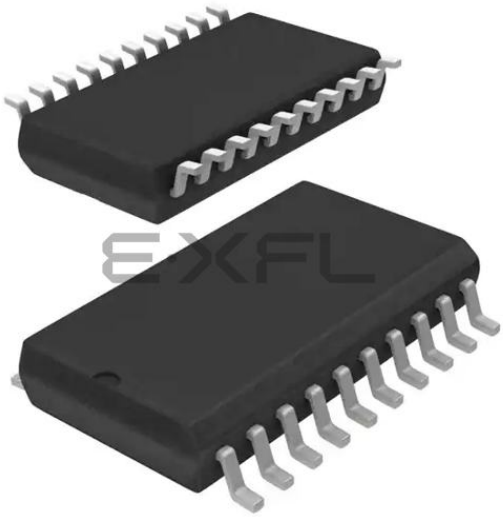


Welcome to E-XFL.COM

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	I²C, LINbus, SPI, UART/USART, USI
Peripherals	Brown-out Detect/Reset, POR, PWM, Temp Sensor, WDT
Number of I/O	16
Program Memory Size	8KB (4K x 16)
Program Memory Type	FLASH
EEPROM Size	512 x 8
RAM Size	512 x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 5.5V
Data Converters	A/D 11x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	20-SOIC (0.295", 7.50mm Width)
Supplier Device Package	20-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/attiny87-sur

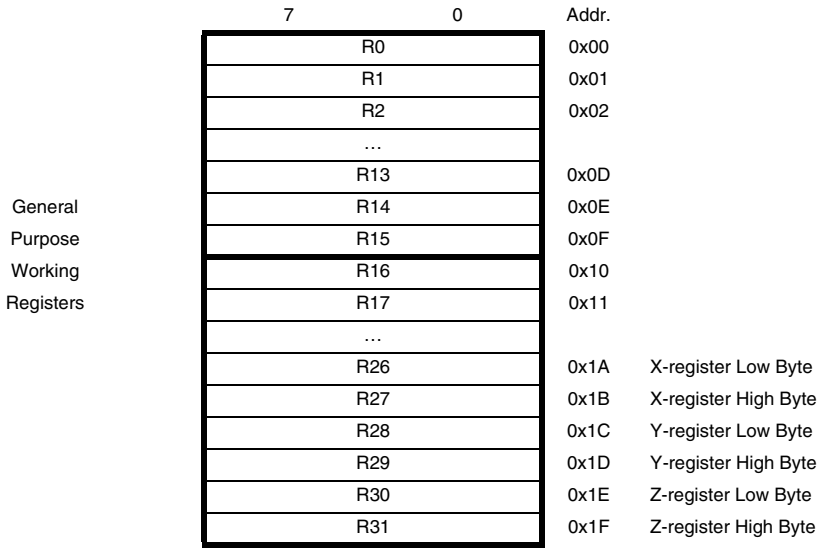
2.4 General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 2-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 2-2. AVR CPU General Purpose Working Registers



Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 2-2, each register is also assigned a Data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

2.4.1 The X-register, Y-register, and Z-register

The registers R26:R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 2-3 on page 11.

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
EEPROM_read:
  ; Wait for completion of previous write
  sbic   EECR,EEPE
  rjmp   EEPROM_read
  ; Set up address (r18:r17) in address register
  out    EEARH, r18
  out    EEARL, r17
  ; Start eeprom read by writing EERE
  sbi    EECR,EERE
  ; Read data from data register
  in     r16,EEDR
  ret
```

C Code Example

```
unsigned char EEPROM_read(unsigned char ucAddress)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEPE))
    ;
  /* Set up address register */
  EEAR = ucAddress;
  /* Start eeprom read by writing EERE */
  EECR |= (1<<EERE);
  /* Return data from data register */
  return EEDR;
}
```

3.3.6 Preventing EEPROM Corruption

During periods of low V_{CC}, the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low V_{CC} reset protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

Table 4-9. Start-up Times for the External Clock Selection

SUT[1:0][1] CSUT[1:0][2]	Start-up Time from Power-down/save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	14CK (+ 4.1 ms[3])	BOD enabled
01	6 CK	14CK + 4.1 ms	Fast rising power
10	6 CK	14CK + 65 ms	Slowly rising power
11	Reserved		

Notes: 1. Flash Fuse bits.
2. CLKSELR register bits.
3. Additional delay (+ 4ms) available if RSTDISBL fuse is set.

Note that the System Clock Prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation. Refer to "System Clock Prescaler" on page 37 for details.

4.2.7 Clock Output Buffer

If not using a crystal oscillator, the device can output the system clock on the CLKO pin. To enable the output, the CKOUT Fuse or COUT bit of CLKSELR register has to be programmed. This option is useful when the device clock is needed to drive other circuits on the system. Note that the clock will not be output during reset and the normal operation of I/O pin will be overridden when the fuses are programmed. If the System Clock Prescaler is used, it is the divided system clock that is output.

