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	23
Program Memory Size	8KB (4K 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 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	32-TQFP
Supplier Device Package	32-TQFP (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega88-15at

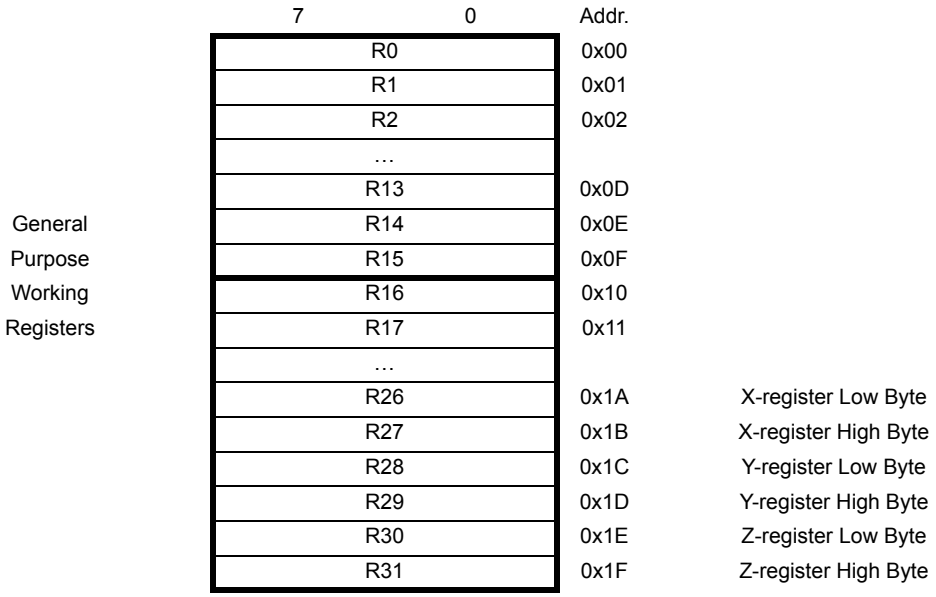
4.5 General Purpose Register File

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 4-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 4-2. 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 4-2, 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.

5. AVR ATmega48/88/168 Memories

This section describes the different memories in the Atmel® ATmega48/88/168. The AVR® architecture has two main memory spaces, the data memory and the program memory space. In addition, the Atmel ATmega48/88/168 features an EEPROM memory for data storage. All three memory spaces are linear and regular.

5.1 In-System Reprogrammable Flash Program Memory

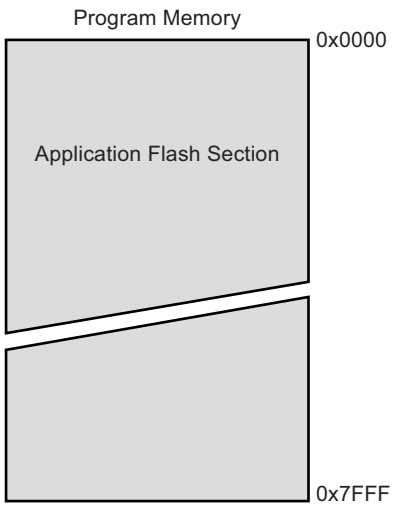
The Atmel ATmega48/88/168 contains 4/8/16K bytes on-chip in-system reprogrammable flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the flash is organized as 2/4/8K x 16. For software security, the flash program memory space is divided into two sections, boot loader section and application program section in Atmel ATmega88 and ATmega168. ATmega48 does not have separate boot loader and application program sections, and the SPM instruction can be executed from the entire flash. See SELFPRGEN description in Section 23.4.1 “Store Program Memory Control and Status Register – SPMCSR” on page 225 and Section 24.5.1 “Store Program Memory Control and Status Register – SPMCSR” on page 233 for more details.

The flash memory has an endurance of at least 75,000 write/erase cycles. The Atmel ATmega48/88/168 program counter (PC) is 11/12/13 bits wide, thus addressing the 2/4/8K program memory locations. The operation of boot program section and associated boot lock bits for software protection are described in detail in Section 23. “Self-Programming the Flash, ATmega48” on page 223 and Section 24. “Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168” on page 229. Section 25. “Memory Programming” on page 242 contains a detailed description on flash programming in SPI- or parallel programming mode.

Constant tables can be allocated within the entire program memory address space (see the LPM – load program memory instruction description).

Timing diagrams for instruction fetch and execution are presented in Section 4.7 “Instruction Execution Timing” on page 13.

Figure 5-1. Program Memory Map, ATmega48



The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

Assembly Code Example

```
EEPROM_write:
        ; Wait for completion of previous write
        sbic    EECR,EEPE
        rjmp    EEPROM_write
        ; Set up address (r18:r17) in address register
        out     EEARH, r18
        out     EEARL, r17
        ; Write data (r16) to Data Register
        out     EEDR,r16
        ; Write logical one to EEMPE
        sbi     EECR,EEMPE
        ; Start eeprom write by setting EEPE
        sbi     EECR,EEPE
        ret
```

