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	8MHz
Connectivity	EBI/EMI, I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	54
Program Memory Size	128KB (64K x 16)
Program Memory Type	FLASH
EEPROM Size	4K x 8
RAM Size	8K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	64-TQFP
Supplier Device Package	64-TQFP (14x14)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega1281v-8aur

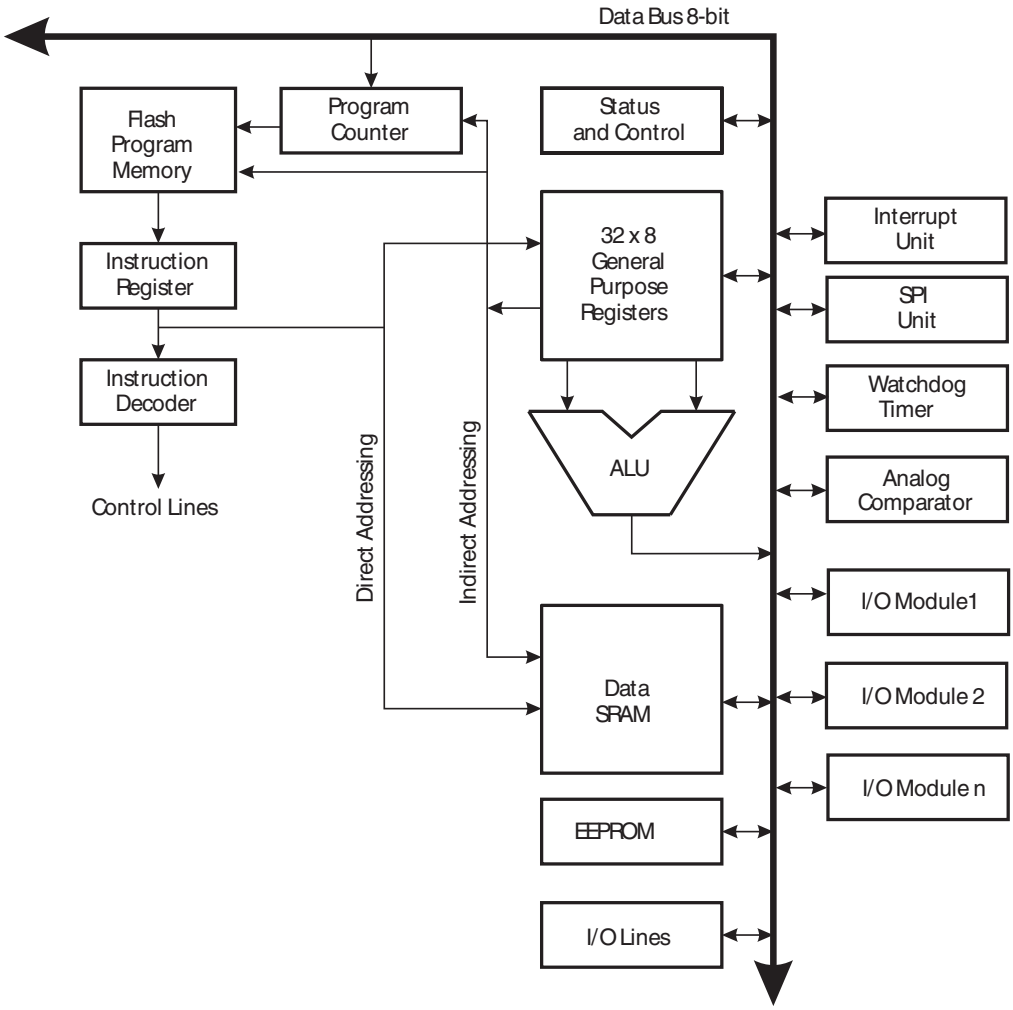
7. AVR CPU Core

7.1 Introduction

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

7.2 Architectural Overview

Figure 7-1. Block Diagram of the AVR Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 × 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two oper-

ands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16-bit or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, the ATmega640/1280/1281/2560/2561 has Extended I/O space from 0x60 - 0x1FF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

7.3 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the “Instruction Set Summary” on page 404 for a detailed description.