4.3 Dynamic Clock Switch

4.3.1 Features

The ATtiny87/167 provides a powerful dynamic clock switch circuit that allows users to turn on and off clocks of the device on the fly. The built-in de-glitching circuitry allows clocks to be enabled or disabled asynchronously. This enables efficient power management schemes to be implemented easily and quickly. In a safety application, the dynamic clock switch circuit allows continuous monitoring of the external clock permitting a fallback scheme in case of clock failure.

The control of the dynamic clock switch circuit must be supervised by software. This operation is facilitated by the following features:

- **Safe commands**, to avoid unintentional commands, a special write procedure must be followed to change the CLKCSR register bits (See "CLKPR – Clock Prescaler Register" on page 38.):
- **Exclusive action**, the actions are controlled by a decoding table (commands) written to the CLKCSR register. This ensures that only one command operation can be launched at any time. The main actions of the decoding table are:
  - '*Disable Clock Source*',
  - '*Enable Clock Source*',
  - '*Request Clock Availability*',
  - '*Clock Source Switching*',
  - '*Recover System Clock Source*',

The '*Disable Clock Source*' command disables the clock source indicated by the settings of CLKSELR register (only CSEL[3:0]). If the clock source indicated is currently the one that is used to drive the system clock, the command is not executed.

Because the selected configuration is latched at clock source level, it is possible to enable many clock sources at a given time (ex: the internal RC oscillator for system clock + an oscillator with external crystal). The user (code) is responsible of this management.

4.3.4 COUT Command

The '*CKOUT*' command allows to drive the CLKO pin. Refer to Section 4.2.7 "Clock Output Buffer" on page 31 for using.

4.3.5 Clock Availability

'Request for Clock Availability' command enables a hardware oscillation cycle counter driven by the selected source clock, CSEL[3:0]. The count limit value is determined by the settings of CSUT[1:0]. The clock is declared ready (CLKRDY = 1) when the count limit value is reached. The CLKRDY flag is reset when the count starts. Once set, this flag remains unchanged until a new count is commanded. To perform this checking, the CKSEL and CSUT fields should not be changed while the operation is running.

Note that once the new clock source is selected ('*Enable Clock Source*' command), the count procedure is automatically started. The user (code) should wait for the setting of the CLKRDY flag in CLKSCR register before using a newly selected clock.

At any time, the user (code) can ask for the availability of a clock source. The user (code) can request it by writing the *'Request for Clock Availability'* command in the CLKSCR register. A full polling of the status of clock sources can thus be done.

4.3.6 System Clock Source Recovering

The '*Recover System Clock Source*' command returns the current clock source used to drive the system clock as per Table 4-1 on page 25. The CKSEL field of CLKSELR register is then updated with this returned value. There is no information on the SUT used or status on CKOUT.

4.3.7 Clock Switching

To drive the system clock, the user can switch from the current clock source to any other of the following ones (one of them being the current clock source):

1. Calibrated internal RC oscillator 8.0 MHz,
2. Internal watchdog oscillator 128 kHz,
3. External clock,
4. External low-frequency oscillator,
5. External Crystal/Ceramic Resonator.

The clock switching is performed by a sequence of commands. First, the user (code) must make sure that the new clock source is operating. Then the '*Clock Source Switching*' command can be issued. Once this command has been successfully completed using the '*Recover System Clock Source*' command, the user (code) may stop the previous clock source.

It is strongly recommended to run this sequence only once the interrupts have been disabled. The user (code) is responsible for the correct implementation of the clock switching sequence.

The following code example shows one assembly and one C function for changing the time-out value of the Watchdog Timer.

Assembly Code Example[1]

```
WDT_Prescaler_Change:
  ; Turn off global interrupt
  cli
  ; Reset Watchdog Timer
  wdr
  ; Start timed sequence
  lds r16, WDTCR
  ori   r16, (1<<WDCE) | (1<<WDE)
  sts WDTCR, r16
  ; -- Got four cycles to set the new values from here -
  ; Set new prescaler(time-out) value = 64K cycles (~0.5 s)
  ldi   r16, (1<<WDE) | (1<<WDP2) | (1<<WDP0)
  sts WDTCR, r16
  ; -- Finished setting new values, used 2 cycles -
  ; Turn on global interrupt
  sei
  ret
```

C Code Example[1]

```
void WDT_Prescaler_Change(void)
{
  __disable_interrupt();
  __watchdog_reset();
  /* Start timed sequence */
  WDTCR |= (1<<WDCE) | (1<<WDE);
  /* Set new prescaler(time-out) value = 64K cycles (~0.5 s) */
  WDTCR  = (1<<WDE) | (1<<WDP2) | (1<<WDP0);
  __enable_interrupt();
}
```

Notes: 1. See "About Code Examples" on page 6.

