.10 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.

25.58.3.11 gmtime_r() `void gmtime_r (`
 `const time_t * timer,`
 `struct tm * timeptr)`

Re entrant version of [gmtime\(\)](#).

25.58.3.12 is_leap_year() `uint8_t is_leap_year (`
 `int16_t year)`

Return 1 if year is a leap year, zero if it is not.

25.58.3.13 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 for more information.

25.58.3.14 iso_week_date_r() `void iso_week_date_r (`
 `int year,`
 `int yday,`
 `struct week_date * iso)`

Re-entrant version of [iso-week_date](#).

25.58.3.15 isotime() `char * isotime (`
 `const struct tm * tmptr)`

The isotime function constructs an ascii string in the form
2013-03-23 01:03:52

25.58.3.16 isotime_r() `void isotime_r (`
 `const struct tm * tmptr,`
 `char * buffer)`

Re entrant version of [isotime\(\)](#)

25.58.3.17 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.

25.58.3.18 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.

25.58.3.19 localtime_r() `void localtime_r (`
 `const time_t * timer,`
 `struct tm * timeptr)`

Re entrant version of [localtime\(\)](#).

25.58.3.20 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.

25.58.3.21 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.

On successful completion, the values of all elements of timeptr are set to the appropriate range.

25.58.3.22 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.

25.58.3.23 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.

25.58.3.24 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.

25.58.3.25 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);
```

25.58.3.26 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);
```

25.58.3.27 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.

25.58.3.28 solar_declination() double solar_declination (
 const time_t * timer)

Returns the declination of the sun in radians.

25.58.3.29 solar_noon() time_t solar_noon (
 const time_t * timer)

Computes the time of solar noon, at the location of the observer.

25.58.3.30 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.

25.58.3.31 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\(\)](#).

25.58.3.32 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\(\)](#).

25.58.3.33 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();
}
```

25.58.3.34 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.

25.58.3.35 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.

25.58.3.36 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.

25.59 time.h

[Go to the documentation of this file.](#)

```

1 /*
2  * (C)2012 Michael Duane Rice All rights reserved.
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are
6  * met:
7  *
8  * Redistributions of source code must retain the above copyright notice, this
9  * list of conditions and the following disclaimer. Redistributions in binary
10 * form must reproduce the above copyright notice, this list of conditions
11 * and the following disclaimer in the documentation and/or other materials
12 * provided with the distribution. Neither the name of the copyright holders
13 * nor the names of contributors may be used to endorse or promote products
14 * derived from this software without specific prior written permission.
15 *
16 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
17 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
18 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
19 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
20 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
21 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
22 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
23 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
24 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
26 * POSSIBILITY OF SUCH DAMAGE.
27 */
28
29 /* $Id: time.h 2503 2016-02-07 22:59:47Z joerg_wunsch $ */
30
31 /** \file */
32
33 /** \defgroup avr_time <time.h>: Time
34 \code #include <time.h> \endcode
35 <h3>Introduction to the Time functions</h3>
36 This file declares the time functions implemented in \c avr-libc.
37
38 The implementation aspires to conform with ISO/IEC 9899 (C90). However, due to limitations of the
39 target processor and the nature of its development environment, a practical implementation must
40 of necessity deviate from the standard.
41
42
43
44 Section 7.23.2.1 clock()
45 The type clock_t, the macro CLOCKS_PER_SEC, and the function clock() are not implemented. We
46 consider these items belong to operating system code, or to application code when no operating
47 system is present.
48
49 Section 7.23.2.3 mktime()
50 The standard specifies that mktime() should return (time_t) -1, if the time cannot be represented.
51 This implementation always returns a 'best effort' representation.
52
53 Section 7.23.2.4 time()
54 The standard specifies that time() should return (time_t) -1, if the time is not available.
55 Since the application must initialize the time system, this functionality is not implemented.
56
57 Section 7.23.2.2, difftime()
58 Due to the lack of a 64 bit double, the function difftime() returns a long integer. In most cases
59 this change will be invisible to the user, handled automatically by the compiler.
60
61 Section 7.23.1.4 struct tm
62 Per the standard, struct tm->tm_isdst is greater than zero when Daylight Saving time is in effect.
63 This implementation further specifies that, when positive, the value of tm_isdst represents
64 the amount time is advanced during Daylight Saving time.
65
66 Section 7.23.3.5 strptime()
67 Only the 'C' locale is supported, therefore the modifiers 'E' and 'O' are ignored.
68 The 'Z' conversion is also ignored, due to the lack of time zone name.
69
70 In addition to the above departures from the standard, there are some behaviors which are different
71 from what is often expected, though allowed under the standard.
72
73 There is no 'platform standard' method to obtain the current time, time zone, or
74 daylight savings 'rules' in the AVR environment. Therefore the application must initialize
75 the time system with this information. The functions set_zone(), set_dst(), and
76 set_system_time() are provided for initialization. Once initialized, system time is maintained by
77 calling the function system_tick() at one second intervals.
78
79 Though not specified in the standard, it is often expected that time_t is a signed integer
80 representing an offset in seconds from Midnight Jan 1 1970... i.e. 'Unix time'. This implementation
81 uses an unsigned 32 bit integer offset from Midnight Jan 1 2000. The use of this 'epoch' helps to
82 simplify the conversion functions, while the 32 bit value allows time to be properly represented
83 until Tue Feb 7 06:28:15 2136 UTC. The macros UNIX_OFFSET and NTP_OFFSET are defined to assist in

```

```

84 converting to and from Unix and NTP time stamps.
85
86 Unlike desktop counterparts, it is impractical to implement or maintain the 'zoneinfo' database.
87 Therefore no attempt is made to account for time zone, daylight saving, or leap seconds in past dates.
88 All calculations are made according to the currently configured time zone and daylight saving 'rule'.
89
90 In addition to C standard functions, re-entrant versions of ctime(), asctime(), gmtime() and
91 localtime() are provided which, in addition to being re-entrant, have the property of claiming
92 less permanent storage in RAM. An additional time conversion, isotime() and its re-entrant version,
93 uses far less storage than either ctime() or asctime().
94
95 Along with the usual smattering of utility functions, such as is_leap_year(), this library includes
96 a set of functions related the sun and moon, as well as sidereal time functions.
97 */
98
99 #ifndef TIME_H
100 #define TIME_H
101
102 #ifdef __cplusplus
103 extern      "C" {
104 #endif
105
106 #include <inttypes.h>
107 #include <stdlib.h>
108
109 /** \ingroup avr_time */
110 /* @{ */
111
112 /**
113 time_t represents seconds elapsed from Midnight, Jan 1 2000 UTC (the Y2K 'epoch').
114 Its range allows this implementation to represent time up to Tue Feb 7 06:28:15 2136 UTC.
115 */
116     typedef uint32_t time_t;
117
118 /**
119 The time function returns the systems current time stamp.
120 If timer is not a null pointer, the return value is also assigned to the object it points to.
121 */
122     time_t      time(time_t *timer);
123
124 /**
125 The difftime function returns the difference between two binary time stamps,
126 time1 - time0.
127 */
128     int32_t      difftime(time_t time1, time_t time0);
129
130
131 /**
132 The tm structure contains a representation of time 'broken down' into components of the
133 Gregorian calendar.
134
135 The value of tm_isdst is zero if Daylight Saving Time is not in effect, and is negative if
136 the information is not available.
137
138 When Daylight Saving Time is in effect, the value represents the number of
139 seconds the clock is advanced.
140
141 See the set_dst() function for more information about Daylight Saving.
142 */
143 */
144     struct tm {
145         int8_t      tm_sec; /**< seconds after the minute - [ 0 to 59 ] */
146         int8_t      tm_min; /**< minutes after the hour - [ 0 to 59 ] */
147         int8_t      tm_hour; /**< hours since midnight - [ 0 to 23 ] */
148         int8_t      tm_mday; /**< day of the month - [ 1 to 31 ] */
149         int8_t      tm_wday; /**< days since Sunday - [ 0 to 6 ] */
150         int8_t      tm_mon; /**< months since January - [ 0 to 11 ] */
151         int16_t     tm_year; /**< years since 1900 */
152         int16_t     tm_yday; /**< days since January 1 - [ 0 to 365 ] */
153         int16_t     tm_isdst; /**< Daylight Saving Time flag */
154     };
155
156 #ifndef __DOXYGEN__
157     /* We have to provide clock_t / CLOCKS_PER_SEC so that libstdc++-v3 can
158     be built. We define CLOCKS_PER_SEC via a symbol _CLOCKS_PER_SEC_
159     so that the user can provide the value on the link line, which should
160     result in little or no run-time overhead compared with a constant. */
161     typedef unsigned long clock_t;
162     extern char *_CLOCKS_PER_SEC_;
163     #define CLOCKS_PER_SEC ((clock_t) _CLOCKS_PER_SEC_)
164     extern clock_t clock(void);
165 #endif /* !__DOXYGEN__ */
166
167 /**
168 This function 'compiles' the elements of a broken-down time structure, returning a binary time stamp.
169 The elements of timeptr are interpreted as representing Local Time.
170

```

```

171 The original values of the tm_wday and tm_yday elements of the structure are ignored,
172 and the original values of the other elements are not restricted to the ranges stated for struct tm.
173
174 On successful completion, the values of all elements of timeptr are set to the appropriate range.
175 */
176     time_t          mktime(struct tm * timeptr);
177
178 /**
179 This function 'compiles' the elements of a broken-down time structure, returning a binary time stamp.
180 The elements of timeptr are interpreted as representing UTC.
181
182 The original values of the tm_wday and tm_yday elements of the structure are ignored,
183 and the original values of the other elements are not restricted to the ranges stated for struct tm.
184
185 Unlike mktime(), this function DOES NOT modify the elements of timeptr.
186 */
187     time_t          mk_gmtime(const struct tm * timeptr);
188
189 /**
190 The gmtime function converts the time stamp pointed to by timer into broken-down time,
191 expressed as UTC.
192 */
193     struct tm       *gmtime(const time_t * timer);
194
195 /**
196 Re entrant version of gmtime().
197 */
198     void            gmtime_r(const time_t * timer, struct tm * timeptr);
199
200 /**
201 The localtime function converts the time stamp pointed to by timer into broken-down time,
202 expressed as Local time.
203 */
204     struct tm       *localtime(const time_t * timer);
205
206 /**
207 Re entrant version of localtime().
208 */
209     void            localtime_r(const time_t * timer, struct tm * timeptr);
210
211 /**
212 The asctime function converts the broken-down time of timeptr, into an ascii string in the form
213
214 Sun Mar 23 01:03:52 2013
215 */
216     char            *asctime(const struct tm * timeptr);
217
218 /**
219 Re entrant version of asctime().
220 */
221     void            asctime_r(const struct tm * timeptr, char *buf);
222
223 /**
224 The ctime function is equivalent to asctime(localtime(timer))
225 */
226     char            *ctime(const time_t * timer);
227
228 /**
229 Re entrant version of ctime().
230 */
231     void            ctime_r(const time_t * timer, char *buf);
232
233 /**
234 The isotime function constructs an ascii string in the form
235 \code2013-03-23 01:03:52\endcode
236 */
237     char            *isotime(const struct tm * tmptr);
238
239 /**
240 Re entrant version of isotime()
241 */
242     void            isotime_r(const struct tm *, char *);
243
244 /**
245 A complete description of strftime() is beyond the pale of this document.
246 Refer to ISO/IEC document 9899 for details.
247
248 All conversions are made using the 'C Locale', ignoring the E or O modifiers. Due to the lack of
249 a time zone 'name', the 'Z' conversion is also ignored.
250 */
251     size_t          strftime(char *s, size_t maxsize, const char *format, const struct tm * timeptr);
252
253 /**
254 Specify the Daylight Saving function.
255
256 The Daylight Saving function should examine its parameters to determine whether
257 Daylight Saving is in effect, and return a value appropriate for tm_isdst.

```

```

258
259 Working examples for the USA and the EU are available..
260
261 \code #include <util/eu_dst.h>\endcode
262 for the European Union, and
263 \code #include <util/usa_dst.h>\endcode
264 for the United States
265
266 If a Daylight Saving function is not specified, the system will ignore Daylight Saving.
267 */
268 void set_dst(int (*) (const time_t *, int32_t *));
269
270 /**
271 Set the 'time zone'. The parameter is given in seconds East of the Prime Meridian.
272 Example for New York City:
273 \code set_zone(-5 * ONE_HOUR);\endcode
274
275 If the time zone is not set, the time system will operate in UTC only.
276 */
277 void set_zone(int32_t);
278
279 /**
280 Initialize the system time. Examples are...
281
282 From a Clock / Calendar type RTC:
283 \code
284 struct tm rtc_time;
285
286 read_rtc(&rtc_time);
287 rtc_time.tm_isdst = 0;
288 set_system_time( mktime(&rtc_time) );
289 \endcode
290
291 From a Network Time Protocol time stamp:
292 \code
293 set_system_time(ntp_timestamp - NTP_OFFSET);
294 \endcode
295
296 From a UNIX time stamp:
297 \code
298 set_system_time(unix_timestamp - UNIX_OFFSET);
299 \endcode
300
301 */
302 void set_system_time(time_t timestamp);
303
304 /**
305 Maintain the system time by calling this function at a rate of 1 Hertz.
306
307 It is anticipated that this function will typically be called from within an
308 Interrupt Service Routine, (though that is not required). It therefore includes code which
309 makes it simple to use from within a 'Naked' ISR, avoiding the cost of saving and restoring
310 all the cpu registers.
311
312 Such an ISR may resemble the following example...
313 \code
314 ISR(RTC_OVF_vect, ISR_NAKED)
315 {
316 system_tick();
317 reti();
318 }
319 \endcode
320 */
321 void system_tick(void);
322
323 /**
324 Enumerated labels for the days of the week.
325 */
326 enum _WEEK_DAYS_ {
327     SUNDAY,
328     MONDAY,
329     TUESDAY,
330     WEDNESDAY,
331     THURSDAY,
332     FRIDAY,
333     SATURDAY
334 };
335
336 /**
337 Enumerated labels for the months.
338 */
339 enum _MONTHS_ {
340     JANUARY,
341     FEBRUARY,
342     MARCH,
343     APRIL,
344     MAY,

```

```

345     JUNE,
346     JULY,
347     AUGUST,
348     SEPTEMBER,
349     OCTOBER,
350     NOVEMBER,
351     DECEMBER
352 };
353
354 /**
355 Return 1 if year is a leap year, zero if it is not.
356 */
357     uint8_t         is_leap_year(int16_t year);
358
359 /**
360 Return the length of month, given the year and month, where month is in the range 1 to 12.
361 */
362     uint8_t         month_length(int16_t year, uint8_t month);
363
364 /**
365 Return the calendar week of year, where week 1 is considered to begin on the
366 day of week specified by 'start'. The returned value may range from zero to 52.
367 */
368     uint8_t         week_of_year(const struct tm * timeptr, uint8_t start);
369
370 /**
371 Return the calendar week of month, where the first week is considered to begin on the
372 day of week specified by 'start'. The returned value may range from zero to 5.
373 */
374     uint8_t         week_of_month(const struct tm * timeptr, uint8_t start);
375
376 /**
377 Structure which represents a date as a year, week number of that year, and day of week.
378 See http://en.wikipedia.org/wiki/ISO\_week\_date for more information.
379 */
380     struct week_date {
381         int year; /**< year number (Gregorian calendar) */
382         int week; /**< week number (#1 is where first Thursday is in) */
383         int day; /**< day within week */
384     };
385
386 /**
387 Return a week_date structure with the ISO_8601 week based date corresponding to the given
388 year and day of year. See http://en.wikipedia.org/wiki/ISO\_week\_date for more
389 information.
390 */
391     struct week_date * iso_week_date( int year, int yday);
392
393 /**
394 Re-entrant version of iso-week_date.
395 */
396     void iso_week_date_r( int year, int yday, struct week_date *);
397
398 /**
399 Convert a Y2K time stamp into a FAT file system time stamp.
400 */
401     uint32_t         fatfs_time(const struct tm * timeptr);
402
403 /** One hour, expressed in seconds */
404 #define ONE_HOUR 3600
405
406 /** Angular degree, expressed in arc seconds */
407 #define ONE_DEGREE 3600
408
409 /** One day, expressed in seconds */
410 #define ONE_DAY 86400
411
412 /** Difference between the Y2K and the UNIX epochs, in seconds. To convert a Y2K
413 timestamp to UNIX...
414 \code
415 long unix;
416 time_t y2k;
417
418 y2k = time(NULL);
419 unix = y2k + UNIX_OFFSET;
420 \endcode
421 */
422 #define UNIX_OFFSET 946684800
423
424 /** Difference between the Y2K and the NTP epochs, in seconds. To convert a Y2K
425 timestamp to NTP...
426 \code
427 unsigned long ntp;
428 time_t y2k;
429
430 y2k = time(NULL);
431 ntp = y2k + NTP_OFFSET;

```



```

432 \endcode
433 */
434 #define NTP_OFFSET 3155673600
435
436 /*
437 * =====
438 *                      Ephemeris
439 */
440
441 /**
442 Set the geographic coordinates of the 'observer', for use with several of the
443 following functions. Parameters are passed as seconds of North Latitude, and seconds
444 of East Longitude.
445
446 For New York City...
447 \code set_position( 40.7142 * ONE_DEGREE, -74.0064 * ONE_DEGREE); \endcode
448 */
449 void          set_position(int32_t latitude, int32_t longitude);
450
451 /**
452 Computes the difference between apparent solar time and mean solar time.
453 The returned value is in seconds.
454 */
455 int16_t       equation_of_time(const time_t * timer);
456
457 /**
458 Computes the amount of time the sun is above the horizon, at the location of the observer.
459
460 NOTE: At observer locations inside a polar circle, this value can be zero during the winter,
461 and can exceed ONE_DAY during the summer.
462
463 The returned value is in seconds.
464 */
465 int32_t       daylight_seconds(const time_t * timer);
466
467 /**
468 Computes the time of solar noon, at the location of the observer.
469 */
470 time_t        solar_noon(const time_t * timer);
471
472 /**
473 Return the time of sunrise, at the location of the observer. See the note about daylight_seconds().
474 */
475 time_t        sun_rise(const time_t * timer);
476
477 /**
478 Return the time of sunset, at the location of the observer. See the note about daylight_seconds().
479 */
480 time_t        sun_set(const time_t * timer);
481
482 /** Returns the declination of the sun in radians. */
483 double        solar_declination(const time_t * timer);
484
485 /**
486 Returns an approximation to the phase of the moon.
487 The sign of the returned value indicates a waning or waxing phase.
488 The magnitude of the returned value indicates the percentage illumination.
489 */
490 int8_t        moon_phase(const time_t * timer);
491
492 /**
493 Returns Greenwich Mean Sidereal Time, as seconds into the sidereal day.
494 The returned value will range from 0 through 86399 seconds.
495 */
496 unsigned long gm_sidereal(const time_t * timer);
497
498 /**
499 Returns Local Mean Sidereal Time, as seconds into the sidereal day.
500 The returned value will range from 0 through 86399 seconds.
501 */
502 unsigned long lm_sidereal(const time_t * timer);
503
504 /* @} */
505 #ifdef __cplusplus
506 }
507 #endif
508
509 #endif          /* TIME_H */

```

25.60 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`

25.61 atomic.h

[Go to the documentation of this file.](#)

```

1  /* Copyright (c) 2007 Dean Camera
2  All rights reserved.
3
4  Redistribution and use in source and binary forms, with or without
5  modification, are permitted provided that the following conditions are met:
6
7  * Redistributions of source code must retain the above copyright
8  notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE.
30 */
31
32 /* $Id: atomic.h 2541 2017-06-11 15:41:47Z joerg_wunsch $ */
33
34 #ifndef _UTIL_ATOMIC_H_
35 #define _UTIL_ATOMIC_H_ 1
36
37 #include <avr/io.h>
38 #include <avr/interrupt.h>
39
40 #if !defined(__DOXYGEN__)
41 /* Internal helper functions. */
42 static __inline__ uint8_t __iSeiRetVal(void)
43 {
44     sei();
45     return 1;
46 }
47
48 static __inline__ uint8_t __iCliRetVal(void)
49 {
50     cli();
51     return 1;
52 }
53
54 static __inline__ void __iSeiParam(const uint8_t *__s)
55 {
56     sei();
57     __asm__ volatile (" : :: \"memory\"; " : __s);
58 }
59
60 static __inline__ void __iCliParam(const uint8_t *__s)
61 {
62     cli();
63     __asm__ volatile (" : :: \"memory\"; " : __s);
64 }
65
66 static __inline__ void __iRestore(const uint8_t *__s)
67 {
68     SREG = *__s;
69     __asm__ volatile (" : :: \"memory\"; " : __s);
70 }
71

```