C Code Example

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
        /* Wait for completion of previous write */
        while(EECR & (1<<EEPE))
                ;
        /* Set up address and Data Registers */
        EEAR = uiAddress;
        EEDR = ucData;
        /* Write logical one to EEMPE */
        EECR |= (1<<EEMPE);
        /* Start eeprom write by setting EEPE */
        EECR |= (1<<EEPE);
}
```

11. External Interrupts

The external interrupts are triggered by the INT0 and INT1 pins or any of the PCINT23..0 pins. Observe that, if enabled, the interrupts will trigger even if the INT0 and INT1 or PCINT23..0 pins are configured as outputs. This feature provides a way of generating a software interrupt. The pin change interrupt PCI2 will trigger if any enabled PCINT23..16 pin toggles. The pin change interrupt PCI1 will trigger if any enabled PCINT14..8 pin toggles. The pin change interrupt PCI0 will trigger if any enabled PCINT7..0 pin toggles. The PCMSK2, PCMSK1 and PCMSK0 registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT23..0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than idle mode.

The INT0 and INT1 interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the external interrupt control register A – EICRA. When the INT0 or INT1 interrupts are enabled and are configured as level triggered, the interrupts will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT0 or INT1 requires the presence of an I/O clock, described in Section 6.1 “Clock Systems and their Distribution” on page 23. Low level interrupt on INT0 and INT1 is detected asynchronously. This implies that this interrupt can be used for waking the part also from sleep modes other than idle mode. The I/O clock is halted in all sleep modes except idle mode.

Note that if a level triggered interrupt is used for wake-up from power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL fuses as described in Section 6. “System Clock and Clock Options” on page 23.

11.1 External Interrupt Control Register A – EICRA

The external interrupt control register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..4 – Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega48/88/168, and will always read as zero.

- **Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**

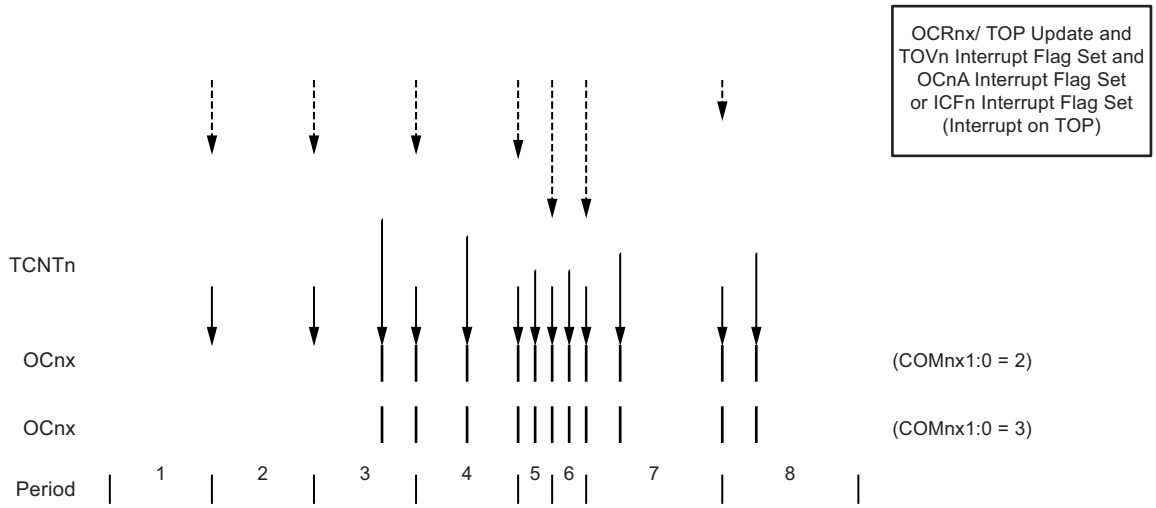
The external interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 11-1. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 11-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 5, 6, or 7), the value in ICR1 (WGM13:0 = 14), or the value in OCR1A (WGM13:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 14-7. The figure shows fast 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 single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x interrupt flag will be set when a compare match occurs.

Figure 14-7. Fast PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOV1) is set each time the counter reaches TOP. In addition the OC1A or ICF1 flag is set at the same timer clock cycle as TOV1 is set when either OCR1A or ICR1 is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

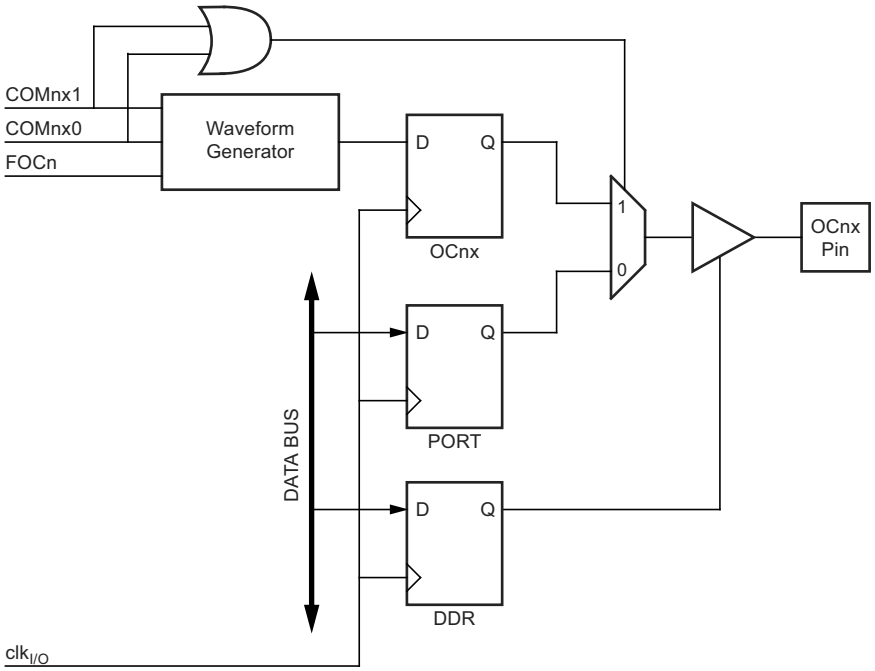
When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCR1x registers are written.

The procedure for updating ICR1 differs from updating OCR1A when used for defining the TOP value. The ICR1 register is not double buffered. This means that if ICR1 is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICR1 value written is lower than the current value of TCNT1. The result will then be that the counter will miss the compare match at the TOP value. The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. The OCR1A register however, is double buffered. This feature allows the OCR1A I/O location to be written anytime. When the OCR1A I/O location is written the value written will be put into the OCR1A buffer register. The OCR1A compare register will then be updated with the value in the buffer register at the next timer clock cycle the TCNT1 matches TOP. The update is done at the same timer clock cycle as the TCNT1 is cleared and the TOV1 flag is set.