2. The Watchdog Timer should be reset before any change of the WDP bits, since a change in the WDP bits can result in a time-out when switching to a shorter time-out period.

6.3.2 Clock monitoring

The Watchdog Timer can be used to detect a loss of system clock. This configuration is driven by the dynamic clock switch circuit. Please refer to Section 4.3.8 "Clock Monitoring" on page 34 for more information.

9.3.3 Alternate Functions of Port A

The Port A pins with alternate functions are shown in Table 9-3.

Table 9-3. Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	PCINT7 (Pin Change Interrupt 7) ADC7 (ADC Input Channel 7) AIN1 (Analog Comparator Positive Input) XREF (Internal Voltage Reference Output) AREF (External Voltage Reference Input)
PA6	PCINT6 (Pin Change Interrupt 6) ADC6 (ADC Input Channel 6) AIN0 (Analog Comparator Negative Input) $\overline{SS}$ (SPI Slave Select Input)
PA5	PCINT5 (Pin Change Interrupt 5) ADC5 (ADC Input Channel 5) T1 (Timer/Counter1 Clock Input) USCK (Three-wire Mode USI *Alternate* Clock Input) SCL (Two-wire Mode USI *Alternate* Clock Input) SCK (SPI Master Clock)
PA4	PCINT4 (Pin Change Interrupt 4) ADC4 (ADC Input Channel 4) ICP1 (Timer/Counter1 Input Capture Trigger) DI (Three-wire Mode USI *Alternate* Data Input) SDA (Two-wire Mode USI *Alternate* Data Input / Output) MOSI (SPI Master Output / Slave Input)
PA3	PCINT3 (Pin Change Interrupt 3) ADC3 (ADC Input Channel 3) ISRC (Current Source Pin) INT1 (External Interrupt1 Input)
PA2	PCINT2 (Pin Change Interrupt 2) ADC2 (ADC Input Channel 2) OC0A (Output Compare and PWM Output A for Timer/Counter0) DO (Three-wire Mode USI *Alternate* Data Output) MISO (SPI Master Input / Slave Output)
PA1	PCINT1 (Pin Change Interrupt 1) ADC1 (ADC Input Channel 1) TXD (UART Transmit Pin) TXLIN (LIN Transmit Pin)
PA0	PCINT0 (Pin Change Interrupt 0) ADC0 (ADC Input Channel 0) RXD (UART Receive Pin) RXLIN (LIN Receive Pin)

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Table 10-5.

Table 10-5. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_T0_S (No prescaling)
0	1	0	clk_T0_S/8 (From prescaler)