```

72 }
73 #endif /* !__DOXYGEN__ */
74
75 /** \file */
76 /** \defgroup util_atomic <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks
77
78 \code
79 #include <util/atomic.h>
80 \endcode
81
82 \note The macros in this header file require the ISO/IEC 9899:1999
83 ("ISO C99") feature of for loop variables that are declared inside
84 the for loop itself. For that reason, this header file can only
85 be used if the standard level of the compiler (option --std=) is
86 set to either \c c99 or \c gnu99.
87
88 The macros in this header file deal with code blocks that are
89 guaranteed to be executed Atomically or Non-Atomically. The term
90 "Atomic" in this context refers to the inability of the respective
91 code to be interrupted.
92
93 These macros operate via automatic manipulation of the Global
94 Interrupt Status (I) bit of the SREG register. Exit paths from
95 both block types are all managed automatically without the need
96 for special considerations, i. e. the interrupt status will be
97 restored to the same value it had when entering the respective
98 block (unless ATOMIC_FORCEON or NONATOMIC_FORCEOFF are used).
99
100 A typical example that requires atomic access is a 16 (or more)
101 bit variable that is shared between the main execution path and an
102 ISR. While declaring such a variable as volatile ensures that the
103 compiler will not optimize accesses to it away, it does not
104 guarantee atomic access to it. Assuming the following example:
105
106 \code
107 #include <inttypes.h>
108 #include <avr/interrupt.h>
109 #include <avr/io.h>
110
111 volatile uint16_t ctr;
112
113 ISR(TIMER1_OVF_vect)
114 {
115     ctr--;
116 }
117
118 ...
119 int
120 main(void)
121 {
122     ...
123     ctr = 0x200;
124     start_timer();
125     while (ctr != 0)
126         // wait
127     ;
128     ...
129 }
130 \endcode
131
132 There is a chance where the main context will exit its wait loop
133 when the variable \c ctr just reached the value 0xFF. This happens
134 because the compiler cannot natively access a 16-bit variable
135 atomically in an 8-bit CPU. So the variable is for example at
136 0x100, the compiler then tests the low byte for 0, which succeeds.
137 It then proceeds to test the high byte, but that moment the ISR
138 triggers, and the main context is interrupted. The ISR will
139 decrement the variable from 0x100 to 0xFF, and the main context
140 proceeds. It now tests the high byte of the variable which is
141 (now) also 0, so it concludes the variable has reached 0, and
142 terminates the loop.
143
144 Using the macros from this header file, the above code can be
145 rewritten like:
146
147 \code
148 #include <inttypes.h>
149 #include <avr/interrupt.h>
150 #include <avr/io.h>
151 #include <util/atomic.h>
152
153 volatile uint16_t ctr;
154
155 ISR(TIMER1_OVF_vect)
156 {
157     ctr--;
158 }

```

```

159
160 ...
161 int
162 main(void)
163 {
164 ...
165 ctr = 0x200;
166 start_timer();
167 sei();
168 uint16_t ctr_copy;
169 do
170 {
171 ATOMIC_BLOCK(ATOMIC_FORCEON)
172 {
173 ctr_copy = ctr;
174 }
175 }
176 while (ctr_copy != 0);
177 ...
178 }
179 \endcode
180
181 This will install the appropriate interrupt protection before
182 accessing variable \c ctr, so it is guaranteed to be consistently
183 tested. If the global interrupt state were uncertain before
184 entering the ATOMIC_BLOCK, it should be executed with the
185 parameter ATOMIC_RESTORESTATE rather than ATOMIC_FORCEON.
186
187 See \ref optim_code_reorder for things to be taken into account
188 with respect to compiler optimizations.
189 */
190
191 /** \def ATOMIC_BLOCK(type)
192 \ingroup util_atomic
193
194 Creates a block of code that is guaranteed to be executed
195 atomically. Upon entering the block the Global Interrupt Status
196 flag in SREG is disabled, and re-enabled upon exiting the block
197 from any exit path.
198
199 Two possible macro parameters are permitted, ATOMIC_RESTORESTATE
200 and ATOMIC_FORCEON.
201 */
202 #if defined(__DOXYGEN__)
203 #define ATOMIC_BLOCK(type)
204 #else
205 #define ATOMIC_BLOCK(type) for ( type, __ToDo = __iCliRetVal(); \
206 __ToDo ; __ToDo = 0 )
207 #endif /* __DOXYGEN__ */
208
209 /** \def NONATOMIC_BLOCK(type)
210 \ingroup util_atomic
211
212 Creates a block of code that is executed non-atomically. Upon
213 entering the block the Global Interrupt Status flag in SREG is
214 enabled, and disabled upon exiting the block from any exit
215 path. This is useful when nested inside ATOMIC_BLOCK sections,
216 allowing for non-atomic execution of small blocks of code while
217 maintaining the atomic access of the other sections of the parent
218 ATOMIC_BLOCK.
219
220 Two possible macro parameters are permitted,
221 NONATOMIC_RESTORESTATE and NONATOMIC_FORCEOFF.
222 */
223 #if defined(__DOXYGEN__)
224 #define NONATOMIC_BLOCK(type)
225 #else
226 #define NONATOMIC_BLOCK(type) for ( type, __ToDo = __iSeiRetVal(); \
227 __ToDo ; __ToDo = 0 )
228 #endif /* __DOXYGEN__ */
229
230 /** \def ATOMIC_RESTORESTATE
231 \ingroup util_atomic
232
233 This is a possible parameter for ATOMIC_BLOCK. When used, it will
234 cause the ATOMIC_BLOCK to restore the previous state of the SREG
235 register, saved before the Global Interrupt Status flag bit was
236 disabled. The net effect of this is to make the ATOMIC_BLOCK's
237 contents guaranteed atomic, without changing the state of the
238 Global Interrupt Status flag when execution of the block
239 completes.
240 */
241 #if defined(__DOXYGEN__)
242 #define ATOMIC_RESTORESTATE
243 #else
244 #define ATOMIC_RESTORESTATE uint8_t sreg_save \
245 __attribute__((__cleanup__(__iRestore))) = SREG

```

```

246 #endif /* __DOXYGEN__ */
247
248 /** \def ATOMIC_FORCEON
249 \ingroup util_atomic
250
251 This is a possible parameter for ATOMIC_BLOCK. When used, it will
252 cause the ATOMIC_BLOCK to force the state of the SREG register on
253 exit, enabling the Global Interrupt Status flag bit. This saves a
254 small amount of flash space, a register, and one or more processor
255 cycles, since the previous value of the SREG register does not need
256 to be saved at the start of the block.
257
258 Care should be taken that ATOMIC_FORCEON is only used when it is
259 known that interrupts are enabled before the block's execution or
260 when the side effects of enabling global interrupts at the block's
261 completion are known and understood.
262 */
263 #if defined(__DOXYGEN__)
264 #define ATOMIC_FORCEON
265 #else
266 #define ATOMIC_FORCEON uint8_t sreg_save \
267 __attribute__((__cleanup__((iSeiParam))) = 0
268 #endif /* __DOXYGEN__ */
269
270 /** \def NONATOMIC_RESTORESTATE
271 \ingroup util_atomic
272
273 This is a possible parameter for NONATOMIC_BLOCK. When used, it
274 will cause the NONATOMIC_BLOCK to restore the previous state of
275 the SREG register, saved before the Global Interrupt Status flag
276 bit was enabled. The net effect of this is to make the
277 NONATOMIC_BLOCK's contents guaranteed non-atomic, without changing
278 the state of the Global Interrupt Status flag when execution of
279 the block completes.
280 */
281 #if defined(__DOXYGEN__)
282 #define NONATOMIC_RESTORESTATE
283 #else
284 #define NONATOMIC_RESTORESTATE uint8_t sreg_save \
285 __attribute__((__cleanup__((iRestore))) = SREG
286 #endif /* __DOXYGEN__ */
287
288 /** \def NONATOMIC_FORCEOFF
289 \ingroup util_atomic
290
291 This is a possible parameter for NONATOMIC_BLOCK. When used, it
292 will cause the NONATOMIC_BLOCK to force the state of the SREG
293 register on exit, disabling the Global Interrupt Status flag
294 bit. This saves a small amount of flash space, a register, and one
295 or more processor cycles, since the previous value of the SREG
296 register does not need to be saved at the start of the block.
297
298 Care should be taken that NONATOMIC_FORCEOFF is only used when it
299 is known that interrupts are disabled before the block's execution
300 or when the side effects of disabling global interrupts at the
301 block's completion are known and understood.
302 */
303 #if defined(__DOXYGEN__)
304 #define NONATOMIC_FORCEOFF
305 #else
306 #define NONATOMIC_FORCEOFF uint8_t sreg_save \
307 __attribute__((__cleanup__((iCliParam))) = 0
308 #endif /* __DOXYGEN__ */
309
310 #endif

```

25.62 crc16.h File Reference

Functions

- static inline __uint16_t __crc16_update (__uint16_t __crc, __uint8_t __data)
- static inline __uint16_t __crc_xmodem_update (__uint16_t __crc, __uint8_t __data)
- static inline __uint16_t __crc_ccitt_update (__uint16_t __crc, __uint8_t __data)
- static inline __uint8_t __crc_ibutton_update (__uint8_t __crc, __uint8_t __data)
- static inline __uint8_t __crc8_ccitt_update (__uint8_t __crc, __uint8_t __data)

25.63 crc16.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, 2003, 2004 Marek Michalkiewicz
2 Copyright (c) 2005, 2007 Joerg Wunsch
3 Copyright (c) 2013 Dave Hylands
4 Copyright (c) 2013 Frederic Nadeau
5 All rights reserved.
6
7 Redistribution and use in source and binary forms, with or without
8 modification, are permitted provided that the following conditions are met:
9
10 * Redistributions of source code must retain the above copyright
11 notice, this list of conditions and the following disclaimer.
12
13 * Redistributions in binary form must reproduce the above copyright
14 notice, this list of conditions and the following disclaimer in
15 the documentation and/or other materials provided with the
16 distribution.
17
18 * Neither the name of the copyright holders nor the names of
19 contributors may be used to endorse or promote products derived
20 from this software without specific prior written permission.
21
22 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
23 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
24 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
25 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
26 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
27 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
28 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
29 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
30 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
31 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
32 POSSIBILITY OF SUCH DAMAGE. */
33
34 /* $Id: crc16.h 2398 2013-05-08 11:45:54Z joerg_wunsch $ */
35
36 #ifndef _UTIL_CRC16_H_
37 #define _UTIL_CRC16_H_
38
39 #include <stdint.h>
40
41 /** \file */
42 /** \defgroup util_crc <util/crc16.h>: CRC Computations
43 \code#include <util/crc16.h>\endcode
44
45 This header file provides a optimized inline functions for calculating
46 cyclic redundancy checks (CRC) using common polynomials.
47
48 \par References:
49
50 \par
51
52 See the Dallas Semiconductor app note 27 for 8051 assembler example and
53 general CRC optimization suggestions. The table on the last page of the
54 app note is the key to understanding these implementations.
55
56 \par
57
58 Jack Crenshaw's "Implementing CRCs" article in the January 1992 issue of \e
59 Embedded \e Systems \e Programming. This may be difficult to find, but it
60 explains CRC's in very clear and concise terms. Well worth the effort to
61 obtain a copy.
62
63 A typical application would look like:
64
65 \code
66 // Dallas iButton test vector.
67 uint8_t serno[] = { 0x02, 0x1c, 0xb8, 0x01, 0, 0, 0, 0xa2 };
68
69 int
70 checkcrc(void)
71 {
72     uint8_t crc = 0, i;
73
74     for (i = 0; i < sizeof serno / sizeof serno[0]; i++)
75         crc = _crc_ibutton_update(crc, serno[i]);
76
77     return crc; // must be 0
78 }
79 \endcode
80 */
81
82 /** \ingroup util_crc
83 Optimized CRC-16 calculation.

```

```

84
85 Polynomial:  x^16 + x^15 + x^2 + 1 (0xa001)<br>
86 Initial value:  0xffff
87
88 This CRC is normally used in disk-drive controllers.
89
90 The following is the equivalent functionality written in C.
91
92 \code
93 uint16_t
94 crc16_update(uint16_t crc, uint8_t a)
95 {
96     int i;
97
98     crc ^= a;
99     for (i = 0; i < 8; ++i)
100     {
101         if (crc & 1)
102             crc = (crc >> 1) ^ 0xA001;
103         else
104             crc = (crc >> 1);
105     }
106
107     return crc;
108 }
109
110 \endcode */
111
112 static __inline__ uint16_t
113 _crc16_update(uint16_t __crc, uint8_t __data)
114 {
115     uint8_t __tmp;
116     uint16_t __ret;
117
118     __asm__ __volatile__ (
119         "eor %A0,%2" "\n\t"
120         "mov %1,%A0" "\n\t"
121         "swap %1" "\n\t"
122         "eor %1,%A0" "\n\t"
123         "mov __tmp_reg__,%1" "\n\t"
124         "lsr %1" "\n\t"
125         "lsr %1" "\n\t"
126         "eor %1,__tmp_reg__" "\n\t"
127         "mov __tmp_reg__,%1" "\n\t"
128         "lsr %1" "\n\t"
129         "eor %1,__tmp_reg__" "\n\t"
130         "andi %1,0x07" "\n\t"
131         "mov __tmp_reg__,%A0" "\n\t"
132         "mov %A0,%B0" "\n\t"
133         "lsr %1" "\n\t"
134         "ror __tmp_reg__" "\n\t"
135         "ror %1" "\n\t"
136         "mov %B0,__tmp_reg__" "\n\t"
137         "eor %A0,%1" "\n\t"
138         "lsr __tmp_reg__" "\n\t"
139         "ror %1" "\n\t"
140         "eor %B0,__tmp_reg__" "\n\t"
141         "eor %A0,%1"
142         : "=r" (__ret), "=d" (__tmp)
143         : "r" (__data), "0" (__crc)
144         : "r0"
145     );
146     return __ret;
147 }
148
149 /** \ingroup util_crc
150 Optimized CRC-XMODEM calculation.
151
152 Polynomial:  x^16 + x^12 + x^5 + 1 (0x1021)<br>
153 Initial value:  0x0
154
155 This is the CRC used by the Xmodem-CRC protocol.
156
157 The following is the equivalent functionality written in C.
158
159 \code
160 uint16_t
161 crc_xmodem_update (uint16_t crc, uint8_t data)
162 {
163     int i;
164
165     crc = crc ^ ((uint16_t)data << 8);
166     for (i=0; i<8; i++)
167     {
168         if (crc & 0x8000)
169             crc = (crc << 1) ^ 0x1021;
170         else

```

```

171 crc <= 1;
172 }
173
174 return crc;
175 }
176 \endcode */
177
178 static __inline__ uint16_t
179 _crc_xmodem_update(uint16_t __crc, uint8_t __data)
180 {
181     uint16_t __ret;          /* %B0:%A0 (alias for __crc) */
182     uint8_t __tmp1;          /* %1 */
183     uint8_t __tmp2;          /* %2 */
184                             /* %3 __data */
185
186     __asm__ __volatile__ (
187         "eor    %B0,%3"      "\n\t" /* crc.hi ^ data */
188         "mov    __tmp_reg__,%B0" "\n\t"
189         "swap   __tmp_reg__"  "\n\t" /* swap(crc.hi ^ data) */
190
191         /* Calculate the ret.lo of the CRC. */
192         "mov    %1,__tmp_reg__" "\n\t"
193         "andi   %1,0x0f"      "\n\t"
194         "eor    %1,%B0"      "\n\t"
195         "mov    %2,%B0"      "\n\t"
196         "eor    %2,__tmp_reg__" "\n\t"
197         "lsl    %2"          "\n\t"
198         "andi   %2,0xe0"      "\n\t"
199         "eor    %1,%2"      "\n\t" /* __tmp1 is now ret.lo. */
200
201         /* Calculate the ret.hi of the CRC. */
202         "mov    %2,__tmp_reg__" "\n\t"
203         "eor    %2,%B0"      "\n\t"
204         "andi   %2,0xf0"      "\n\t"
205         "lsr    %2"          "\n\t"
206         "mov    __tmp_reg__,%B0" "\n\t"
207         "lsl    __tmp_reg__"  "\n\t"
208         "rol    %2"          "\n\t"
209         "lsr    %B0"          "\n\t"
210         "lsr    %B0"          "\n\t"
211         "lsr    %B0"          "\n\t"
212         "andi   %B0,0x1f"      "\n\t"
213         "eor    %B0,%2"      "\n\t"
214         "eor    %B0,%A0"      "\n\t" /* ret.hi is now ready. */
215         "mov    %A0,%1"      "\n\t" /* ret.lo is now ready. */
216         : "=d" (__ret), "=d" (__tmp1), "=d" (__tmp2)
217         : "r" (__data), "0" (__crc)
218         : "r0"
219     );
220     return __ret;
221 }
222
223 /** \ingroup util_crc
224 Optimized CRC-CCITT calculation.
225
226 Polynomial:  x^16 + x^12 + x^5 + 1 (0x8408)<br>
227 Initial value:  0xffff
228
229 This is the CRC used by PPP and IrDA.
230
231 See RFC1171 (PPP protocol) and IrDA IrLAP 1.1
232
233 \note Although the CCITT polynomial is the same as that used by the Xmodem
234 protocol, they are quite different. The difference is in how the bits are
235 shifted through the algorithm. Xmodem shifts the MSB of the CRC and the
236 input first, while CCITT shifts the LSB of the CRC and the input first.
237
238 The following is the equivalent functionality written in C.
239
240 \code
241 uint16_t
242 crc_ccitt_update (uint16_t crc, uint8_t data)
243 {
244     data ^= lo8 (crc);
245     data ^= data << 4;
246
247     return (((uint16_t)data << 8) | hi8 (crc)) ^ (uint8_t)(data >> 4)
248     ^ ((uint16_t)data << 3));
249 }
250 \endcode */
251
252 static __inline__ uint16_t
253 _crc_ccitt_update (uint16_t __crc, uint8_t __data)
254 {
255     uint16_t __ret;
256
257     __asm__ __volatile__ (

```



```

258     "eor    %A0,%1"           "\n\t"
259
260     "mov    __tmp_reg__,%A0"  "\n\t"
261     "swap   %A0"              "\n\t"
262     "andi   %A0,0xf0"         "\n\t"
263     "eor    %A0,__tmp_reg__"  "\n\t"
264
265     "mov    __tmp_reg__,%B0"  "\n\t"
266
267     "mov    %B0,%A0"          "\n\t"
268
269     "swap   %A0"              "\n\t"
270     "andi   %A0,0x0f"         "\n\t"
271     "eor    __tmp_reg__,%A0"  "\n\t"
272
273     "lsr    %A0"              "\n\t"
274     "eor    %B0,%A0"          "\n\t"
275
276     "eor    %A0,%B0"          "\n\t"
277     "lsl    %A0"              "\n\t"
278     "lsl    %A0"              "\n\t"
279     "lsl    %A0"              "\n\t"
280     "eor    %A0,__tmp_reg__"
281
282     : "=d" (__ret)
283     : "r" (__data), "0" (__crc)
284     : "r0"
285 );
286     return __ret;
287 }
288
289 /** \ingroup util_crc
290     Optimized Dallas (now Maxim) iButton 8-bit CRC calculation.
291
292     Polynomial:   $x^8 + x^5 + x^4 + 1$  (0x8C) <br>
293     Initial value: 0x0
294
295     See http://www.maxim-ic.com/appnotes.cfm/appnote\_number/27
296
297     The following is the equivalent functionality written in C.
298
299     \code
300     uint8_t
301     _crc_ibutton_update(uint8_t crc, uint8_t data)
302     {
303     uint8_t i;
304
305     crc = crc ^ data;
306     for (i = 0; i < 8; i++)
307     {
308     if (crc & 0x01)
309     crc = (crc >> 1) ^ 0x8C;
310     else
311     crc >>= 1;
312     }
313
314     return crc;
315     }
316     \endcode
317 */
318
319 static __inline__ uint8_t
320 _crc_ibutton_update(uint8_t __crc, uint8_t __data)
321 {
322     uint8_t __i, __pattern;
323     __asm__ __volatile__ (
324     "    eor %0, %4" "\n\t"
325     "    ldi %1, 8" "\n\t"
326     "    ldi %2, 0x8C" "\n\t"
327     "1:  lsr %0" "\n\t"
328     "    brcc 2f" "\n\t"
329     "    eor %0, %2" "\n\t"
330     "2:  dec %1" "\n\t"
331     "    brne 1b" "\n\t"
332     : "=r" (__crc), "=d" (__i), "=d" (__pattern)
333     : "0" (__crc), "r" (__data));
334     return __crc;
335 }
336
337 /** \ingroup util_crc
338     Optimized CRC-8-CCITT calculation.
339
340     Polynomial:   $x^8 + x^2 + x + 1$  (0xE0) <br>
341
342     For use with simple CRC-8 <br>
343     Initial value: 0x0
344

