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	Obsolete
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	32
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)	4.5V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	44-TQFP
Supplier Device Package	44-TQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega16-16ai

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry is useful in BCD arithmetic. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S: Sign Bit, S = N ⊕ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two’s Complement Overflow Flag V. See the “Instruction Set Description” for detailed information.

- **Bit 3 – V: Two’s Complement Overflow Flag**

The Two’s Complement Overflow Flag V supports two’s complement arithmetics. See the “Instruction Set Description” for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

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) */
```

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.

SRAM Data Memory

Figure 9 shows how the ATmega16 SRAM Memory is organized.

The lower 1120 Data Memory locations address the Register File, the I/O Memory, and the internal data SRAM. The first 96 locations address the Register File and I/O Memory, and the next 1024 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

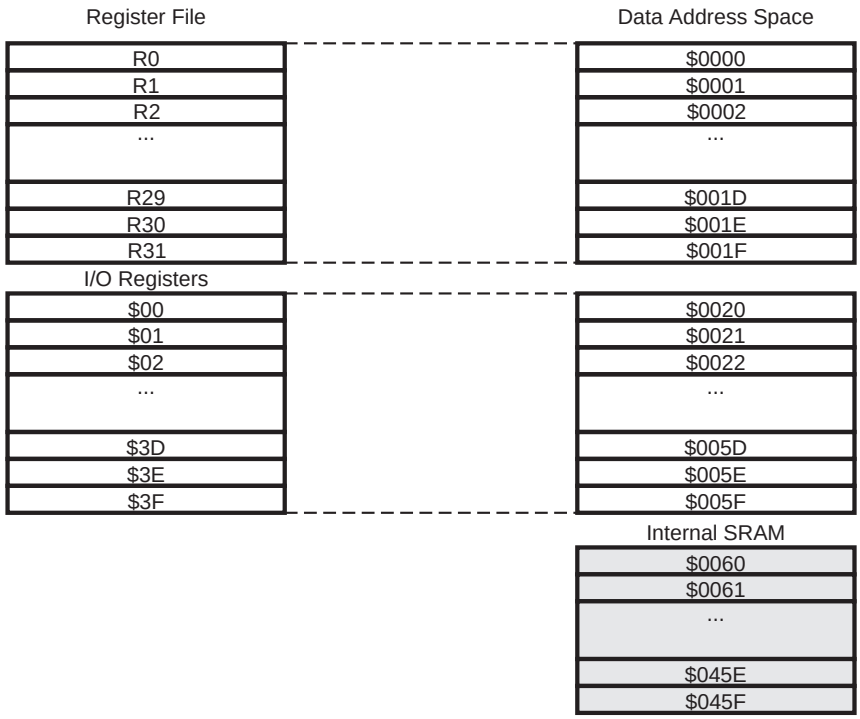
The direct addressing reaches the entire data space.

The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y-register or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O Registers, and the 1024 bytes of internal data SRAM in the ATmega16 are all accessible through all these addressing modes. The Register File is described in “General Purpose Register File” on page 11.

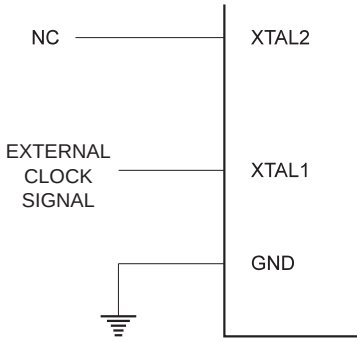
Figure 9. Data Memory Map



External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 14. To run the device on an external clock, the CKSEL Fuses must be programmed to “0000”. By programming the CKOPT Fuse, the user can enable an internal 36 pF capacitor between XTAL1 and GND.

Figure 14. External Clock Drive Configuration



When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 12.

Table 12. Start-up Times for the External Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved		

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. It is required to ensure that the MCU is kept in reset during such changes in the clock frequency.

Timer/Counter Oscillator

For AVR microcontrollers with Timer/Counter Oscillator pins (TOSC1 and TOSC2), the crystal is connected directly between the pins. No external capacitors are needed. The Oscillator is optimized for use with a 32.768 kHz watch crystal. Applying an external clock source to TOSC1 is not recommended.