7.4 Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the “Instruction Set Summary” on page 404. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

Figure 7-6. The Parallel Instruction Fetches and Instruction Executions

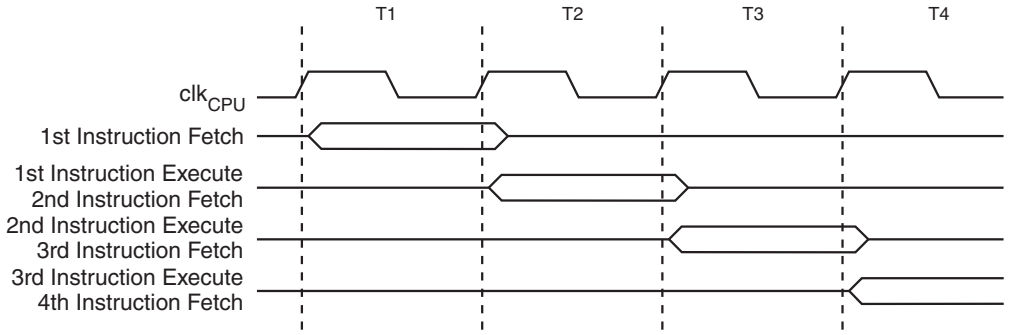
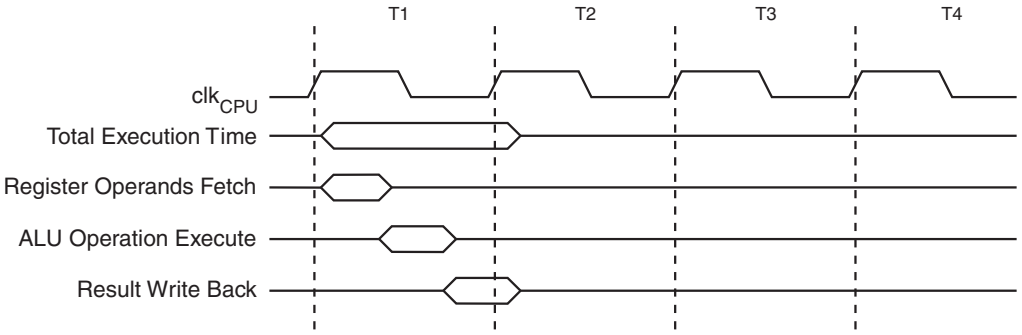


Figure 7-7 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 7-7. Single Cycle ALU Operation



7.8 Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section “Memory Programming” on page 325 for details.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in “Interrupts” on page 101. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). Refer to “Interrupts” on page 101 for more information. The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse, see “Memory Programming” on page 325.

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

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the

flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.

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

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

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

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

Assembly Code Example

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

C Code Example

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

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

Assembly Code Example

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

C Code Example

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

11.10.2 PRR0 – Power Reduction Register 0

Bit	7	6	5	4	3	2	1	0	
(0x64)	PRTWI	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	PRR0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 - PRTWI: Power Reduction TWI**

Writing a logic one to this bit shuts down the TWI by stopping the clock to the module. When waking up the TWI again, the TWI should be re initialized to ensure proper operation.

- **Bit 6 - PRTIM2: Power Reduction Timer/Counter2**

Writing a logic one to this bit shuts down the Timer/Counter2 module in synchronous mode (AS2 is 0). When the Timer/Counter2 is enabled, operation will continue like before the shutdown.

- **Bit 5 - PRTIM0: Power Reduction Timer/Counter0**

Writing a logic one to this bit shuts down the Timer/Counter0 module. When the Timer/Counter0 is enabled, operation will continue like before the shutdown.

- **Bit 4 - Res: Reserved bit**

This bit is reserved bit and will always read as zero.

- **Bit 3 - PRTIM1: Power Reduction Timer/Counter1**

Writing a logic one to this bit shuts down the Timer/Counter1 module. When the Timer/Counter1 is enabled, operation will continue like before the shutdown.

- **Bit 2 - PRSPI: Power Reduction Serial Peripheral Interface**

Writing a logic one to this bit shuts down the Serial Peripheral Interface by stopping the clock to the module. When waking up the SPI again, the SPI should be re initialized to ensure proper operation.

- **Bit 1 - PRUSART0: Power Reduction USART0**

Writing a logic one to this bit shuts down the USART0 by stopping the clock to the module. When waking up the USART0 again, the USART0 should be re initialized to ensure proper operation.

