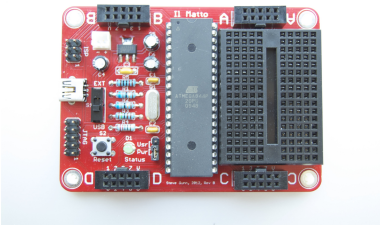
Quick Reference

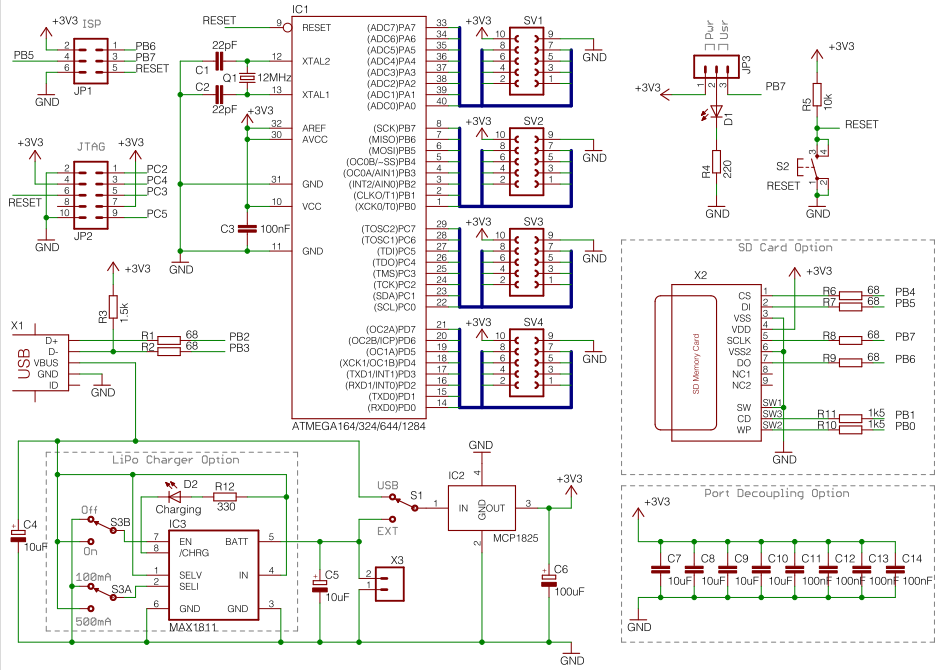
Il Matto



Version: August 24, 2012

Steve R. Gunn

Electronics and Computer Science
University of Southampton



Features

ATmega	164P/PA	324P/PA	644P/PA	1284P
Flash	16K	32K	64K	128K
Boot	2K	4K	8K	8K
EEPROM	512	1K	2K	4K
SRAM	1K	2K	4K	16K

Architecture	Harvard, RISC
Instructions	131 (most single cycle) 8-bit × 8-bit multiply (two cycles)
Registers	32×8-bit general purpose
Frequency	12MHz
Speed	Up to 12 MIPS
Voltage	3.3V (CMOS Logic levels)
Output	Symmetrical pin drive (40mA max.) $V_{OH} \geq 2.53V$, $V_{OL} \leq 0.66V$ (@10mA)
Input	Tri-state or internal pull-up (~35kΩ) $V_{IH} \geq 1.98V$, $V_{IL} \leq 0.99V$
Serial	2×UART, I²C, SPI
Analogue	ADC (8 channels, 10-bit, 15kSPS) Analogue comparator
Timer/Counter	2×8-bit, 1×16-bit
PWM	six (two for each timer/counter)
Programming	JTAG, ISP, USB
Power Source	USB, 3.3V, or external (3.6–6.0V)
Power Usage	<5mA in native operation
Debug	JTAG boundary scan & On-chip debug
Options	SD Card (SPI interface), LiPo charger

Configuration

```
avrdude -c programmer -p device -U memtype:op:filename[:format]
```

programmer	Interface	Flash	EEPROM	Fuses	Lock
usbasp	USB	☒	☒	☐	☒
c232hm	ISP	☒	☒	☒	☒
c232hm	JTAG	☒	☒	☒	☒

