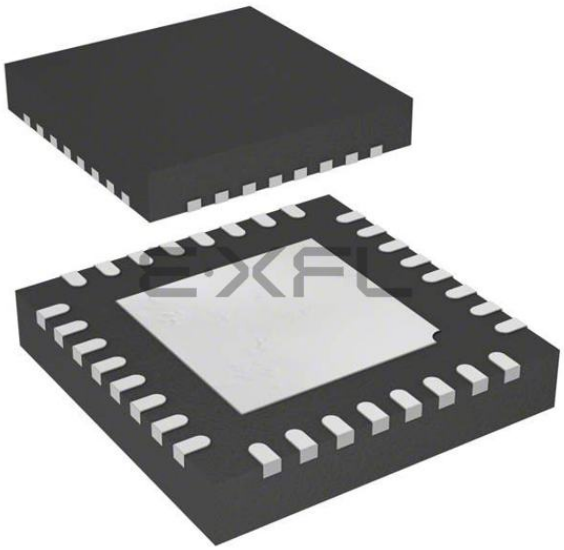


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

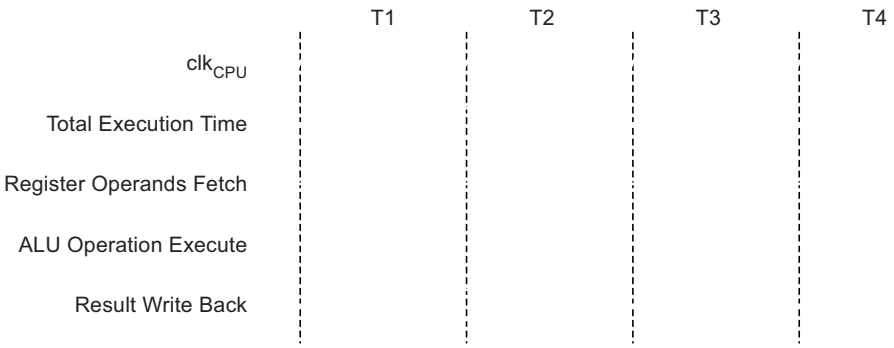
"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	CANbus, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, Temp Sensor, WDT
Number of I/O	-
Program Memory Size	16KB (8K x 16)
Program Memory Type	FLASH
EEPROM Size	512 x 8
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 11x10b; D/A 1x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 150°C (TA)
Mounting Type	Surface Mount
Package / Case	32-VQFN Exposed Pad
Supplier Device Package	32-QFN (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega16m1-15md

Figure 3-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 3-5. Single Cycle ALU Operation



3.8 Reset and Interrupt Handling

The AVR® provides several different interrupt sources. These interrupts and the separate reset vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the global interrupt enable bit in the status register in order to enable the interrupt. Depending on the program counter value, interrupts may be automatically disabled when boot lock bits BLB02 or BLB12 are programmed. This feature improves software security. See Section 25. “Memory Programming” on page 255 for details.

The lowest addresses in the program memory space are by default defined as the reset and interrupt vectors. The complete list of vectors is shown in Section 8. “Interrupts” on page 47. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is ANACOMP0 – the analog comparator 0 interrupt. The interrupt vectors can be moved to the start of the boot flash section by setting the IVSEL bit in the MCU control register (MCUCR). Refer to Section 8. “Interrupts” on page 47 for more information. The reset vector can also be moved to the start of the boot flash section by programming the BOOTRST fuse, see Section 24. “Boot Loader Support – Read-while-write Self-Programming ATmega16/32/64/M1/C1” on page 241.

3.8.1 Interrupt Behavior

When an interrupt occurs, the global interrupt enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a return from interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the interrupt flag. For these interrupts, the program counter is vectored to the actual interrupt vector in order to execute the interrupt handling routine, and hardware clears the corresponding interrupt flag. Interrupt flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the interrupt flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the global interrupt enable bit is cleared, the corresponding interrupt flag(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have interrupt flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the status register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
in      r16, SREG               ; store SREG value
cli                             ; disable interrupts during timed sequence
sbi     EECR, EEMWE             ; start EEPROM write
sbi     EECR, EEWE
out     SREG, r16               ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG;                   /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMWE); /* start EEPROM write */
EECR |= (1<<EEWE);
SREG = cSREG;                   /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

Assembly Code Example

```
sei             ; set Global Interrupt Enable
sleep           ; enter sleep, waiting for interrupt
; note: will enter sleep before any pending
; interrupt(s)
```

C Code Example

```
_SEI(); /* set Global Interrupt Enable */
_SLEEP(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

3.8.2 Interrupt Response Time

The interrupt execution response for all the enabled AVR® interrupts is four clock cycles minimum. After four clock cycles the program vector address for the actual interrupt handling routine is executed. during this four clock cycle period, the program counter is pushed onto the stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. during these four clock cycles, the program counter (two bytes) is popped back from the stack, the stack pointer is incremented by two, and the I-bit in SREG is set.

Therefore it is recommended not to take the OSCCAL adjustments to a higher frequency than 8MHz in order to keep the PLL in the correct operating range.

The internal PLL is enabled only when the PLLE bit in the register PLLCSR is set. The bit PLOCK from the register PLLCSR is set when PLL is locked.