Note: The Timer/Counter Oscillator uses the same type of crystal oscillator as Low-Frequency Oscillator and the internal capacitors have the same nominal value of 36 pF.

Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to "Analog to Digital Converter" on page 204 for details on ADC operation.

Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. When entering ADC Noise Reduction mode, the Analog Comparator should be disabled. In the other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. Refer to "Analog Comparator" on page 201 for details on how to configure the Analog Comparator.

Brown-out Detector

If the Brown-out Detector is not needed in the application, this module should be turned off. If the Brown-out Detector is enabled by the BODEN Fuse, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Brown-out Detection" on page 40 for details on how to configure the Brown-out Detector.

Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-out Detector, the Analog Comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to "Internal Voltage Reference" on page 42 for details on the start-up time.

Watchdog Timer

If the Watchdog Timer is not needed in the application, this module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Watchdog Timer" on page 42 for details on how to configure the Watchdog Timer.

Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important thing is then to ensure that no pins drive resistive loads. In sleep modes where the both the I/O clock ($clk_{I/O}$) and the ADC clock (clk_{ADC}) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section "Digital Input Enable and Sleep Modes" on page 54 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T0}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set. Figure 34 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

Figure 34. Timer/Counter Timing Diagram, no Prescaling

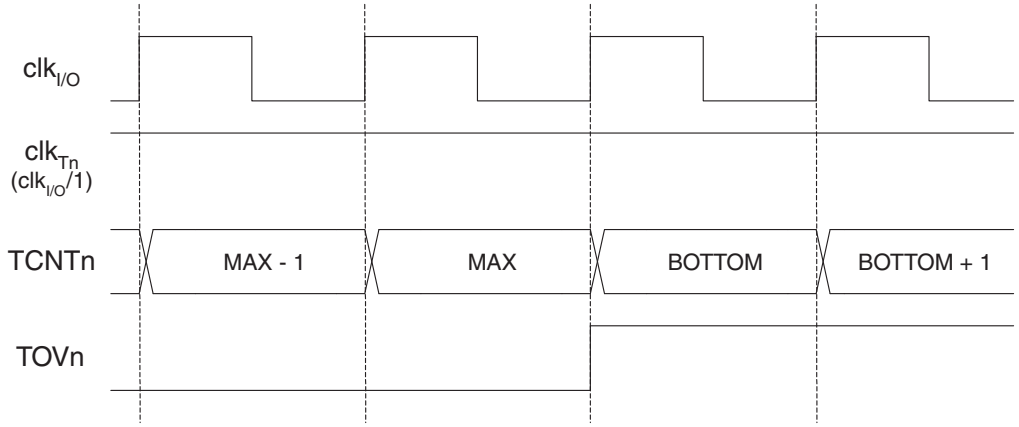


Figure 35 shows the same timing data, but with the prescaler enabled.

Figure 35. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

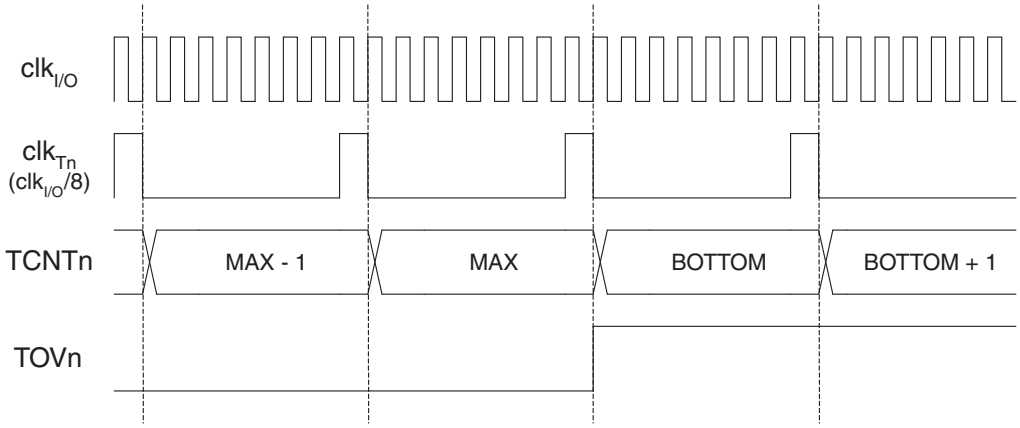


Figure 36 shows the setting of OCF0 in all modes except CTC mode.

The following code examples show how to do an atomic read of the TCNT1 Register contents. Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_ReadTCNT1:
  ; Save global interrupt flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Read TCNT1 into r17:r16
  in  r16,TCNT1L
  in  r17,TCNT1H
  ; Restore global interrupt flag
  out SREG,r18
  ret
```