0	1	1	clk_T0_S/32 (From prescaler)
1	0	0	clk_T0_S/64 (From prescaler)
1	0	1	clk_T0_S/128 (From prescaler)
1	1	0	clk_T0_S/256 (From prescaler)
1	1	1	clk_T0_S/1024 (From prescaler)

10.11.3 TCNT0 – Timer/Counter0 Register

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	TCNT07	TCNT06	TCNT05	TCNT04	TCNT03	TCNT02	TCNT01	TCNT00	TCNT0
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	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the Compare Match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a Compare Match between TCNT0 and the OCR0x Register.

10.11.4 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0A7	OCR0A6	OCR0A5	OCR0A4	OCR0A3	OCR0A2	OCR0A1	OCR0A0	OCR0A
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	

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

10.11.5 ASSR – Asynchronous Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB6)	–	EXCLK	AS0	TCN0UB	OCR0AUB	–	TCR0AUB	TCR0BUB	ASSR
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: Reserved Bit**

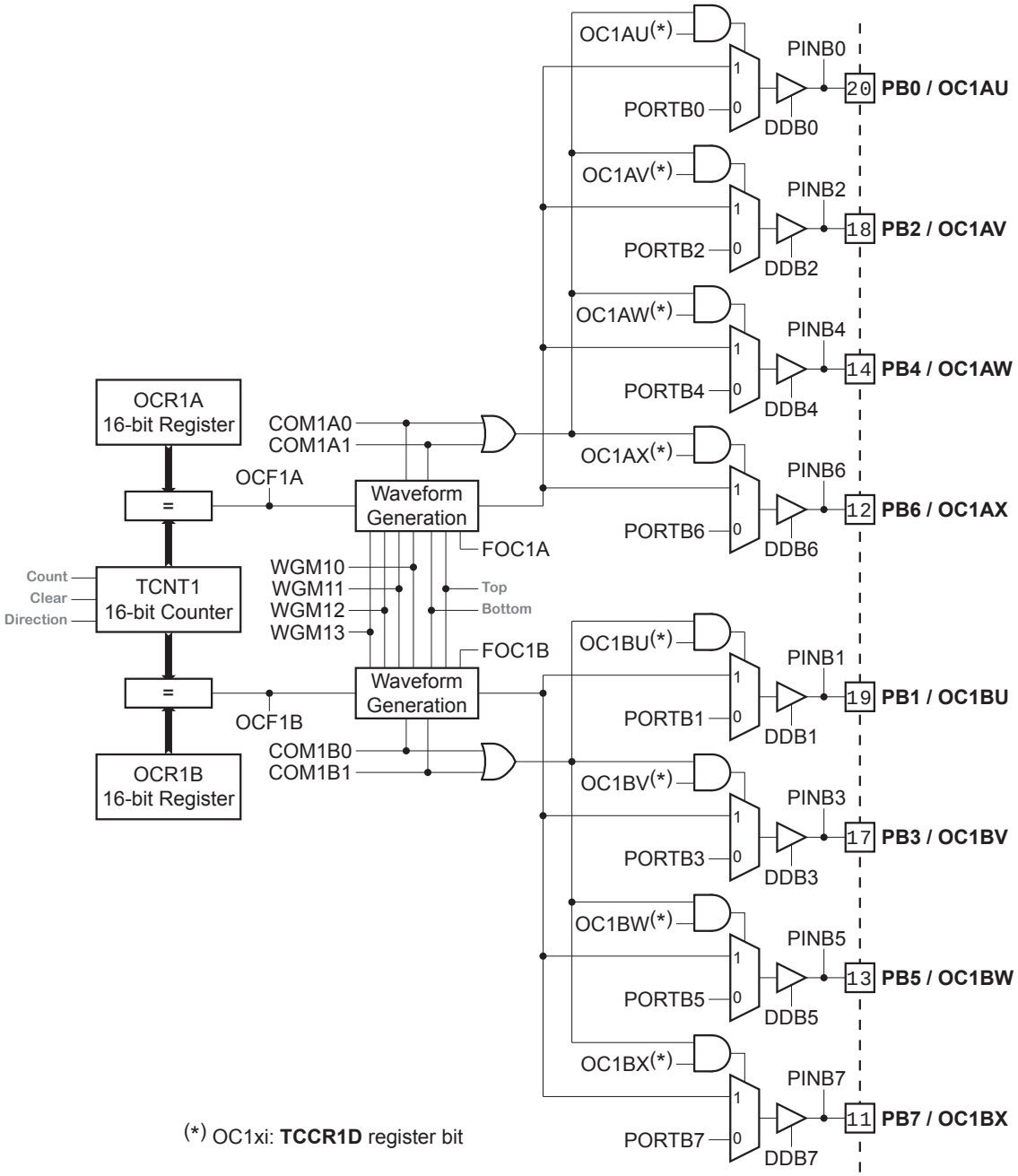
This bit is reserved in the ATtiny87/167 and will always read as zero.

- **Bit 6 – EXCLK: Enable External Clock Input**

When EXCLK is written to one, and asynchronous clock is selected, the external clock input buffer is enabled and an external clock can be input on XTAL1 pin instead of an external crystal. Writing to EXCLK should be done before asynchronous operation is selected. Note that the crystal oscillator will only run when this bit is zero.

of the general I/O port control registers (DDR and PORT) that are affected by the COM1A/B[1:0] and OCnxi bits are shown. When referring to the OC1A/B state, the reference is for the internal OC1A/B Register, not the OC1A/Bi pin. If a system reset occur, the OC1A/B Register is reset to “0”.

Figure 12-5. Compare Match Output



(*) OC1xi: **TCCR1D** register bit

or toggle at a compare match (See ”Compare Match Output Unit” on page 119.). The OCnxi bits over control the setting of the COM1A/B[1:0] bits as shown in Figure 12-6 on page 121.

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 129.