```

```

345 For use with CRC-8-ROHC<br>
346 Initial value: 0xff<br>
347 Reference: http://tools.ietf.org/html/rfc3095#section-5.9.1
348
349 For use with CRC-8-ATM/ITU<br>
350 Initial value: 0xff<br>
351 Final XOR value: 0x55<br>
352 Reference: http://www.itu.int/rec/T-REC-I.432.1-199902-I/en
353
354 The C equivalent has been originally written by Dave Hylands.
355 Assembly code is based on _crc_ibutton_update optimization.
356
357 The following is the equivalent functionality written in C.
358
359 \code
360 uint8_t
361 _crc8_ccitt_update (uint8_t inCrc, uint8_t inData)
362 {
363     uint8_t i;
364     uint8_t data;
365
366     data = inCrc ^ inData;
367
368     for ( i = 0; i < 8; i++ )
369     {
370         if (( data & 0x80 ) != 0 )
371         {
372             data <<= 1;
373             data ^= 0x07;
374         }
375         else
376         {
377             data <<= 1;
378         }
379     }
380     return data;
381 }
382 \endcode
383 */
384
385 static __inline__ uint8_t
386 _crc8_ccitt_update(uint8_t __crc, uint8_t __data)
387 {
388     uint8_t __i, __pattern;
389     __asm__ __volatile__ (
390         "    eor    %0, %4" "\n\t"
391         "    ldi    %1, 8" "\n\t"
392         "    ldi    %2, 0x07" "\n\t"
393         "1:    lsl    %0" "\n\t"
394         "        brcc 2f" "\n\t"
395         "        eor    %0, %2" "\n\t"
396         "2:    dec    %1" "\n\t"
397         "        brne 1b" "\n\t"
398         : "=r" (__crc), "=d" (__i), "=d" (__pattern)
399         : "0" (__crc), "r" (__data));
400     return __crc;
401 }
402
403 #endif /* _UTIL_CRC16_H_ */

```

25.64 delay.h File Reference

Macros

- `#define F_CPU 1000000UL`

Functions

- void `_delay_ms` (double __ms)
- void `_delay_us` (double __us)

25.65 delay.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, Marek Michalkiewicz
2 Copyright (c) 2004,2005,2007 Joerg Wunsch
3 Copyright (c) 2007 Florin-Viorel Petrov
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 * Redistributions of source code must retain the above copyright
10 notice, this list of conditions and the following disclaimer.
11
12 * Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in
14 the documentation and/or other materials provided with the
15 distribution.
16
17 * Neither the name of the copyright holders nor the names of
18 contributors may be used to endorse or promote products derived
19 from this software without specific prior written permission.
20
21 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
22 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
23 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
24 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
25 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
26 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
27 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
28 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
29 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
30 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
31 POSSIBILITY OF SUCH DAMAGE. */
32
33 /* $Id: delay.h.in 2551 2020-10-10 20:33:35Z joerg_wunsch $ */
34
35 #ifndef _UTIL_DELAY_H_
36 #define _UTIL_DELAY_H_ 1
37
38 #ifndef __DOXYGEN__
39 #   ifndef __HAS_DELAY_CYCLES
40 #       define __HAS_DELAY_CYCLES 1
41 #   endif
42 #endif /* __DOXYGEN__ */
43
44 #include <inttypes.h>
45 #include <util/delay_basic.h>
46 #include <math.h>
47
48 /** \file */
49 /** \defgroup util_delay <util/delay.h>: Convenience functions for busy-wait delay loops
50 \code
51 #define F_CPU 1000000UL // 1 MHz
52 //#define F_CPU 14.7456E6
53 #include <util/delay.h>
54 \endcode
55
56 \note As an alternative method, it is possible to pass the
57 F_CPU macro down to the compiler from the Makefile.
58 Obviously, in that case, no \c \#define statement should be
59 used.
60
61 The functions in this header file are wrappers around the basic
62 busy-wait functions from <util/delay_basic.h>. They are meant as
63 convenience functions where actual time values can be specified
64 rather than a number of cycles to wait for. The idea behind is
65 that compile-time constant expressions will be eliminated by
66 compiler optimization so floating-point expressions can be used
67 to calculate the number of delay cycles needed based on the CPU
68 frequency passed by the macro F_CPU.
69
70 \note In order for these functions to work as intended, compiler
71 optimizations <em>must</em> be enabled, and the delay time
72 <em>must</em> be an expression that is a known constant at
73 compile-time. If these requirements are not met, the resulting
74 delay will be much longer (and basically unpredictable), and
75 applications that otherwise do not use floating-point calculations
76 will experience severe code bloat by the floating-point library
77 routines linked into the application.
78
79 The functions available allow the specification of microsecond, and
80 millisecond delays directly, using the application-supplied macro
81 F_CPU as the CPU clock frequency (in Hertz).
82
83 */

```

```

84
85 #if !defined(__DOXYGEN__)
86 static __inline__ void _delay_us(double __us) __attribute__((__always_inline__));
87 static __inline__ void _delay_ms(double __ms) __attribute__((__always_inline__));
88 #endif
89
90 #ifndef F_CPU
91 /* prevent compiler error by supplying a default */
92 # warning "F_CPU not defined for <util/delay.h>"
93 /** \ingroup util_delay
94 \def F_CPU
95 \brief CPU frequency in Hz
96
97 The macro F_CPU specifies the CPU frequency to be considered by
98 the delay macros. This macro is normally supplied by the
99 environment (e.g. from within a project header, or the project's
100 Makefile). The value 1 MHz here is only provided as a "vanilla"
101 fallback if no such user-provided definition could be found.
102
103 In terms of the delay functions, the CPU frequency can be given as
104 a floating-point constant (e.g. 3.6864E6 for 3.6864 MHz).
105 However, the macros in <util/setbaud.h> require it to be an
106 integer value.
107 */
108 # define F_CPU 1000000UL
109 #endif
110
111 #ifndef __OPTIMIZE__
112 # warning "Compiler optimizations disabled; functions from <util/delay.h> won't work as designed"
113 #endif
114
115 #if __HAS_DELAY_CYCLES && defined(__OPTIMIZE__) && \
116 !defined(__DELAY_BACKWARD_COMPATIBLE__) && \
117 __STDC_HOSTED__
118 # include <math.h>
119 #endif
120
121 /**
122 \ingroup util_delay
123
124 Perform a delay of \c __ms milliseconds, using _delay_loop_2().
125
126 The macro F_CPU is supposed to be defined to a
127 constant defining the CPU clock frequency (in Hertz).
128
129 The maximal possible delay is 262.14 ms / F_CPU in MHz.
130
131 When the user request delay which exceed the maximum possible one,
132 _delay_ms() provides a decreased resolution functionality. In this
133 mode _delay_ms() will work with a resolution of 1/10 ms, providing
134 delays up to 6.5535 seconds (independent from CPU frequency). The
135 user will not be informed about decreased resolution.
136
137 If the avr-gcc toolchain has __builtin_avr_delay_cycles()
138 support, maximal possible delay is 4294967.295 ms/ F_CPU in MHz. For
139 values greater than the maximal possible delay, overflows results in
140 no delay i.e., 0ms.
141
142 Conversion of \c __ms into clock cycles may not always result in
143 integer. By default, the clock cycles rounded up to next
144 integer. This ensures that the user gets at least \c __ms
145 microseconds of delay.
146
147 Alternatively, by defining the macro \c __DELAY_ROUND_DOWN__, or
148 \c __DELAY_ROUND_CLOSEST__, before including this header file, the
149 algorithm can be made to round down, or round to closest integer,
150 respectively.
151
152 \note
153
154 The implementation of _delay_ms() based on
155 __builtin_avr_delay_cycles() is not backward compatible with older
156 implementations. In order to get functionality backward compatible
157 with previous versions, the macro \c "__DELAY_BACKWARD_COMPATIBLE__"
158 must be defined before including this header file. Also, the
159 backward compatible algorithm will be chosen if the code is
160 compiled in a <em>freestanding environment</em> (GCC option
161 \c -ffreestanding), as the math functions required for rounding are
162 not available to the compiler then.
163
164 */
165 void
166 _delay_ms(double __ms)
167 {
168     double __tmp ;
169     #if __HAS_DELAY_CYCLES && defined(__OPTIMIZE__) && \
170 !defined(__DELAY_BACKWARD_COMPATIBLE__) && \

```

```

171 __STDC_HOSTED__
172 uint32_t __ticks_dc;
173 extern void __builtin_avr_delay_cycles(unsigned long);
174 __tmp = ((F_CPU) / 1e3) * __ms;
175
176 #if defined(__DELAY_ROUND_DOWN__)
177     __ticks_dc = (uint32_t)fabs(__tmp);
178
179 #elif defined(__DELAY_ROUND_CLOSEST__)
180     __ticks_dc = (uint32_t)(fabs(__tmp)+0.5);
181
182 #else
183     //round up by default
184     __ticks_dc = (uint32_t)(ceil(fabs(__tmp)));
185 #endif
186
187     __builtin_avr_delay_cycles(__ticks_dc);
188
189 #else
190     uint16_t __ticks;
191     __tmp = ((F_CPU) / 4e3) * __ms;
192     if (__tmp < 1.0)
193         __ticks = 1;
194     else if (__tmp > 65535)
195     {
196         // __ticks = requested delay in 1/10 ms
197         __ticks = (uint16_t) (__ms * 10.0);
198         while(__ticks)
199         {
200             // wait 1/10 ms
201             _delay_loop_2(((F_CPU) / 4e3) / 10);
202             __ticks --;
203         }
204         return;
205     }
206     else
207         __ticks = (uint16_t)__tmp;
208     _delay_loop_2(__ticks);
209 #endif
210 }
211
212 /**
213 \ingroup util_delay
214
215 Perform a delay of \c __us microseconds, using _delay_loop_1().
216
217 The macro F_CPU is supposed to be defined to a
218 constant defining the CPU clock frequency (in Hertz).
219
220 The maximal possible delay is 768 us / F_CPU in MHz.
221
222 If the user requests a delay greater than the maximal possible one,
223 _delay_us() will automatically call _delay_ms() instead. The user
224 will not be informed about this case.
225
226 If the avr-gcc toolchain has __builtin_avr_delay_cycles()
227 support, maximal possible delay is 4294967.295 us/ F_CPU in MHz. For
228 values greater than the maximal possible delay, overflow results in
229 no delay i.e., 0us.
230
231 Conversion of \c __us into clock cycles may not always result in
232 integer. By default, the clock cycles rounded up to next
233 integer. This ensures that the user gets at least \c __us
234 microseconds of delay.
235
236 Alternatively, by defining the macro \c __DELAY_ROUND_DOWN__, or
237 \c __DELAY_ROUND_CLOSEST__, before including this header file, the
238 algorithm can be made to round down, or round to closest integer,
239 respectively.
240
241 \note
242
243 The implementation of _delay_ms() based on
244 __builtin_avr_delay_cycles() is not backward compatible with older
245 implementations. In order to get functionality backward compatible
246 with previous versions, the macro \c __DELAY_BACKWARD_COMPATIBLE__
247 must be defined before including this header file. Also, the
248 backward compatible algorithm will be chosen if the code is
249 compiled in a <em>freestanding environment</em> (GCC option
250 \c -ffreestanding), as the math functions required for rounding are
251 not available to the compiler then.
252
253 */
254 void
255 _delay_us(double __us)
256 {
257     double __tmp ;

```

```

258 #if __HAS_DELAY_CYCLES && defined(__OPTIMIZE__) && \
259 !defined(__DELAY_BACKWARD_COMPATIBLE__) && \
260 __STDC_HOSTED__
261     uint32_t __ticks_dc;
262     extern void __builtin_avr_delay_cycles(unsigned long);
263     __tmp = ((F_CPU) / 1e6) * __us;
264
265 #if defined(__DELAY_ROUND_DOWN__)
266     __ticks_dc = (uint32_t) fabs(__tmp);
267
268 #elif defined(__DELAY_ROUND_CLOSEST__)
269     __ticks_dc = (uint32_t) (fabs(__tmp)+0.5);
270
271 #else
272     //round up by default
273     __ticks_dc = (uint32_t) (ceil(fabs(__tmp)));
274 #endif
275
276     __builtin_avr_delay_cycles(__ticks_dc);
277
278 #else
279     uint8_t __ticks;
280     double __tmp2 ;
281     __tmp = ((F_CPU) / 3e6) * __us;
282     __tmp2 = ((F_CPU) / 4e6) * __us;
283     if (__tmp < 1.0)
284         __ticks = 1;
285     else if (__tmp2 > 65535)
286     {
287         _delay_ms(__us / 1000.0);
288         return;
289     }
290     else if (__tmp > 255)
291     {
292         uint16_t __ticks=(uint16_t) __tmp2;
293         _delay_loop_2(__ticks);
294         return;
295     }
296     else
297         __ticks = (uint8_t) __tmp;
298     _delay_loop_1(__ticks);
299 #endif
300 }
301
302
303 #endif /* _UTIL_DELAY_H_ */

```

25.66 delay_basic.h File Reference

Functions

- void [_delay_loop_1](#) (uint8_t __count)
- void [_delay_loop_2](#) (uint16_t __count)

25.67 delay_basic.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, Marek Michalkiewicz
2 Copyright (c) 2007 Joerg Wunsch
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15
16 * Neither the name of the copyright holders nor the names of
17 contributors may be used to endorse or promote products derived
18 from this software without specific prior written permission.
19

```

```

20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE. */
31
32 /* $Id: delay_basic.h 2453 2014-10-19 08:18:11Z saaadhu $ */
33
34 #ifndef _UTIL_DELAY_BASIC_H_
35 #define _UTIL_DELAY_BASIC_H_ 1
36
37 #include <inttypes.h>
38
39 #if !defined(__DOXYGEN__)
40 static __inline__ void _delay_loop_1(uint8_t __count) __attribute__((__always_inline__));
41 static __inline__ void _delay_loop_2(uint16_t __count) __attribute__((__always_inline__));
42 #endif
43
44 /** \file */
45 /** \defgroup util_delay_basic <util/delay_basic.h>: Basic busy-wait delay loops
46 \code
47 #include <util/delay_basic.h>
48 \endcode
49
50 The functions in this header file implement simple delay loops
51 that perform a busy-waiting. They are typically used to
52 facilitate short delays in the program execution. They are
53 implemented as count-down loops with a well-known CPU cycle
54 count per loop iteration. As such, no other processing can
55 occur simultaneously. It should be kept in mind that the
56 functions described here do not disable interrupts.
57
58 In general, for long delays, the use of hardware timers is
59 much preferable, as they free the CPU, and allow for
60 concurrent processing of other events while the timer is
61 running. However, in particular for very short delays, the
62 overhead of setting up a hardware timer is too much compared
63 to the overall delay time.
64
65 Two inline functions are provided for the actual delay algorithms.
66
67 */
68
69 /** \ingroup util_delay_basic
70
71 Delay loop using an 8-bit counter \c __count, so up to 256
72 iterations are possible. (The value 256 would have to be passed
73 as 0.) The loop executes three CPU cycles per iteration, not
74 including the overhead the compiler needs to setup the counter
75 register.
76
77 Thus, at a CPU speed of 1 MHz, delays of up to 768 microseconds
78 can be achieved.
79 */
80 void
81 _delay_loop_1(uint8_t __count)
82 {
83     __asm__ volatile (
84         "1:  dec %0" "\n\t"
85         "brne lb"
86         : "=r" (__count)
87         : "0" (__count)
88     );
89 }
90
91 /** \ingroup util_delay_basic
92
93 Delay loop using a 16-bit counter \c __count, so up to 65536
94 iterations are possible. (The value 65536 would have to be
95 passed as 0.) The loop executes four CPU cycles per iteration,
96 not including the overhead the compiler requires to setup the
97 counter register pair.
98
99 Thus, at a CPU speed of 1 MHz, delays of up to about 262.1
100 milliseconds can be achieved.
101 */
102 void
103 _delay_loop_2(uint16_t __count)
104 {
105     __asm__ volatile (
106         "1:  sbiw %0,1" "\n\t"

```

```

107         "brne lb"
108         : "=w" (__count)
109         : "0" (__count)
110     );
111 }
112
113 #endif /* _UTIL_DELAY_BASIC_H_ */

```

25.68 eu_dst.h

```

1 /*
2  * (c)2012 Michael Duane Rice All rights reserved.
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are
6  * met:
7  *
8  * Redistributions of source code must retain the above copyright notice, this
9  * list of conditions and the following disclaimer.  Redistributions in binary
10 * form must reproduce the above copyright notice, this list of conditions
11 * and the following disclaimer in the documentation and/or other materials
12 * provided with the distribution.  Neither the name of the copyright holders
13 * nor the names of contributors may be used to endorse or promote products
14 * derived from this software without specific prior written permission.
15 *
16 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
17 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
18 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
19 * ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
20 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
21 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
22 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
23 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
24 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
26 * POSSIBILITY OF SUCH DAMAGE.
27 */
28
29 /* $Id: eu_dst.h 2537 2017-03-31 19:46:35Z joerg_wunsch $ */
30
31 /**
32 Daylight Saving function for the European Union.  To utilize this function, you must
33 \code #include <util/eu_dst.h> \endcode
34 and
35 \code set_dst(eu_dst); \endcode
36
37 Given the time stamp and time zone parameters provided, the Daylight Saving function must
38 return a value appropriate for the tm structures' tm_isdst element.  That is...
39
40 0 : If Daylight Saving is not in effect.
41
42 -1 : If it cannot be determined if Daylight Saving is in effect.
43
44 A positive integer : Represents the number of seconds a clock is advanced for Daylight Saving.
45 This will typically be ONE_HOUR.
46
47 Daylight Saving 'rules' are subject to frequent change.  For production applications it is
48 recommended to write your own DST function, which uses 'rules' obtained from, and modifiable by,
49 the end user ( perhaps stored in EEPROM ).
50 */
51
52 #ifndef EU_DST_H
53 #define EU_DST_H
54
55 #ifdef __cplusplus
56 extern "C" {
57 #endif
58
59 #include <time.h>
60 #include <inttypes.h>
61
62 int eu_dst(const time_t * timer, int32_t * z) {
63     struct tm tmptr;
64     uint8_t month, mday, hour, day_of_week, d;
65     int n;
66
67     /* obtain the variables */
68     gmtime_r(timer, &tmptr);
69     month = tmptr.tm_mon;
70     day_of_week = tmptr.tm_wday;
71     mday = tmptr.tm_mday - 1;
72     hour = tmptr.tm_hour;
73
74     if ((month > MARCH) && (month < OCTOBER))

```



```

75         return ONE_HOUR;
76
77     if (month < MARCH)
78         return 0;
79     if (month > OCTOBER)
80         return 0;
81
82     /* determine mday of last Sunday */
83     n = tmptr.tm_mday - 1;
84     n -= day_of_week;
85     n += 7;
86     d = n % 7; /* date of first Sunday */
87
88     n = 31 - d;
89     n /= 7; /* number of Sundays left in the month */
90
91     d = d + 7 * n; /* d: mday of final Sunday */
92
93     if (month == MARCH) {
94         if (mday < d) /* before last Sunday */
95             return 0;
96         if (mday > d) /* past last Sunday */
97             return ONE_HOUR;
98         if (hour < 2) /* at last Sunday, before switching */
99             return 0;
100        return ONE_HOUR; /* at last Sunday, after switching */
101    }
102    /* month == OCTOBER */
103    if (mday < d) /* before last Sunday */
104        return ONE_HOUR;
105    if (mday > d) /* past last Sunday */
106        return 0;
107    if (hour < 2) /* at last Sunday, before switching */
108        return ONE_HOUR;
109    return 0; /* at last Sunday, after switchin */
110
111 }
112
113 #ifdef __cplusplus
114 }
115 #endif
116
117 #endif

```

25.69 parity.h File Reference

Macros

- `#define parity_even_bit(val)`

25.70 parity.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, Marek Michalkiewicz
2 Copyright (c) 2004,2005,2007 Joerg Wunsch
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15
16 * Neither the name of the copyright holders nor the names of
17 contributors may be used to endorse or promote products derived
18 from this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

```

```

24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE. */
31
32 /* $Id: parity.h 1196 2007-01-23 15:34:58Z joerg_wunsch $ */
33
34 #ifndef _UTIL_PARITY_H_
35 #define _UTIL_PARITY_H_
36
37 /** \file */
38 /** \defgroup util_parity <util/parity.h> Parity bit generation
39 \code #include <util/parity.h> \endcode
40
41 This header file contains optimized assembler code to calculate
42 the parity bit for a byte.
43 */
44 /** \def parity_even_bit
45 \ingroup util_parity
46 \returns 1 if \c val has an odd number of bits set. */
47 #define parity_even_bit(val) \
48 (__extension__({ \
49   unsigned char __t; \
50   __asm__ ( \
51     "mov __tmp_reg__,%0" "\n\t" \
52     "swap %0" "\n\t" \
53     "eor %0,__tmp_reg__" "\n\t" \
54     "mov __tmp_reg__,%0" "\n\t" \
55     "lsr %0" "\n\t" \
56     "lsr %0" "\n\t" \
57     "eor %0,__tmp_reg__" \
58     : "=r" (__t) \
59     : "0" ((unsigned char)(val)) \
60     : "r0" \
61   ); \
62   (((__t + 1) >> 1) & 1); \
63   }))
64
65 #endif /* _UTIL_PARITY_H_ */

```

25.71 setbaud.h File Reference

Macros

- #define [BAUD_TOL](#) 2
- #define [UBRR_VALUE](#)
- #define [UBRRL_VALUE](#)
- #define [UBRRH_VALUE](#)
- #define [USE_2X](#) 0

25.72 setbaud.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2007 Cliff Lawson
2 Copyright (c) 2007 Carlos Lamas
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15
16 * Neither the name of the copyright holders nor the names of
17 contributors may be used to endorse or promote products derived
18 from this software without specific prior written permission.