Using the ICR1 register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (see Table on page 114). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x register at the compare match between OCR1x and TCNT1, and clearing (or setting) the OC1x register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

Figure 15-4. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the output compare (OC2x) from the waveform generator if either of the COM2x1:0 bits are set. However, the OC2x pin direction (input or output) is still controlled by the data direction register (DDR) for the port pin. The data direction register bit for the OC2x pin (DDR_OC2x) must be set as output before the OC2x value is visible on the pin. The port override function is independent of the waveform generation mode.

The design of the output compare pin logic allows initialization of the OC2x state before the output is enabled. Note that some COM2x1:0 bit settings are reserved for certain modes of operation. See Section 15.8 “8-bit Timer/Counter Register Description” on page 129

15.5.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COM2x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM2x1:0 = 0 tells the waveform generator that no action on the OC2x register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 15-5 on page 130. For fast PWM mode, refer to Table 15-6 on page 130, and for phase correct PWM refer to Table 15-7 on page 130.

A change of the COM2x1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC2x strobe bits.

15.6 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the waveform generation mode (WGM22:0) and compare output mode (COM2x1:0) bits. The compare output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COM2x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM2x1:0 bits control whether the output should be set, cleared, or toggled at a compare match (see Section 15.5 “Compare Match Output Unit” on page 122).

For detailed timing information refer to Section 15.7 “Timer/Counter Timing Diagrams” on page 127.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC2x pin. Setting the COM2x1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM2x1:0 to three. TOP is defined as 0xFF when WGM2:0 = 3, and OCR2A when MGM2:0 = 7 (See Table 15-4 on page 129). The actual OC2x value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC2x register at the compare match between OCR2x and TCNT2 when the counter increments, and setting (or clearing) the OC2x register at compare match between OCR2x and TCNT2 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{\text{clk\_I/O}}}{N \cdot 510}$$

The *N* variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2A register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR2A is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

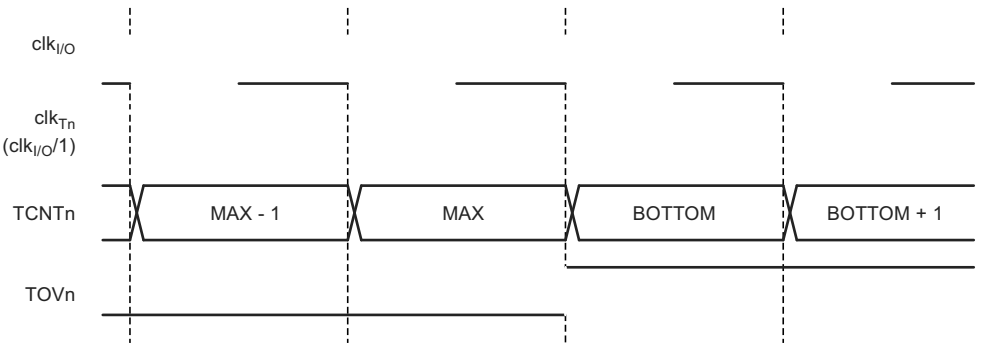
At the very start of period 2 in Figure 15-7 on page 126 OCnx has a transition from high to low even though there is no compare match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without compare match.

- OCR2A changes its value from MAX, like in Figure 15-7 on page 126. When the OCR2A value is MAX the OCn pin value is the same as the result of a down-counting compare match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting compare match.
- The timer starts counting from a value higher than the one in OCR2A, and for that reason misses the compare match and hence the OCn change that would have happened on the way up.

15.7 Timer/Counter Timing Diagrams

The following figures show the Timer/Counter in synchronous mode, and the timer clock (clk_{T2}) is therefore shown as a clock enable signal. In asynchronous mode, $clk_{I/O}$ should be replaced by the Timer/Counter oscillator clock. The figures include information on when interrupt flags are set. Figure 15-8 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 15-8. Timer/Counter Timing Diagram, no Prescaling



15.9 Asynchronous operation of the Timer/Counter

15.9.1 Asynchronous Operation of Timer/Counter2

When Timer/Counter2 operates asynchronously, some considerations must be taken.