12.9.1 Normal Mode

The simplest mode of operation is the Normal mode (WGM1[3:0] = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the Timer/Counter Overflow Flag (TOV1) will be set in the same timer clock cycle as the TCNT1 becomes zero. The TOV1 flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV1 flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Input Capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

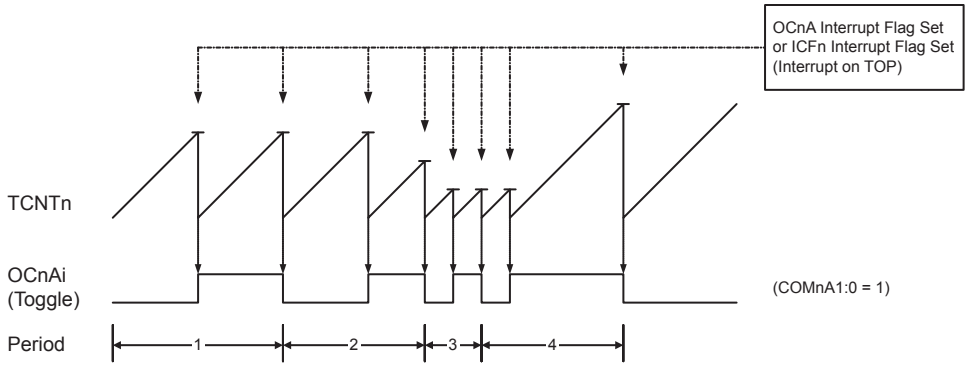
The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

12.9.2 Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM1[3:0] = 4 or 12), the OCR1A or ICR1 Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT1) matches either the OCR1A (WGM1[3:0] = 4) or the ICR1 (WGM1[3:0] = 12). The OCR1A or ICR1 define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

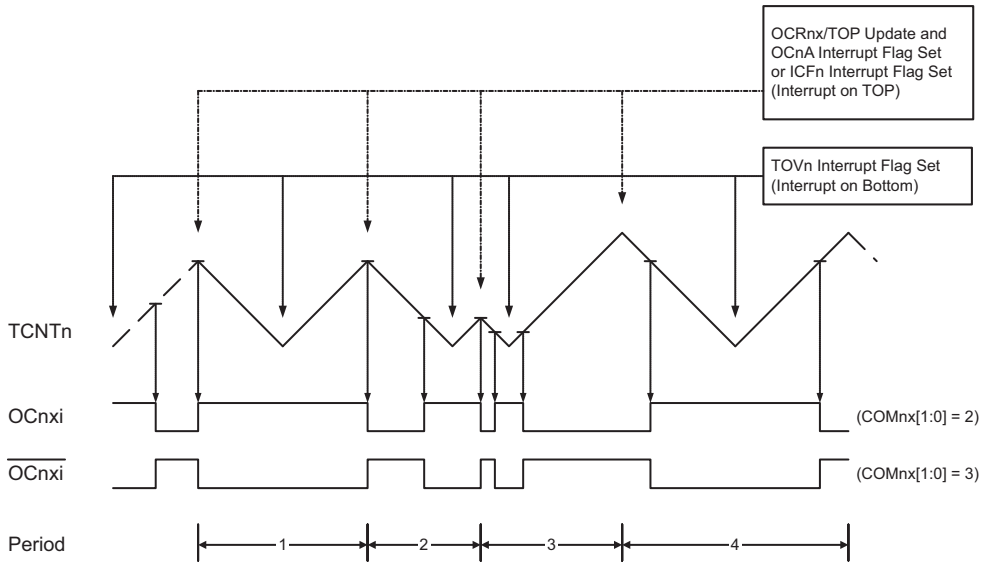
The timing diagram for the CTC mode is shown in Figure 12-7. The counter value (TCNT1) increases until a compare match occurs with either OCR1A or ICR1, and then counter (TCNT1) is cleared.

Figure 12-7. CTC Mode, Timing Diagram



In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM1[3:0] = 1, 2, or 3), the value in ICR1 (WGM1[3:0] = 10), or the value in OCR1A (WGM1[3:0] = 11). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 12-9. The figure shows phase correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1A/B and TCNT1. The OC1A/B interrupt flag will be set when a compare match occurs.

Figure 12-9. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches BOTTOM. When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 flag is set accordingly at the same timer clock cycle as the OCR1A/B Registers are updated with the double buffer value (at TOP). The interrupt flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNT1 and the OCR1A/B. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCR1A/B Registers are written. As the third period shown in Figure 12-9 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCR1A/B Register. Since the OCR1A/B update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

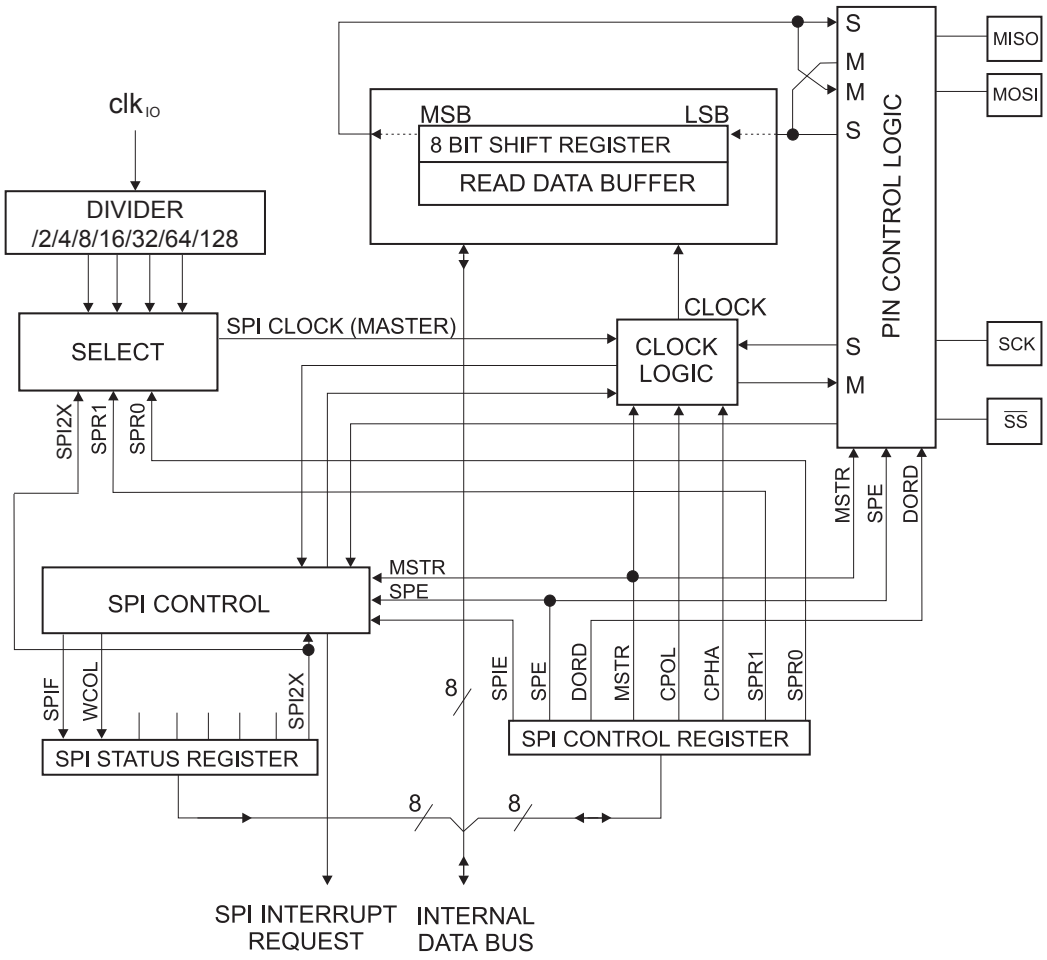
13. SPI - Serial Peripheral Interface

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATtiny87/167 and peripheral devices or between several AVR devices. The ATtiny87/167 SPI includes the following features:

13.1 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 13-1. SPI Block Diagram[1]



Note: 1. Refer to Figure 1.4 on page 4, and Table 9-3 on page 76 for SPI pin placement.

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

Assembly Code Example[1]

```
SPI_SlaveInit:
  ; Set MISO output, all others input
  ldi     r17,(1<<DD_MISO)
  out     DDR_SPI,r17
  ; Enable SPI
  ldi     r17,(1<<SPE)
  out     SPCR,r17
  ret

SPI_SlaveReceive:
  ; Wait for reception complete