```

```

19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE. */
31
32 /* $Id: setbaud.h 2424 2014-04-29 13:08:15Z pitchumani $ */
33
34 /**
35  \file
36  */
37
38 /**
39  \defgroup util_setbaud <util/setbaud.h>: Helper macros for baud rate calculations
40  \code
41  #define F_CPU 11059200
42  #define BAUD 38400
43  #include <util/setbaud.h>
44  \endcode
45
46  This header file requires that on entry values are already defined
47  for F_CPU and BAUD. In addition, the macro BAUD_TOL will define
48  the baud rate tolerance (in percent) that is acceptable during
49  the calculations. The value of BAUD_TOL will default to 2 %.
50
51  This header file defines macros suitable to setup the UART baud
52  rate prescaler registers of an AVR. All calculations are done
53  using the C preprocessor. Including this header file causes no
54  other side effects so it is possible to include this file more than
55  once (supposedly, with different values for the BAUD parameter),
56  possibly even within the same function.
57
58  Assuming that the requested BAUD is valid for the given F_CPU then
59  the macro UBRR_VALUE is set to the required prescaler value. Two
60  additional macros are provided for the low and high bytes of the
61  prescaler, respectively: UBRRRL_VALUE is set to the lower byte of
62  the UBRR_VALUE and UBRRLH_VALUE is set to the upper byte. An
63  additional macro USE_2X will be defined. Its value is set to 1 if
64  the desired BAUD rate within the given tolerance could only be
65  achieved by setting the U2X bit in the UART configuration. It will
66  be defined to 0 if U2X is not needed.
67
68  Example usage:
69
70  \code
71  #include <avr/io.h>
72
73  #define F_CPU 4000000
74
75  static void
76  uart_9600(void)
77  {
78  #define BAUD 9600
79  #include <util/setbaud.h>
80  UBRRH = UBRRLH_VALUE;
81  UBRRL = UBRRRL_VALUE;
82  #if USE_2X
83  UCSRA |= (1 << U2X);
84  #else
85  UCSRA &= ~(1 << U2X);
86  #endif
87  }
88
89  static void
90  uart_38400(void)
91  {
92  #undef BAUD // avoid compiler warning
93  #define BAUD 38400
94  #include <util/setbaud.h>
95  UBRRH = UBRRLH_VALUE;
96  UBRRL = UBRRRL_VALUE;
97  #if USE_2X
98  UCSRA |= (1 << U2X);
99  #else
100  UCSRA &= ~(1 << U2X);
101  #endif
102  }
103  \endcode
104
105  In this example, two functions are defined to setup the UART

```

```

106 to run at 9600 Bd, and 38400 Bd, respectively.    Using a CPU
107 clock of 4 MHz, 9600 Bd can be achieved with an acceptable
108 tolerance without setting U2X (prescaler 25), while 38400 Bd
109 require U2X to be set (prescaler 12).
110 */
111
112 #ifndef F_CPU
113 #   error "setbaud.h requires F_CPU to be defined"
114 #endif
115
116 #ifndef BAUD
117 #   error "setbaud.h requires BAUD to be defined"
118 #endif
119
120 #if !(F_CPU)
121 #   error "F_CPU must be a constant value"
122 #endif
123
124 #if !(BAUD)
125 #   error "BAUD must be a constant value"
126 #endif
127
128 #if defined(__DOXYGEN__)
129 /**
130  \def BAUD_TOL
131  \ingroup util_setbaud
132
133  Input and output macro for <util/setbaud.h>
134
135  Define the acceptable baud rate tolerance in percent.    If not set
136  on entry, it will be set to its default value of 2.
137  */
138  #define BAUD_TOL 2
139
140  /**
141  \def UBRR_VALUE
142  \ingroup util_setbaud
143
144  Output macro from <util/setbaud.h>
145
146  Contains the calculated baud rate prescaler value for the UBRR
147  register.
148  */
149  #define UBRR_VALUE
150
151  /**
152  \def UBRRL_VALUE
153  \ingroup util_setbaud
154
155  Output macro from <util/setbaud.h>
156
157  Contains the lower byte of the calculated prescaler value
158  (UBRR_VALUE).
159  */
160  #define UBRRL_VALUE
161
162  /**
163  \def UBRRH_VALUE
164  \ingroup util_setbaud
165
166  Output macro from <util/setbaud.h>
167
168  Contains the upper byte of the calculated prescaler value
169  (UBRR_VALUE).
170  */
171  #define UBRRH_VALUE
172
173  /**
174  \def USE_2X
175  \ingroup util_setbaud
176
177  Output macro from <util/setbaud.h>
178
179  Contains the value 1 if the desired baud rate tolerance could only
180  be achieved by setting the U2X bit in the UART configuration.
181  Contains 0 otherwise.
182  */
183  #define USE_2X 0
184
185  #else /* !__DOXYGEN__ */
186
187  #undef USE_2X
188
189  /* Baud rate tolerance is 2 % unless previously defined */
190  #ifndef BAUD_TOL
191  #   define BAUD_TOL 2
192  #endif

```

```

193
194 #ifndef __ASSEMBLER__
195 #define UBRR_VALUE (((F_CPU) + 8 * (BAUD)) / (16 * (BAUD)) - 1)
196 #else
197 #define UBRR_VALUE (((F_CPU) + 8UL * (BAUD)) / (16UL * (BAUD)) - 1UL)
198 #endif
199
200 #if 100 * (F_CPU) > \
201 (16 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) + (BAUD) * (BAUD_TOL))
202 # define USE_2X 1
203 #elif 100 * (F_CPU) < \
204 (16 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) - (BAUD) * (BAUD_TOL))
205 # define USE_2X 1
206 #else
207 # define USE_2X 0
208 #endif
209
210 #if USE_2X
211 /* U2X required, recalculate */
212 #undef UBRR_VALUE
213
214 #ifndef __ASSEMBLER__
215 #define UBRR_VALUE (((F_CPU) + 4 * (BAUD)) / (8 * (BAUD)) - 1)
216 #else
217 #define UBRR_VALUE (((F_CPU) + 4UL * (BAUD)) / (8UL * (BAUD)) - 1UL)
218 #endif
219
220 #if 100 * (F_CPU) > \
221 (8 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) + (BAUD) * (BAUD_TOL))
222 # warning "Baud rate achieved is higher than allowed"
223 #endif
224
225 #if 100 * (F_CPU) < \
226 (8 * ((UBRR_VALUE) + 1)) * (100 * (BAUD) - (BAUD) * (BAUD_TOL))
227 # warning "Baud rate achieved is lower than allowed"
228 #endif
229
230 #endif /* USE_U2X */
231
232 #if UBRR_VALUE
233 /* Check for overflow */
234 # if UBRR_VALUE >= (1 << 12)
235 # warning "UBRR value overflow"
236 # endif
237
238 # define UBRR_L_VALUE (UBRR_VALUE & 0xff)
239 # define UBRR_H_VALUE (UBRR_VALUE >> 8)
240 #endif
241
242 #endif /* __DOXYGEN__ */
243 /* end of util/setbaud.h */

```

25.73 compat/twi.h

```

1 /* Copyright (c) 2005 Joerg Wunsch
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright
11 notice, this list of conditions and the following disclaimer in
12 the documentation and/or other materials provided with the
13 distribution.
14
15 * Neither the name of the copyright holders nor the names of
16 contributors may be used to endorse or promote products derived
17 from this software without specific prior written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
26 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
29 POSSIBILITY OF SUCH DAMAGE. */
30

```

```

31 /* $Id: twi.h 933 2005-11-05 22:23:16Z joerg_wunsch $ */
32
33 #ifndef _COMPAT_TWI_H_
34 #define _COMPAT_TWI_H_
35
36 #include <util/twi.h>
37
38 #endif /* _COMPAT_TWI_H_ */

```

25.74 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

25.75 util/twi.h

[Go to the documentation of this file.](#)

```

1 /* Copyright (c) 2002, Marek Michalkiewicz
2 Copyright (c) 2005, 2007 Joerg Wunsch
3 All rights reserved.
4
5 Redistribution and use in source and binary forms, with or without
6 modification, are permitted provided that the following conditions are met:
7
8 * Redistributions of source code must retain the above copyright
9 notice, this list of conditions and the following disclaimer.
10
11 * Redistributions in binary form must reproduce the above copyright
12 notice, this list of conditions and the following disclaimer in
13 the documentation and/or other materials provided with the
14 distribution.
15
16 * Neither the name of the copyright holders nor the names of
17 contributors may be used to endorse or promote products derived
18 from this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE. */
31
32 /* $Id: twi.h 1196 2007-01-23 15:34:58Z joerg_wunsch $ */
33 /* copied from: Id: avr/twi.h,v 1.4 2004/11/01 21:19:54 arcanum Exp */
34
35 #ifndef _UTIL_TWI_H_
36 #define _UTIL_TWI_H_ 1
37
38 #include <avr/io.h>
39
40 /** \file */
41 /** \defgroup util_twi <util/twi.h>: TWI bit mask definitions
42 \code #include <util/twi.h> \endcode
43
44 This header file contains bit mask definitions for use with
45 the AVR TWI interface.
46 */
47 /** \name TWSR values
48
49 Mnemonics:
50 <br>TW_MT_XXX - master transmitter
51 <br>TW_MR_XXX - master receiver
52 <br>TW_ST_XXX - slave transmitter
53 <br>TW_SR_XXX - slave receiver
54 */
55
56 @{
57 /* Master */
58 /** \ingroup util_twi
59 \def TW_START
60 start condition transmitted */
61 #define TW_START 0x08
62
63 /** \ingroup util_twi
64 \def TW_REP_START
65 repeated start condition transmitted */
66 #define TW_REP_START 0x10
67
68 /* Master Transmitter */
69 /** \ingroup util_twi
70 \def TW_MT_SLA_ACK
71 SLA+W transmitted, ACK received */
72 #define TW_MT_SLA_ACK 0x18
73
74 /** \ingroup util_twi
75 \def TW_MT_SLA_NACK
76 SLA+W transmitted, NACK received */
77 #define TW_MT_SLA_NACK 0x20
78
79 /** \ingroup util_twi
80 \def TW_MT_DATA_ACK
81 data transmitted, ACK received */
82 #define TW_MT_DATA_ACK 0x28
83

```

```
84 /** \ingroup util_twi
85 \def TW_MT_DATA_NACK
86 data transmitted, NACK received */
87 #define TW_MT_DATA_NACK    0x30
88
89 /** \ingroup util_twi
90 \def TW_MT_ARB_LOST
91 arbitration lost in SLA+W or data */
92 #define TW_MT_ARB_LOST    0x38
93
94 /* Master Receiver */
95 /** \ingroup util_twi
96 \def TW_MR_ARB_LOST
97 arbitration lost in SLA+R or NACK */
98 #define TW_MR_ARB_LOST    0x38
99
100 /** \ingroup util_twi
101 \def TW_MR_SLA_ACK
102 SLA+R transmitted, ACK received */
103 #define TW_MR_SLA_ACK    0x40
104
105 /** \ingroup util_twi
106 \def TW_MR_SLA_NACK
107 SLA+R transmitted, NACK received */
108 #define TW_MR_SLA_NACK    0x48
109
110 /** \ingroup util_twi
111 \def TW_MR_DATA_ACK
112 data received, ACK returned */
113 #define TW_MR_DATA_ACK    0x50
114
115 /** \ingroup util_twi
116 \def TW_MR_DATA_NACK
117 data received, NACK returned */
118 #define TW_MR_DATA_NACK    0x58
119
120 /* Slave Transmitter */
121 /** \ingroup util_twi
122 \def TW_ST_SLA_ACK
123 SLA+R received, ACK returned */
124 #define TW_ST_SLA_ACK    0xA8
125
126 /** \ingroup util_twi
127 \def TW_ST_ARB_LOST_SLA_ACK
128 arbitration lost in SLA+RW, SLA+R received, ACK returned */
129 #define TW_ST_ARB_LOST_SLA_ACK    0xB0
130
131 /** \ingroup util_twi
132 \def TW_ST_DATA_ACK
133 data transmitted, ACK received */
134 #define TW_ST_DATA_ACK    0xB8
135
136 /** \ingroup util_twi
137 \def TW_ST_DATA_NACK
138 data transmitted, NACK received */
139 #define TW_ST_DATA_NACK    0xC0
140
141 /** \ingroup util_twi
142 \def TW_ST_LAST_DATA
143 last data byte transmitted, ACK received */
144 #define TW_ST_LAST_DATA    0xC8
145
146 /* Slave Receiver */
147 /** \ingroup util_twi
148 \def TW_SR_SLA_ACK
149 SLA+W received, ACK returned */
150 #define TW_SR_SLA_ACK    0x60
151
152 /** \ingroup util_twi
153 \def TW_SR_ARB_LOST_SLA_ACK
154 arbitration lost in SLA+RW, SLA+W received, ACK returned */
155 #define TW_SR_ARB_LOST_SLA_ACK    0x68
156
157 /** \ingroup util_twi
158 \def TW_SR_GCALL_ACK
159 general call received, ACK returned */
160 #define TW_SR_GCALL_ACK    0x70
161
162 /** \ingroup util_twi
163 \def TW_SR_ARB_LOST_GCALL_ACK
164 arbitration lost in SLA+RW, general call received, ACK returned */
165 #define TW_SR_ARB_LOST_GCALL_ACK    0x78
166
167 /** \ingroup util_twi
168 \def TW_SR_DATA_ACK
169 data received, ACK returned */
170 #define TW_SR_DATA_ACK    0x80
```



```

171
172 /** \ingroup util_twi
173 \def TW_SR_DATA_NACK
174 data received, NACK returned */
175 #define TW_SR_DATA_NACK    0x88
176
177 /** \ingroup util_twi
178 \def TW_SR_GCALL_DATA_ACK
179 general call data received, ACK returned */
180 #define TW_SR_GCALL_DATA_ACK    0x90
181
182 /** \ingroup util_twi
183 \def TW_SR_GCALL_DATA_NACK
184 general call data received, NACK returned */
185 #define TW_SR_GCALL_DATA_NACK    0x98
186
187 /** \ingroup util_twi
188 \def TW_SR_STOP
189 stop or repeated start condition received while selected */
190 #define TW_SR_STOP    0xA0
191
192 /* Misc */
193 /** \ingroup util_twi
194 \def TW_NO_INFO
195 no state information available */
196 #define TW_NO_INFO    0xF8
197
198 /** \ingroup util_twi
199 \def TW_BUS_ERROR
200 illegal start or stop condition */
201 #define TW_BUS_ERROR    0x00
202
203
204 /**
205 * \ingroup util_twi
206 * \def TW_STATUS_MASK
207 * The lower 3 bits of TWSR are reserved on the ATmega163.
208 * The 2 LSB carry the prescaler bits on the newer ATmegs.
209 */
210 #define TW_STATUS_MASK    (_BV(TWS7) | _BV(TWS6) | _BV(TWS5) | _BV(TWS4) | \
211 _BV(TWS3))
212 /**
213 * \ingroup util_twi
214 * \def TW_STATUS
215 *
216 * TWSR, masked by TW_STATUS_MASK
217 */
218 #define TW_STATUS    (TWSR & TW_STATUS_MASK)
219 /* @} */
220
221 /**
222 * \name R/~W bit in SLA+R/W address field.
223 */
224
225 /* @{ */
226 /** \ingroup util_twi
227 \def TW_READ
228 SLA+R address */
229 #define TW_READ    1
230
231 /** \ingroup util_twi
232 \def TW_WRITE
233 SLA+W address */
234 #define TW_WRITE    0
235 /* @} */
236
237 #endif /* _UTIL_TWI_H_ */

```

25.76 usa_dst.h

```

1 /*
2 * (c)2012 Michael Duane Rice All rights reserved.
3 *
4 * Redistribution and use in source and binary forms, with or without
5 * modification, are permitted provided that the following conditions are
6 * met:
7 *
8 * Redistributions of source code must retain the above copyright notice, this
9 * list of conditions and the following disclaimer.  Redistributions in binary
10 * form must reproduce the above copyright notice, this list of conditions
11 * and the following disclaimer in the documentation and/or other materials
12 * provided with the distribution.  Neither the name of the copyright holders
13 * nor the names of contributors may be used to endorse or promote products
14 * derived from this software without specific prior written permission.

```

```

15 *
16 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
17 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
18 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
19 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
20 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
21 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
22 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
23 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
24 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
26 * POSSIBILITY OF SUCH DAMAGE.
27 */
28
29 /* $Id: usa_dst.h 2344 2013-04-10 19:52:09Z swfltek $ */
30
31 /**
32 Daylight Saving function for the USA. To utilize this function, you must
33 \code #include <util/usa_dst.h> \endcode
34 and
35 \code set_dst(usa_dst); \endcode
36
37 Given the time stamp and time zone parameters provided, the Daylight Saving function must
38 return a value appropriate for the tm structures' tm_isdst element. That is...
39
40 0 : If Daylight Saving is not in effect.
41
42 -1 : If it cannot be determined if Daylight Saving is in effect.
43
44 A positive integer : Represents the number of seconds a clock is advanced for Daylight Saving.
45 This will typically be ONE_HOUR.
46
47 Daylight Saving 'rules' are subject to frequent change. For production applications it is
48 recommended to write your own DST function, which uses 'rules' obtained from, and modifiable by,
49 the end user ( perhaps stored in EEPROM ).
50
51 */
52
53 #ifndef USA_DST_H
54 #define USA_DST_H
55
56 #ifdef __cplusplus
57 extern "C" {
58 #endif
59
60 #include <time.h>
61 #include <inttypes.h>
62
63 #ifndef DST_START_MONTH
64 #define DST_START_MONTH MARCH
65 #endif
66
67 #ifndef DST_END_MONTH
68 #define DST_END_MONTH NOVEMBER
69 #endif
70
71 #ifndef DST_START_WEEK
72 #define DST_START_WEEK 2
73 #endif
74
75 #ifndef DST_END_WEEK
76 #define DST_END_WEEK 1
77 #endif
78
79 int usa_dst(const time_t * timer, int32_t * z) {
80     time_t t;
81     struct tm tmptr;
82     uint8_t month, week, hour, day_of_week, d;
83     int n;
84
85     /* obtain the variables */
86     t = *timer + *z;
87     gmtime_r(&t, &tmptr);
88     month = tmptr.tm_mon;
89     day_of_week = tmptr.tm_wday;
90     week = week_of_month(&tmptr, 0);
91     hour = tmptr.tm_hour;
92
93     if ((month > DST_START_MONTH) && (month < DST_END_MONTH))
94         return ONE_HOUR;
95
96     if (month < DST_START_MONTH)
97         return 0;
98     if (month > DST_END_MONTH)
99         return 0;
100
101     if (month == DST_START_MONTH) {

```

```

102
103     if (week < DST_START_WEEK)
104         return 0;
105     if (week > DST_START_WEEK)
106         return ONE_HOUR;
107
108     if (day_of_week > SUNDAY)
109         return ONE_HOUR;
110     if (hour >= 2)
111         return ONE_HOUR;
112     return 0;
113 }
114 if (week > DST_END_WEEK)
115     return 0;
116 if (week < DST_END_WEEK)
117     return ONE_HOUR;
118 if (day_of_week > SUNDAY)
119     return 0;
120 if (hour >= 1)
121     return 0;
122 return ONE_HOUR;
123
124 }
125
126 #ifdef __cplusplus
127 }
128 #endif
129
130 #endif

```

25.77 eedef.h

```

1 /* Copyright (c) 2009 Dmitry Xmelkov
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9 * Redistributions in binary form must reproduce the above copyright
10 notice, this list of conditions and the following disclaimer in
11 the documentation and/or other materials provided with the
12 distribution.
13 * Neither the name of the copyright holders nor the names of
14 contributors may be used to endorse or promote products derived
15 from this software without specific prior written permission.
16
17 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
21 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
27 POSSIBILITY OF SUCH DAMAGE. */
28
29 /* $Id: eedef.h 2468 2015-02-25 12:53:38Z pitchumani $ */
30
31 #ifndef _EEDEF_H_
32 #define _EEDEF_H_ 1
33
34 #ifndef __DOXYGEN__
35
36 /* EEPROM address arg for a set of byte/word/dword functions and for
37 the internal eeprom_read_block(). */
38 #define addr_lo r24
39 #define addr_hi r25
40
41 /* Number of bytes arg for all block read/write functions, include
42 internal. */
43 #define n_lo r20
44 #define n_hi r21
45
46 #if __AVR_XMEGA__
47
48 # define NVM_BASE NVM_ADDR0
49
50 #else
51
52 # if !defined (EECR) && defined (DEECR) /* AT86RF401 */