- Warning: When switching between asynchronous and synchronous clocking of Timer/Counter2, the timer registers TCNT2, OCR2x, and TCCR2x might be corrupted. A safe procedure for switching clock source is:
  a. Disable the Timer/Counter2 interrupts by clearing OCIE2x and TOIE2.
  b. Select clock source by setting AS2 as appropriate.
  c. Write new values to TCNT2, OCR2x, and TCCR2x.
  d. To switch to asynchronous operation: Wait for TCN2xUB, OCR2xUB, and TCR2xUB.
  e. Clear the Timer/Counter2 interrupt flags.
  f. Enable interrupts, if needed.
- The CPU main clock frequency must be more than four times the oscillator frequency.
- When writing to one of the registers TCNT2, OCR2x, or TCCR2x, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the temporary register have been transferred to its destination. Each of the five mentioned registers have their individual temporary register, which means that e.g. writing to TCNT2 does not disturb an OCR2x write in progress. To detect that a transfer to the destination register has taken place, the asynchronous status register – ASSR has been implemented.
- When entering power-save or ADC noise reduction mode after having written to TCNT2, OCR2x, or TCCR2x, the user must wait until the written register has been updated if Timer/Counter2 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if any of the output compare2 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR2x or TCNT2. If the write cycle is not finished, and the MCU enters sleep mode before the corresponding OCR2xUB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter2 is used to wake the device up from power-save or ADC noise reduction mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering power-save or ADC noise reduction mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  a. Write a value to TCCR2x, TCNT2, or OCR2x.
  b. Wait until the corresponding update busy flag in ASSR returns to zero.
  c. Enter power-save or ADC noise reduction mode.
- When the asynchronous operation is selected, the 32.768kHz oscillator for Timer/Counter2 is always running, except in power-down and standby modes. After a power-up reset or wake-up from power-down or standby mode, the user should be aware of the fact that this oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter2 after power-up or wake-up from power-down or standby mode. The contents of all Timer/Counter2 registers must be considered lost after a wake-up from power-down or standby mode due to unstable clock signal upon start-up, no matter whether the oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from power-save or ADC noise reduction mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.

18.4.1 USART MSPIM Initialization

The USART in MSPIM mode has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting master mode of operation (by setting DDR_XCKn to one), setting frame format and enabling the transmitter and the receiver. Only the transmitter can operate independently. For interrupt driven USART operation, the global interrupt flag should be cleared (and thus interrupts globally disabled) when doing the initialization.

Note: To ensure immediate initialization of the XCKn output the baud-rate register (UBRRn) must be zero at the time the transmitter is enabled. Contrary to the normal mode USART operation the UBRRn must then be written to the desired value after the transmitter is enabled, but before the first transmission is started. Setting UBRRn to zero before enabling the transmitter is not necessary if the initialization is done immediately after a reset since UBRRn is reset to zero.

Before doing a re-initialization with changed baud rate, data mode, or frame format, be sure that there is no ongoing transmissions during the period the registers are changed. The TXCn flag can be used to check that the transmitter has completed all transfers, and the RXCn flag can be used to check that there are no unread data in the receive buffer. Note that the TXCn flag must be cleared before each transmission (before UDRn is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume polling (no interrupts enabled). The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 registers.

Assembly Code Example[1]