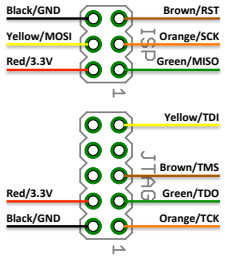
device	Compatible devices
m164p	ATmega164P/PA
m324p	ATmega324P/PA
m644p	ATmega644P/PA
m1284p	ATmega1284P

memtype	Description
flash	flash ROM
eeprom	EEPROM
lfuse	low fuse byte
hfuse	high fuse byte
efuse	extended fuse byte
lock	lock byte
calibration	RC oscillator cal. byte
signature	three device sig. bytes

op	Description
r	read
w	write
v	verify

format	Description
i	Intel Hex
s	Motorola S-record
r	raw binary (l. endian)
m	byte values (on CL)
a	auto-detect (input)
d	decimal (output)
h	hexadecimal (output)
o	octal (output)
b	binary (output)

	JTAG	ISP	I²C	SPI	UART
red			Vcc		
ora	TCK	SCK	SCL	SK	TXD
ylw	TDI	MOSI	SDA	DO	RXD
grn	TDO	MISO	SDA	DI	$\overline{RTS}$
brn	TMS	RST	—	CS	$\overline{CTS}$
blk			Gnd		



Pin Functions

Port	Pin	Name	Description
A	PA7	ADC7	ADC input channel 7
	PA*n*	ADC*n*	ADC input channel *n*
	PA0	ADC0	ADC input channel 0
B	PB7	SCK	SPI Bus Master clock input
	PB6	MISO	SPI Bus Master Input/Slave Output
	PB5	MOSI	SPI Bus Master Output/Slave Input
	PB4	$\overline{SS}$	SPI Slave Select input
	PB3	AIN1 OC0B	Analog Comparator Negative Input Timer/Counter 0 Output Compare Match B Output
	PB2	AIN0 OC0A	Analog Comparator Positive Input Timer/Counter 0 Output Compare Match A Output
	PB1	T1 CLKO	Timer/Counter 1 External Counter Input Divided System Clock Output
	PB0	T0 XCK0	Timer/Counter 0 External Counter Input USART0 External Clock Input/Output
C	PC7	TOSC2	Timer Oscillator pin 2
	PC6	TOSC1	Timer Oscillator pin 1
	PC5	TDI	JTAG Test Data Input
	PC4	TDO	JTAG Test Data Output
	PC3	TMS	JTAG Test Mode Select
	PC2	TCK	JTAG Test Clock
	PC1	SDA	2-wire Serial Bus Data Input/Output Line
	PC0	SCL	2-wire Serial Bus Clock Line
D	PD7	OC2A	Timer/Counter2 Output Compare Match A Output
	PD6	ICP1 OC2B	Timer/Counter1 Input Capture Trigger Timer/Counter2 Output Compare Match B Output
	PD5	OC1A	Timer/Counter1 Output Compare Match A Output
	PD4	OC1B XCK1	Timer/Counter1 Output Compare Match B Output USART1 External Clock Input/Output
	PD3	INT1 TXD1	External Interrupt1 Input USART1 Transmit Pin
	PD2	INT0 RXD1	External Interrupt0 Input USART1 Receive Pin
	PD1	TXD0	USART0 Transmit Pin
	PD0	RXD0	USART0 Receive Pin

Notes: Each pin also has a pin change interrupt.
Each pin can also be configured as GPIO.