```

```

53 # define EECR DEECR
54 # define EEARL DEEAR
55 # define EEDR DEEDR
56 # endif
57
58 # if !defined (EERE) && defined (EER) /* AT86RF401 */
59 # define EERE EER
60 # endif
61
62 # if !defined (EWE) && defined (EEPE) /* A part of Mega and Tiny */
63 # define EWE EEPE
64 # endif
65 # if !defined (EWE) && defined (EEL) /* AT86RF401 */
66 # define EWE EEL
67 # endif
68
69 # if !defined (EEMWE) && defined (EEMPE) /* A part of Mega and Tiny */
70 # define EEMWE EEMPE
71 # endif
72 # if !defined (EEMWE) && defined (EEU) /* AT86RF401 */
73 # define EEMWE EEU
74 # endif
75
76 # if !_SFR_IO_REG_P (EECR) \
77 || !_SFR_IO_REG_P (EEDR) \
78 || !_SFR_IO_REG_P (EEARL) \
79 || (defined (EEARH) && !_SFR_IO_REG_P (EEARH))
80 # error
81 # endif
82
83 #endif /* __AVR_XMEGA__ */
84 #endif /* __DOXYGEN__ */
85 #endif /* !__EDEF_H__ */

```

25.78 fdevopen.c File Reference

Functions

- [FILE *](#) [fdevopen](#) (int(*put)(char, [FILE *](#)), int(*get)([FILE *](#)))

25.79 stdio_private.h

```

1 /* Copyright (c) 2002,2005, Joerg Wunsch
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9 * Redistributions in binary form must reproduce the above copyright
10 notice, this list of conditions and the following disclaimer in
11 the documentation and/or other materials provided with the
12 distribution.
13 * Neither the name of the copyright holders nor the names of
14 contributors may be used to endorse or promote products derived
15 from this software without specific prior written permission.
16
17 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
21 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
27 POSSIBILITY OF SUCH DAMAGE.
28 */
29
30 /* $Id: stdio_private.h 847 2005-09-06 18:49:15Z joerg_wunsch $ */
31
32 #include <stdint.h>
33 #include <stdio.h>
34
35 /* values for PRINTF_LEVEL */
36 #define PRINTF_MIN 1

```

```

37 #define PRINTF_STD 2
38 #define PRINTF_FLT 3
39
40 /* values for SCANF_LEVEL */
41 #define SCANF_MIN 1
42 #define SCANF_STD 2
43 #define SCANF_FLT 3

```

25.80 xtoa_fast.h

```

1 /* Copyright (c) 2005, Dmitry Xmelkov
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9 * Redistributions in binary form must reproduce the above copyright
10 notice, this list of conditions and the following disclaimer in
11 the documentation and/or other materials provided with the
12 distribution.
13 * Neither the name of the copyright holders nor the names of
14 contributors may be used to endorse or promote products derived
15 from this software without specific prior written permission.
16
17 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
21 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
27 POSSIBILITY OF SUCH DAMAGE. */
28
29 /* $Id: xtoa_fast.h 1223 2007-02-18 13:33:09Z dmix $ */
30
31 #ifndef _XTOA_FAST_H_
32 #define _XTOA_FAST_H_
33
34 #ifndef __ASSEMBLER__
35
36 #include <stddef.h> /* for 'size_t' */
37
38 char * itoa_fast (int val, char *s, int base);
39 char * utoa_fast (unsigned val, char *s, int base);
40 char * ltoa_fast (long val, char *s, int base);
41 char * ultoa_fast (unsigned long val, char *s, int base);
42
43 char * itoa_width (int val, char *s, int base, size_t width);
44 char * utoa_width (unsigned val, char *s, int base, size_t width);
45 char * ltoa_width (long val, char *s, int base, size_t width);
46 char * ultoa_width (unsigned long val, char *s, int base, size_t width);
47
48 /* Internal function for use from 'printf'. */
49 char * __ultoa_invert (unsigned long val, char *s, int base);
50
51 #endif /* ifndef __ASSEMBLER__ */
52
53 /* Next flags are to use with 'base'. Unused fields are reserved. */
54 #define XTOA_PREFIX 0x0100 /* put prefix for octal or hex */
55 #define XTOA_UPPER 0x0200 /* use upper case letters */
56
57 #endif /* _XTOA_FAST_H_ */

```

25.81 dtoa_conv.h

```

1 /* Copyright (c) 2005, Dmitry Xmelkov
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9 * Redistributions in binary form must reproduce the above copyright
10 notice, this list of conditions and the following disclaimer in
11 the documentation and/or other materials provided with the

```

```

12 distribution.
13 * Neither the name of the copyright holders nor the names of
14 contributors may be used to endorse or promote products derived
15 from this software without specific prior written permission.
16
17 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
21 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
27 POSSIBILITY OF SUCH DAMAGE. */
28
29 /* $Id: dtoa_conv.h 1175 2007-01-14 15:18:18Z dmix $ */
30
31 #ifndef _DTOA_CONV_H
32 #define _DTOA_CONV_H
33
34 #include <stdio.h>
35
36 int dtoa_prf (double val, char *s, unsigned char width, unsigned char prec,
37              unsigned char flags);
38 int dtoa_lim (double val, char *s, unsigned char width, unsigned char prec,
39              unsigned char flags);
40 int dtoa_cln (double val, char *s, unsigned char ndigs, unsigned char flags);
41
42 void d2stream (double val, FILE *stream,
43               unsigned char width, unsigned char prec, unsigned char flags);
44
45 #define DTOA_SPACE 0x01 /* put space for positives */
46 #define DTOA_PLUS 0x02 /* put '+' for positives */
47 #define DTOA_UPPER 0x04 /* use uppercase letters */
48 #define DTOA_ZFILL 0x08 /* fill zeroes */
49 #define DTOA_LEFT 0x10 /* adjust to left */
50 #define DTOA_NOFILL 0x20 /* do not fill to width */
51 #define DTOA_EXP 0x40 /* d2stream: 'e(E)' format */
52 #define DTOA_FIX 0x80 /* d2stream: 'f(F)' format */
53
54 #define DTOA_EWIDTH (-1) /* Width too small */
55 #define DTOA_NONFINITE (-2) /* Value is NaN or Inf */
56
57 #endif /* !_DTOA_CONV_H */

```

25.82 stdlib_private.h

```

1 /* Copyright (c) 2004, Joerg Wunsch
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright
8 notice, this list of conditions and the following disclaimer.
9 * Redistributions in binary form must reproduce the above copyright
10 notice, this list of conditions and the following disclaimer in
11 the documentation and/or other materials provided with the
12 distribution.
13 * Neither the name of the copyright holders nor the names of
14 contributors may be used to endorse or promote products derived
15 from this software without specific prior written permission.
16
17 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
21 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
24 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
25 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
26 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
27 POSSIBILITY OF SUCH DAMAGE.
28 */
29
30 /* $Id: stdlib_private.h 1657 2008-03-24 17:11:08Z arcanum $ */
31
32 #include <inttypes.h>
33 #include <stdlib.h>
34 #include <avr/io.h>
35

```

```

36 #if !defined(__DOXYGEN__)
37
38 struct __freelist {
39     size_t sz;
40     struct __freelist *nx;
41 };
42
43 #endif
44
45 extern char *__brkval; /* first location not yet allocated */
46 extern struct __freelist *__flp; /* freelist pointer (head of freelist) */
47 extern size_t __malloc_margin; /* user-changeable before the first malloc() */
48 extern char *__malloc_heap_start;
49 extern char *__malloc_heap_end;
50
51 #ifndef __AVR__
52
53 /*
54 * When compiling malloc.c/realloc.c natively on a host machine, it will
55 * include a main() that performs a regression test. This is meant as
56 * a debugging aid, where a normal source-level debugger will help to
57 * verify that the various allocator structures have the desired
58 * appearance at each stage.
59 *
60 * When cross-compiling with avr-gcc, it will compile into just the
61 * library functions malloc() and free().
62 */
63 #define MALLOC_TEST
64
65 #endif /* !__AVR__ */
66
67 #ifdef MALLOC_TEST
68
69 extern void *mymalloc(size_t);
70 extern void myfree(void *);
71 extern void myrealloc(void *, size_t);
72
73 #define malloc mymalloc
74 #define free myfree
75 #define realloc myrealloc
76
77 #define __heap_start mymem[0]
78 #define __heap_end mymem[256]
79 extern char mymem[];
80 #define STACK_POINTER() (mymem + 256)
81
82 #else /* !MALLOC_TEST */
83
84 extern char __heap_start;
85 extern char __heap_end;
86
87 /* Needed for definition of AVR_STACK_POINTER_REG. */
88 #include <avr/io.h>
89
90 #define STACK_POINTER() ((char *)AVR_STACK_POINTER_REG)
91
92 #endif /* MALLOC_TEST */

```

25.83 ephemera_common.h

```

1 /*
2 * (C)2012 Michael Duane Rice All rights reserved.
3 *
4 * Redistribution and use in source and binary forms, with or without
5 * modification, are permitted provided that the following conditions are
6 * met:
7 *
8 * Redistributions of source code must retain the above copyright notice, this
9 * list of conditions and the following disclaimer. Redistributions in binary
10 * form must reproduce the above copyright notice, this list of conditions
11 * and the following disclaimer in the documentation and/or other materials
12 * provided with the distribution. Neither the name of the copyright holders
13 * nor the names of contributors may be used to endorse or promote products
14 * derived from this software without specific prior written permission.
15 *
16 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
17 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
18 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
19 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
20 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
21 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
22 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
23 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
24 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

```

```
25 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
26 * POSSIBILITY OF SUCH DAMAGE.
27 */
28
29 /* $Id: ephemerata_common.h 2345 2013-04-11 23:58:48Z swfltek $ */
30
31 #ifndef EPHEMERATA_PRIVATE_H
32 #define EPHEMERATA_PRIVATE_H
33
34 #define TROP_YEAR 31556925
35 #define ANOM_YEAR 31558433
36 #define INCLINATION 0.409105176667471 /* Earths axial tilt at the epoch */
37 #define PERIHELION 31316400 /* perihelion of 1999, 03 jan 13:00 UTC */
38 #define SOLSTICE 836160 /* winter solstice of 1999, 22 Dec 07:44 UTC */
39 #define TWO_PI 6.283185307179586
40 #define TROP_CYCLE 5022440.6025
41 #define ANOM_CYCLE 5022680.6082
42 #define DELTA_V 0.03342044 /* 2x orbital eccentricity */
43
44 #endif
```


Index

- <alloca.h>: Allocate space in the stack, [106](#)
 - alloca, [106](#)
- <assert.h>: Diagnostics, [106](#)
- <avr/boot.h>: Bootloader Support Utilities, [160](#)
 - boot_is_spm_interrupt, [161](#)
 - boot_lock_bits_set, [161](#)
 - boot_lock_bits_set_safe, [161](#)
 - boot_lock_fuse_bits_get, [162](#)
 - boot_page_erase, [162](#)
 - boot_page_erase_safe, [162](#)
 - boot_page_fill, [162](#)
 - boot_page_fill_safe, [163](#)
 - boot_page_write, [163](#)
 - boot_page_write_safe, [163](#)
 - boot_rww_busy, [163](#)
 - boot_rww_enable, [163](#)
 - boot_rww_enable_safe, [164](#)
 - boot_signature_byte_get, [164](#)
 - boot_spm_busy, [164](#)
 - boot_spm_busy_wait, [164](#)
 - boot_spm_interrupt_disable, [164](#)
 - boot_spm_interrupt_enable, [164](#)
 - BOOTLOADER_SECTION, [165](#)
 - GET_EXTENDED_FUSE_BITS, [165](#)
 - GET_HIGH_FUSE_BITS, [165](#)
 - GET_LOCK_BITS, [165](#)
 - GET_LOW_FUSE_BITS, [165](#)
- <avr/cpufunc.h>: Special AVR CPU functions, [165](#)
 - _MemoryBarrier, [165](#)
 - _NOP, [166](#)
 - ccp_write_io, [166](#)
- <avr/eeprom.h>: EEPROM handling, [166](#)
 - _EEGET, [167](#)
 - _EEPUT, [168](#)
 - __EEGET, [167](#)
 - __EEPUT, [167](#)
 - EEMEM, [168](#)
 - eeprom_busy_wait, [168](#)
 - eeprom_is_ready, [168](#)
 - eeprom_read_block, [168](#)
 - eeprom_read_byte, [168](#)
 - eeprom_read_dword, [168](#)
 - eeprom_read_float, [169](#)
 - eeprom_read_word, [169](#)
 - eeprom_update_block, [169](#)
 - eeprom_update_byte, [169](#)
 - eeprom_update_dword, [169](#)
 - eeprom_update_float, [169](#)
 - eeprom_update_word, [169](#)
 - eeprom_write_block, [169](#)
 - eeprom_write_byte, [170](#)
 - eeprom_write_dword, [170](#)
 - eeprom_write_float, [170](#)
 - eeprom_write_word, [170](#)
- <avr/fuse.h>: Fuse Support, [170](#)
- <avr/interrupt.h>: Interrupts, [173](#)
 - BADISR_vect, [191](#)
 - cli, [191](#)
 - EMPTY_INTERRUPT, [191](#)
 - ISR, [192](#)
 - ISR_ALIAS, [192](#)
 - ISR_ALIASOF, [192](#)
 - ISR_BLOCK, [192](#)
 - ISR_FLATTEN, [193](#)
 - ISR_NAKED, [193](#)
 - ISR_NOBLOCK, [193](#)
 - ISR_NOICF, [193](#)
 - reti, [193](#)
 - sei, [193](#)
 - SIGNAL, [193](#)
- <avr/io.h>: AVR device-specific IO definitions, [194](#)
 - _PROTECTED_WRITE, [195](#)
- <avr/lock.h>: Lockbit Support, [195](#)
- <avr/pgmspace.h>: Program Space Utilities, [197](#)
 - memcpy_P, [206](#)
 - memchr_P, [206](#)
 - memcmp_P, [206](#)
 - memcmp_PF, [207](#)
 - memcpy_P, [207](#)
 - memcpy_PF, [207](#)
 - memmem_P, [208](#)
 - memrchr_P, [208](#)
 - pgm_get_far_address, [199](#)
 - PGM_P, [199](#)
 - pgm_read_byte, [199](#)
 - pgm_read_byte_far, [199](#)
 - pgm_read_byte_near, [200](#)
 - pgm_read_dword, [200](#)
 - pgm_read_dword_far, [200](#)
 - pgm_read_dword_near, [200](#)
 - pgm_read_float, [200](#)
 - pgm_read_float_far, [201](#)
 - pgm_read_float_near, [201](#)
 - pgm_read_ptr, [201](#)
 - pgm_read_ptr_far, [201](#)
 - pgm_read_ptr_near, [201](#)
 - pgm_read_word, [202](#)
 - pgm_read_word_far, [202](#)
 - pgm_read_word_near, [202](#)
 - PGM_VOID_P, [202](#)
 - prog_char, [203](#)
 - prog_int16_t, [203](#)
 - prog_int32_t, [203](#)
 - prog_int64_t, [203](#)
 - prog_int8_t, [204](#)
 - prog_uchar, [204](#)
 - prog_uint16_t, [204](#)
 - prog_uint32_t, [205](#)
 - prog_uint64_t, [205](#)
 - prog_uint8_t, [205](#)

- prog_void, [206](#)
- PROGMEM, [202](#)
- PSTR, [203](#)
- strcasemp_P, [208](#)
- strcasemp_PF, [209](#)
- strcasestr_P, [209](#)
- strcat_P, [209](#)
- strcat_PF, [209](#)
- strchr_P, [210](#)
- strchrnul_P, [210](#)
- strcmp_P, [210](#)
- strcmp_PF, [210](#)
- strcpy_P, [212](#)
- strcpy_PF, [212](#)
- strcspn_P, [212](#)
- strlcat_P, [213](#)
- strlcat_PF, [213](#)
- strncpy_P, [213](#)
- strncpy_PF, [214](#)
- strlen_P, [214](#)
- strlen_PF, [214](#)
- strncasemp_P, [215](#)
- strncasemp_PF, [215](#)
- strncat_P, [216](#)
- strncat_PF, [216](#)
- strncmp_P, [216](#)
- strncmp_PF, [217](#)
- strncpy_P, [217](#)
- strncpy_PF, [217](#)
- strnlen_P, [218](#)
- strnlen_PF, [218](#)
- strpbrk_P, [218](#)
- strrchr_P, [219](#)
- strsep_P, [219](#)
- strspn_P, [219](#)
- strstr_P, [220](#)
- strstr_PF, [220](#)
- strtok_P, [220](#)
- strtok_rP, [221](#)
- <avr/power.h>: Power Reduction Management, [221](#)
 - clock_prescale_set, [225](#)
- <avr/sfr_defs.h>: Special function registers, [226](#)
 - _BV, [227](#)
 - bit_is_clear, [227](#)
 - bit_is_set, [228](#)
 - loop_until_bit_is_clear, [228](#)
 - loop_until_bit_is_set, [228](#)
- <avr/signature.h>: Signature Support, [228](#)
- <avr/sleep.h>: Power Management and Sleep Modes, [229](#)
 - sleep_bod_disable, [230](#)
 - sleep_cpu, [230](#)
 - sleep_disable, [230](#)
 - sleep_enable, [230](#)
 - sleep_mode, [230](#)
- <avr/version.h>: avr-libc version macros, [230](#)
 - __AVR_LIBC_DATE__, [231](#)
 - __AVR_LIBC_DATE_STRING__, [231](#)
 - __AVR_LIBC_MAJOR__, [231](#)
 - __AVR_LIBC_MINOR__, [231](#)
 - __AVR_LIBC_REVISION__, [231](#)
 - __AVR_LIBC_VERSION_STRING__, [231](#)
 - __AVR_LIBC_VERSION__, [231](#)
- <avr/wdt.h>: Watchdog timer handling, [232](#)
 - __attribute__, [234](#)
 - wdt_reset, [233](#)
 - WDTO_120MS, [233](#)
 - WDTO_15MS, [233](#)
 - WDTO_1S, [233](#)
 - WDTO_250MS, [233](#)
 - WDTO_2S, [233](#)
 - WDTO_30MS, [233](#)
 - WDTO_4S, [233](#)
 - WDTO_500MS, [234](#)
 - WDTO_60MS, [234](#)
 - WDTO_8S, [234](#)
- <compat/deprecated.h>: Deprecated items, [250](#)
 - cbi, [251](#)
 - enable_external_int, [251](#)
 - inb, [251](#)
 - inp, [251](#)
 - INTERRUPT, [251](#)
 - outb, [252](#)
 - outp, [252](#)
 - sbi, [252](#)
 - timer_enable_int, [252](#)
- <compat/ina90.h>: Compatibility with IAR EWB 3.x, [253](#)
- <ctype.h>: Character Operations, [107](#)
 - isalnum, [107](#)
 - isalpha, [107](#)
 - isascii, [107](#)
 - isblank, [108](#)
 - iscntrl, [108](#)
 - isdigit, [108](#)
 - isgraph, [108](#)
 - islower, [108](#)
 - isprint, [108](#)
 - ispunct, [108](#)
 - isspace, [108](#)
 - isupper, [108](#)
 - isxdigit, [109](#)
 - toascii, [109](#)
 - tolower, [109](#)
 - toupper, [109](#)
- <errno.h>: System Errors, [109](#)
 - EDOM, [110](#)
 - ERANGE, [110](#)
 - errno, [110](#)
- <inttypes.h>: Integer Type conversions, [110](#)
 - int_farptr_t, [123](#)
 - PRId16, [113](#)
 - PRId32, [113](#)
 - PRId8, [113](#)
 - PRIdFAST16, [113](#)
 - PRIdFAST32, [113](#)