Both internal 8MHz RC Oscillator, Crystal Oscillator and PLL are switched off in Power-down and Standby sleep modes.01/15

Table 5-7. Start-up Times when the PLL is selected as system clock

CKSEL3..0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)
0011 RC Osc	00	1K CK	14CK
	01	1K CK	14CK + 4ms
	10	1K CK	14CK + 64ms
	11	16K CK	14CK
0101 Ext Osc	00	1K CK	14CK
	01	1K CK	14CK + 4ms
	10	16K CK	14CK + 4ms
	11	16K CK	14CK + 64ms
0001 Ext Clk	00	6 CK(1)	14CK
	01	6 CK(1)	14CK + 4ms
	10	6 CK(1)	14CK + 64ms
	11	Reserved	

Note: 1. This value do not provide a proper restart; do not use PD in this clock scheme.

Figure 5-3. PLL Clocking System

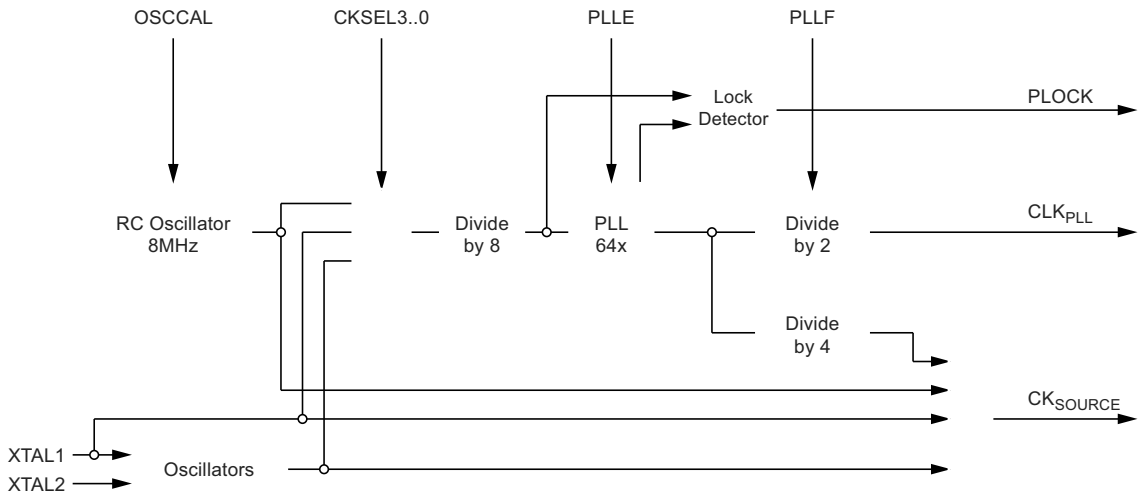


Table 9-4 and Table 9-5 relates the alternate functions of Port B to the overriding signals shown in Figure 9-5 on page 56.

Table 9-4. Overriding Signals for Alternate Functions in PB7..PB4

Signal Name	PB7/ADC4/ PSCOUT0B/SCK/ PCINT7	PB6/ADC7/ PSCOUT1B/ PCINT6	PB5/ADC6/ INT2/ACMPN1/ AMP2-/PCINT5	PB4/AMP0+/ PCINT4
PUOE	SPE × $\overline{\text{MSTR}}$ × $\overline{\text{SPIPS}}$	0	0	0
PUOV	PB7 × $\overline{\text{PUD}}$ × $\overline{\text{SPIPS}}$	0	0	0
DDOE	SPE × $\overline{\text{MSTR}}$ × $\overline{\text{SPIPS}}$ + PSCen01	PSCen11	0	0
DDOV	PSCen01	1	0	0
PVOE	SPE × MSTR × $\overline{\text{SPIPS}}$	PSCen11	0	0
PVOV	PSCout01 × SPIPS + $\overline{\text{PSCout01}}$ × PSCen01 × $\overline{\text{SPIPS}}$ + PSCout01 × PSCen01 × $\overline{\text{SPIPS}}$	PSCOUT11	0	0
DIEOE	ADC4D	ADC7D	ADC6D + In2en	AMP0ND
DIEOV	0	0	In2en	0
DI	SCKin × $\overline{\text{SPIPS}}$ × $\overline{\text{ireset}}$	ICP1B	INT2	
AIO	ADC4	ADC7	ADC6	AMP0+

Table 9-5. Overriding Signals for Alternate Functions in PB3..PB0

Signal Name	PB3/AMP0-/ PCINT3	PB2/ADC5/INT1/ ACMPN0/PCINT2	PB1/MOSI/ PSCOUT2B/ PCINT1	PB0/MISO/ PSCOUT2A/ PCINT0