- **Bit 0 - PRADC: Power Reduction ADC**

Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot use the ADC input MUX when the ADC is shut down.

11.10.3 PRR1 – Power Reduction Register 1

Bit	7	6	5	4	3	2	1	0	
(0x65)	–	–	PRTIM5	PRTIM4	PRTIM3	PRUSART3	PRUSART2	PRUSART1	PRR1
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 - Res: Reserved bits**

These bits are reserved and will always read as zero.

- **Bit 5 - PRTIM5: Power Reduction Timer/Counter5**

Writing a logic one to this bit shuts down the Timer/Counter5 module. When the Timer/Counter5 is enabled, operation will continue like before the shutdown.

- **Bit 4 - PRTIM4: Power Reduction Timer/Counter4**

Writing a logic one to this bit shuts down the Timer/Counter4 module. When the Timer/Counter4 is enabled, operation will continue like before the shutdown.

- **Bit 3 - PRTIM3: Power Reduction Timer/Counter3**

Writing a logic one to this bit shuts down the Timer/Counter3 module. When the Timer/Counter3 is enabled, operation will continue like before the shutdown.

- **Bit 2 - PRUSART3: Power Reduction USART3**

Writing a logic one to this bit shuts down the USART3 by stopping the clock to the module. When waking up the USART3 again, the USART3 should be re initialized to ensure proper operation.

- **Bit 1 - PRUSART2: Power Reduction USART2**

Writing a logic one to this bit shuts down the USART2 by stopping the clock to the module. When waking up the USART2 again, the USART2 should be re initialized to ensure proper operation.

- **Bit 0 - PRUSART1: Power Reduction USART1**

Writing a logic one to this bit shuts down the USART1 by stopping the clock to the module. When waking up the USART1 again, the USART1 should be re initialized to ensure proper operation.

12. System Control and Reset

12.1 Resetting the AVR

During reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be a JMP – Absolute Jump – instruction to the reset handling routine. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa. The circuit diagram in Figure 12-1 on page 58 shows the reset logic. "System and Reset Characteristics" on page 360 defines the electrical parameters of the reset circuitry.

The I/O ports of the AVR are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running.

After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the SUT and CKSEL Fuses. The different selections for the delay period are presented in "Clock Sources" on page 40.

12.2 Reset Sources

The ATmega640/1280/1281/2560/2561 has five sources of reset:

- **Power-on Reset**. The MCU is reset when the supply voltage is below the Power-on Reset threshold (V_{POT})
- **External Reset**. The MCU is reset when a low level is present on the $\overline{\text{RESET}}$ pin for longer than the minimum pulse length
- **Watchdog Reset**. The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled
- **Brown-out Reset**. The MCU is reset when the supply voltage AV_{CC} is below the Brown-out Reset threshold (V_{BOT}) and the Brown-out Detector is enabled
- **JTAG AVR Reset**. The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system. Refer to the section "IEEE 1149.1 (JTAG) Boundary-scan" on page 295 for details

12.5.2 WDTCSR – Watchdog Timer Control Register

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

- **Bit 7 - WDIF: Watchdog Interrupt Flag**

This bit is set when a time-out occurs in the Watchdog Timer and the Watchdog Timer is configured for interrupt. WDIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, WDIF is cleared by writing a logic one to the flag. When the I-bit in SREG and WDIE are set, the Watchdog Time-out Interrupt is executed.

- **Bit 6 - WDIE: Watchdog Interrupt Enable**

When this bit is written to one and the I-bit in the Status Register is set, the Watchdog Interrupt is enabled. If WDE is cleared in combination with this setting, the Watchdog Timer is in Interrupt Mode, and the corresponding interrupt is executed if time-out in the Watchdog Timer occurs.

If WDE is set, the Watchdog Timer is in Interrupt and System Reset Mode. The first time-out in the Watchdog Timer will set WDIF. Executing the corresponding interrupt vector will clear WDIE and WDIF automatically by hardware (the Watchdog goes to System Reset Mode). This is useful for keeping the Watchdog Timer security while using the interrupt. To stay in Interrupt and System Reset Mode, WDIE must be set after each interrupt. This should however not be done within the interrupt service routine itself, as this might compromise the safety-function of the Watchdog System Reset mode. If the interrupt is not executed before the next time-out, a System Reset will be applied.