- PRIdFAST8, [113](#)
- PRIdLEAST16, [114](#)
- PRIdLEAST32, [114](#)
- PRIdLEAST8, [114](#)
- PRIdPTR, [114](#)
- PRli16, [114](#)
- PRli32, [114](#)
- PRli8, [114](#)
- PRliFAST16, [114](#)
- PRliFAST32, [114](#)
- PRliFAST8, [114](#)
- PRliLEAST16, [114](#)
- PRliLEAST32, [115](#)
- PRliLEAST8, [115](#)
- PRliPTR, [115](#)
- PRlo16, [115](#)
- PRlo32, [115](#)
- PRlo8, [115](#)
- PRloFAST16, [115](#)
- PRloFAST32, [115](#)
- PRloFAST8, [115](#)
- PRloLEAST16, [115](#)
- PRloLEAST32, [115](#)
- PRloLEAST8, [116](#)
- PRloPTR, [116](#)
- PRlu16, [116](#)
- PRlu32, [116](#)
- PRlu8, [116](#)
- PRluFAST16, [116](#)
- PRluFAST32, [116](#)
- PRluFAST8, [116](#)
- PRluLEAST16, [116](#)
- PRluLEAST32, [116](#)
- PRluLEAST8, [116](#)
- PRluPTR, [117](#)
- PRIX16, [117](#)
- PRIx16, [117](#)
- PRIX32, [117](#)
- PRIx32, [117](#)
- PRIX8, [117](#)
- PRIx8, [117](#)
- PRIXFAST16, [117](#)
- PRIxFAST16, [117](#)
- PRIXFAST32, [117](#)
- PRIxFAST32, [117](#)
- PRIXFAST8, [118](#)
- PRIxFAST8, [118](#)
- PRIXLEAST16, [118](#)
- PRIxLEAST16, [118](#)
- PRIXLEAST32, [118](#)
- PRIxLEAST32, [118](#)
- PRIXLEAST8, [118](#)
- PRIxLEAST8, [118](#)
- PRIXPTR, [118](#)
- PRIxPTR, [118](#)
- SCNd16, [118](#)
- SCNd32, [119](#)
- SCNd8, [119](#)
- SCNdFAST16, [119](#)
- SCNdFAST32, [119](#)
- SCNdFAST8, [119](#)
- SCNdLEAST16, [119](#)
- SCNdLEAST32, [119](#)
- SCNdLEAST8, [119](#)
- SCNdPTR, [119](#)
- SCNi16, [119](#)
- SCNi32, [119](#)
- SCNi8, [120](#)
- SCNiFAST16, [120](#)
- SCNiFAST32, [120](#)
- SCNiFAST8, [120](#)
- SCNiLEAST16, [120](#)
- SCNiLEAST32, [120](#)
- SCNiLEAST8, [120](#)
- SCNiPTR, [120](#)
- SCNo16, [120](#)
- SCNo32, [120](#)
- SCNo8, [120](#)
- SCNoFAST16, [121](#)
- SCNoFAST32, [121](#)
- SCNoFAST8, [121](#)
- SCNoLEAST16, [121](#)
- SCNoLEAST32, [121](#)
- SCNoLEAST8, [121](#)
- SCNoPTR, [121](#)
- SCNu16, [121](#)
- SCNu32, [121](#)
- SCNu8, [121](#)
- SCNuFAST16, [121](#)
- SCNuFAST32, [122](#)
- SCNuFAST8, [122](#)
- SCNuLEAST16, [122](#)
- SCNuLEAST32, [122](#)
- SCNuLEAST8, [122](#)
- SCNuPTR, [122](#)
- SCNx16, [122](#)
- SCNx32, [122](#)
- SCNx8, [122](#)
- SCNxFAST16, [122](#)
- SCNxFAST32, [122](#)
- SCNxFAST8, [123](#)
- SCNxLEAST16, [123](#)
- SCNxLEAST32, [123](#)
- SCNxLEAST8, [123](#)
- SCNxPTR, [123](#)
- uint_farptr_t, [123](#)
- <math.h>: Mathematics, [123](#)
- M_E, [124](#)
- <setjmp.h>: Non-local goto, [124](#)
- longjmp, [125](#)
- setjmp, [125](#)
- <stdint.h>: Standard Integer Types, [126](#)
- INT16_C, [129](#)
- INT16_MAX, [129](#)
- INT16_MIN, [129](#)
- int16_t, [134](#)

INT32_C, 129
INT32_MAX, 129
INT32_MIN, 129
int32_t, 134
INT64_C, 129
INT64_MAX, 129
INT64_MIN, 129
int64_t, 134
INT8_C, 129
INT8_MAX, 129
INT8_MIN, 130
int8_t, 134
INT_FAST16_MAX, 130
INT_FAST16_MIN, 130
int_fast16_t, 134
INT_FAST32_MAX, 130
INT_FAST32_MIN, 130
int_fast32_t, 134
INT_FAST64_MAX, 130
INT_FAST64_MIN, 130
int_fast64_t, 134
INT_FAST8_MAX, 130
INT_FAST8_MIN, 130
int_fast8_t, 135
INT_LEAST16_MAX, 130
INT_LEAST16_MIN, 130
int_least16_t, 135
INT_LEAST32_MAX, 131
INT_LEAST32_MIN, 131
int_least32_t, 135
INT_LEAST64_MAX, 131
INT_LEAST64_MIN, 131
int_least64_t, 135
INT_LEAST8_MAX, 131
INT_LEAST8_MIN, 131
int_least8_t, 135
INTMAX_C, 131
INTMAX_MAX, 131
INTMAX_MIN, 131
intmax_t, 135
INTPTR_MAX, 131
INTPTR_MIN, 131
intptr_t, 135
PTRDIFF_MAX, 132
PTRDIFF_MIN, 132
SIG_ATOMIC_MAX, 132
SIG_ATOMIC_MIN, 132
SIZE_MAX, 132
UINT16_C, 132
UINT16_MAX, 132
uint16_t, 135
UINT32_C, 132
UINT32_MAX, 132
uint32_t, 135
UINT64_C, 132
UINT64_MAX, 132
uint64_t, 136
UINT8_C, 133
UINT8_MAX, 133
uint8_t, 136
UINT_FAST16_MAX, 133
uint_fast16_t, 136
UINT_FAST32_MAX, 133
uint_fast32_t, 136
UINT_FAST64_MAX, 133
uint_fast64_t, 136
UINT_FAST8_MAX, 133
uint_fast8_t, 136
UINT_LEAST16_MAX, 133
uint_least16_t, 136
UINT_LEAST32_MAX, 133
uint_least32_t, 136
UINT_LEAST64_MAX, 133
uint_least64_t, 136
UINT_LEAST8_MAX, 133
uint_least8_t, 137
UINTMAX_C, 133
UINTMAX_MAX, 134
uintmax_t, 137
UINTPTR_MAX, 134
uintptr_t, 137
<stdio.h>: Standard IO facilities, 137
 fdevopen, 139
<stdlib.h>: General utilities, 140
 atof, 141
 DOSTR_ALWAYS_SIGN, 140
 DOSTR_PLUS_SIGN, 141
 DOSTR_UPPERCASE, 141
 dostre, 141
 dosttrf, 142
 EXIT_FAILURE, 141
 EXIT_SUCCESS, 141
 itoa, 142
 ltoa, 142
 random, 143
 RANDOM_MAX, 141
 random_r, 143
 srandom, 143
 ultoa, 143
 utoa, 144
<string.h>: Strings, 145
 _FFS, 146
 ffs, 146
 ffsl, 146
 ffsll, 146
 memccpy, 146
 memchr, 147
 memcmp, 147
 memcpy, 147
 memmem, 148
 memmove, 148
 memrchr, 148
 memset, 149
 strcasecmp, 149
 strcasestr, 149
 strcat, 149

- strchr, [150](#)
- strchrnul, [150](#)
- strcmp, [150](#)
- strcpy, [151](#)
- strcspn, [151](#)
- strdup, [151](#)
- strlcat, [152](#)
- strncpy, [152](#)
- strlen, [153](#)
- strlwr, [153](#)
- strncasecmp, [153](#)
- strncat, [154](#)
- strncmp, [154](#)
- strncpy, [154](#)
- strnlen, [155](#)
- strpbrk, [155](#)
- strrchr, [155](#)
- strrev, [156](#)
- strsep, [156](#)
- strspn, [156](#)
- strstr, [157](#)
- strtok, [157](#)
- strtok_r, [157](#)
- strupr, [158](#)
- <time.h>: Time, [158](#)
- time_t, [159](#)
- <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks, [235](#)
- ATOMIC_BLOCK, [236](#)
- ATOMIC_FORCEON, [236](#)
- ATOMIC_RESTORESTATE, [236](#)
- NONATOMIC_BLOCK, [236](#)
- NONATOMIC_FORCEOFF, [237](#)
- NONATOMIC_RESTORESTATE, [237](#)
- <util/crc16.h>: CRC Computations, [237](#)
- _crc16_update, [238](#)
- _crc8_ccitt_update, [238](#)
- _crc_ccitt_update, [239](#)
- _crc_ibutton_update, [240](#)
- _crc_xmodem_update, [240](#)
- <util/delay.h>: Convenience functions for busy-wait delay loops, [241](#)
- _delay_ms, [242](#)
- _delay_us, [242](#)
- F_CPU, [241](#)
- <util/delay_basic.h>: Basic busy-wait delay loops, [243](#)
- _delay_loop_1, [243](#)
- _delay_loop_2, [243](#)
- <util/parity.h>: Parity bit generation, [243](#)
- parity_even_bit, [244](#)
- <util/setbaud.h>: Helper macros for baud rate calculations, [244](#)
- BAUD_TOL, [245](#)
- UBRR_VALUE, [245](#)
- UBRRH_VALUE, [245](#)
- UBRRL_VALUE, [245](#)
- USE_2X, [246](#)
- <util/twi.h>: TWI bit mask definitions, [246](#)
- TW_BUS_ERROR, [247](#)
- TW_MR_ARB_LOST, [247](#)
- TW_MR_DATA_ACK, [247](#)
- TW_MR_DATA_NACK, [247](#)
- TW_MR_SLA_ACK, [247](#)
- TW_MR_SLA_NACK, [247](#)
- TW_MT_ARB_LOST, [247](#)
- TW_MT_DATA_ACK, [247](#)
- TW_MT_DATA_NACK, [247](#)
- TW_MT_SLA_ACK, [248](#)
- TW_MT_SLA_NACK, [248](#)
- TW_NO_INFO, [248](#)
- TW_READ, [248](#)
- TW_REP_START, [248](#)
- TW_SR_ARB_LOST_GCALL_ACK, [248](#)
- TW_SR_ARB_LOST_SLA_ACK, [248](#)
- TW_SR_DATA_ACK, [248](#)
- TW_SR_DATA_NACK, [248](#)
- TW_SR_GCALL_ACK, [248](#)
- TW_SR_GCALL_DATA_ACK, [248](#)
- TW_SR_GCALL_DATA_NACK, [249](#)
- TW_SR_SLA_ACK, [249](#)
- TW_SR_STOP, [249](#)
- TW_ST_ARB_LOST_SLA_ACK, [249](#)
- TW_ST_DATA_ACK, [249](#)
- TW_ST_DATA_NACK, [249](#)
- TW_ST_LAST_DATA, [249](#)
- TW_ST_SLA_ACK, [249](#)
- TW_START, [249](#)
- TW_STATUS, [249](#)
- TW_STATUS_MASK, [249](#)
- TW_WRITE, [250](#)
- prefix, [70](#)
- \$PATH, [70](#)
- \$PREFIX, [70](#)
- _BV
 - <avr/sfr_defs.h>: Special function registers, [227](#)
- _EETGET
 - <avr/eeprom.h>: EEPROM handling, [167](#)
- _EETPUT
 - <avr/eeprom.h>: EEPROM handling, [168](#)
- _FDEV_EOF
 - stdio.h, [443](#)
- _FDEV_ERR
 - stdio.h, [444](#)
- _FDEV_SETUP_READ
 - stdio.h, [444](#)
- _FDEV_SETUP_RW
 - stdio.h, [444](#)
- _FDEV_SETUP_WRITE
 - stdio.h, [444](#)
- _FFS
 - <string.h>: Strings, [146](#)
- _MONTHS_
 - time.h, [492](#)
- _MemoryBarrier
 - <avr/cpufunc.h>: Special AVR CPU functions, [165](#)
- _NOP

- `<avr/cpufunc.h>`: Special AVR CPU functions, 166
- `__PROTECTED_WRITE`
- `<avr/io.h>`: AVR device-specific IO definitions, 195
- `__WEEK_DAYS__`
- `time.h`, 492
- `__AVR_LIBC_DATE__`
- `<avr/version.h>`: avr-libc version macros, 231
- `__AVR_LIBC_DATE_STRING__`
- `<avr/version.h>`: avr-libc version macros, 231
- `__AVR_LIBC_MAJOR__`
- `<avr/version.h>`: avr-libc version macros, 231
- `__AVR_LIBC_MINOR__`
- `<avr/version.h>`: avr-libc version macros, 231
- `__AVR_LIBC_REVISION__`
- `<avr/version.h>`: avr-libc version macros, 231
- `__AVR_LIBC_VERSION_STRING__`
- `<avr/version.h>`: avr-libc version macros, 231
- `__AVR_LIBC_VERSION__`
- `<avr/version.h>`: avr-libc version macros, 231
- `__EEGET`
- `<avr/eeprom.h>`: EEPROM handling, 167
- `__EEPUT`
- `<avr/eeprom.h>`: EEPROM handling, 167
- `__attribute__`
- `<avr/wdt.h>`: Watchdog timer handling, 234
- `power.h`, 352
- `__compar_fn_t`
- `stdlib.h`, 469
- `__malloc_heap_end`
- `stdlib.h`, 473
- `__malloc_heap_start`
- `stdlib.h`, 474
- `__malloc_margin`
- `stdlib.h`, 474
- `__crc16_update`
- `<util/crc16.h>`: CRC Computations, 238
- `__crc8_ccitt_update`
- `<util/crc16.h>`: CRC Computations, 238
- `__crc_ccitt_update`
- `<util/crc16.h>`: CRC Computations, 239
- `__crc_ibutton_update`
- `<util/crc16.h>`: CRC Computations, 240
- `__crc_xmodem_update`
- `<util/crc16.h>`: CRC Computations, 240
- `__delay_loop_1`
- `<util/delay_basic.h>`: Basic busy-wait delay loops, 243
- `__delay_loop_2`
- `<util/delay_basic.h>`: Basic busy-wait delay loops, 243
- `__delay_ms`
- `<util/delay.h>`: Convenience functions for busy-wait delay loops, 242
- `__delay_us`
- `<util/delay.h>`: Convenience functions for busy-wait delay loops, 242
- A more sophisticated project, 267
- A simple project, 256
- `abort`
- `stdlib.h`, 469
- `abs`
- `stdlib.h`, 469
- `acos`
- `math.h`, 414
- `acosh`
- `math.h`, 414
- Additional notes from `<avr/sfr_defs.h>`, 225
- `alloca`
- `<alloca.h>`: Allocate space in the stack, 106
- `alloca.h`, 290
- `asctime`
- `time.h`, 492
- `asctime_r`
- `time.h`, 492
- `asin`
- `math.h`, 414
- `asinh`
- `math.h`, 414
- `assert`
- `assert.h`, 291
- `assert.h`, 291
- `assert`, 291
- `atan`
- `math.h`, 414
- `atan2`
- `math.h`, 414
- `atan2f`
- `math.h`, 414
- `atanf`
- `math.h`, 414
- `atof`
- `<stdlib.h>`: General utilities, 141
- `atoi`
- `stdlib.h`, 469
- `atol`
- `stdlib.h`, 469
- `atomic.h`, 503, 504
- `ATOMIC_BLOCK`
- `<util/atomic.h>` Atomically and Non-Atomically Executed Code Blocks, 236
- `ATOMIC_FORCEON`
- `<util/atomic.h>` Atomically and Non-Atomically Executed Code Blocks, 236
- `ATOMIC_RESTORESTATE`
- `<util/atomic.h>` Atomically and Non-Atomically Executed Code Blocks, 236
- `avrdude`, usage, 97
- `avrprog`, usage, 97
- `BADISR_vect`
- `<avr/interrupt.h>`: Interrupts, 191
- `BAUD_TOL`
- `<util/setbaud.h>`: Helper macros for baud rate calculations, 245
- `bit_is_clear`
- `<avr/sfr_defs.h>`: Special function registers, 227
- `bit_is_set`

- `<avr/sfr_defs.h>`: Special function registers, 228
- `boot.h`, 293
- `boot_is_spm_interrupt`
 - `<avr/boot.h>`: Bootloader Support Utilities, 161
- `boot_lock_bits_set`
 - `<avr/boot.h>`: Bootloader Support Utilities, 161
- `boot_lock_bits_set_safe`
 - `<avr/boot.h>`: Bootloader Support Utilities, 161
- `boot_lock_fuse_bits_get`
 - `<avr/boot.h>`: Bootloader Support Utilities, 162
- `boot_page_erase`
 - `<avr/boot.h>`: Bootloader Support Utilities, 162
- `boot_page_erase_safe`
 - `<avr/boot.h>`: Bootloader Support Utilities, 162
- `boot_page_fill`
 - `<avr/boot.h>`: Bootloader Support Utilities, 162
- `boot_page_fill_safe`
 - `<avr/boot.h>`: Bootloader Support Utilities, 163
- `boot_page_write`
 - `<avr/boot.h>`: Bootloader Support Utilities, 163
- `boot_page_write_safe`
 - `<avr/boot.h>`: Bootloader Support Utilities, 163
- `boot_rww_busy`
 - `<avr/boot.h>`: Bootloader Support Utilities, 163
- `boot_rww_enable`
 - `<avr/boot.h>`: Bootloader Support Utilities, 163
- `boot_rww_enable_safe`
 - `<avr/boot.h>`: Bootloader Support Utilities, 164
- `boot_signature_byte_get`
 - `<avr/boot.h>`: Bootloader Support Utilities, 164
- `boot_spm_busy`
 - `<avr/boot.h>`: Bootloader Support Utilities, 164
- `boot_spm_busy_wait`
 - `<avr/boot.h>`: Bootloader Support Utilities, 164
- `boot_spm_interrupt_disable`
 - `<avr/boot.h>`: Bootloader Support Utilities, 164
- `boot_spm_interrupt_enable`
 - `<avr/boot.h>`: Bootloader Support Utilities, 164
- BOOTLOADER_SECTION
 - `<avr/boot.h>`: Bootloader Support Utilities, 165
- `bsearch`
 - `stdlib.h`, 470
- `calloc`
 - `stdlib.h`, 470
- `cbi`
 - `<compat/deprecated.h>`: Deprecated items, 251
- `cbrt`
 - `math.h`, 415
- `cbrtf`
 - `math.h`, 415
- `ccp_write_io`
 - `<avr/cpufunc.h>`: Special AVR CPU functions, 166
- `ceil`
 - `math.h`, 415
- `ceilf`
 - `math.h`, 415
- `clearerr`
 - `stdio.h`, 446
- `cli`
 - `<avr/interrupt.h>`: Interrupts, 191
- `clock_prescale_get`
 - `power.h`, 352
- `clock_prescale_set`
 - `<avr/power.h>`: Power Reduction Management, 225
- Combining C and assembly source files, 254
- `copysignf`
 - `math.h`, 415
- `cos`
 - `math.h`, 415
- `cosf`
 - `math.h`, 415
- `cosh`
 - `math.h`, 415
- `coshf`
 - `math.h`, 415
- `cpufunc.h`, 301
- `crc16.h`, 507, 508
- `ctime`
 - `time.h`, 492
- `ctime_r`
 - `time.h`, 493
- `ctype.h`, 397, 398
- `day`
 - `week_date`, 284
- `daylight_seconds`
 - `time.h`, 493
- `defines.h`, 287
- `delay.h`, 512, 513
- `delay_basic.h`, 516
- Demo projects, 253
- `deprecated.h`, 393
- `difftime`
 - `time.h`, 493
- `disassembling`, 260
- `div`
 - `stdlib.h`, 470
- `div_t`, 281
 - `quot`, 282
 - `rem`, 282
- `dtoa_conv.h`, 531
- DTOSTR_ALWAYS_SIGN
 - `<stdlib.h>`: General utilities, 140
- DTOSTR_PLUS_SIGN
 - `<stdlib.h>`: General utilities, 141
- DTOSTR_UPPERCASE
 - `<stdlib.h>`: General utilities, 141
- `dtostre`
 - `<stdlib.h>`: General utilities, 141
- `dtostrf`
 - `<stdlib.h>`: General utilities, 142
- EDOM
 - `<errno.h>`: System Errors, 110
- `eedef.h`, 529
- EEMEM