Interrupts

No.	Address[1]	Source	Interrupt Definition
1	$0000	RESET[2]	AVR system reset condition[3]
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	PCINT0	Pin Change Interrupt Request 0
6	$000A	PCINT1	Pin Change Interrupt Request 1
7	$000C	PCINT2	Pin Change Interrupt Request 2
8	$000E	PCINT3	Pin Change Interrupt Request 3
9	$0010	WDT	Watchdog Time-out Interrupt
10	$0012	TIMER2_COMPA	Timer/Counter2 Compare Match A
11	$0014	TIMER2_COMPB	Timer/Counter2 Compare Match B
12	$0016	TIMER2_OVF	Timer/Counter2 Overflow
13	$0018	TIMER1_CAPT	Timer/Counter1 Capture Event
14	$001A	TIMER1_COMPA	Timer/Counter1 Compare Match A
15	$001C	TIMER1_COMPB	Timer/Counter1 Compare Match B
16	$001E	TIMER1_OVF	Timer/Counter1 Overflow
17	$0020	TIMER0_COMPA	Timer/Counter0 Compare Match A
18	$0022	TIMER0_COMPB	Timer/Counter0 Compare match B
19	$0024	TIMER0_OVF	Timer/Counter0 Overflow
20	$0026	SPI_STC	SPI Serial Transfer Complete
21	$0028	USART0_RX	USART0 Rx Complete
22	$002A	USART0_UDRE	USART0 Data Register Empty
23	$002C	USART0_TX	USART0 Tx Complete
24	$002E	ANALOG_COMP	Analog Comparator
25	$0030	ADC	ADC Conversion Complete
26	$0032	EE_READY	EEPROM Ready
27	$0034	TWI	2-wire Serial Interface
28	$0036	SPM_READY	Store Program Memory Ready
29	$0038	USART1_RX	USART1 Rx Complete
30	$003A	USART1_UDRE	USART1 Data Register Empty
31	$003C	USART1_TX	USART1 Tx Complete

Notes: [1]If IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section.
[2]If BOOTRST Fuse is set, it will jump to the Boot Loader address.
[3]External Pin, Power-on, Brown-out, Watchdog, and JTAG Reset.

Fuses & Lock Bits

Fuse	Bit	Default	Name	Description
Low	7	☒	CKDIV8	Divide clock by 8
	6	☐	CKOUT	Enable clock output on pin PB1
	5	☐	SUT1	Select start-up time
	4	☒	SUT0	
	3	☒	CKSEL3	Select Clock Source
	2	☒	CKSEL2	
	1	☐	CKSEL1	
	0	☒	CKSEL0	
High	7	☐	OCDEN	Enable On Chip Debug
	6	☒	JTAGEN	Enable JTAG
	5	☒	SPIEN	Enable Serial programming and Data Downloading
	4	☐	WDTON	Watchdog timer always on
	3	☐	EESAVE	EEPROM memory is preserved through chip erase
	2	☒	BOOTSZ1	Select Boot Size
	1	☒	BOOTSZ0	
	0	☐	BOOTRST	Select Reset Vector
Extended	7	☐		
	6	☐		
	5	☐		
	4	☐		
	3	☐		
	2	☐	BODLEVEL2	
	1	☐	BODLEVEL1	Brown-out Detector trigger level
	0	☐	BODLEVEL0	
Lock	7	☐		
	6	☐		
	5	☐	BLB12	
	4	☐	BLB11	
	3	☐	BLB02	
	2	☐	BLB01	
	1	☐	LB2	
	0	☐	LB1	

☐ means unprogrammed (1); ☒ means programmed (0).

AVR C Library

`<alloca.h>`	Allocate space in the stack
`<assert.h>`	Diagnostics
`<ctype.h>`	Character Operations
`<errno.h>`	System Errors
`<inttypes.h>`	Integer Type conversions
`<math.h>`	Mathematics
`<setjmp.h>`	Non-local goto
`<stdint.h>`	Standard Integer Types
`<stdio.h>`	Standard IO facilities
`<stdlib.h>`	General utilities
`<string.h>`	Strings

`<avr/*.h>`

`<boot.h>`	Bootloader Support Utilities
`<cpufunc.h>`	Special AVR CPU functions
`<eeprom.h>`	EEPROM handling
`<fuse.h>`	Fuse Support
`<interrupt.h>`	Interrupts
`<io.h>`	AVR device-specific IO definitions
`<lock.h>`	Lockbit Support
`<pgmspace.h>`	Program Space Utilities
`<power.h>`	Power Reduction Management
`<sfr_defs.h>`	Special function registers
`<signature.h>`	Signature Support
`<sleep.h>`	Power Management and Sleep Modes
`<version.h>`	avr-libc version macros
`<wdt.h>`	Watchdog timer handling

`<util/*.h>`

`<atomic.h>`	Atomically and Non-Atomically Executed Code Blocks
`<crc16.h>`	CRC Computations
`<delay.h>`	Convenience functions for busy-wait delay loops
`<delay_basic.h>`	Basic busy-wait delay loops
`<parity.h>`	Parity bit generation
`<setbaud.h>`	Helper macros for baud rate calculations
`<twi.h>`	TWI bit mask definitions