PUOE	0	0	–	–
PUOV	0	0	–	–
DDOE	0	0	–	–
DDOV	0	0	–	–
PVOE	0	0	–	–
PVOV	0	0	–	–
DIEOE	AMP0ND	ADC5D + In1en	0	0
DIEOV	0	In1en	0	0
DI		INT1	MOSI_IN × $\overline{\text{SPIPS}}$ × $\overline{\text{ireset}}$	MISO_IN × $\overline{\text{SPIPS}}$ × $\overline{\text{ireset}}$
AIO	AMP0-	ADC5	–	–

12. 8-bit Timer/Counter0 with PWM

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and with PWM support. It allows accurate program execution timing (event management) and wave generation. The main features are:

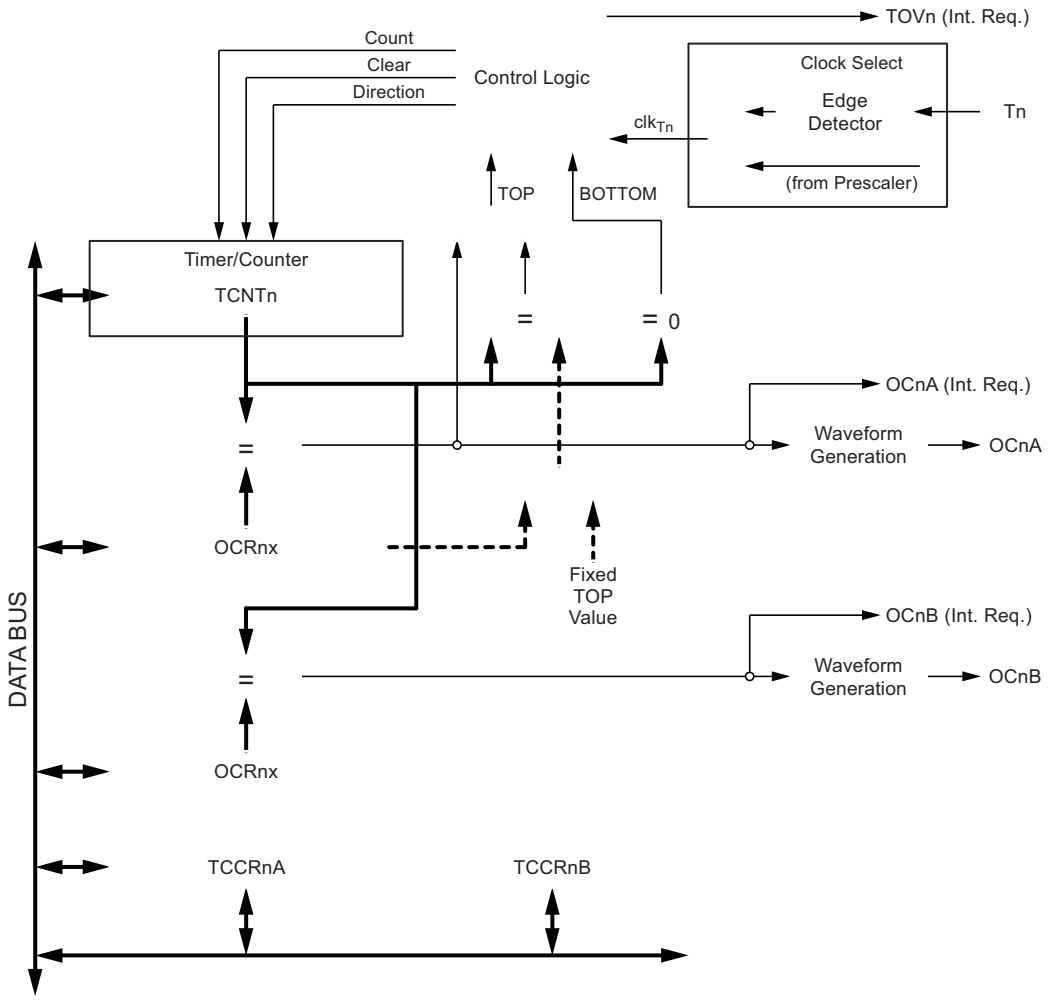
- Two independent output compare units
- Double buffered output compare registers
- Clear timer on compare match (auto reload)
- Glitch free, phase correct pulse width modulator (PWM)
- Variable PWM period
- Frequency generator
- Three independent interrupt sources (TOV0, OCF0A, and OCF0B)

12.1 Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 12-1. For the actual placement of I/O pins, refer to Section 2.3 “Pin Descriptions” on page 9. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in Section 12.8 “8-bit Timer/Counter Register Description” on page 86.

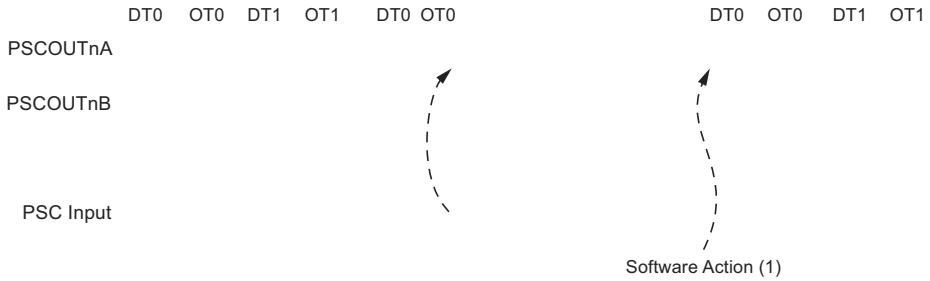
The PRTIM0 bit in Section 6.6 “Power Reduction Register” on page 36 must be written to zero to enable Timer/Counter0 module.

Figure 12-1. 8-bit Timer/Counter Block Diagram



14.11 PSC Input Mode 11xb: Halt PSC and Wait for Software Action

Figure 14-14. PSC Behavior versus PSCn Input A in Fault Mode 11xb



Note: Software action is the setting of the PRUNn bit in PCTLn register.

Used in fault mode 7, PSCn input A or PSCn input B act indifferently on On-Time0/Dead-Time0 or on On-Time1/Dead-Time1.

14.12 Analog Synchronization

Each PSC module generates a signal to synchronize the ADC sample and hold; synchronisation is mandatory for measurements.