  sbis    SPSR,SPIF
  rjmp    SPI_SlaveReceive
  ; Read received data and return
  in      r16,SPDR
  ret
```

C Code Example[1]

```
void SPI_SlaveInit(void)
{
  /* Set MISO output, all others input */
  DDR_SPI = (1<<DD_MISO);
  /* Enable SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* Wait for reception complete */
  while(!(SPSR & (1<<SPIF)));
  /* Return data register */
  return SPDR;
}
```

Note: 1. The example code assumes that the part specific header file is included.

14.3.3 SPI Slave Operation Example

The following code demonstrates how to use the USI module as a SPI Slave:

```
init:
    ldi     r16,(1<<USIWM0)|(1<<USICS1)
    sts     USICR,r16
...
SlaveSPITransfer:
    sts     USIDR,r16
    ldi     r16,(1<<USIOIF)
    sts     USISR,r16
SlaveSPITransfer_loop:
    lds     r16, USISR
    sbrs    r16, USIOIF
    rjmp    SlaveSPITransfer_loop
    lds     r16,USIDR
    ret
```

The code is size optimized using only eight instructions (+ ret). The code example assumes that the DO is configured as output and USCK pin is configured as input in the DDR Register. The value stored in register r16 prior to the function is called is transferred to the master device, and when the transfer is completed the data received from the Master is stored back into the r16 Register.

Note that the first two instructions is for initialization only and needs only to be executed once.These instructions sets Three-wire mode and positive edge USI Data Register clock. The loop is repeated until the USI Counter Overflow Flag is set.

14.3.4 Two-wire Mode

The USI Two-wire mode is compliant to the Inter IC (TWI) bus protocol, but without slew rate limiting on outputs and input noise filtering. Pin names used by this mode are SCL and SDA.

The LIN protocol says that a message with an identifier from 60 (0x3C) up to 63 (0x3F) uses a classic checksum (sum over the data bytes only). Software will be responsible for switching correctly the LIN13 bit to provide/check this expected checksum (the insertion of the ID field in the computation of the CRC is set - or not - just after entering the Rx or Tx Response command).

15.5.15 Data Management

15.5.15.1 LIN FIFO Data Buffer

To preserve register allocation, the LIN data buffer is seen as a FIFO (with address pointer accessible). This FIFO is accessed via the LINDX[2:0] field of LINSEL register through the LINDAT register.

LINDX[2:0], the data index, is the address pointer to the required data byte. The data byte can be read or written. The data index is automatically incremented after each LINDAT access if the $\overline{\text{LAINC}}$ (active low) bit is cleared. A roll-over is implemented, after data index=7 it is data index=0. Otherwise, if $\overline{\text{LAINC}}$ bit is set, the data index needs to be written (updated) before each LINDAT access.