C Code Example[1]

```
unsigned int TIM16_ReadTCNT1( void )
{
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Read TCNT1 into i */
  i = TCNT1;
  /* Restore global interrupt flag */
  SREG = sreg;
  return i;
}
```

Note: 1. See “About Code Examples” on page 7.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

$\overline{SS}$ Pin Functionality

Slave Mode

When the SPI is configured as a Slave, the Slave Select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs except MISO which can be user configured as an output, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

The $\overline{SS}$ pin is useful for packet/byte synchronization to keep the Slave Bit Counter synchronous with the Master Clock generator. When the $\overline{SS}$ pin is driven high, the SPI Slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

Master Mode

When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the $\overline{SS}$ pin.

If $\overline{SS}$ is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the $\overline{SS}$ pin of the SPI Slave.

If $\overline{SS}$ is configured as an input, it must be held high to ensure Master SPI operation. If the $\overline{SS}$ pin is driven low by peripheral circuitry when the SPI is configured as a Master with the $\overline{SS}$ pin defined as an input, the SPI system interprets this as another Master selecting the SPI as a Slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a Slave. As a result of the SPI becoming a Slave, the MOSI and SCK pins become inputs.
2. The SPIF Flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in Master mode, and there exists a possibility that $\overline{SS}$ is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a Slave Select, it must be set by the user to re-enable SPI Master mode.

SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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	

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the global interrupt enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

Asynchronous Operational Range

The operational range of the receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the receiver does not have a similar (see Table 61) base frequency, the receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F}$$

$$R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D Sum of character size and parity size (D = 5 to 10 bit)

S Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.

S_F First sample number used for majority voting. S_F = 8 for Normal Speed and S_F = 4 for Double Speed mode.

S_M Middle sample number used for majority voting. S_M = 9 for Normal Speed and S_M = 5 for Double Speed mode.

R_{slow} is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. R_{fast} is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 61 and Table 62 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

Table 61. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2X = 0)

D # (Data+Parity Bit)	R_{slow} (%)	R_{fast} (%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	93.20	106.67	+6.67/-6.8	±3.0
6	94.12	105.79	+5.79/-5.88	±2.5
7	94.81	105.11	+5.11/-5.19	±2.0
8	95.36	104.58	+4.58/-4.54	±2.0
9	95.81	104.14	+4.14/-4.19	±1.5
10	96.17	103.78	+3.78/-3.83	±1.5

Table 62. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2X = 1)

D # (Data+Parity Bit)	R_{slow} (%)	R_{fast} (%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	94.12	105.66	+5.66/-5.88	±2.5
6	94.92	104.92	+4.92/-5.08	±2.0
7	95.52	104.35	+4.35/-4.48	±1.5

Multi-processor Communication Mode

Setting the Multi-processor Communication mode (MPCM) bit in UCSRA enables a filtering function of incoming frames received by the USART Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCM setting, but has to be used differently when it is a part of a system utilizing the Multi-processor Communication mode.

If the receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the receiver is set up for frames with nine data bits, then the ninth bit (RXB8) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multi-processor Communication mode enables several Slave MCUs to receive data from a Master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular Slave MCU has been addressed, it will receive the following data frames as normal, while the other Slave MCUs will ignore the received frames until another address frame is received.

Using MPCM

For an MCU to act as a Master MCU, it can use a 9-bit character frame format (UCSZ = 7). The ninth bit (TXB8) must be set when an address frame (TXB8 = 1) or cleared when a data frame (TXB = 0) is being transmitted. The Slave MCUs must in this case be set to use a 9-bit character frame format.