Table 12-1. Watchdog Timer Configuration

WDTON[1]	WDE	WDIE	Mode	Action on Time-out
1	0	0	Stopped	None
1	0	1	Interrupt Mode	Interrupt
1	1	0	System Reset Mode	Reset
1	1	1	Interrupt and System Reset Mode	Interrupt, then go to System Reset Mode
0	x	x	System Reset Mode	Reset

Note: 1. WDTON Fuse set to “0“ means programmed and “1” means unprogrammed.

- **Bit 4 - WDCE: Watchdog Change Enable**

This bit is used in timed sequences for changing WDE and prescaler bits. To clear the WDE bit, and/or change the prescaler bits, WDCE must be set.

Once written to one, hardware will clear WDCE after four clock cycles.

- **Bit 3 - WDE: Watchdog System Reset Enable**

WDE is overridden by WDRF in MCUSR. This means that WDE is always set when WDRF is set. To clear WDE, WDRF must be cleared first. This feature ensures multiple resets during conditions causing failure, and a safe start-up after the failure.

- **Bit 5, 2:0 - WDP3:0: Watchdog Timer Prescaler 3, 2, 1 and 0**

The WDP3:0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is running. The different prescaling values and their corresponding time-out periods are shown in Table 12-2 on page 66.

Table 13-14. Overriding Signals for Alternate Functions in PD3:PD0[1]

Signal Name	PD3/INT3/TXD1	PD2/INT2/RXD1	PD1/INT1/SDA	PD0/INT0/SCL
PUOE	TXEN1	RXEN1	TWEN	TWEN
PUOV	0	PORTD2 • $\overline{PUD}$	PORTD1 • $\overline{PUD}$	PORTD0 • $\overline{PUD}$
DDOE	TXEN1	RXEN1	TWEN	TWEN
DDOV	1	0	SDA_OUT	SCL_OUT
PVOE	TXEN1	0	TWEN	TWEN
PVOV	TXD1	0	0	0
DIEOE	INT3 ENABLE	INT2 ENABLE	INT1 ENABLE	INT0 ENABLE
DIEOV	1	1	1	1
DI	INT3 INPUT	INT2 INPUT/RXD1	INT1 INPUT	INT0 INPUT
AIO	–	–	SDA INPUT	SCL INPUT

Note: 1. When enabled, the 2-wire Serial Interface enables Slew-Rate controls on the output pins PD0 and PD1. This is not shown in this table. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module.

13.3.5 Alternate Functions of Port E

The Port E pins with alternate functions are shown in Table 13-15.

Table 13-15. Port E Pins Alternate Functions

Port Pin	Alternate Function
PE7	INT7/ICP3/CLK0 (External Interrupt 7 Input, Timer/Counter3 Input Capture Trigger or Divided System Clock)
PE6	INT6/ T3 (External Interrupt 6 Input or Timer/Counter3 Clock Input)
PE5	INT5/OC3C (External Interrupt 5 Input or Output Compare and PWM Output C for Timer/Counter3)
PE4	INT4/OC3B (External Interrupt4 Input or Output Compare and PWM Output B for Timer/Counter3)
PE3	AIN1/OC3A (Analog Comparator Negative Input or Output Compare and PWM Output A for Timer/Counter3)
PE2	AIN0/XCK0 (Analog Comparator Positive Input or USART0 external clock input/output)
PE1	PDO[1]/TXD0 (Programming Data Output or USART0 Transmit Pin)
PE0	PDI[1]/RXD0/PCINT8 (Programming Data Input, USART0 Receive Pin or Pin Change Interrupt 8)

Note: 1. Only for ATmega1281/2561. For ATmega640/1280/2560 these functions are placed on MISO/MOSI pins.