The first byte of a LIN frame is stored at the data index=0, the second one at the data index=1, and so on. Nevertheless, LINSEL must be initialized by the user before use.

15.5.15.2 UART Data Register

The LINDAT register is the data register (no buffering - no FIFO). In write access, LINDAT will be for data out and in read access, LINDAT will be for data in.

In UART mode the LINSEL register is unused.

15.5.16 OCD Support

When a debugger break occurs, the state machine of the LIN/UART controller is stopped (included frame time-out) and further communication may be corrupted.

- Pull-up resistors on the dW/(RESET) line must not be smaller than 10kΩ. The pull-up resistor is not required for debugWIRE functionality.
- Connecting the RESET pin directly to V_{CC} will not work.
- Capacitors connected to the RESET pin must be disconnected when using debugWire.
- All external reset sources must be disconnected.

19.4 Software Break Points

DebugWIRE supports Program memory break points by the AVR BREAK instruction. Setting a break point in AVR Studio® will insert a BREAK instruction in the Program memory. The instruction replaced by the BREAK instruction will be stored. When program execution is continued, the stored instruction will be executed before continuing from the Program memory. A break can be inserted manually by putting the BREAK instruction in the program.

The Flash must be re-programmed each time a break point is changed. This is automatically handled by AVR Studio through the debugWIRE interface. The use of break points will therefore reduce the Flash Data retention. Devices used for debugging purposes should not be shipped to end customers.

19.5 Limitations of DebugWIRE

The debugWIRE communication pin (dW) is physically located on the same pin as External Reset (RESET). An External Reset source is therefore not supported when the debugWIRE is enabled.

The debugWIRE system accurately emulates all I/O functions when running at full speed, i.e., when the program in the CPU is running. When the CPU is stopped, care must be taken while accessing some of the I/O Registers via the debugger (AVR Studio).

A programmed DWEN Fuse enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption while in sleep. Thus, the DWEN Fuse should be disabled when debugWire is not used.

19.6 DebugWIRE Related Register in I/O Memory

The following section describes the registers used with the debugWire.

19.6.1 DWDR – DebugWIRE Data Register

Bit	7	6	5	4	3	2	1	0	
0x31 (0x51)	DWDR[7:0]								DWDR
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	

The DWDR Register provides a communication channel from the running program in the MCU to the debugger. This register is only accessible by the debugWIRE and can therefore not be used as a general purpose register in the normal operations.

22. Electrical Characteristics

22.1 Absolute Maximum Ratings*

Operating Temperature.....– 40°C to +85°C Storage Temperature.....– 65°C to +150°C Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground .....– 0.5V to V_{CC}+0.5V Voltage on $\overline{\text{RESET}}$ with respect to Ground....– 0.5V to +13.0V Voltage on V_{CC} with respect to Ground.....– 0.5V to 6.0V DC Current per I/O Pin ..... 40.0 mA DC Current V_{CC} and GND Pins ..... 200.0 mA Injection Current at VCC = 0V to 5V(2)..... ±5.0mA(1)	*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Notes: 1. Maximum current per port = ±30mA
2. Functional corruption may occur .

22.2 DC Characteristics

T_A = -40°C to +85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.(1)	Max.	Units
V_{IL}	Input Low Voltage	Except XTAL1 and $\overline{\text{RESET}}$ pins	- 0.5		0.2 V_{CC} (2)	V
V_{IL1}		XTAL1 pin - External Clock Selected	- 0.5		0.1 V_{CC} (2)	V