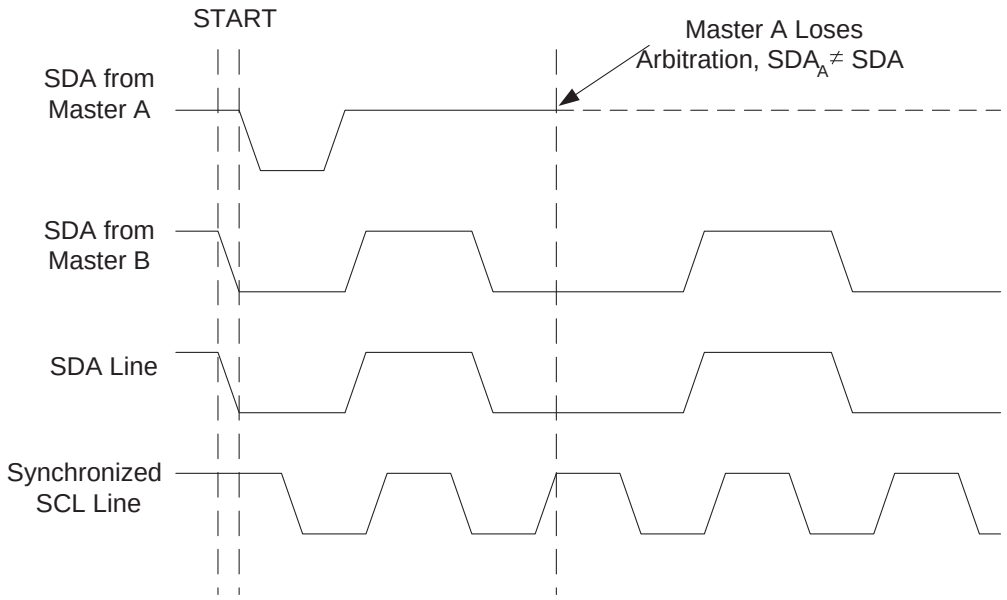
The following procedure should be used to exchange data in Multi-processor Communication mode:

1. All Slave MCUs are in Multi-processor Communication mode (MPCM in UCSRA is set).
2. The Master MCU sends an address frame, and all Slaves receive and read this frame. In the Slave MCUs, the RXC Flag in UCSRA will be set as normal.
3. Each Slave MCU reads the UDR Register and determines if it has been selected. If so, it clears the MPCM bit in UCSRA, otherwise it waits for the next address byte and keeps the MPCM setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other Slave MCUs, which still have the MPCM bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPCM bit and waits for a new address frame from Master. The process then repeats from 2.

Using any of the 5-bit to 8-bit character frame formats is possible, but impractical since the receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult since the transmitter and receiver uses the same character size setting. If 5-bit to 8-bit character frames are used, the transmitter must be set to use two stop bit (USBS = 1) since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPCM bit. The MPCM bit shares the same I/O location as the TXC Flag and this might accidentally be cleared when using SBI or CBI instructions.

Figure 83. Arbitration between Two Masters



Note that arbitration is not allowed between:

- A REPEATED START condition and a data bit
- A STOP condition and a data bit
- A REPEATED START and a STOP condition

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.

4. When the address packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made, for example by using include-files.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Boundary-scan Chain is shifted by the TCK input.
- Update-DR: Data from the Boundary-scan Chain is applied to the output latches. However, the output latches are not connected to the pins.

AVR_RESET; $C

The AVR specific public JTAG instruction for forcing the AVR device into the Reset mode or releasing the JTAG Reset source. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic 'one' in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

BYPASS; $F

Mandatory JTAG instruction selecting the Bypass Register for Data Register.

The active states are:

- Capture-DR: Loads a logic “0” into the Bypass Register.
- Shift-DR: The Bypass Register cell between TDI and TDO is shifted.

Boundary-scan Related Register in I/O Memory

MCU Control and Status Register – MCUCSR

The MCU Control and Status Register contains control bits for general MCU functions, and provides information on which reset source caused an MCU Reset.