13.4.32 PORTL – Port L Data Register

Bit	7	6	5	4	3	2	1	0	
(0x10B)	PORTL7	PORTL6	PORTL5	PORTL4	PORTL3	PORTL2	PORTL1	PORTL0	PORTL
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	

13.4.33 DDRL – Port L Data Direction Register

Bit	7	6	5	4	3	2	1	0	
(0x10A)	DDL7	DDL6	DDL5	DDL4	DDL3	DDL2	DDL1	DDL0	DDRL
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	

13.4.34 PINL – Port L Input Pins Address

Bit	7	6	5	4	3	2	1	0	
(0x109)	PINL5	PINL5	PINL5	PINL4	PINL3	PINGL	PINL1	PINL0	PINL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Table 16-7 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to phase correct PWM mode.

Table 16-7. Compare Output Mode, Phase Correct PWM Mode[1]

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected
0	1	Reserved
1	0	Clear OC0B on Compare Match when up-counting. Set OC0B on Compare Match when down-counting
1	1	Set OC0B on Compare Match when up-counting. Clear OC0B on Compare Match when down-counting

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See "Phase Correct PWM Mode" on page 122 for more details.

- **Bits 3, 2 – Res: Reserved Bits**

These bits are reserved bits and will always read as zero.

- **Bits 1:0 – WGM01:0: Waveform Generation Mode**

Combined with the WGM02 bit found in the TCCR0B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 16-8. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes (see "Modes of Operation" on page 144).

Table 16-8. Waveform Generation Mode Bit Description

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1][2]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	TOP	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Note: 1. MAX = 0xFF
2. BOTTOM = 0x00

The following code examples show how to do an atomic write of the TCNTn Register contents. Writing any of the OCRnA/B/C or ICRn Registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_WriteTCNTn:
  ; Save global interrupt flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Set TCNTn to r17:r16
  out TCNTnH,r17
  out TCNTnL,r16
  ; Restore global interrupt flag
  out SREG,r18
  ret
```

C Code Example[1]