```
USART_Init:
        clr r18
        out UBRRnH,r18
        out UBRRnL,r18
        ; Setting the XCKn port pin as output, enables master mode.
        sbi XCKn_DDR, XCKn
        ; Set MSPI mode of operation and SPI data mode 0.
        ldi r18, (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn)
        out UCSRnC,r18
        ; Enable receiver and transmitter.
        ldi r18, (1<<RXENn)|(1<<TXENn)
        out UCSRnB,r18
        ; Set baud rate.
        ; IMPORTANT: The Baud Rate must be set after the transmitter is
                     enabled!
        out UBRRnH, r17
        out UBRRnL, r18
        ret
```

C Code Example[1]

```
void USART_Init( unsigned int baud )
{
        UBRRn = 0;
        /* Setting the XCKn port pin as output, enables master mode. */
        XCKn_DDR |= (1<<XCKn);
        /* Set MSPI mode of operation and SPI data mode 0. */
        UCSRnC = (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn);
        /* Enable receiver and transmitter. */
        UCSRnB = (1<<RXENn)|(1<<TXENn);
        /* Set baud rate. */
        /* IMPORTANT: The Baud Rate must be set after the transmitter
                      is enabled   */
        UBRRn = baud;
}
```

Note: 1. The example code assumes that the part specific header file is included. For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

19. 2-wire Serial Interface

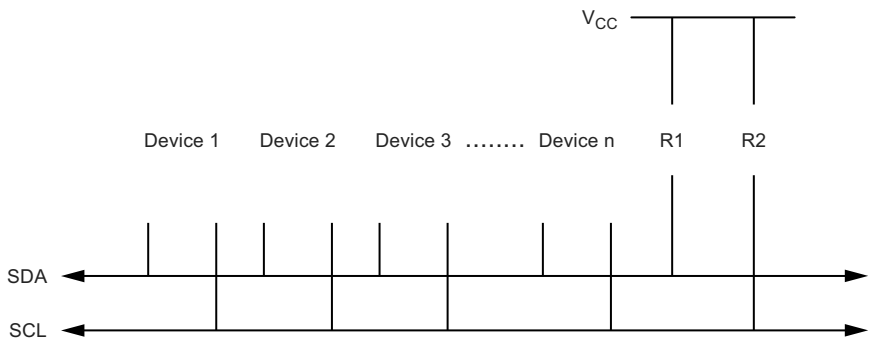
19.1 Features

- Simple yet powerful and flexible communication interface, only two bus lines needed
- Both master and slave operation supported
- Device can operate as transmitter or receiver
- 7-bit address space allows up to 128 different slave addresses
- Multi-master arbitration support
- Up to 400kHz data transfer speed
- Slew-rate limited output drivers
- Noise suppression circuitry rejects spikes on bus lines
- Fully programmable slave address with general call support
- Address recognition causes wake-up when AVR® is in sleep mode

19.2 2-wire Serial Interface Bus Definition

The 2-wire serial interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

Figure 19-1. TWI Bus Interconnection



19.2.1 TWI Terminology

The following definitions are frequently encountered in this section.

Table 19-1. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

The PRTWI bit in Section 7.7.1 “Power Reduction Register - PRR” on page 35 must be written to zero to enable the 2-wire serial interface.

23.4.2 EEPROM Write Prevents Writing to SPMCSR

Note that an EEPROM write operation will block all software programming to flash. Reading the fuses and lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEPE) in the EECR register and verifies that the bit is cleared before writing to the SPMCSR register.

23.4.3 Reading the Fuse and Lock Bits from Software

It is possible to read both the fuse and lock bits from software. To read the lock bits, load the Z-pointer with 0x0001 and set the BLBSET and SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the BLBSET and SELFPRGEN bits are set in SPMCSR, the value of the lock bits will be loaded in the destination register. The BLBSET and SELFPRGEN bits will auto-clear upon completion of reading the lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When BLBSET and SELFPRGEN are cleared, LPM will work as described in the Instruction set manual.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	–	–	–	–	**LB2**	**LB1**

The algorithm for reading the fuse low byte is similar to the one described above for reading the lock bits. To read the fuse low byte, load the Z-pointer with 0x0000 and set the BLBSET and SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the fuse low byte (FLB) will be loaded in the destination register as shown below.See Table 25-5 on page 244 for a detailed description and mapping of the fuse low byte.

Bit	7	6	5	4	3	2	1	0
Rd	**FLB7**	**FLB6**	**FLB5**	**FLB4**	**FLB3**	**FLB2**	**FLB1**	**FLB0**

Similarly, when reading the fuse high byte (FHB), load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the fuse high byte will be loaded in the destination register as shown below. See Table 25-4 on page 243 for detailed description and mapping of the extended fuse byte.

Bit	7	6	5	4	3	2	1	0
Rd	**FHB7**	**FHB6**	**FHB5**	**FHB4**	**FHB3**	**FHB2**	**FHB1**	**FHB0**

Similarly, when reading the extended fuse byte (EFB), load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the extended fuse byte will be loaded in the destination register as shown below. See Table 25-5 on page 244 for detailed description and mapping of the extended fuse byte.

Bit	7	6	5	4	3	2	1	0
Rd	**FHB7**	**FHB6**	**FHB5**	**FHB4**	**FHB3**	**FHB2**	**FHB1**	**FHB0**

Fuse and lock bits that are programmed, will be read as zero. Fuse and lock bits that are unprogrammed, will be read as one.

23.4.4 Preventing Flash Corruption

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

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

24. Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168

In Atmel® ATmega88 and Atmel ATmega168, the boot loader support provides a real read-while-write self-programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a flash-resident boot loader program. The boot loader program can use any available data interface and associated protocol to read code and write (program) that code into the flash memory, or read the code from the program memory. The program code within the boot loader section has the capability to write into the entire flash, including the boot loader memory. The boot loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the boot loader memory is configurable with fuses and the boot loader has two separate sets of boot lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

24.1 Boot Loader Features

- Read-while-write self-programming
- Flexible boot memory size
- High security (separate boot lock bits for a flexible protection)
- Separate fuse to select reset vector
- Optimized page(1) size
- Code efficient algorithm
- Efficient read-modify-write support

Note: 1. A page is a section in the flash consisting of several bytes (see Table 25-12 on page 247) used during programming. The page organization does not affect normal operation.

24.2 Application and Boot Loader Flash Sections

The flash memory is organized in two main sections, the application section and the boot loader section (see Figure 24-2 on page 231). The size of the different sections is configured by the BOOTSZ fuses as shown in Table 24-6 on page 240 and Figure 24-2 on page 231. These two sections can have different level of protection since they have different sets of lock bits.

24.2.1 Application Section

The application section is the section of the flash that is used for storing the application code. The protection level for the application section can be selected by the application boot lock bits (boot lock bits 0), see Table 24-2 on page 232. The application section can never store any boot loader code since the SPM instruction is disabled when executed from the application section.