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	**–**	**JTRF**	**WDRF**	**BORF**	**EXTRF**	**PORF**	**MCUCSR**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

- **Bit 7 – JTD: JTAG Interface Disable**

When this bit is zero, the JTAG interface is enabled if the JTAGEN Fuse is programmed. If this bit is one, the JTAG interface is disabled. In order to avoid unintentional disabling or enabling of the JTAG interface, a timed sequence must be followed when changing this bit: The application software must write this bit to the desired value twice within four cycles to change its value.

If the JTAG interface is left unconnected to other JTAG circuitry, the JTD bit should be set to one. The reason for this is to avoid static current at the TDO pin in the JTAG interface.

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

Idle Supply Current

Figure 156. Idle Supply Current vs. Frequency (0.1 MHz - 1.0 MHz)

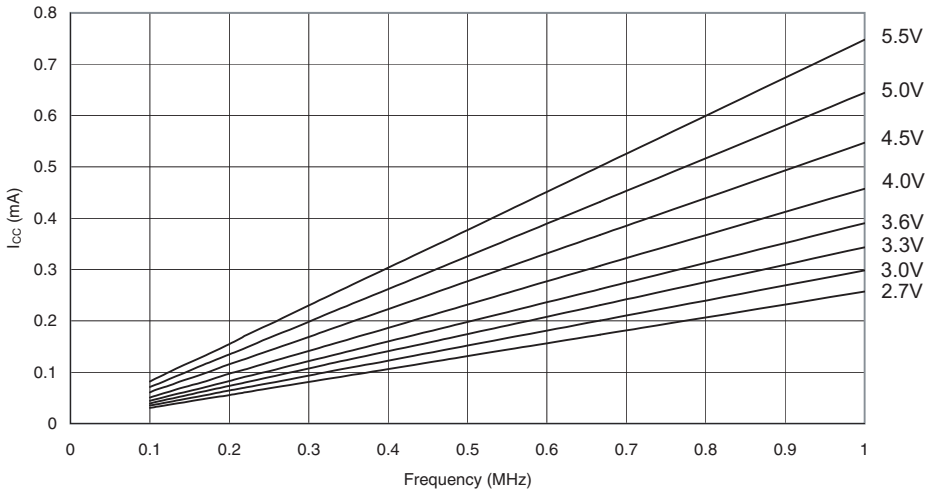
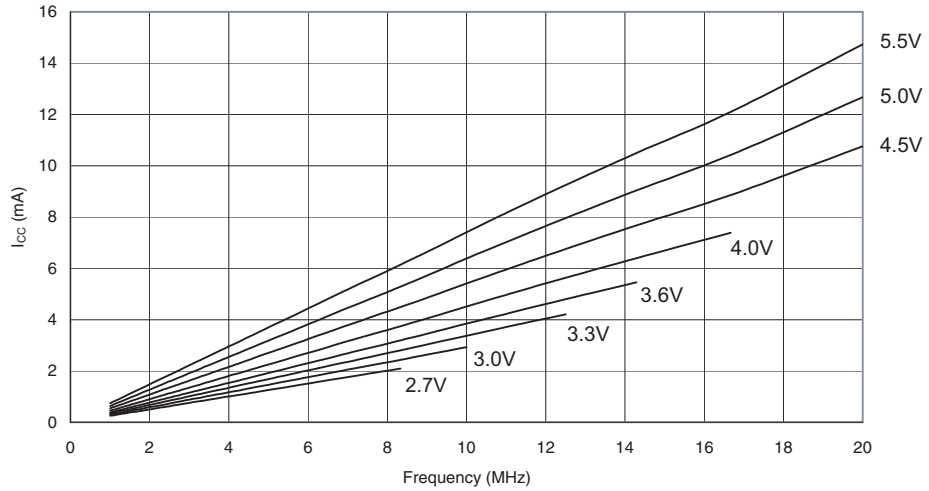


Figure 157. Idle Supply Current vs. Frequency (1 MHz - 20 MHz)



Pin Pullup

Figure 174. I/O Pin Pull-Up Resistor Current vs. Input Voltage (V_{CC} = 5V)