```
void TIM16_WriteTCNTn( unsigned int i )
{
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  __disable_interrupt();
  /* Set TCNTn to i */
  TCNTn = i;
  /* Restore global interrupt flag */
  SREG = sreg;
}
```

Note: 1. See “About Code Examples” on page 10.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNTn.

17.3.1 Reusing the Temporary High Byte Register

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

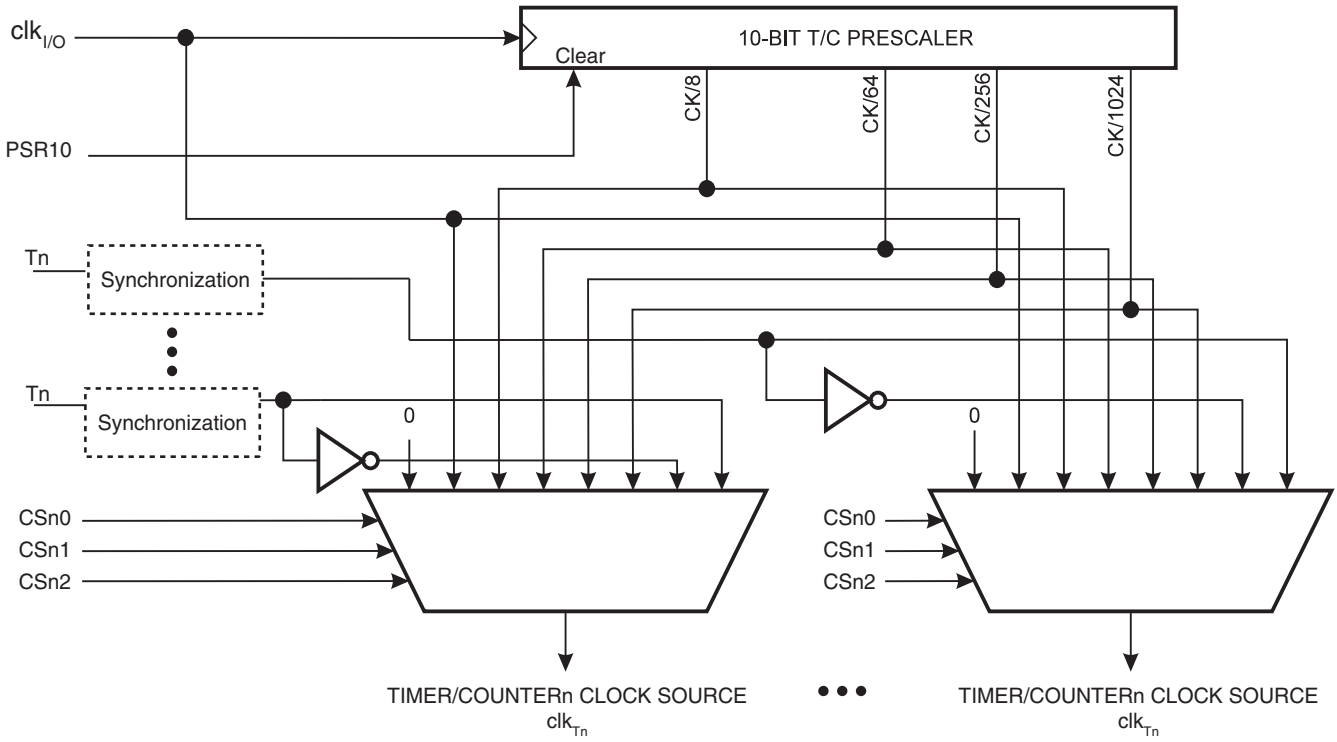
17.4 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the *Clock Select* (CSn2:0) bits located in the *Timer/Counter control Register B* (TCCRnB). For details on clock sources and prescaler, see “Timer/Counter 0, 1, 3, 4, and 5 Prescaler” on page 164.

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ($f_{ExtClk} < f_{clk\_I/O}/2$) given a 50/50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by Oscillator source (crystal, resonator, and capacitors) tolerances, it is recommended that maximum frequency of an external clock source is less than $f_{clk\_I/O}/2.5$.

An external clock source can not be prescaled.

Figure 18-2. Prescaler for synchronous Timer/Counters



When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 21-1. For more details on automatic port overrides, refer to “Alternate Port Functions” on page 72.

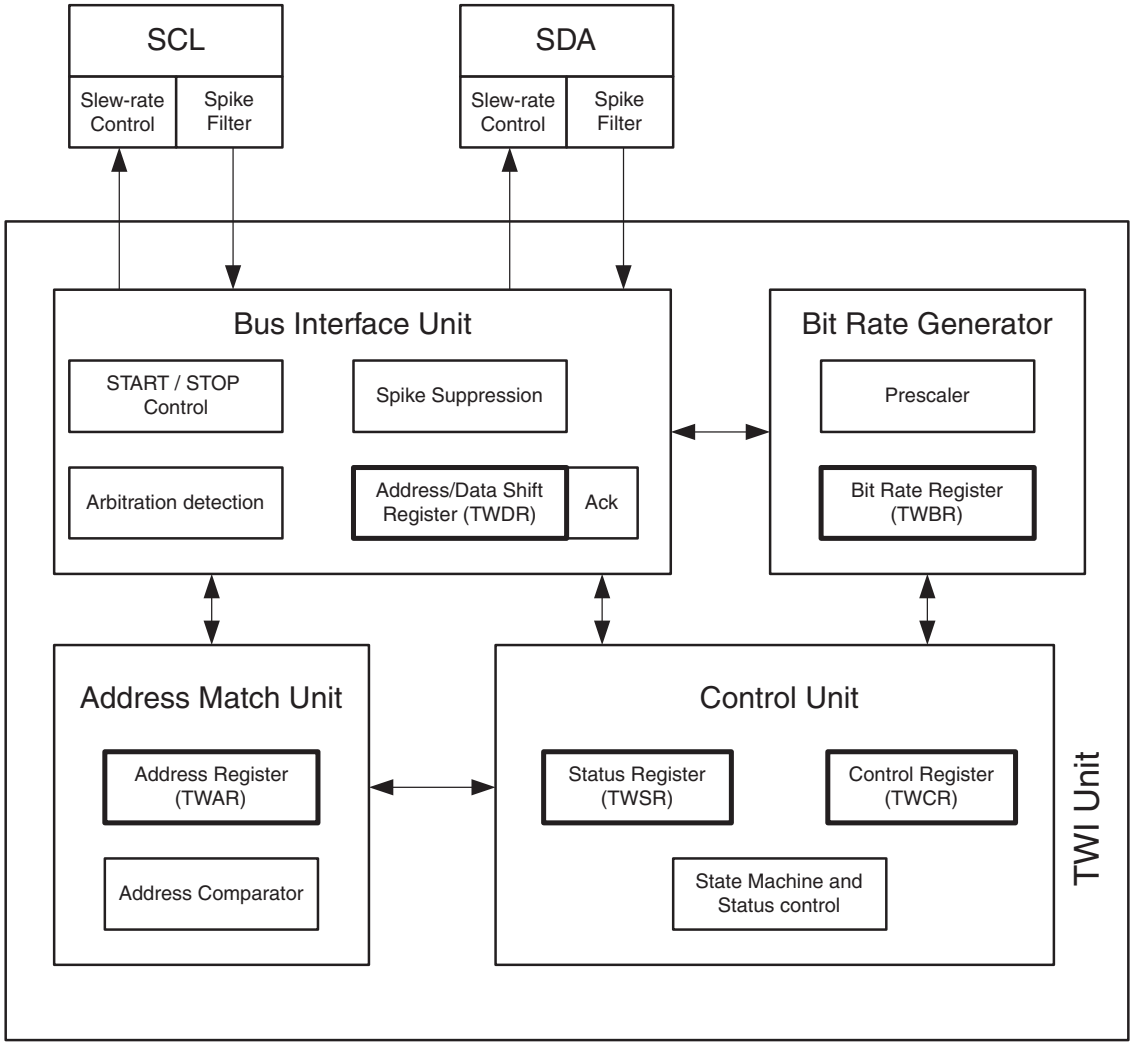
Table 21-1. SPI Pin Overrides[1]

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Note: 1. See “Alternate Functions of Port B” on page 76 for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a Master and how to perform a simple transmission. DDR_SPI in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. For example, if MOSI is placed on pin PB5, replace DD_MOSI with DDB5 and DDR_SPI with DDRB.

Figure 24-9. Overview of the TWI Module



24.5.1 SCL and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

24.5.2 Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the Slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period.

The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

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.

29.6.12 Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. Table 29-6 shows the typical programming time for Flash accesses from the CPU.

Table 29-6. 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

29.6.13 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<<SPMEN)
  call Do_spm

  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  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<<SPMEN)
  call Do_spm
  adiw ZH:ZL, 2
  sbiw loophi:looplo, 2         ;use subi for PAGESIZEB<=256
  brne Wrloop
```