- `<avr/eeprom.h>`: EEPROM handling, 168
- `eeprom.h`, 302
- `eeprom_busy_wait`
 - `<avr/eeprom.h>`: EEPROM handling, 168
- `eeprom_is_ready`
 - `<avr/eeprom.h>`: EEPROM handling, 168
- `eeprom_read_block`
 - `<avr/eeprom.h>`: EEPROM handling, 168
- `eeprom_read_byte`
 - `<avr/eeprom.h>`: EEPROM handling, 168
- `eeprom_read_dword`
 - `<avr/eeprom.h>`: EEPROM handling, 168
- `eeprom_read_float`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_read_word`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_update_block`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_update_byte`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_update_dword`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_update_float`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_update_word`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_write_block`
 - `<avr/eeprom.h>`: EEPROM handling, 169
- `eeprom_write_byte`
 - `<avr/eeprom.h>`: EEPROM handling, 170
- `eeprom_write_dword`
 - `<avr/eeprom.h>`: EEPROM handling, 170
- `eeprom_write_float`
 - `<avr/eeprom.h>`: EEPROM handling, 170
- `eeprom_write_word`
 - `<avr/eeprom.h>`: EEPROM handling, 170
- `EMPTY_INTERRUPT`
 - `<avr/interrupt.h>`: Interrupts, 191
- `enable_external_int`
 - `<compat/deprecated.h>`: Deprecated items, 251
- `EOF`
 - `stdio.h`, 444
- `ephemera_common.h`, 533
- `equation_of_time`
 - `time.h`, 493
- `ERANGE`
 - `<errno.h>`: System Errors, 110
- `errno`
 - `<errno.h>`: System Errors, 110
- `errno.h`, 400
- `eu_dst.h`, 518
- Example using the two-wire interface (TWI), 277
- `exit`
 - `stdlib.h`, 470
- `EXIT_FAILURE`
 - `<stdlib.h>`: General utilities, 141
- `EXIT_SUCCESS`
 - `<stdlib.h>`: General utilities, 141
- `exp`
 - `math.h`, 416
- `expf`
 - `math.h`, 416
- `F_CPU`
 - `<util/delay.h>`: Convenience functions for busy-wait delay loops, 241
- `fabsf`
 - `math.h`, 416
- `FAQ`, 48
- `fatfs_time`
 - `time.h`, 493
- `fclose`
 - `stdio.h`, 446
- `fdev_close`
 - `stdio.h`, 444
- `fdev_get_adata`
 - `stdio.h`, 444
- `fdev_set_adata`
 - `stdio.h`, 444
- `FDEV_SETUP_STREAM`
 - `stdio.h`, 444
- `fdev_setup_stream`
 - `stdio.h`, 445
- `fdevopen`
 - `<stdio.h>`: Standard IO facilities, 139
- `fdevopen.c`, 530
- `fdim`
 - `math.h`, 416
- `fdimf`
 - `math.h`, 416
- `feof`
 - `stdio.h`, 446
- `ferror`
 - `stdio.h`, 447
- `fflush`
 - `stdio.h`, 447
- `ffs`
 - `<string.h>`: Strings, 146
- `ffsl`
 - `<string.h>`: Strings, 146
- `ffsll`
 - `<string.h>`: Strings, 146
- `fgetc`
 - `stdio.h`, 447
- `fgets`
 - `stdio.h`, 447
- `FILE`
 - `stdio.h`, 446
- `floor`
 - `math.h`, 416
- `floorf`
 - `math.h`, 416
- `fma`
 - `math.h`, 416
- `fmaf`
 - `math.h`, 416
- `fmax`

- math.h, [417](#)
- fmaxf
 - math.h, [417](#)
- fmin
 - math.h, [417](#)
- fminf
 - math.h, [417](#)
- fmod
 - math.h, [417](#)
- fmodf
 - math.h, [417](#)
- fprintf
 - stdio.h, [447](#)
- fprintf_P
 - stdio.h, [447](#)
- fputc
 - stdio.h, [447](#)
- fputs
 - stdio.h, [448](#)
- fputs_P
 - stdio.h, [448](#)
- fread
 - stdio.h, [448](#)
- free
 - stdlib.h, [470](#)
- frexp
 - math.h, [417](#)
- frexpf
 - math.h, [417](#)
- fscanf
 - stdio.h, [448](#)
- fscanf_P
 - stdio.h, [448](#)
- fuse.h, [305](#)
- fwrite
 - stdio.h, [448](#)
- GET_EXTENDED_FUSE_BITS
 - <avr/boot.h>: Bootloader Support Utilities, [165](#)
- GET_HIGH_FUSE_BITS
 - <avr/boot.h>: Bootloader Support Utilities, [165](#)
- GET_LOCK_BITS
 - <avr/boot.h>: Bootloader Support Utilities, [165](#)
- GET_LOW_FUSE_BITS
 - <avr/boot.h>: Bootloader Support Utilities, [165](#)
- getc
 - stdio.h, [445](#)
- getchar
 - stdio.h, [445](#)
- gets
 - stdio.h, [449](#)
- gm_sideral
 - time.h, [493](#)
- gmtime
 - time.h, [493](#)
- gmtime_r
 - time.h, [493](#)
- hd44780.h, [288](#)
- hypot
 - math.h, [418](#)
- hypotf
 - math.h, [418](#)
- ina90.h, [396](#)
- inb
 - <compat/deprecated.h>: Deprecated items, [251](#)
- INFINITY
 - math.h, [412](#)
- inp
 - <compat/deprecated.h>: Deprecated items, [251](#)
- installation, [69](#)
- installation, avarice, [74](#)
- installation, avr-libc, [73](#)
- installation, avrdude, [73](#)
- installation, avrprog, [73](#)
- installation, binutils, [71](#)
- installation, gcc, [72](#)
- Installation, gdb, [74](#)
- installation, simulavr, [74](#)
- INT16_C
 - <stdint.h>: Standard Integer Types, [129](#)
- INT16_MAX
 - <stdint.h>: Standard Integer Types, [129](#)
- INT16_MIN
 - <stdint.h>: Standard Integer Types, [129](#)
- int16_t
 - <stdint.h>: Standard Integer Types, [134](#)
- INT32_C
 - <stdint.h>: Standard Integer Types, [129](#)
- INT32_MAX
 - <stdint.h>: Standard Integer Types, [129](#)
- INT32_MIN
 - <stdint.h>: Standard Integer Types, [129](#)
- int32_t
 - <stdint.h>: Standard Integer Types, [134](#)
- INT64_C
 - <stdint.h>: Standard Integer Types, [129](#)
- INT64_MAX
 - <stdint.h>: Standard Integer Types, [129](#)
- INT64_MIN
 - <stdint.h>: Standard Integer Types, [129](#)
- int64_t
 - <stdint.h>: Standard Integer Types, [134](#)
- INT8_C
 - <stdint.h>: Standard Integer Types, [129](#)
- INT8_MAX
 - <stdint.h>: Standard Integer Types, [129](#)
- INT8_MIN
 - <stdint.h>: Standard Integer Types, [130](#)
- int8_t
 - <stdint.h>: Standard Integer Types, [134](#)
- int_farptr_t
 - <inttypes.h>: Integer Type conversions, [123](#)
- INT_FAST16_MAX
 - <stdint.h>: Standard Integer Types, [130](#)
- INT_FAST16_MIN
 - <stdint.h>: Standard Integer Types, [130](#)

- `int_fast16_t`
 - `<stdint.h>`: Standard Integer Types, [134](#)
- `INT_FAST32_MAX`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `INT_FAST32_MIN`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `int_fast32_t`
 - `<stdint.h>`: Standard Integer Types, [134](#)
- `INT_FAST64_MAX`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `INT_FAST64_MIN`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `int_fast64_t`
 - `<stdint.h>`: Standard Integer Types, [134](#)
- `INT_FAST8_MAX`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `INT_FAST8_MIN`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `int_fast8_t`
 - `<stdint.h>`: Standard Integer Types, [135](#)
- `INT_LEAST16_MAX`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `INT_LEAST16_MIN`
 - `<stdint.h>`: Standard Integer Types, [130](#)
- `int_least16_t`
 - `<stdint.h>`: Standard Integer Types, [135](#)
- `INT_LEAST32_MAX`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `INT_LEAST32_MIN`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `int_least32_t`
 - `<stdint.h>`: Standard Integer Types, [135](#)
- `INT_LEAST64_MAX`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `INT_LEAST64_MIN`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `int_least64_t`
 - `<stdint.h>`: Standard Integer Types, [135](#)
- `INT_LEAST8_MAX`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `INT_LEAST8_MIN`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `int_least8_t`
 - `<stdint.h>`: Standard Integer Types, [135](#)
- `INTERRUPT`
 - `<compat/deprecated.h>`: Deprecated items, [251](#)
- `interrupt.h`, [309](#), [310](#)
- `INTMAX_C`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `INTMAX_MAX`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `INTMAX_MIN`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `intmax_t`
 - `<stdint.h>`: Standard Integer Types, [135](#)
- `INTPTR_MAX`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `INTPTR_MIN`
 - `<stdint.h>`: Standard Integer Types, [131](#)
- `<stdint.h>`: Standard Integer Types, [131](#)
- `intptr_t`
 - `<stdint.h>`: Standard Integer Types, [135](#)
- `inttypes.h`, [402](#), [404](#)
- `io.h`, [314](#)
- `iocompat.h`, [285](#)
- `is_leap_year`
 - `time.h`, [494](#)
- `isalnum`
 - `<ctype.h>`: Character Operations, [107](#)
- `isalpha`
 - `<ctype.h>`: Character Operations, [107](#)
- `isascii`
 - `<ctype.h>`: Character Operations, [107](#)
- `isblank`
 - `<ctype.h>`: Character Operations, [108](#)
- `iscntrl`
 - `<ctype.h>`: Character Operations, [108](#)
- `isdigit`
 - `<ctype.h>`: Character Operations, [108](#)
- `isfinite`
 - `math.h`, [418](#)
- `isfinitef`
 - `math.h`, [418](#)
- `isgraph`
 - `<ctype.h>`: Character Operations, [108](#)
- `isinf`
 - `math.h`, [418](#)
- `isinff`
 - `math.h`, [418](#)
- `islower`
 - `<ctype.h>`: Character Operations, [108](#)
- `isnan`
 - `math.h`, [419](#)
- `isnanf`
 - `math.h`, [419](#)
- `iso_week_date`
 - `time.h`, [494](#)
- `iso_week_date_r`
 - `time.h`, [494](#)
- `isotime`
 - `time.h`, [494](#)
- `isotime_r`
 - `time.h`, [494](#)
- `isprint`
 - `<ctype.h>`: Character Operations, [108](#)
- `ispunct`
 - `<ctype.h>`: Character Operations, [108](#)
- `ISR`
 - `<avr/interrupt.h>`: Interrupts, [192](#)
- `ISR_ALIAS`
 - `<avr/interrupt.h>`: Interrupts, [192](#)
- `ISR_ALIASOF`
 - `<avr/interrupt.h>`: Interrupts, [192](#)
- `ISR_BLOCK`
 - `<avr/interrupt.h>`: Interrupts, [192](#)
- `ISR_FLATTEN`
 - `<avr/interrupt.h>`: Interrupts, [193](#)

- ISR_NAKED
 - <avr/interrupt.h>: Interrupts, [193](#)
- ISR_NOBLOCK
 - <avr/interrupt.h>: Interrupts, [193](#)
- ISR_NOICF
 - <avr/interrupt.h>: Interrupts, [193](#)
- isspace
 - <ctype.h>: Character Operations, [108](#)
- isupper
 - <ctype.h>: Character Operations, [108](#)
- isxdigit
 - <ctype.h>: Character Operations, [109](#)
- itoa
 - <stdlib.h>: General utilities, [142](#)
- labs
 - stdlib.h, [471](#)
- lcd.h, [289](#)
- ldexp
 - math.h, [419](#)
- ldexpf
 - math.h, [419](#)
- ldiv
 - stdlib.h, [471](#)
- ldiv_t, [282](#)
 - quot, [282](#)
 - rem, [282](#)
- lm_sideréal
 - time.h, [494](#)
- localtime
 - time.h, [494](#)
- localtime_r
 - time.h, [494](#)
- lock.h, [321](#)
- log
 - math.h, [419](#)
- log10
 - math.h, [419](#)
- log10f
 - math.h, [419](#)
- logf
 - math.h, [419](#)
- longjmp
 - <setjmp.h>: Non-local goto, [125](#)
- loop_until_bit_is_clear
 - <avr/sfr_defs.h>: Special function registers, [228](#)
- loop_until_bit_is_set
 - <avr/sfr_defs.h>: Special function registers, [228](#)
- lrint
 - math.h, [420](#)
- lrintf
 - math.h, [420](#)
- lround
 - math.h, [420](#)
- lroundf
 - math.h, [420](#)
- ltoa
 - <stdlib.h>: General utilities, [142](#)
- M_1_PI
 - math.h, [412](#)
- M_2_PI
 - math.h, [412](#)
- M_2_SQRTPI
 - math.h, [413](#)
- M_E
 - <math.h>: Mathematics, [124](#)
- M_LN10
 - math.h, [413](#)
- M_LN2
 - math.h, [413](#)
- M_LOG10E
 - math.h, [413](#)
- M_LOG2E
 - math.h, [413](#)
- M_PI
 - math.h, [413](#)
- M_PI_2
 - math.h, [413](#)
- M_PI_4
 - math.h, [413](#)
- M_SQRT1_2
 - math.h, [413](#)
- M_SQRT2
 - math.h, [413](#)
- malloc
 - stdlib.h, [471](#)
- math.h, [410](#), [423](#)
 - acos, [414](#)
 - acosf, [414](#)
 - asin, [414](#)
 - asinf, [414](#)
 - atan, [414](#)
 - atan2, [414](#)
 - atan2f, [414](#)
 - atanf, [414](#)
 - cbrt, [415](#)
 - cbrtf, [415](#)
 - ceil, [415](#)
 - ceilf, [415](#)
 - copysignf, [415](#)
 - cos, [415](#)
 - cosf, [415](#)
 - cosh, [415](#)
 - coshf, [415](#)
 - exp, [416](#)
 - expf, [416](#)
 - fabsf, [416](#)
 - fdim, [416](#)
 - fdimf, [416](#)
 - floor, [416](#)
 - floorf, [416](#)
 - fma, [416](#)
 - fmaf, [416](#)
 - fmax, [417](#)
 - fmaxf, [417](#)
 - fmin, [417](#)

- fminf, [417](#)
- fmod, [417](#)
- fmodf, [417](#)
- frexp, [417](#)
- frexpf, [417](#)
- hypot, [418](#)
- hypotf, [418](#)
- INFINITY, [412](#)
- isfinite, [418](#)
- isfinitef, [418](#)
- isinf, [418](#)
- isinff, [418](#)
- isnan, [419](#)
- isnanf, [419](#)
- ldexp, [419](#)
- ldexpf, [419](#)
- log, [419](#)
- log10, [419](#)
- log10f, [419](#)
- logf, [419](#)
- lrint, [420](#)
- lrintf, [420](#)
- lround, [420](#)
- lroundf, [420](#)
- M_1_PI, [412](#)
- M_2_PI, [412](#)
- M_2_SQRTPI, [413](#)
- M_LN10, [413](#)
- M_LN2, [413](#)
- M_LOG10E, [413](#)
- M_LOG2E, [413](#)
- M_PI, [413](#)
- M_PI_2, [413](#)
- M_PI_4, [413](#)
- M_SQRT1_2, [413](#)
- M_SQRT2, [413](#)
- modf, [420](#)
- modff, [420](#)
- NAN, [413](#)
- pow, [421](#)
- powf, [421](#)
- round, [421](#)
- roundf, [421](#)
- signbit, [421](#)
- signbitf, [421](#)
- sin, [422](#)
- sinf, [422](#)
- sinh, [422](#)
- sinhf, [422](#)
- sqrt, [422](#)
- sqrtf, [422](#)
- square, [422](#)
- squaref, [422](#)
- tan, [422](#)
- tanf, [423](#)
- tanh, [423](#)
- tanhf, [423](#)
- trunc, [423](#)
- truncf, [423](#)
- memcpy
 - <string.h>: Strings, [146](#)
- memcpy_P
 - <avr/pgmspace.h>: Program Space Utilities, [206](#)
- memchr
 - <string.h>: Strings, [147](#)
- memchr_P
 - <avr/pgmspace.h>: Program Space Utilities, [206](#)
- memcmp
 - <string.h>: Strings, [147](#)
- memcmp_P
 - <avr/pgmspace.h>: Program Space Utilities, [206](#)
- memcmp_PF
 - <avr/pgmspace.h>: Program Space Utilities, [207](#)
- memcpy
 - <string.h>: Strings, [147](#)
- memcpy_P
 - <avr/pgmspace.h>: Program Space Utilities, [207](#)
- memcpy_PF
 - <avr/pgmspace.h>: Program Space Utilities, [207](#)
- memmem
 - <string.h>: Strings, [148](#)
- memmem_P
 - <avr/pgmspace.h>: Program Space Utilities, [208](#)
- memmove
 - <string.h>: Strings, [148](#)
- memrchr
 - <string.h>: Strings, [148](#)
- memrchr_P
 - <avr/pgmspace.h>: Program Space Utilities, [208](#)
- memset
 - <string.h>: Strings, [149](#)
- mk_gmtime
 - time.h, [495](#)
- mktime
 - time.h, [495](#)
- modf
 - math.h, [420](#)
- modff
 - math.h, [420](#)
- month_length
 - time.h, [495](#)
- moon_phase
 - time.h, [495](#)
- NAN
 - math.h, [413](#)
- NONATOMIC_BLOCK
 - <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks, [236](#)
- NONATOMIC_FORCEOFF
 - <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks, [237](#)
- NONATOMIC_RESTORESTATE
 - <util/atomic.h> Atomically and Non-Atomically Executed Code Blocks, [237](#)
- NTP_OFFSET
 - time.h, [491](#)

- ONE_DAY
 - time.h, [491](#)
- ONE_DEGREE
 - time.h, [492](#)
- ONE_HOUR
 - time.h, [492](#)
- outb
 - <compat/deprecated.h>: Deprecated items, [252](#)
- outp
 - <compat/deprecated.h>: Deprecated items, [252](#)
- parity.h, [519](#)
- parity_even_bit
 - <util/parity.h>: Parity bit generation, [244](#)
- pgm_get_far_address
 - <avr/pgmspace.h>: Program Space Utilities, [199](#)
- PGM_P
 - <avr/pgmspace.h>: Program Space Utilities, [199](#)
- pgm_read_byte
 - <avr/pgmspace.h>: Program Space Utilities, [199](#)
- pgm_read_byte_far
 - <avr/pgmspace.h>: Program Space Utilities, [199](#)
- pgm_read_byte_near
 - <avr/pgmspace.h>: Program Space Utilities, [200](#)
- pgm_read_dword
 - <avr/pgmspace.h>: Program Space Utilities, [200](#)
- pgm_read_dword_far
 - <avr/pgmspace.h>: Program Space Utilities, [200](#)
- pgm_read_dword_near
 - <avr/pgmspace.h>: Program Space Utilities, [200](#)
- pgm_read_float
 - <avr/pgmspace.h>: Program Space Utilities, [200](#)
- pgm_read_float_far
 - <avr/pgmspace.h>: Program Space Utilities, [201](#)
- pgm_read_float_near
 - <avr/pgmspace.h>: Program Space Utilities, [201](#)
- pgm_read_ptr
 - <avr/pgmspace.h>: Program Space Utilities, [201](#)
- pgm_read_ptr_far
 - <avr/pgmspace.h>: Program Space Utilities, [201](#)
- pgm_read_ptr_near
 - <avr/pgmspace.h>: Program Space Utilities, [201](#)
- pgm_read_word
 - <avr/pgmspace.h>: Program Space Utilities, [202](#)
- pgm_read_word_far
 - <avr/pgmspace.h>: Program Space Utilities, [202](#)
- pgm_read_word_near
 - <avr/pgmspace.h>: Program Space Utilities, [202](#)
- PGM_VOID_P
 - <avr/pgmspace.h>: Program Space Utilities, [202](#)
- pgmspace.h, [324](#), [326](#)
- portpins.h, [345](#)
- pow
 - math.h, [421](#)
- power.h, [352](#), [353](#)
 - __attribute__, [352](#)
 - clock_prescale_get, [352](#)
- powf
 - math.h, [421](#)
- PRId16
 - <inttypes.h>: Integer Type conversions, [113](#)
- PRId32
 - <inttypes.h>: Integer Type conversions, [113](#)
- PRId8
 - <inttypes.h>: Integer Type conversions, [113](#)
- PRIdFAST16
 - <inttypes.h>: Integer Type conversions, [113](#)
- PRIdFAST32
 - <inttypes.h>: Integer Type conversions, [113](#)
- PRIdFAST8
 - <inttypes.h>: Integer Type conversions, [113](#)
- PRIdLEAST16
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIdLEAST32
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIdLEAST8
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIdPTR
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIf16
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIf32
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIf8
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIfAST16
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIfAST32
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIfAST8
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIfLEAST16
 - <inttypes.h>: Integer Type conversions, [114](#)
- PRIfLEAST32
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIfLEAST8
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIfPTR
 - <inttypes.h>: Integer Type conversions, [115](#)
- printf
 - stdio.h, [449](#)
- printf_P
 - stdio.h, [449](#)
- PRIo16
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIo32
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIo8
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIoFAST16
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIoFAST32
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIoFAST8
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIoLEAST16
 - <inttypes.h>: Integer Type conversions, [115](#)