Interrupts

Interrupts ($s, t \in$ {RESET, INT0, ..., TIMER2_OVF, ..., USART1_TX})

```
ISR(s_vect) { /* Handler code */ }
ISR(s_vect, ISR_NOBLOCK) { /* Handler code (interruptable) */ }
ISR(t_vect, ISR_ALIASOF(s_vect)); /* Shared handler */
sei(); /* Enable interrupts */
cli(); /* Disable all interrupts */
ISR(BADISR_vect) { /* Handle any undefined interrupt */ }
EMPTY_INTERRUPT(s_vect); /* Empty Handler */ }
ISR(s_vect, ISR_NAKED) { /* Handler code (no prolog/epilog) */
    /* Save SREG if modified. */ reti(); }
```

External Interrupts ($n \in \{0,1,2\}$, $c \in \{\_\_, \_\!\sqcap\!\_, \_\!\urcorner\!\_, \_\!\sqcap\!\urcorner\}$)

```
EIMSK &= ~_BV(INTn); /* Disable */
EICRA |= _BV(ISCn1) | _BV(ISCn0); /* c=_⌐ */
ISR(INTn_vect) { /* Handler code */ }
EIMSK |= _BV(INTn); /* Enable */
```

Pin Change Interrupts ($x \in \{0,1,2,3\}$, $n \in \{0,1,\ldots,7\}$)

```
ISR(PCINTx_vect) { /* Handler code */ }
PCICR = _BV(PCIEx); /* Enable Portx (0→A,1→B,2→C,3→D) */
PCMSKx &= _BV(n); /* Enable bit n on Portx */
```

Integer Types

`uint8_t`	8-bit unsigned int	(0 ... 255)
`int8_t`	8-bit signed int	(-128 ... 127)
`uint16_t`	16-bit unsigned int	(0 ... 65535)
`int16_t`	16-bit signed int	(-32768 ... 32767)
`uintptr_t`	unsigned int pointer	(0x0000 ... 0xFFFF)
`uint32_t`	32-bit unsigned int	(0 ... 4294967295)
`int32_t`	32-bit signed int	(-2147483648 ... 2147483647)
`uint64_t`	64-bit unsigned int	(0 ... 2^{64}-1)
`int64_t`	64-bit signed int	(-2^{63} ... 2^{63}-1)

```
typedef uint8_t byte;
typedef uint16_t word;
typedef uint32_t dword;
```

Input/Output

Port Registers ($x \in$ {A,B,C,D})

`PINx, PORTx, DDRx`	Input, Output and Direction registers

Bit Manipulation ($n \in \{0,1,\ldots,7\}$, $r \in$ {I/O Registers})

`uint8_t value;`	Declare `value` as an 8-bit byte
`#define _BV(n) (1 << (n))`	Bit Value
`value = 0xFF;`	Set all 8-bits of byte `value`
`value = 0x00;`	Clear all 8-bits of byte `value`
`value = ~value;`	Invert all bits of byte `value`
`value |= _BV(n);`	Set bit n of byte `value`
`value &= ~_BV(n);`	Clear bit n of byte `value`
`if bit_is_set(r, n) { ... }`	Test if bit n of r is set
`if bit_is_clear(r, n) { ... }`	Test if bit n of r is clear
`loop_until_bit_is_set(r, n);`	Wait until bit n of r is set
`loop_until_bit_is_clear(r, n);`	Wait until bit n of r is clear

Input

`DDRx = 0x00;`	Set 8-bits of port x as inputs
`PORTx = 0xFF;`	Enable pull-ups on input port x
`PORTx = 0x00;`	Configure inputs as tri-state on port x
`value = PINx;`	Read value of port x
`DDRx &= ~_BV(n);`	Set bit n of port x as input
`PORTx |= _BV(n);`	Enable pull-up on bit n of port x
`PORTx &= ~_BV(n);`	Configure tri-state on bit n of port x
`if (PINx & _BV(n)) { ... }`	Test value of pin n on port x