V_{IL2}		$\overline{\text{RESET}}$ pin	- 0.5		0.2 V_{CC} (2)	V
V_{IL3}		$\overline{\text{RESET}}$ pin as I/O	- 0.5		0.2 V_{CC} (2)	V
V_{IH}	Input High Voltage	Except XTAL1 and $\overline{\text{RESET}}$ pins	0.7 V_{CC} (3)		V_{CC} + 0.5	V
V_{IH1}		XTAL1 pin - External Clock Selected	0.8 V_{CC} (3)		V_{CC} + 0.5	V
V_{IH2}		$\overline{\text{RESET}}$ pin	0.9 V_{CC} (3)		V_{CC} + 0.5	V
V_{IH3}		$\overline{\text{RESET}}$ pin as I/O	0.7 V_{CC} (3)		V_{CC} + 0.5	V
V_{OL}	Output Low Voltage (4) (Ports A, B,)	I_{OL} = 10 mA, V_{CC} = 5V I_{OL} = 5 mA, V_{CC} = 3V			0.6 0.5	V
V_{OH}	Output High Voltage (5) (Ports A, B)	I_{OH} = – 10 mA, V_{CC} = 5V I_{OH} = – 5 mA, V_{CC} = 3V	4.3 2.5			V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)		< 0.05	1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)		< 0.05	1	µA

Table 22-10. ADC Characteristics, Differential Channels (-40°C/+85°C)

Symbol	Parameter	Condition	Min	Typ	Max	Units
	Resolution	Differential conversion		8		
TUE	Absolute accuracy	Gain = 8x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		1.0	3.0	LSB
		Gain = 20x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		1.5	3.5	
		Gain = 8x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		2.0	4.5	
		Gain = 20x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		2.0	6.0	
INL	Integral Non Linearity	Gain = 8x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		0.2	1.0	LSB
		Gain = 20x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		0.4	1.5	
		Gain = 8x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		0.5	2.0	
		Gain = 20x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		1.6	5.0	
DNL	Differential Non Linearity	Gain = 8x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		0.3	0.8	LSB
		Gain = 20x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		0.3	0.8	
		Gain = 8x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		0.4	0.8	
		Gain = 20x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz		0.6	1.6	
	Gain error	Gain = 8x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz	-3.0	1.0	3.0	LSB
		Gain = 20x, BIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz	-4.0	1.5	4.0	
		Gain = 8x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz	-5.0	-2.5	0.0	
		Gain = 20x, UNIPOLAR V_{REF} = 4V, V_{CC} = 5V ADC clock = 200 kHz	-4.0	-0.5	4.0	

23.3 Supply Current of I/O modules

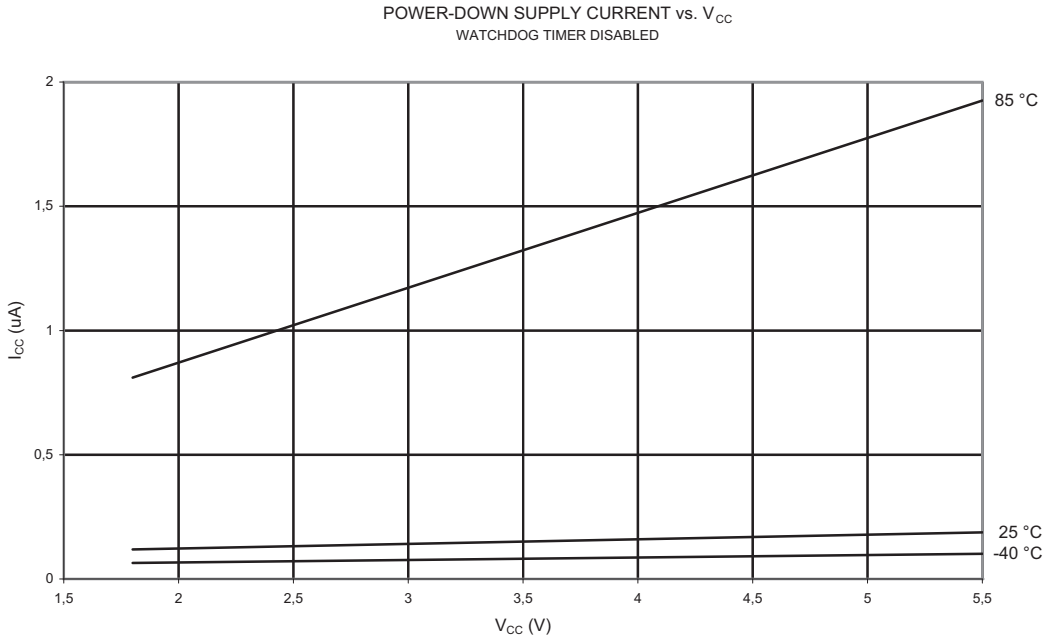
The table below can be used to calculate the additional current consumption for the different I/O modules Idle mode. The enabling or disabling of the I/O modules are controlled by the Power Reduction Register. See Section 5.9.3 “PRR – Power Reduction Register” on page 47 for details.

Table 23-1. Additional Current Consumption for the different I/O modules (absolute values)

Module	V_{CC} = 5.0 V Freq. = 16 MHz	V_{CC} = 5.0 V Freq. = 8 MHz	V_{CC} = 3.0 V Freq. = 8 MHz	V_{CC} = 3.0 V Freq. = 4 MHz	Units
LIN/UART	0.77	0.37	0.20	0.10	mA
SPI	0.31	0.14	0.08	0.04	mA
TIMER-1	0.28	0.13	0.08	0.04	mA
TIMER-0	0.41	0.20	0.10	0.05	mA
USI	0.14	0.05	0.04	0.02	mA
ADC	0.48	0.22	0.10	0.05	mA

23.4 Current Consumption in Power-down Mode

Figure 23-8. Power-down Supply Current vs. V_{CC} (Watchdog Timer Disabled)



25. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clock
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 Reg.	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
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
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1

Mnemonics	Operands	Description	Operation	Flags	#Clock
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
		DATA TRANSFER INSTRUCTIONS			
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
		MCU CONTROL INSTRUCTIONS			
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A