This signal can be selected between all falling or rising edge of PSCOUTnA or PSCOUTnB outputs.

In center aligned mode, OCRnRAH/L is not used, so it can be used to specified the synchronization of the ADC. It this case, it's minimum value is 1.

14.13 Interrupt Handling

As each PSC module can be dedicated for one function, each PSC has its own interrupt system (vector ..)

List of interrupt sources:

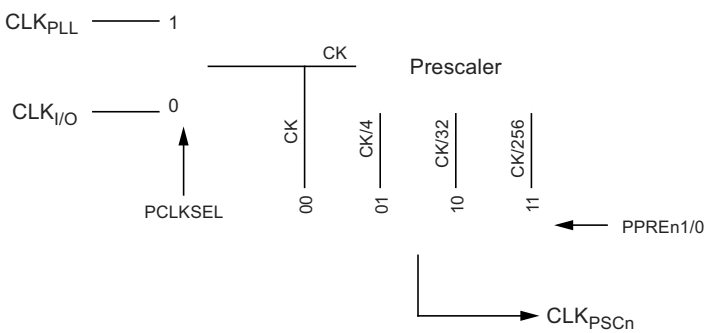
- Counter reload (end of on time 1)
- PSC input event (active edge or at the beginning of level configured event)
- PSC mutual synchronization error

14.14 PSC Clock Sources

Each PSC has two clock inputs:

- CLK PLL from the PLL
- CLK I/O

Figure 14-15. Clock Selection



PCLKSELn bit in PSC control register (PCTL) is used to select the clock source.

PPREn1/0 bits in PSC control register (PCTL) are used to select the divide factor of the clock.

Table 14-9. Synchronization Source Description in Centered Mode

PSYNCn1	PSYNCn0	Description
0	0	Send signal on match with OCRnRA (during counting down of PSC). The min value of OCRnRA must be 1.
0	1	Send signal on match with OCRnRA (during counting up of PSC). The min value of OCRnRA must be 1.
1	0	no synchronization signal
1	1	no synchronization signal

14.16.3 PSC Output Compare SA Register – POCRnSAH and POCRnSAL

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	POCRnSA[11:8]				POCRnSAH
	POCRnSA[7:0]								POCRnSAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.16.4 PSC Output Compare RA Register – POCRnRAH and POCRnRAL

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	POCRnRA[11:8]				POCRnRAH
	POCRnRA[7:0]								POCRnRAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.16.5 PSCOutput Compare SB Register – POCRnSBH and POCRnSBL

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	POCRnSB[11:8]				POCRnSBH
	POCRnSB[7:0]								OCRnSBL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.16.6 PSC Output Compare RB Register – POCR_RBH and POCR_RBL

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	POCRnRB[11:8]				POCR_RBH
	POCRnRB[7:0]								POCR_RBL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: n = 0 to 2 according to module number.

The output compare registers RA, RB, SA and SB contain a 12-bit value that is continuously compared with the PSC counter value. A match can be used to generate an output compare interrupt, or to generate a waveform output on the associated pin.

The output compare registers are 16bit and 12-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers.

15. Serial Peripheral Interface – SPI

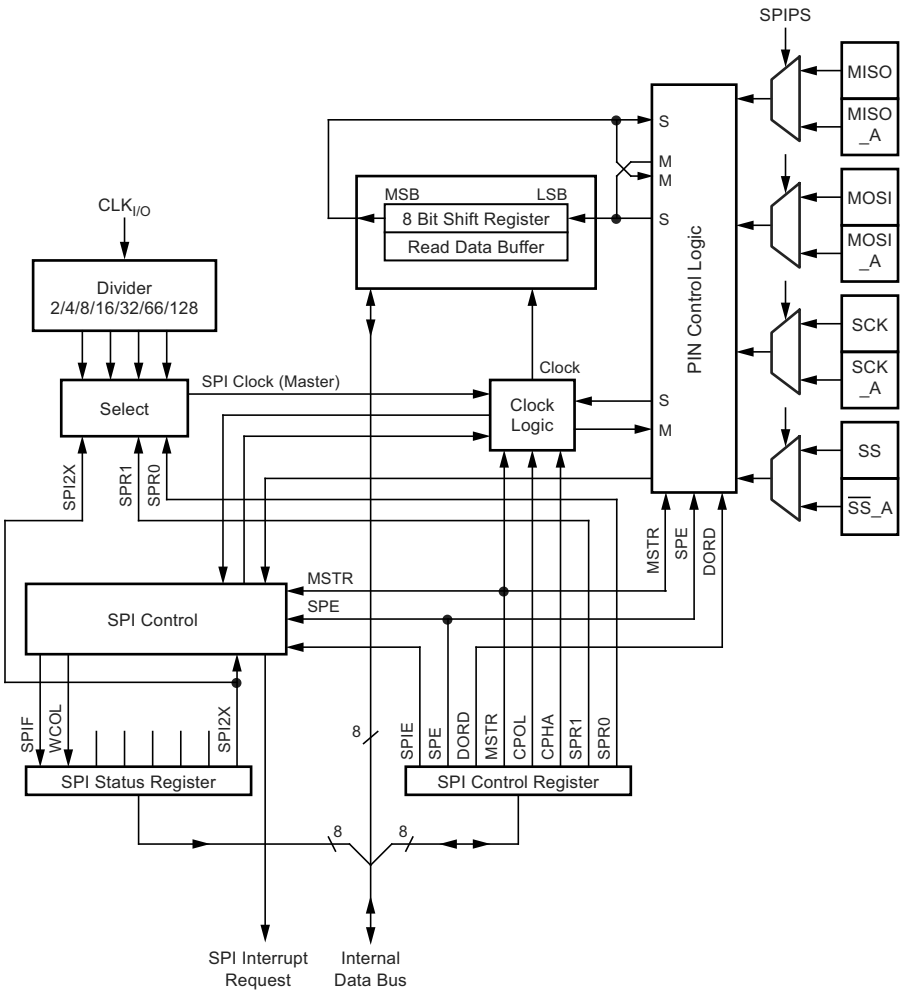
The serial peripheral interface (SPI) allows high-speed synchronous data transfer between the ATmega16/32/64/M1/C1 and peripheral devices or between several AVR devices.