Output

`DDRx = 0xFF;`	Set 8-bits of port x as outputs
`PORTx = 0xFF;`	Set all output bits on port x high
`PORTx = 0x00;`	Set all output bits on port x low
`PINx = 0xFF;`	Toggle all output bits on port x high
`DDRx |= _BV(n);`	Set bit n of port x as output
`PORTx |= _BV(n);`	Set bit n of port x high
`PORTx &= ~_BV(n);`	Set bit n of port x low
`PINx |= _BV(n);`	Toggle bit n of port x

Input/Output

`DDRx = 0x0F;`	High 4-bits input, low 4-bits output
`PORTx |= 0x0F;`	Set all output bits high
`PORTx &= 0xF0;`	Set all output bits low
`value = PINx & 0xF0;`	Read input bits on port x

Communication

UART ($n \in \{0,1\}$, BAUD $\in$ {1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200})

```
#define BAUD 57600
#include <util/setbaud.h>
void init_uart(void) { /* 8N1 */
  UBRRnH = UBRRH_VALUE; UBRRnL = UBRRL_VALUE;
  UCSRnA = USE_2X << U2Xn;
  UCSRnB = _BV(RXENn) | _BV(TXENn);
  UCSRnC = _BV(UCSZn0) | _BV(UCSZn1); }
void tx(uint8_t b) { while(!(UCSRnA & _BV(UDREn))); UDRn = b; }
uint8_t rx(void) { while(!(UCSRnA & _BV(RXCn))); return UDRn; }
```

SPI ($F\_SCK = F\_CPU/d$, $d \in \{2,4,8,16,32,64,128\}$)

```
void init_spi_master(void) { /* out: MOSI, SCK, /SS, in: MISO */
  DDRB = _BV(PB4) | _BV(PB5) | _BV(PB7);
  SPCR = _BV(SPE) | _BV(MSTR) | _BV(SPI2X); /* F_SCK = F_CPU/2 */ }
void tx(uint8_t b) { SPDR = b; while(!(SPSR & _BV(SPIF))); }
void init_spi_slave(void) { /* out: MISO, in: MOSI, SCK, /SS */
  DDRB = _BV(PB6); SPCR = _BV(SPE); /* Enable SPI */ }
uint8_t rx(void) { while(!(SPSR & _BV(SPIF))); return SPDR; }
```

I²C ($F\_SCL = F\_CPU/d$, d=2(8+b4²), $b \in \{0,1,\ldots,255\}$, $p \in \{0,1,2,3\}$)

```
#include <util/twi.h>
void start(void) { TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN);
  while(!(TWCR & _BV(TWINT))); }
void stop(void) { TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); }
void tx(uint8_t b) { TWDR = b; TWCR = _BV(TWINT) | _BV(TWEN);
  while(!(TWCR & _BV(TWINT))); }
uint8_t rx(void) { TWCR = _BV(TWINT) | _BV(TWEN);
  while(!(TWCR & _BV(TWINT))); return TWDR; }
TWBR = 0x34; TWSR = 0x00; /* F_SCL = F_CPU/120 */
start();              assert(TW_STATUS == TW_START);
tx(SLA | TW_READ);    assert(TW_STATUS == TW_MR_SLA_ACK);
uint8_t b = rx();     assert(TW_STATUS == TW_MR_DATA_NACK);
start();              assert(TW_STATUS == TW_REP_START);
tx(SLA | TW_WRITE);   assert(TW_STATUS == TW_MT_SLA_ACK);
tx(b);                assert(TW_STATUS == TW_MT_DATA_ACK);
stop();
```

Utilities

Single source file project: Command to compile and link

```
avr-gcc -mmcu=d -DF_CPU=f -Wall -Os prog.c -o prog.elf
```

Multiple source file project: Command to compile

```
avr-gcc -mmcu=d -DF_CPU=f -Wall -Os -c prog1.c -o prog1.o
```

Multiple source file project: Command to link

```
avr-gcc -mmcu=d -DF_CPU=f -Wall -Os -o prog.elf prog1.o prog2.o...
```

$d \in$ {atmega164p, atmega324p, atmega644p, atmega1284p}, f=12000000

Create target file `avr-objcopy -O ihex prog.elf prog.hex`

Disassembly `avr-objdump -h -S prog.elf` Size `avr-size prog.elf`

CPU Functions

```
#include <avr/cpufunc.h>
_NOP(); /* Execute a no operation (one clock cycle) */
_MemoryBarrier(); /* Do not cache memory data in registers */
```