29.7 Register Description

29.7.1 SPMCSR – Store Program Memory Control and Status Register

The Store Program Memory Control and Status Register contains the control bits needed to control the Boot Loader operations.

Bit	7	6	5	4	3	2	1	0	
0x37 (0x57)	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCSR
Read/Write	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPMIE: SPM Interrupt Enable**

When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCSR Register is cleared.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**

When a Self-Programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

- **Bit 5 – SIGRD: Signature Row Read**

If this bit is written to one at the same time as SPMEN, the next LPM instruction within three clock cycles will read a byte from the signature row into the destination register. see “Reading the Signature Row from Software” on page 317 for details. An SPM instruction within four cycles after SIGRD and SPMEN are set will have no effect. This operation is reserved for future use and should not be used.

- **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**

When programming (Page Erase or Page Write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a Page Erase or a Page Write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

- **Bit 3 – BLBSET: Boot Lock Bit Set**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets Boot Lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer are ignored. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

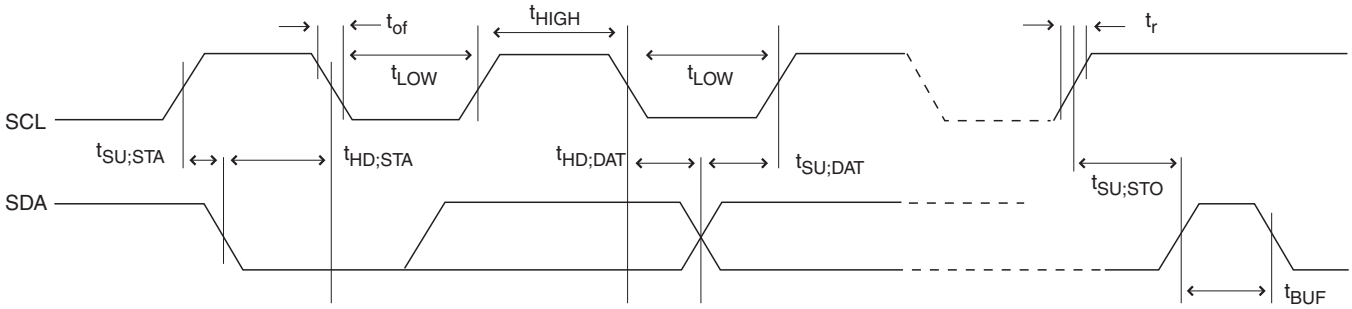
An (E)LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCSR Register, will read either the Lock bits or the Fuse bits (depending on Z0 in the Z-pointer) into the destination register. See “Reading the Fuse and Lock Bits from Software” on page 316 for details.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SPMEN, 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.