The ATmega16/32/64/M1/C1 SPI includes the following features:

15.1 Features

- Full-duplex, three-wire synchronous data transfer
- Master or slave operation
- LSB first or MSB first data transfer
- Seven programmable bit rates
- End of transmission interrupt flag
- Write collision flag protection
- Wake-up from idle mode
- Double speed (CK/2) master SPI mode

Figure 15-1. SPI Block Diagram(1)



Note: 1. Refer to Figure 1-1 on page 3, and Table 9-3 on page 58 for SPI pin placement.

16.6.4 Stamping Message

The capture of the timer value is done in the MOb which receives or sends the frame. All managed MOb are stamped, the stamping of a received (sent) frame occurs on RxOk (TXOK).

16.7 Error Management

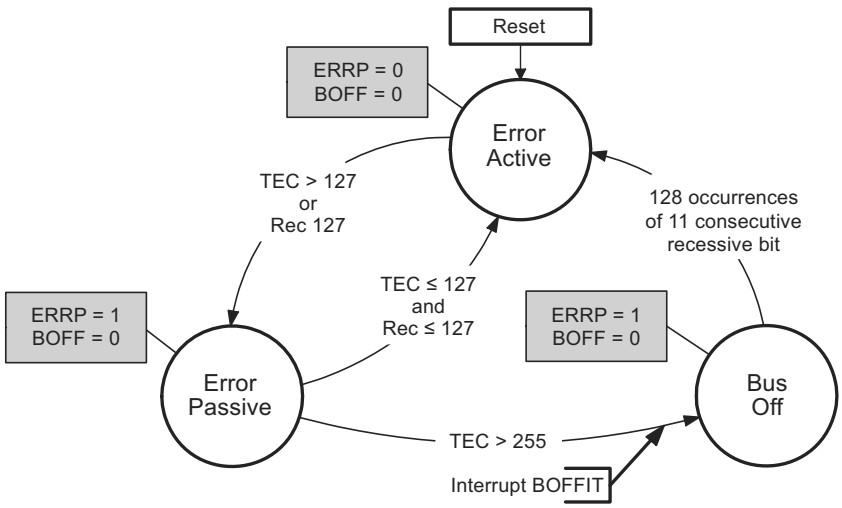
16.7.1 Fault Confinement

The CAN channel may be in one of the three following states:

- Error active (default):
  The CAN channel takes part in bus communication and can send an active error frame when the CAN macro detects an error.
- Error passive:
  The CAN channel cannot send an active error frame. It takes part in bus communication, but when an error is detected, a passive error frame is sent. Also, after a transmission, an error passive unit will wait before initiating further transmission.
- Bus off:
  The CAN channel is not allowed to have any influence on the bus.

For fault confinement, a transmit error counter (TEC) and a receive error counter (REC) are implemented. BOFF and ERRP bits give the information of the state of the CAN channel. Setting BOFF to one may generate an interrupt.

Figure 16-12. Line Error Mode



Note: More than one REC/TEC change may apply during a given message transfer.

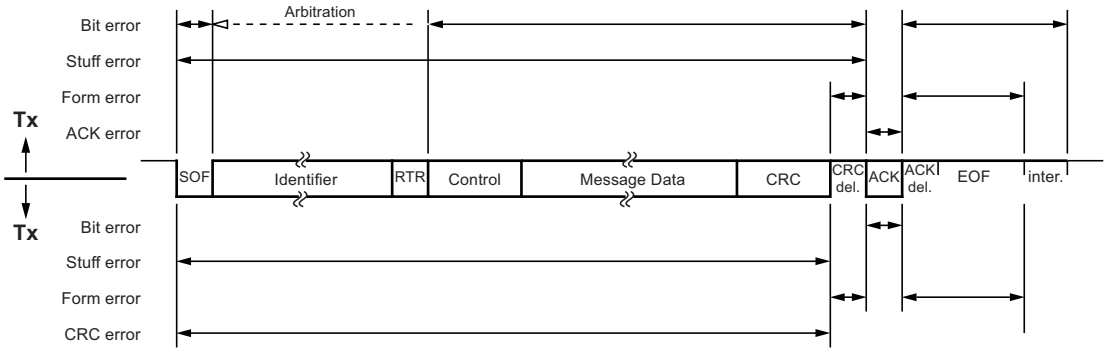
16.7.2 Error Types

- **BERR**: Bit error. The bit value which is monitored is different from the bit value sent.

Note: Exceptions:
- Recessive bit sent monitored as dominant bit during the arbitration field and the acknowledge slot.
- Detecting a dominant bit during the sending of an error frame.