Delays

```
#include <util/delay.h>
void _delay_us(double x); /* x microseconds */
void _delay_ms(double x); /* x milliseconds */
```

Redirect stdin, stdout and stderr (with assertions) to UARTn ($n \in \{0,1\}$)

```
#define __ASSERT_USE_STDERR
#include <assert.h>
#include <stdio.h>
#include <avr/io.h>
void uputchar(char c, FILE *stream) {
  if (c == '\n') uputchar('\r', stream);
  loop_until_bit_is_set(UCSRnA, UDREn); UDRn = c; }
char ugetchar(FILE *stream) {
  loop_until_bit_is_set(UCSRnA, RXCn); return UDRn;}
FILE uout = FDEV_SETUP_STREAM(uputchar, NULL, _FDEV_SETUP_WRITE);
FILE uin = FDEV_SETUP_STREAM(NULL, ugetchar, _FDEV_SETUP_READ);
init_uart(); stdout = &uout; stderr = &uout; stdin = &uin;
```

Memory

Program Memory (16-bit)

Application Flash ROM	0x0000	
Boot Flash ROM	FLASHEND	

Data Memory (8-bit)

32 Registers		0x0000-0x001F
64 I/O Registers		0x0020-0x005F
160 Ext I/O Reg.		0x0060-0x00FF
	.data vars	0x0100
	.bss vars	__bss_start
SRAM	heap ↓	__heap_start
	stack ↑	SP -RAMEND

EEPROM (8-bit)

	0x0000
	E2END

	ATmega	164P/PA	324P/PA	644P/PA	1284P
word*	FLASHEND	0x1FFF	0x3FFF	0x7FFF	0xFFFF
byte*	RAMEND	0x04FF	0x08FF	0x10FF	0x40FF
byte*	E2END	0x01FF	0x03FF	0x07FF	0x0FFF
	11	0x0080	0x0100	0x0200	0x0200
	10	0x0100	0x0200	0x0400	0x0400
word*	BOOTSZ 01	0x0200	0x0400	0x0800	0x0800
	00	0x0400	0x0800	0x1000	0x1000

EEPROM ($t \in$ {byte, word, dword, float})

```
#include <avr/eeprom.h>
t eeprom_read_t(t *p);
void eeprom_write_t(t *p, t value);
void eeprom_update_t(t *p, t value);
void eeprom_read_block(void *dst, const void *src, size_t n);
void eeprom_write_block(const void *src, void *dst, size_t n);
void eeprom_update_block(const void *src, void *dst, size_t n);
```

Program Memory ($t \in$ {byte, word, dword, float})

```
#include <avr/pgmspace.h>
const float f PROGMEM = 3.14; /* Declare object in flash ROM */
const uint8_t data[] PROGMEM = {0x01,0x45,0x76,0xA6,...,0x61};
PGM_P s_ptr; /* Pointer to string in program space */
PGM_VOID_P g_ptr; /* Pointer to generic object in program space */
PSTR(s_ptr); /* Static pointer to string in program space */
t pgm_read_t(uint16_t byte_address);[1]
t pgm_read_t_far(uint32_t byte_address);
```

[1] Data must be in lower 64KB for ATmega1284P

Timers & Counters

Timers/Counters ($n \in \{0,1,2\}$, $x \in \{A,B\}$, $m \in \{0,1,2,3\}$)

n	Bits	BOT	MAX	Internal Clock (F_CPU/d)	External Clock
0	8	0x00	0xFF	$d \in \{1,8,64,256,1024\}$	T0
1	16	0x0000	0xFFFF	$d \in \{1,8,64,256,1024\}$	T1
2	8	0x00	0xFF	$d \in \{1,8,32,64,128,256,1024\}$	TOSC1