24.2.2 BLS – Boot Loader Section

While the application section is used for storing the application code, the The boot loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire flash, including the BLS itself. The protection level for the boot loader section can be selected by the boot loader lock bits (boot lock bits 1), see Table 24-3 on page 232.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SELFPRGEN, the next SPM instruction within four clock cycles executes page write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a page write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation if the NRWW section is addressed.

- **Bit 1 – PGERS: Page Erase**

If this bit is written to one at the same time as SELFPRGEN, the next SPM instruction within four clock cycles executes page erase. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a page erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation if the NRWW section is addressed.

- **Bit 0 – SELFPRGEN: Self Programming Enable**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT or PGERS, the following SPM instruction will have a special meaning, see description above. If only SELFPRGEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer. The LSB of the Z-pointer is ignored. The SELFPRGEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During page erase and page write, the SELFPRGEN bit remains high until the operation is completed.

Writing any other combination than “10001”, “01001”, “00101”, “00011” or “00001” in the lower five bits will have no effect.

24.6 Addressing the Flash During Self-Programming

The Z-pointer is used to address the SPM commands.

Bit	15	14	13	12	11	10	9	8
ZH (R31)	**Z15**	**Z14**	**Z13**	**Z12**	**Z11**	**Z10**	**Z9**	**Z8**
ZL (R30)	**Z7**	**Z6**	**Z5**	**Z4**	**Z3**	**Z2**	**Z1**	**Z0**
	7	6	5	4	3	2	1	0

Since the flash is organized in pages (see Table 25-12 on page 247), the program counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is1 shown in Figure 24-3 on page 235. Note that the page erase and page write operations are addressed independently. Therefore it is of major importance that the boot loader software addresses the same page in both the page erase and page write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

The only SPM operation that does not use the Z-pointer is setting the boot loader lock bits. The content of the Z-pointer is ignored and will have no effect on the operation. The LPM instruction does also use the Z-pointer to store the address. Since this instruction addresses the flash byte-by-byte, also the LSB (bit Z0) of the Z-pointer is used.

24.7.8 EEPROM Write Prevents Writing to SPMCSR

Note that an EEPROM write operation will block all software programming to flash. Reading the fuses and lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEPE) in the EECR register and verifies that the bit is cleared before writing to the SPMCSR register.

24.7.9 Reading the Fuse and Lock Bits from Software

It is possible to read both the fuse and lock bits from software. To read the lock bits, load the Z-pointer with 0x0001 and set the BLBSET and SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the BLBSET and SELFPRGEN bits are set in SPMCSR, the value of the lock bits will be loaded in the destination register. The BLBSET and SELFPRGEN bits will auto-clear upon completion of reading the lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When BLBSET and SELFPRGEN are cleared, LPM will work as described in the Instruction set manual.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	LB2	LB1

The algorithm for reading the fuse low byte is similar to the one described above for reading the lock bits. To read the fuse low byte, load the Z-pointer with 0x0000 and set the BLBSET and SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the fuse low byte (FLB) will be loaded in the destination register as shown below. Refer to Table 25-5 on page 244 for a detailed description and mapping of the fuse low byte.

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

Similarly, when reading the fuse high byte, load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the fuse high byte (FHB) will be loaded in the destination register as shown below. Refer to Table 25-6 on page 244 for detailed description and mapping of the fuse high byte.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

When reading the extended fuse byte, load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the extended fuse byte (EFB) will be loaded in the destination register as shown below. Refer to Table 25-4 on page 243 for detailed description and mapping of the extended fuse byte.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	–	–	EFB3	EFB2	EFB1	EFB0

Fuse and lock bits that are programmed, will be read as zero. Fuse and lock bits that are unprogrammed, will be read as one.

24.7.10 Preventing Flash Corruption

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

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

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):

1. If there is no need for a boot loader update in the system, program the boot loader lock bits to prevent any boot loader software updates.
2. 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 operating voltage matches the detection level. If not, 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.
3. Keep the AVR® core in power-down sleep mode during periods of low V_{CC}. This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCSR register and thus the flash from unintentional writes.

24.7.11 Programming Time for Flash when Using SPM

The calibrated RC oscillator is used to time flash accesses. Table 24-5 shows the typical programming time for flash accesses from the CPU.

Table 24-5. SPM Programming Time

Symbol	Min Programming Time	Max Programming Time
Flash write (page erase, page write, and write lock bits by SPM)	3.7ms	4.5ms

24.7.12 Simple Assembly Code Example for a Boot Loader

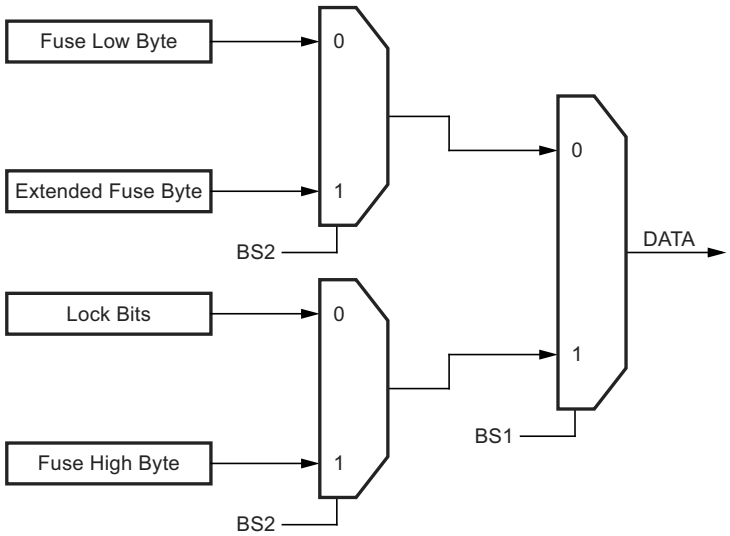
```
        ;-the routine writes one page of data from RAM to Flash
        ; the first data location in RAM is pointed to by the Y pointer
        ; the first data location in Flash is pointed to by the Z-pointer
        ;-error handling is not included
        ;-the routine must be placed inside the Boot space
        ; (at least the Do_spm sub routine). Only code inside NRWW section can
        ; be read during Self-Programming (Page Erase and Page Write).
        ;-registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
        ; loophi (r25), spmcrval (r20)
        ; storing and restoring of registers is not included in the routine
        ; register usage can be optimized at the expense of code size
        ;-It is assumed that either the interrupt table is moved to the Boot
        ; loader section or that the interrupts are disabled.
.equ   PAGESIZEB = PAGESIZE*2     ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
        ;Page Erase
        ldi    spmcrval, (1<<PGERS) | (1<<SELFPRGEN)
        call   Do_spm