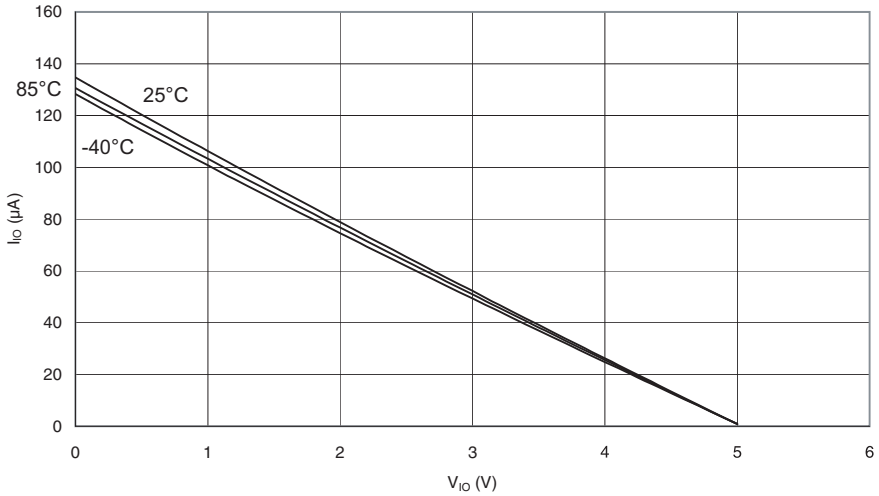


Figure 175. I/O Pin Pull-Up Resistor Current vs. Input Voltage (V_{CC} = 2.7V)

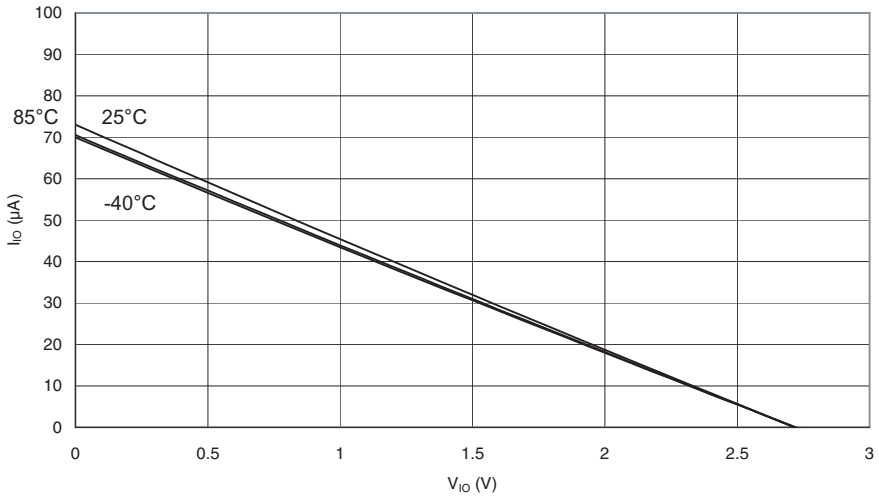


Figure 176. Reset Pull-Up Resistor Current vs. Reset Pin Voltage (V_{CC} = 5V)

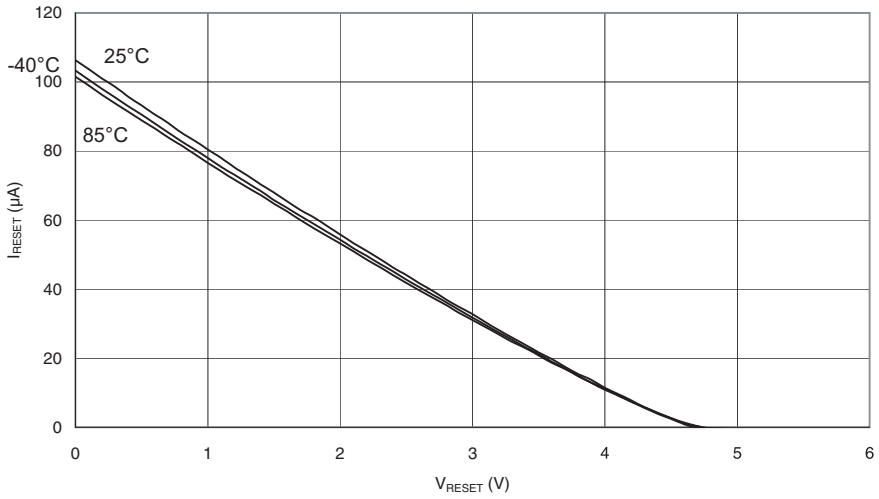


Figure 177. Reset Pull-Up Resistor Current vs. Reset Pin Voltage (V_{CC} = 2.7V)