- **SERR**: Stuff error. Detection of more than five consecutive bit with the same polarity.
- **CERR**: CRC error (Rx only). The receiver performs a CRC check on every destuffed received message from the start of frame up to the data field. If this checking does not match with the destuffed CRC field, an CRC error is set.
- **FERR**: Form error. The form error results from one (or more) violations of the fixed form of the following bit fields:
  - CRC delimiter
  - acknowledgement delimiter
  - end-of-frame
  - error delimiter
  - overload delimiter
- **AERR**: Acknowledgment error (Tx only). No detection of the dominant bit in the acknowledge slot.

Figure 16-13. Error Detection Procedures in a Data Frame



16.7.3 Error Setting

The CAN channel can detect some errors on the CAN network.

- In transmission:
  The error is set at MOb level.
- In reception:
  - The identified has matched:
    - The error is set at MOb level.
  - The identified has not or not yet matched:
    - The error is set at general level.

After detecting an error, the CAN channel sends an error frame on network. If the CAN channel detects an error frame on network, it sends its own error frame.

16.11 MOb Registers

The MOb registers has **no** initial (default) value after RESET.

16.11.1 CAN MOb Status Register - CANSTMOB

Bit	7	6	5	4	3	2	1	0	
	DLCW	TXOK	RXOK	BERR	SERR	CERR	FERR	AERR	CANSTMOB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	-	-	-	-	-	-	-	-	

- **Bit 7 – DLCW: Data Length Code Warning**

The incoming message does not have the DLC expected. Whatever the frame type, the DLC field of the CANCDMOB register is updated by the received DLC.

- **Bit 6 – TXOK: Transmit OK**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The communication enabled by transmission is completed. TxOK rises at the end of EOF field. When the controller is ready to send a frame, if two or more message objects are enabled as producers, the lower MOb index (0 to 14) is supplied first.

- **Bit 5 – RXOK: Receive OK**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The communication enabled by reception is completed. RxOK rises at the end of the 6^{th} bit of EOF field. In case of two or more message object reception hits, the lower MOb index (0 to 14) is updated first.

- **Bit 4 – BERR: Bit Error (Only in Transmission)**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The bit value monitored is different from the bit value sent.

Exceptions: the monitored recessive bit sent as a dominant bit during the arbitration field and the acknowledge slot detecting a dominant bit during the sending of an error frame.

- **Bit 3 – SERR: Stuff Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