- PRIoLEAST32
 - <inttypes.h>: Integer Type conversions, [115](#)
- PRIoLEAST8
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRIoPTR
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRlu16
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRlu32
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRlu8
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRluFAST16
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRluFAST32
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRluFAST8
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRluLEAST16
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRluLEAST32
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRluLEAST8
 - <inttypes.h>: Integer Type conversions, [116](#)
- PRluPTR
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIX16
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIx16
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIX32
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIx32
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIX8
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIx8
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIXFAST16
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIxFAST16
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIXFAST32
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIxFAST32
 - <inttypes.h>: Integer Type conversions, [117](#)
- PRIXFAST8
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIxFAST8
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIXLEAST16
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIxLEAST16
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIXLEAST32
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIxLEAST32
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIXLEAST8
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIxLEAST8
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIXPTR
 - <inttypes.h>: Integer Type conversions, [118](#)
- PRIxPTR
 - <inttypes.h>: Integer Type conversions, [118](#)
- prog_char
 - <avr/pgmspace.h>: Program Space Utilities, [203](#)
- prog_int16_t
 - <avr/pgmspace.h>: Program Space Utilities, [203](#)
- prog_int32_t
 - <avr/pgmspace.h>: Program Space Utilities, [203](#)
- prog_int64_t
 - <avr/pgmspace.h>: Program Space Utilities, [203](#)
- prog_int8_t
 - <avr/pgmspace.h>: Program Space Utilities, [204](#)
- prog_uchar
 - <avr/pgmspace.h>: Program Space Utilities, [204](#)
- prog_uint16_t
 - <avr/pgmspace.h>: Program Space Utilities, [204](#)
- prog_uint32_t
 - <avr/pgmspace.h>: Program Space Utilities, [205](#)
- prog_uint64_t
 - <avr/pgmspace.h>: Program Space Utilities, [205](#)
- prog_uint8_t
 - <avr/pgmspace.h>: Program Space Utilities, [205](#)
- prog_void
 - <avr/pgmspace.h>: Program Space Utilities, [206](#)
- PROGMEM
 - <avr/pgmspace.h>: Program Space Utilities, [202](#)
- project.h, [285](#)
- PSTR
 - <avr/pgmspace.h>: Program Space Utilities, [203](#)
- PTRDIFF_MAX
 - <stdint.h>: Standard Integer Types, [132](#)
- PTRDIFF_MIN
 - <stdint.h>: Standard Integer Types, [132](#)
- putc
 - stdio.h, [445](#)
- putchar
 - stdio.h, [445](#)
- puts
 - stdio.h, [449](#)
- puts_P
 - stdio.h, [449](#)
- qsort
 - stdlib.h, [471](#)
- quot
 - div_t, [282](#)
 - ldiv_t, [282](#)
- rand
 - stdlib.h, [471](#)
- RAND_MAX
 - stdlib.h, [469](#)
- rand_r

- stdlib.h, [472](#)
- random
 - <stdlib.h>: General utilities, [143](#)
- RANDOM_MAX
 - <stdlib.h>: General utilities, [141](#)
- random_r
 - <stdlib.h>: General utilities, [143](#)
- realloc
 - stdlib.h, [472](#)
- rem
 - div_t, [282](#)
 - ldiv_t, [282](#)
- reti
 - <avr/interrupt.h>: Interrupts, [193](#)
- round
 - math.h, [421](#)
- roundf
 - math.h, [421](#)
- sbi
 - <compat/deprecated.h>: Deprecated items, [252](#)
- scanf
 - stdio.h, [449](#)
- scanf_P
 - stdio.h, [449](#)
- SCNd16
 - <inttypes.h>: Integer Type conversions, [118](#)
- SCNd32
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNd8
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNdFAST16
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNdFAST32
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNdFAST8
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNdLEAST16
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNdLEAST32
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNdLEAST8
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNdPTR
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNi16
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNi32
 - <inttypes.h>: Integer Type conversions, [119](#)
- SCNi8
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNiFAST16
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNiFAST32
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNiFAST8
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNiLEAST16
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNiLEAST32
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNiLEAST8
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNiPTR
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNo16
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNo32
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNo8
 - <inttypes.h>: Integer Type conversions, [120](#)
- SCNoFAST16
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNoFAST32
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNoFAST8
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNoLEAST16
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNoLEAST32
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNoLEAST8
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNoPTR
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNu16
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNu32
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNu8
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNuFAST16
 - <inttypes.h>: Integer Type conversions, [121](#)
- SCNuFAST32
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNuFAST8
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNuLEAST16
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNuLEAST32
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNuLEAST8
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNuPTR
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNx16
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNx32
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNx8
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNxFAST16
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNxFAST32
 - <inttypes.h>: Integer Type conversions, [122](#)
- SCNxFAST8
 - <inttypes.h>: Integer Type conversions, [123](#)

- SCNxLEAST16
 - <inttypes.h>: Integer Type conversions, [123](#)
- SCNxLEAST32
 - <inttypes.h>: Integer Type conversions, [123](#)
- SCNxLEAST8
 - <inttypes.h>: Integer Type conversions, [123](#)
- SCNxPTR
 - <inttypes.h>: Integer Type conversions, [123](#)
- sei
 - <avr/interrupt.h>: Interrupts, [193](#)
- set_dst
 - time.h, [495](#)
- set_position
 - time.h, [495](#)
- set_system_time
 - time.h, [496](#)
- set_zone
 - time.h, [496](#)
- setbaud.h, [520](#)
- setjmp
 - <setjmp.h>: Non-local goto, [125](#)
- setjmp.h, [429](#)
- sfr_defs.h, [374](#)
- SIG_ATOMIC_MAX
 - <stdint.h>: Standard Integer Types, [132](#)
- SIG_ATOMIC_MIN
 - <stdint.h>: Standard Integer Types, [132](#)
- SIGNAL
 - <avr/interrupt.h>: Interrupts, [193](#)
- signal.h, [378](#)
- signature.h, [378](#)
- signbit
 - math.h, [421](#)
- signbitf
 - math.h, [421](#)
- sin
 - math.h, [422](#)
- sinf
 - math.h, [422](#)
- sinh
 - math.h, [422](#)
- sinhf
 - math.h, [422](#)
- SIZE_MAX
 - <stdint.h>: Standard Integer Types, [132](#)
- sleep.h, [379](#)
- sleep_bod_disable
 - <avr/sleep.h>: Power Management and Sleep Modes, [230](#)
- sleep_cpu
 - <avr/sleep.h>: Power Management and Sleep Modes, [230](#)
- sleep_disable
 - <avr/sleep.h>: Power Management and Sleep Modes, [230](#)
- sleep_enable
 - <avr/sleep.h>: Power Management and Sleep Modes, [230](#)
- sleep_mode
 - <avr/sleep.h>: Power Management and Sleep Modes, [230](#)
- snprintf
 - stdio.h, [449](#)
- snprintf_P
 - stdio.h, [450](#)
- solar_declination
 - time.h, [496](#)
- solar_noon
 - time.h, [496](#)
- sprintf
 - stdio.h, [450](#)
- sprintf_P
 - stdio.h, [450](#)
- sqrt
 - math.h, [422](#)
- sqrtf
 - math.h, [422](#)
- square
 - math.h, [422](#)
- squaref
 - math.h, [422](#)
- srand
 - stdlib.h, [472](#)
- random
 - <stdlib.h>: General utilities, [143](#)
- sscanf
 - stdio.h, [450](#)
- sscanf_P
 - stdio.h, [450](#)
- stderr
 - stdio.h, [446](#)
- stdin
 - stdio.h, [446](#)
- stdint.h, [431](#), [434](#)
- stdio.h, [442](#), [456](#)
 - _FDEV_EOF, [443](#)
 - _FDEV_ERR, [444](#)
 - _FDEV_SETUP_READ, [444](#)
 - _FDEV_SETUP_RW, [444](#)
 - _FDEV_SETUP_WRITE, [444](#)
 - clearerr, [446](#)
 - EOF, [444](#)
 - fclose, [446](#)
 - fdev_close, [444](#)
 - fdev_get_udata, [444](#)
 - fdev_set_udata, [444](#)
 - FDEV_SETUP_STREAM, [444](#)
 - fdev_setup_stream, [445](#)
 - feof, [446](#)
 - ferror, [447](#)
 - fflush, [447](#)
 - fgetc, [447](#)
 - fgets, [447](#)
 - FILE, [446](#)
 - fprintf, [447](#)
 - fprintf_P, [447](#)

- fputc, 447
- fputs, 448
- fputs_P, 448
- fread, 448
- fscanf, 448
- fscanf_P, 448
- fwrite, 448
- getc, 445
- getchar, 445
- gets, 449
- printf, 449
- printf_P, 449
- putc, 445
- putchar, 445
- puts, 449
- puts_P, 449
- scanf, 449
- scanf_P, 449
- snprintf, 449
- snprintf_P, 450
- sprintf, 450
- sprintf_P, 450
- sscanf, 450
- sscanf_P, 450
- stderr, 446
- stdin, 446
- stdout, 446
- ungetc, 450
- vfprintf, 451
- vfprintf_P, 453
- vfscanf, 453
- vfscanf_P, 454
- vprintf, 455
- vscanf, 455
- vsprintf, 455
- vsprintf_P, 455
- vsprintf_P, 455
- stdio_private.h, 530
- stdlib.h, 467, 474
 - __compar_fn_t, 469
 - __malloc_heap_end, 473
 - __malloc_heap_start, 474
 - __malloc_margin, 474
- abort, 469
- abs, 469
- atoi, 469
- atol, 469
- bsearch, 470
- calloc, 470
- div, 470
- exit, 470
- free, 470
- labs, 471
- ldiv, 471
- malloc, 471
- qsort, 471
- rand, 471
- RAND_MAX, 469
- rand_r, 472
- realloc, 472
- srand, 472
- strtod, 472
- strtol, 472
- strtoul, 473
- stdlib_private.h, 532
- stdout
 - stdio.h, 446
- strcasecmp
 - <string.h>: Strings, 149
- strcasecmp_P
 - <avr/pgmspace.h>: Program Space Utilities, 208
- strcasecmp_PF
 - <avr/pgmspace.h>: Program Space Utilities, 209
- strcasestr
 - <string.h>: Strings, 149
- strcasestr_P
 - <avr/pgmspace.h>: Program Space Utilities, 209
- strcat
 - <string.h>: Strings, 149
- strcat_P
 - <avr/pgmspace.h>: Program Space Utilities, 209
- strcat_PF
 - <avr/pgmspace.h>: Program Space Utilities, 209
- strchr
 - <string.h>: Strings, 150
- strchr_P
 - <avr/pgmspace.h>: Program Space Utilities, 210
- strchrnul
 - <string.h>: Strings, 150
- strchrnul_P
 - <avr/pgmspace.h>: Program Space Utilities, 210
- strcmp
 - <string.h>: Strings, 150
- strcmp_P
 - <avr/pgmspace.h>: Program Space Utilities, 210
- strcmp_PF
 - <avr/pgmspace.h>: Program Space Utilities, 210
- strcpy
 - <string.h>: Strings, 151
- strcpy_P
 - <avr/pgmspace.h>: Program Space Utilities, 212
- strcpy_PF
 - <avr/pgmspace.h>: Program Space Utilities, 212
- strcspn
 - <string.h>: Strings, 151
- strcspn_P
 - <avr/pgmspace.h>: Program Space Utilities, 212
- strdup
 - <string.h>: Strings, 151
- strftime
 - time.h, 496
- string.h, 482, 483
- strlcat
 - <string.h>: Strings, 152
- strlcat_P

- `<avr/pgmspace.h>`: Program Space Utilities, 213
- `strlcat_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 213
- `strncpy`
 - `<string.h>`: Strings, 152
- `strncpy_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 213
- `strncpy_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 214
- `strlen`
 - `<string.h>`: Strings, 153
- `strlen_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 214
- `strlen_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 214
- `strlwr`
 - `<string.h>`: Strings, 153
- `strncasecmp`
 - `<string.h>`: Strings, 153
- `strncasecmp_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 215
- `strncasecmp_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 215
- `strncat`
 - `<string.h>`: Strings, 154
- `strncat_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 216
- `strncat_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 216
- `strncmp`
 - `<string.h>`: Strings, 154
- `strncmp_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 216
- `strncmp_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 217
- `strncpy`
 - `<string.h>`: Strings, 154
- `strncpy_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 217
- `strncpy_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 217
- `strlen`
 - `<string.h>`: Strings, 155
- `strlen_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 218
- `strlen_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 218
- `strpbrk`
 - `<string.h>`: Strings, 155
- `strpbrk_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 218
- `strchr`
 - `<string.h>`: Strings, 155
- `strchr_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 219
- `strrev`
 - `<string.h>`: Strings, 156
- `strsep`
 - `<string.h>`: Strings, 156
- `strsep_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 219
- `strspn`
 - `<string.h>`: Strings, 156
- `strspn_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 219
- `strstr`
 - `<string.h>`: Strings, 157
- `strstr_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 220
- `strstr_PF`
 - `<avr/pgmspace.h>`: Program Space Utilities, 220
- `strtod`
 - `stdlib.h`, 472
- `strtok`
 - `<string.h>`: Strings, 157
- `strtok_P`
 - `<avr/pgmspace.h>`: Program Space Utilities, 220
- `strtok_r`
 - `<string.h>`: Strings, 157
- `strtok_rP`
 - `<avr/pgmspace.h>`: Program Space Utilities, 221
- `strtol`
 - `stdlib.h`, 472
- `strtoul`
 - `stdlib.h`, 473
- `strupr`
 - `<string.h>`: Strings, 158
- `sun_rise`
 - `time.h`, 496
- `sun_set`
 - `time.h`, 497
- supported devices, 2
- `system_tick`
 - `time.h`, 497
- `tan`
 - `math.h`, 422
- `tanf`
 - `math.h`, 423
- `tanh`
 - `math.h`, 423
- `tanhf`
 - `math.h`, 423
- `time`
 - `time.h`, 497
- `time.h`, 490, 498
 - `_MONTHS_`, 492
 - `_WEEK_DAYS_`, 492
 - `asctime`, 492
 - `asctime_r`, 492
 - `ctime`, 492
 - `ctime_r`, 493
 - `daylight_seconds`, 493
 - `difftime`, 493
 - `equation_of_time`, 493
 - `fats_time`, 493
 - `gm_sidereal`, 493

- gmtime, [493](#)
- gmtime_r, [493](#)
- is_leap_year, [494](#)
- iso_week_date, [494](#)
- iso_week_date_r, [494](#)
- isotime, [494](#)
- isotime_r, [494](#)
- lm_sidereal, [494](#)
- localtime, [494](#)
- localtime_r, [494](#)
- mk_gmtime, [495](#)
- mktime, [495](#)
- month_length, [495](#)
- moon_phase, [495](#)
- NTP_OFFSET, [491](#)
- ONE_DAY, [491](#)
- ONE_DEGREE, [492](#)
- ONE_HOUR, [492](#)
- set_dst, [495](#)
- set_position, [495](#)
- set_system_time, [496](#)
- set_zone, [496](#)
- solar_declination, [496](#)
- solar_noon, [496](#)
- strftime, [496](#)
- sun_rise, [496](#)
- sun_set, [497](#)
- system_tick, [497](#)
- time, [497](#)
- UNIX_OFFSET, [492](#)
- week_of_month, [497](#)
- week_of_year, [497](#)
- time_t
 - <time.h>: Time, [159](#)
- timer_enable_int
 - <compat/deprecated.h>: Deprecated items, [252](#)
- tm, [283](#)
 - tm_hour, [283](#)
 - tm_isdst, [283](#)
 - tm_mday, [283](#)
 - tm_min, [283](#)
 - tm_mon, [283](#)
 - tm_sec, [284](#)
 - tm_wday, [284](#)
 - tm_yday, [284](#)
 - tm_year, [284](#)
- tm_hour
 - tm, [283](#)
- tm_isdst
 - tm, [283](#)
- tm_mday
 - tm, [283](#)
- tm_min
 - tm, [283](#)
- tm_mon
 - tm, [283](#)
- tm_sec
 - tm, [284](#)
- tm_wday
 - tm, [284](#)
- tm_yday
 - tm, [284](#)
- tm_year
 - tm, [284](#)
- toascii
 - <ctype.h>: Character Operations, [109](#)
- tolower
 - <ctype.h>: Character Operations, [109](#)
- tools, optional, [71](#)
- tools, required, [70](#)
- toupper
 - <ctype.h>: Character Operations, [109](#)
- trunc
 - math.h, [423](#)
- truncf
 - math.h, [423](#)
- TW_BUS_ERROR
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MR_ARB_LOST
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MR_DATA_ACK
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MR_DATA_NACK
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MR_SLA_ACK
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MR_SLA_NACK
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MT_ARB_LOST
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MT_DATA_ACK
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MT_DATA_NACK
 - <util/twi.h>: TWI bit mask definitions, [247](#)
- TW_MT_SLA_ACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_MT_SLA_NACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_NO_INFO
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_READ
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_REP_START
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_SR_ARB_LOST_GCALL_ACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_SR_ARB_LOST_SLA_ACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_SR_DATA_ACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_SR_DATA_NACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_SR_GCALL_ACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)
- TW_SR_GCALL_DATA_ACK
 - <util/twi.h>: TWI bit mask definitions, [248](#)

- TW_SR_GCALL_DATA_NACK
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_SR_SLA_ACK
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_SR_STOP
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_ST_ARB_LOST_SLA_ACK
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_ST_DATA_ACK
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_ST_DATA_NACK
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_ST_LAST_DATA
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_ST_SLA_ACK
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_START
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_STATUS
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_STATUS_MASK
 - <util/twi.h>: TWI bit mask definitions, [249](#)
- TW_WRITE
 - <util/twi.h>: TWI bit mask definitions, [250](#)
- twi.h, [523–525](#)
- uart.h, [289](#)
- UBRR_VALUE
 - <util/setbaud.h>: Helper macros for baud rate calculations, [245](#)
- UBRRH_VALUE
 - <util/setbaud.h>: Helper macros for baud rate calculations, [245](#)
- UBRR_L_VALUE
 - <util/setbaud.h>: Helper macros for baud rate calculations, [245](#)
- UINT16_C
 - <stdint.h>: Standard Integer Types, [132](#)
- UINT16_MAX
 - <stdint.h>: Standard Integer Types, [132](#)
- uint16_t
 - <stdint.h>: Standard Integer Types, [135](#)
- UINT32_C
 - <stdint.h>: Standard Integer Types, [132](#)
- UINT32_MAX
 - <stdint.h>: Standard Integer Types, [132](#)
- uint32_t
 - <stdint.h>: Standard Integer Types, [135](#)
- UINT64_C
 - <stdint.h>: Standard Integer Types, [132](#)
- UINT64_MAX
 - <stdint.h>: Standard Integer Types, [132](#)
- uint64_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT8_C
 - <stdint.h>: Standard Integer Types, [133](#)
- UINT8_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint8_t
 - <stdint.h>: Standard Integer Types, [136](#)
- <stdint.h>: Standard Integer Types, [136](#)
- uint_farptr_t
 - <inttypes.h>: Integer Type conversions, [123](#)
- UINT_FAST16_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_fast16_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT_FAST32_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_fast32_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT_FAST64_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_fast64_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT_FAST8_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_fast8_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT_LEAST16_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_least16_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT_LEAST32_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_least32_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT_LEAST64_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_least64_t
 - <stdint.h>: Standard Integer Types, [136](#)
- UINT_LEAST8_MAX
 - <stdint.h>: Standard Integer Types, [133](#)
- uint_least8_t
 - <stdint.h>: Standard Integer Types, [137](#)
- UINTMAX_C
 - <stdint.h>: Standard Integer Types, [133](#)
- UINTMAX_MAX
 - <stdint.h>: Standard Integer Types, [134](#)
- uintmax_t
 - <stdint.h>: Standard Integer Types, [137](#)
- UINTPTR_MAX
 - <stdint.h>: Standard Integer Types, [134](#)
- uintptr_t
 - <stdint.h>: Standard Integer Types, [137](#)
- ultoa
 - <stdlib.h>: General utilities, [143](#)
- ungetc
 - stdio.h, [450](#)
- UNIX_OFFSET
 - time.h, [492](#)
- usa_dst.h, [527](#)
- USE_2X
 - <util/setbaud.h>: Helper macros for baud rate calculations, [246](#)
- Using the standard IO facilities, [272](#)
- utoa
 - <stdlib.h>: General utilities, [144](#)

version.h, [384](#)
vfprintf
 stdio.h, [451](#)
vfprintf_P
 stdio.h, [453](#)
vfscanf
 stdio.h, [453](#)
vfscanf_P
 stdio.h, [454](#)
vprintf
 stdio.h, [455](#)
vscanf
 stdio.h, [455](#)
vsprintf
 stdio.h, [455](#)
vsprintf_P
 stdio.h, [455](#)

wdt.h, [385](#)
wdt_reset
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_120MS
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_15MS
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_1S
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_250MS
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_2S
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_30MS
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_4S
 <avr/wdt.h>: Watchdog timer handling, [233](#)
WDTO_500MS
 <avr/wdt.h>: Watchdog timer handling, [234](#)
WDTO_60MS
 <avr/wdt.h>: Watchdog timer handling, [234](#)
WDTO_8S
 <avr/wdt.h>: Watchdog timer handling, [234](#)
week
 week_date, [285](#)
week_date, [284](#)
 day, [284](#)
 week, [285](#)
 year, [285](#)
week_of_month
 time.h, [497](#)
week_of_year
 time.h, [497](#)

xmega.h, [392](#)
xtoa_fast.h, [531](#)

year
 week_date, [285](#)