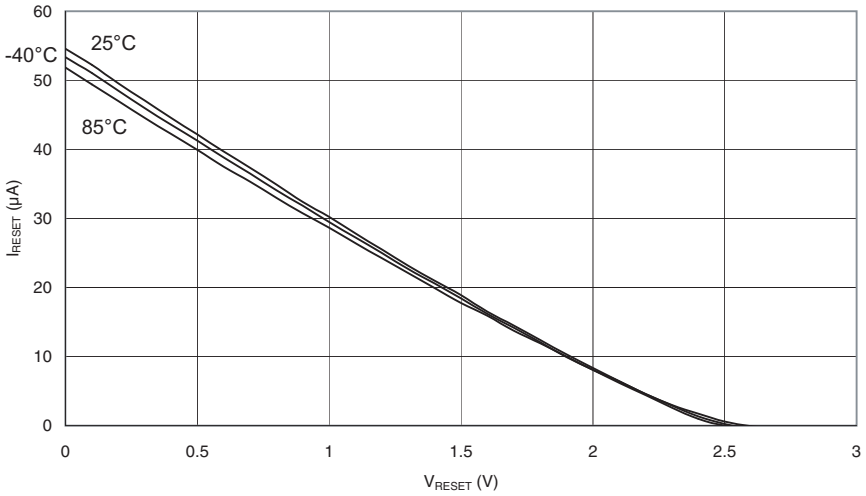


Figure 180. I/O Pin Sink Current vs. Output Voltage (V_{CC} = 5V)

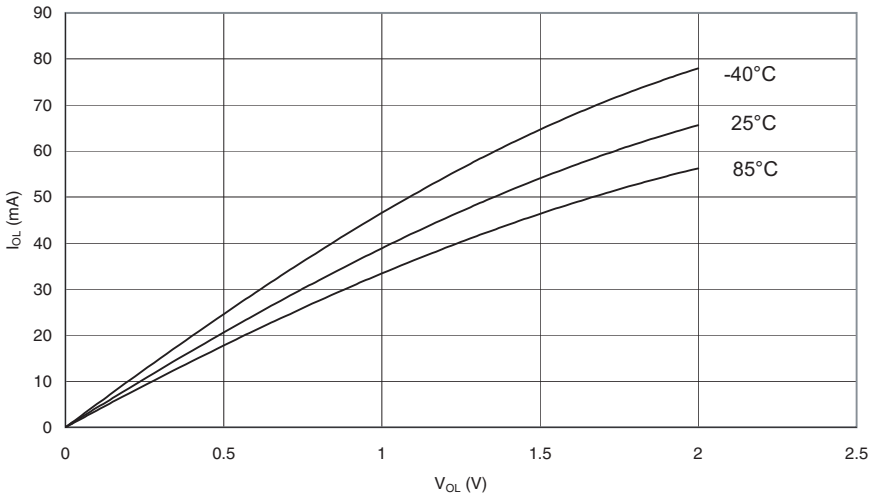


Figure 181. I/O Pin Sink Current vs. Output Voltage (V_{CC} = 2.7V)