        ;      re-enable the RWW section
        ldi    spmcrval, (1<<RWWSRE) | (1<<SELFPRGEN)
        call   Do_spm

        ;      transfer data from RAM to Flash page buffer
        ldi    looplo, low(PAGESIZEB)     ;init loop variable
        ldi    loophi, high(PAGESIZEB)    ;not required for PAGESIZEB<=256
Wrloop:
        ld     r0, Y+
        ld     r1, Y+
        ldi    spmcrval, (1<<SELFPRGEN)
        call   Do_spm
        adiw   ZH:ZL, 2
        sbiw   loophi:looplo, 2    ;use subi for PAGESIZEB<=256
```

25.7.12 Reading the Fuse and Lock Bits

The algorithm for reading the fuse and lock bits is as follows (refer to Section 25.7.4 “Programming the Flash” on page 248 for details on command loading):

1. A: Load command “0000 0100”.
2. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “0”. The status of the fuse low bits can now be read at DATA (“0” means programmed).
3. Set $\overline{OE}$ to “0”, BS2 to “1” and BS1 to “1”. The status of the fuse high bits can now be read at DATA (“0” means programmed).
4. Set OE to “0”, BS2 to “1”, and BS1 to “0”. The status of the extended fuse bits can now be read at DATA (“0” means programmed).
5. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “1”. The status of the lock bits can now be read at DATA (“0” means programmed).
6. Set $\overline{OE}$ to “1”.

Figure 25-6. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read



25.7.13 Reading the Signature Bytes

The algorithm for reading the signature bytes is as follows (refer to Section 25.7.4 “Programming the Flash” on page 248 for details on command and address loading):

1. A: Load command “0000 1000”.
2. B: Load address low byte (0x00 - 0x02).
3. Set $\overline{OE}$ to “0”, and BS1 to “0”. The selected signature byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

25.7.14 Reading the Calibration Byte

The algorithm for reading the calibration byte is as follows (refer to Section 25.7.4 “Programming the Flash” on page 248 for details on command and address loading):

1. A: Load command “0000 1000”.
2. B: Load address low byte, 0x00.
3. Set $\overline{OE}$ to “0”, and BS1 to “1”. The calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

27. 2-wire Serial Interface Characteristics

Table 27-1 describes the requirements for devices connected to the 2-wire serial bus. The Atmel® ATmega48/88/168 2-wire serial interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 27-1.

Table 27-1. 2-wire Serial Bus Requirements

Parameter	Condition	Symbol	Min	Max	Unit
Input low-voltage		V_{IL}	–0.5	0.3 V_{CC}	V
Input high-voltage		V_{IH}	0.7 V_{CC}	V_{CC} + 0.5	V
Hysteresis of schmitt trigger inputs		V_{hys}(1)	0.05 V_{CC}(2)	–	V
Output low-voltage	3mA sink current	V_{OL}(1)	0	0.4	V
Rise time for both SDA and SCL		t_r(1)	20 + 0.1C_b(3)(2)	300	ns
Output fall time from V_{IHmin} to V_{ILmax}	10pF < C_b < 400pF(3)	t_{of}(1)	20 + 0.1C_b(3)(2)	250	ns
Spikes suppressed by input filter		t_{SP}(1)	0	50(2)	ns
Input current each I/O pin	0.1V_{CC} < V_i < 0.9V_{CC}	I_i	–10	10	µA
Capacitance for each I/O Pin		C_i(1)	–	10	pF
SCL clock frequency	f_{CK}(4) > max(16f_{SCL}, 250kHz)(5)	f_{SCL}	0	400	kHz
Value of pull-up resistor	f_{SCL} ≤ 100kHz	Rp	$\frac{V_{CC}-0,4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
	f_{SCL} > 100kHz		$\frac{V_{CC}-0,4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