Mode	WGMn $n\in\{0,2\}$	WGMn n=1	TOP	OCRnx Update	TOVn Set	COMnxm m=0	m=1	m=2	m=3
Normal	0	0	MAX						
CTC	2	4	OCRnA	IMD	MAX	—			
	-	12	ICRn						
PWM (Fast)	3	5	0xFF						
	-	6	0x01FF						
	-	7	0x03FF	BOT	TOP	—			
	7	15	OCRnA				A	0	0
	-	14	ICRn				A		
PWM (PC)	1	1	0xFF						
	-	2	0x01FF						
	-	3	0x03FF	TOP	BOT	—			
	5	11	OCRnA				A		
	-	10	ICRn						
PWM (PFC)	-	9	OCRnA	BOT	BOT	—	A		
	-	8	ICRn						

```
DDRB |= _BV(CT1A); /* Enable CT1 output A */
PORTB &= ~_BV(CT1A);/* Set output A low */
ICR1 = 1000; OCR1A = 950; /* Duty cycle 5% */
TCCR1A = 0; TCCR1B = _BV(WGM13); /* Mode 8: WGM1 = 0b1000 */
TCCR1A |= _BV(COM1A1) | _BV(COM1A0); /* PWM on Output A */
TCCR1B |= _BV(CS12) | _BV(CS10); /* F_TC1 = F_CPU/1024 */
```

Watchdog Timer ($T_WDT = 2^p \times 16$ms, $p \in \{0,1,\ldots,9\}$)

```
#include <avr/wdt.h>
/* Disable Watchdog */
MCUSR &= ~_BV(WDRF); WDTCSR |= _BV(WDCE); WDTSCR = 0x00;
/* Enable 0.5s Watchdog (p=5) */
ISR(WDT_vect) { /* Handler Code */ }
WDTCSR |= _BV(WDCE); /* Four cycles to set-up WDTCSR */
WDTCSR = _BV(WDP2) | _BV(WDP0) | _BV(WDIE);
```

Analogue

ADC ($n \in \{0,1,\ldots,7\}$, $F\_ADC = F\_CPU/d$, $d \in \{2,4,8,16,32,64,128\}$)

```
ADCSRA |= _BV(ADPS2) | _BV(ADPS1); /* F_ADC = F_CPU/64 */
ADMUX = n; /* Select channel n */
ADMUX |= _BV(REFS0); /* AVCC reference */
ADMUX |= _BV(ADLAR); /* ADCH contains 8 MSBs */
ADCSRA |= _BV(ADATE); /* Automatic Trigger Enable */
ISR(ADC_vect) { /* ADCH & ADCL contain result */ }
ADCSRA |= _BV(ADEN); /* Enable ADC */
ADCSRA |= _BV(ADSC); /* Start Conversions */
ADCSRA |= _BV(ADIE); /* Enable interrupt */
```

Comparator ($A+ \in \{AINO, V_{BG}\}$, $A- \in \{AIN1, PAn\}$, $n \in \{0,\ldots,7\}$, $c \in \{\_\!\sqcap\!\_, \_\!\urcorner\!\_, \_\!\sqcap\!\urcorner\}$)

```
ACSR &= ~_BV(ACBG); /* AINO Select */
ADCSRB &= ~_BV(ACME); /* AIN1 Select */
ACSR |= _BV(ACBG); /* Analog Comparator Bandgap Select (1.1V) */
ADCSRA &= ~_BV(ADEN); /* Disable ADC */
ADCSRB |= _BV(ACME); /* Analog Comparator Multiplexer Enable */
ADMUX = n; /* Select input PAn */
ACSR &= ~_BV(ACIE); /* Disable Interrupt */
ISR(ANALOG_COMP_vect) { /* Handler code */ }
ACSR |= _BV(ACIS1) | _BV(ACISO); /* c=_⌐ */
ACSR |= _BV(ACIE); /* Enable Interrupt */
```

Power Management

Sleep ($m \in$ {IDLE, ADC, PWR_DOWN, PWR_SAVE, STANDBY, EXT_STANDBY})

```
#include <avr/sleep.h>
set_sleep_mode(m); sleep_mode();
```

Power Saving ($u \in$ {adc, spi, timer0, timer1, timer2, twi, usart0, usart1, all})

```
#include <avr/power.h>
power_u_enable(); power_u_disable();
```

Frequency Scaling ($F\_CPU = F\_OSC/2^p$, $p \in \{0,1,\ldots,8\}$)

```
#include <avr/power.h>
void clock_prescale_set(clock_div_t p);
clock_div_t clock_prescale_get();
```