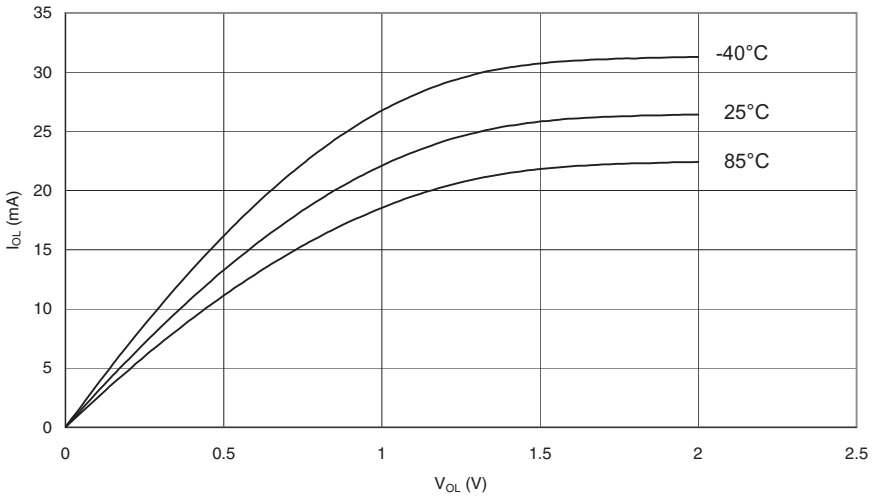


Figure 194. Calibrated 8 MHz RC Oscillator Frequency vs. Temperature

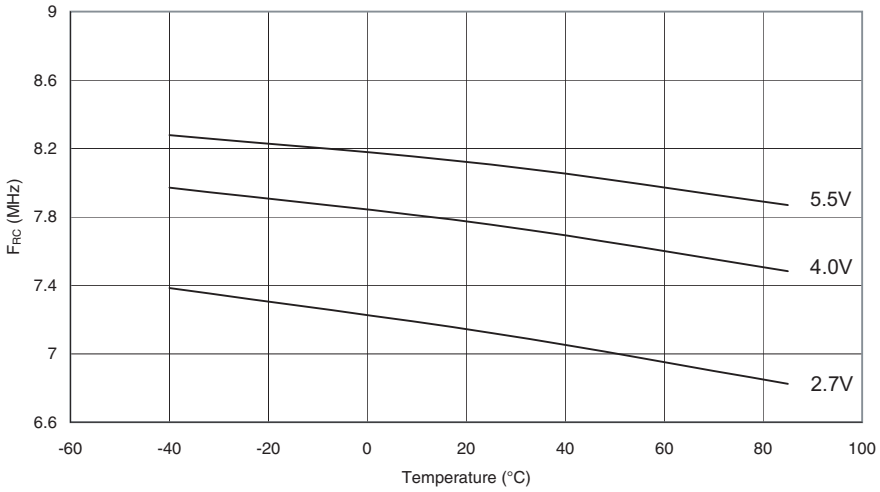
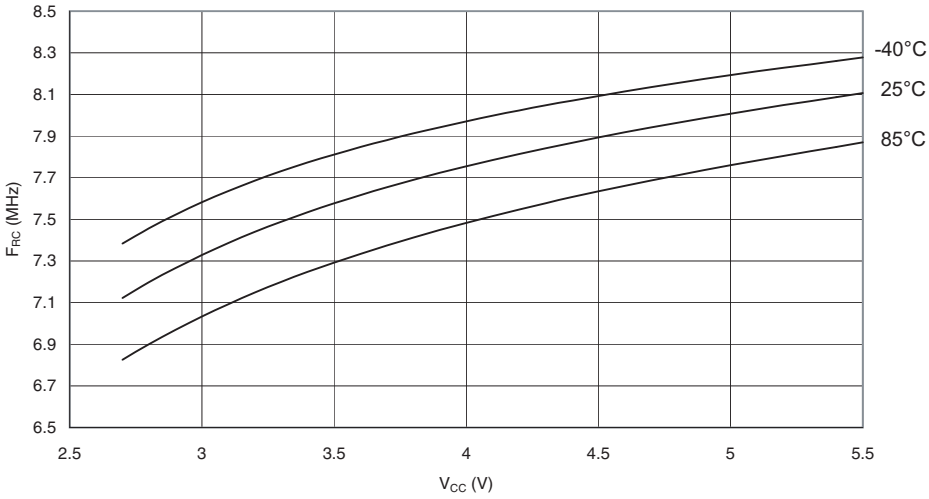


Figure 195. Calibrated 8 MHz RC Oscillator Frequency vs. V_{CC}



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 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 ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - 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 ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x 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
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