Hold time (repeated) START Condition	f_{SCL} ≤ 100kHz	$t_{HD;STA}$	4.0	–	µs
	f_{SCL} > 100kHz		0.6	–	µs
Low period of the SCL clock	f_{SCL} ≤ 100kHz(6)	t_{LOW}	4.7	–	µs
	f_{SCL} > 100kHz(7)		1.3	–	µs
High period of the SCL clock	f_{SCL} ≤ 100kHz	t_{HIGH}	4.0	–	µs
	f_{SCL} > 100kHz		0.6	–	µs
Set-up time for a repeated START condition	f_{SCL} ≤ 100kHz	$t_{SU;STA}$	4.7	–	µs
	f_{SCL} > 100kHz		0.6	–	µs
Data hold time	f_{SCL} ≤ 100kHz	$t_{HD;DAT}$	0	3.45	µs
	f_{SCL} > 100kHz		0	0.9	µs

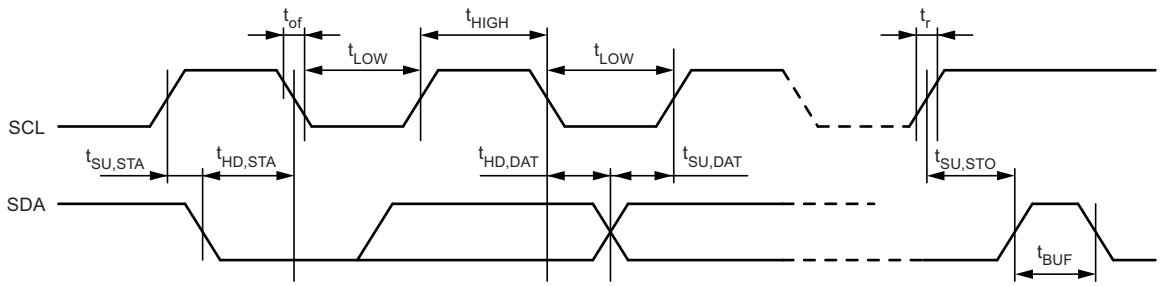
Notes:
1. In Atmel ATmega48/88/168, this parameter is characterized and not 100% tested.
2. Required only for f_{SCL} > 100kHz.
3. C_b = capacitance of one bus line in pF.
4. f_{CK} = CPU clock frequency
5. This requirement applies to all ATmega48/88/168 2-wire serial interface operation. Other devices connected to the 2-wire serial bus need only obey the general f_{SCL} requirement.
6. The actual low period generated by the Atmel ATmega48/88/168 2-wire serial interface is ($1/f_{SCL} – 2/f_{CK}$), thus f_{CK} must be greater than 6MHz for the low time requirement to be strictly met at f_{SCL} = 100kHz.
7. The actual low period generated by the ATmega48/88/168 2-wire serial interface is ($1/f_{SCL} – 2/f_{CK}$), thus the low time requirement will not be strictly met for f_{SCL} > 308kHz when f_{CK} = 8MHz. Still, ATmega48/88/168 devices connected to the bus may communicate at full speed (400kHz) with other ATmega48/88/168 devices, as well as any other device with a proper t_{LOW} acceptance margin.

Table 27-1. 2-wire Serial Bus Requirements (Continued)

Parameter	Condition	Symbol	Min	Max	Unit
Data setup time	$f_{SCL} \leq 100kHz$	$t_{SU;DAT}$	250	–	ns
	$f_{SCL} > 100kHz$		100	–	ns
Setup time for STOP condition	$f_{SCL} \leq 100kHz$	$t_{SU;STO}$	4.0	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Bus free time between a STOP and START condition	$f_{SCL} \leq 100kHz$	t_{BUF}	4.7	–	µs
	$f_{SCL} > 100kHz$		1.3	–	µs

Notes:
1. In Atmel ATmega48/88/168, this parameter is characterized and not 100% tested.
2. Required only for $f_{SCL} > 100kHz$.
3. C_b = capacitance of one bus line in pF.
4. f_{CK} = CPU clock frequency
5. This requirement applies to all ATmega48/88/168 2-wire serial interface operation. Other devices connected to the 2-wire serial bus need only obey the general f_{SCL} requirement.
6. The actual low period generated by the Atmel ATmega48/88/168 2-wire serial interface is $(1/f_{SCL} – 2/f_{CK})$, thus f_{CK} must be greater than 6MHz for the low time requirement to be strictly met at f_{SCL} = 100kHz.
7. The actual low period generated by the ATmega48/88/168 2-wire serial interface is $(1/f_{SCL} – 2/f_{CK})$, thus the low time requirement will not be strictly met for $f_{SCL} > 308kHz$ when f_{CK} = 8MHz. Still, ATmega48/88/168 devices connected to the bus may communicate at full speed (400kHz) with other ATmega48/88/168 devices, as well as any other device with a proper t_{LOW} acceptance margin.

Figure 27-1. 2-wire Serial Bus Timing



28.1.4 Pin Thresholds and Hysteresis

Figure 28-15. I/O Pin Input Threshold versus V_{CC} (VIH, I/O Pin Read as '1

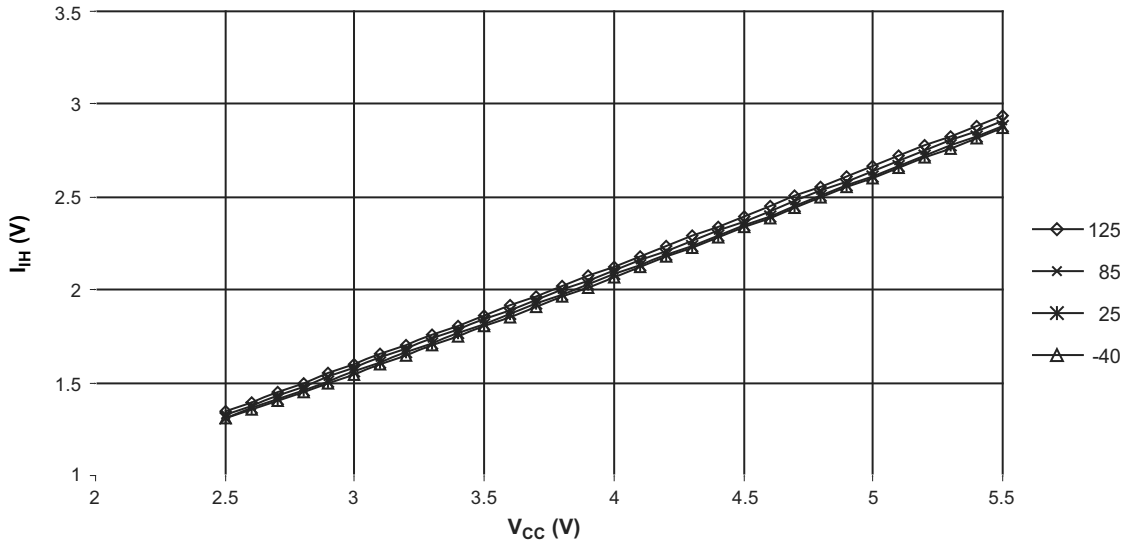


Figure 28-16. I/O Pin Input Threshold versus V_{CC} (VIL, I/O Pin Read as '0')