Detection of more than five consecutive bits with the same polarity. This flag can generate an interrupt.

- **Bit 2 – CERR: CRC Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The receiver performs a CRC check on every de-stuffed received message from the start of frame up to the data field. If this checking does not match with the de-stuffed CRC field, a CRC error is set.

- **Bit 1 – FERR: Form Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The form error results from one or more violations of the fixed form in the following bit fields:

- CRC delimiter.
- Acknowledgment delimiter.
- EOF

17.4.4 LIN/UART Command Overview

Figure 17-5. LIN/UART Command Dependencies

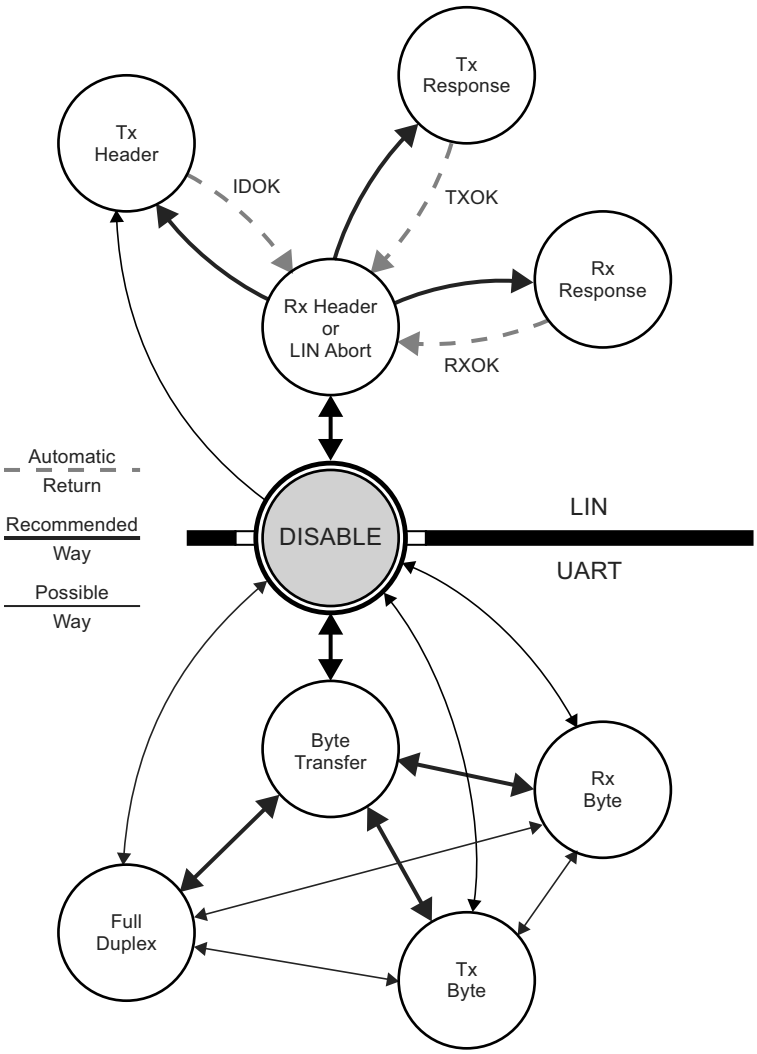


Table 17-1. LIN/UART Command List

LENA	LCMD[2]	LCMD[1]	LCMD[0]	Command	Comment
0	x	x	x	Disable peripheral	
1	0	0	0	Rx Header - LIN abort	LIN withdrawal
			1	Tx Header	LCMD[2..0]=000 after Tx
		1	0	Rx response	LCMD[2..0]=000 after Rx
			1	Tx response	LCMD[2..0]=000 after Tx
	1	0	0	Byte transfer	no CRC, no time out LTXDL=LRXDL=0 (LINDLR: read only register)
		1	0	Rx Byte	
		0	1	Tx Byte	
		1	1	Full duplex	

Table 19-1. Example of Resistor Values (±5%) for a 8-address System (AV_{CC} = 5V(1))

Physical Address	Resistor Value R_{load} (Ohm)	Typical Measured Voltage (V)	Minimum Reading with a 2.56V ref	Typical Reading with a 2.56V ref	Maximum Reading with a 2.56V ref
0	1 000	0.1		40	
1	2 200	0.22		88	
2	3 300	0.33		132	
3	4 700	0.47		188	
4	6 800	0.68		272	
5	10 000	1		400	
6	15 000	1.5		600	
7	22 000	2.2		880	

Table 19-2. Example of Resistor Values (±1%) for a 16-address System (AV_{CC} = 5V1))

Physical Address	Resistor Value R_{load} (Ohm)	Typical Measured Voltage (V)	Minimum Reading with a 2.56V ref	Typical Reading with a 2.56V ref	Miximum Reading with a 2.56V ref
0	1 000	0.1	38	40	45
1	1 200	0.12	46	48	54
2	1500	0.15	57	60	68
3	1800	0.18	69	72	81
4	2200	0.22	84	88	99
5	2700	0.27	104	108	122
6	3300	0.33	127	132	149
7	4700	0.47	181	188	212
8	6 800	0.68	262	272	306
9	8 200	0.82	316	328	369
10	10 000	1.0	386	400	450
11	12 000	1.2	463	480	540
12	15 000	1.5	579	600	675
13	18 000	1.8	694	720	810
14	22 000	2.2	849	880	989
15	27 000	2.7	1023	1023	1023

Note: 1. 5V range: Max R_{load} 30KΩ
3V range: Max R_{load} 15KΩ

19.2.2 Current Source for Low Cost Traducer

An external transducer based on variable resistor can be connected to the current source. This ca be for instance:

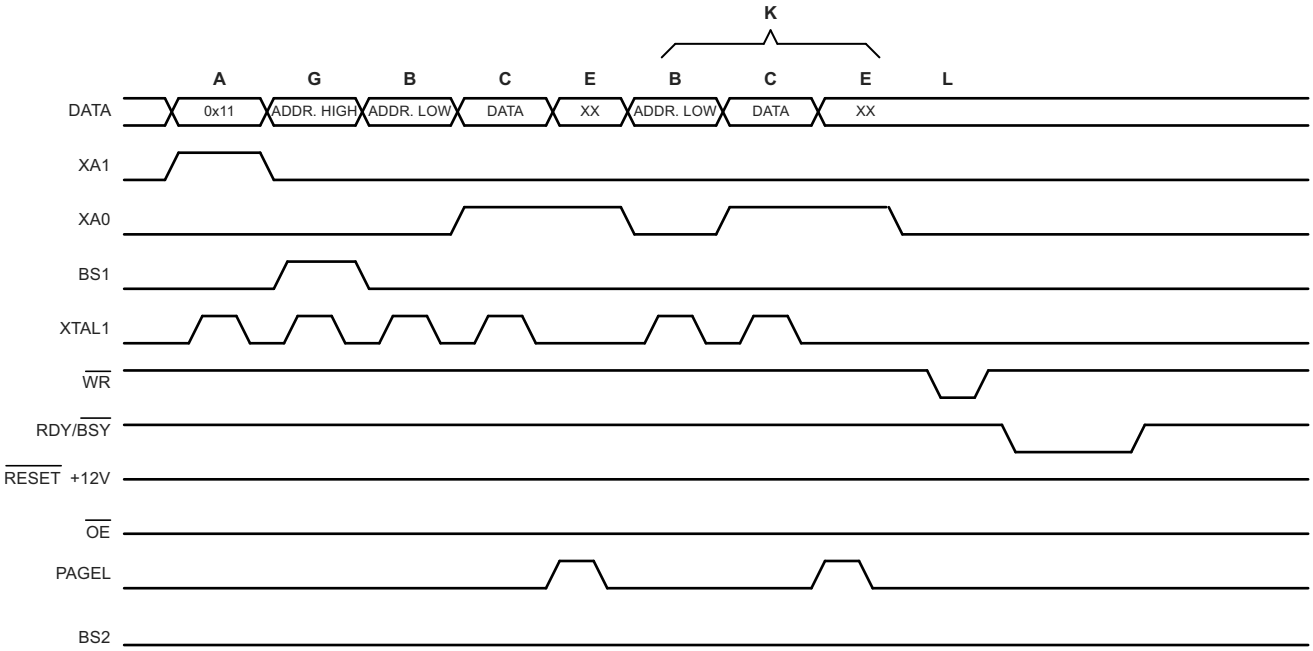
- A thermistor, or temperature-sensitive resistor, used as a temperature sensor
- A CdS photoconductive cell, or luminosity-sensitivity resistor, used as a luminosity sensor.

Using the current source with this type of transducer eliminates the need for additional parts otherwise required in resistor network or Wheatstone bridge.

19.2.3 Voltage Reference for External Devices

An external resistor used in conjunction with the current source can be used as voltage reference for external devices. Using a resistor in serie with a lower tolerance than the current source accuracy (≤ 2%) is recommended. Table 19-2 gives an example of voltage references using standard values of resistors.

Figure 25-4. Programming the EEPROM Waveforms



25.8.6 Reading the Flash

The algorithm for reading the flash memory is as follows (refer to Section 25.8.4 “Programming the Flash” on page 262 for details on command and address loading):

1. A: Load command “0000 0010”.
2. G: Load address High Byte (0x00 - 0xFF).
3. B: Load address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The flash word low byte can now be read at DATA.
5. Set BS1 to “1”. The flash word high byte can now be read at DATA.
6. Set $\overline{OE}$ to “1”.

25.8.7 Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to Section 25.8.4 “Programming the Flash” on page 262 for details on command and address loading):

1. A: Load command “0000 0011”.
2. G: Load address high byte (0x00 - 0xFF).
3. B: Load address low byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM data byte can now be read at DATA.
5. Set $\overline{OE}$ to “1”.

25.8.8 Programming the Fuse Low Bits

The algorithm for programming the fuse low bits is as follows (refer to Section 25.8.4 “Programming the Flash” on page 262 for details on command and data loading):

1. A: Load command “0100 0000”.
2. C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

26.9 ADC Characteristics

Table 26-6. ADC Characteristics in Single Ended Mode - T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Resolution	Single Ended Conversion			10		Bits
Absolute accuracy	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 1MHz	TUE		3.2	5.0	LSB
	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 2MHz	TUE		3.2	5.0	LSB
Integral Non-linearity	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 1MHz	INL		0.7	1.5	LSB
	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 2MHz	INL		0.8	2.0	LSB
Differential Non-linearity	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 1MHz	DNL		0.5	0.8	LSB
	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 2MHz	DNL		0.6	1.4	LSB
Gain error	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 1MHz		–9.0	-5.0	0.0	LSB
	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 2MHz		–9.0	-5.0	0.0	LSB
Offset error	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 1MHz		–2.0	+2.5	+5.0	LSB
	V_{CC} = 5V, V_{REF} = 2.56V ADC clock = 2MHz		–2.0	+2.5	+5.0	LSB
Ref voltage		V_{REF}	2.56		AVCC	V

Figure 27-33. Calibrated 8MHz RC Oscillator Frequency versus V_{CC}

Figure 27-34. Calibrated 8MHz RC Oscillator Frequency versus OSCCAL Value

28. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
Arithmetic and Logic Instructions					
ADD	Rd, Rr	Add two registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with carry two registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add immediate to word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two registers	Rd ← Rd – Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract constant from register	Rd ← Rd – K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with carry two registers	Rd ← Rd – Rr – C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with carry constant from register	Rd ← Rd – K – C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract immediate from word	Rdh:Rdl ← Rdh:Rdl – K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND registers	Rd ← Rd × Rr	Z,N,V	1
ANDI	Rd, K	Logical AND register and constant	Rd ← Rd × K	Z,N,V	1
OR	Rd, Rr	Logical OR registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR register and constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One’s complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two’s complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set bit(s) in register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear bit(s) in register	Rd ← Rd × (0xFF – K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for zero or minus	Rd ← Rd × Rd	Z,N,V	1
CLR	Rd	Clear register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply unsigned	R1:R0 ← Rd × Rr	Z,C	2
MULS	Rd, Rr	Multiply signed	R1:R0 ← Rd × Rr	Z,C	2
MULSU	Rd, Rr	Multiply signed with unsigned	R1:R0 ← Rd × Rr	Z,C	2
FMUL	Rd, Rr	Fractional multiply unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional multiply signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional multiply signed with unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
Branch Instructions					
RJMP	k	Relative jump	PC ← PC + k + 1	None	2
IJMP		Indirect jump to (Z)	PC ← Z	None	2
JMP(*)	k	Direct jump	PC ← k	None	3
RCALL	k	Relative subroutine call	PC ← PC + k + 1	None	3
ICALL		Indirect call to (Z)	PC ← Z	None	3
CALL(*)	k	Direct subroutine call	PC ← k	None	4
RET		Subroutine return	PC ← STACK	None	4
RETI		Interrupt return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, skip if equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare register with immediate	Rd - K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if bit in register cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if bit in register is set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if bit in I/O register cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3

Note: 1. These Instructions are only available in “16K and 32K parts”

29. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	PORTE	–	–	–	–	–	PORTE2	PORTE1	PORTE0	69
0x0D (0x2D)	DDRE	–	–	–	–	–	DDE2	DDE1	DDE0	69
0x0C (0x2C)	PINE	–	–	–	–	–	PINE2	PINE1	PINE0	69
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	69
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	69
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	69
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	68
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	69
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	69
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	68
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	68
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	68
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x00 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega16/32/64/M1/C1 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. These registers are only available on ATmega32/64M1. For other products described in this datasheet, these locations are reserved.