6. The actual low period generated by the ATmega640/1280/1281/2560/2561 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 ATmega640/1280/1281/2560/2561 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, ATmega640/1280/1281/2560/2561 devices connected to the bus may communicate at full speed (400kHz) with other ATmega640/1280/1281/2560/2561 devices, as well as any other device with a proper t_{LOW} acceptance margin.

Figure 31-6. 2-wire Serial Bus Timing



31.7 SPI Timing Characteristics

See Figure 31-7 and Figure 31-8 on page 364 for details.

Table 31-8. SPI Timing Parameters

	Description	Mode	Min.	Typ.	Max.	
1	SCK period	Master		See Table 21-5 on page 198		ns
2	SCK high/low	Master		50% duty cycle		
3	Rise/Fall time	Master		3.6		
4	Setup	Master		10		
5	Hold	Master		10		
6	Out to SCK	Master		0.5 • t_{sck}		
7	SCK to out	Master		10		
8	SCK to out high	Master		10		
9	$\overline{SS}$ low to out	Slave		15		
10	SCK period	Slave	4 • t_{ck}			
11	SCK high/low(1)	Slave	2 • t_{ck}			
12	Rise/Fall time	Slave			1600	
13	Setup	Slave	10			
14	Hold	Slave	t_{ck}			
15	SCK to out	Slave		15		
16	SCK to $\overline{SS}$ high	Slave	20			
17	$\overline{SS}$ high to tri-state	Slave		10		
18	$\overline{SS}$ low to SCK	Slave	20			

Note: 1. In SPI Programming mode the minimum SCK high/low period is:
- 2 t_{CLCL} for f_{CK} < 12MHz
- 3 t_{CLCL} for f_{CK} > 12MHz

Figure 32-8. Idle Supply Current vs. V_{CC} (Internal RC Oscillator, 8MHz)

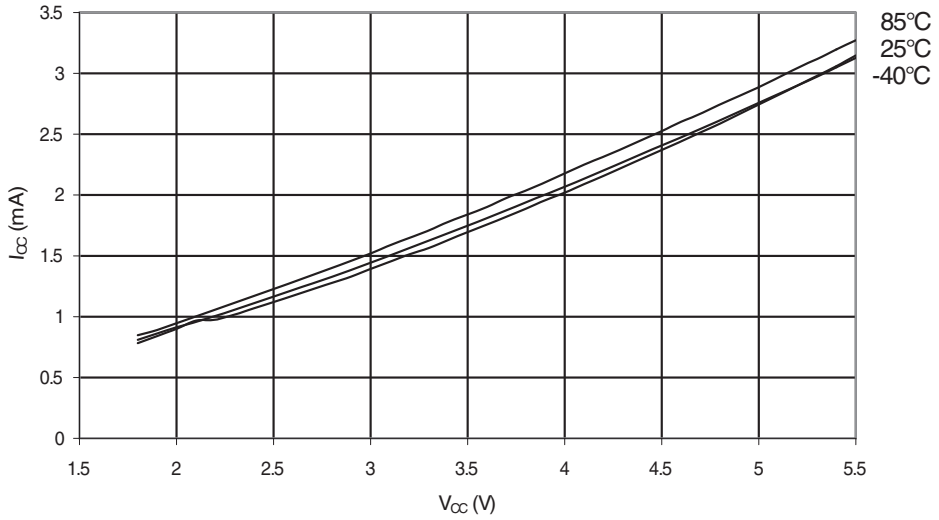
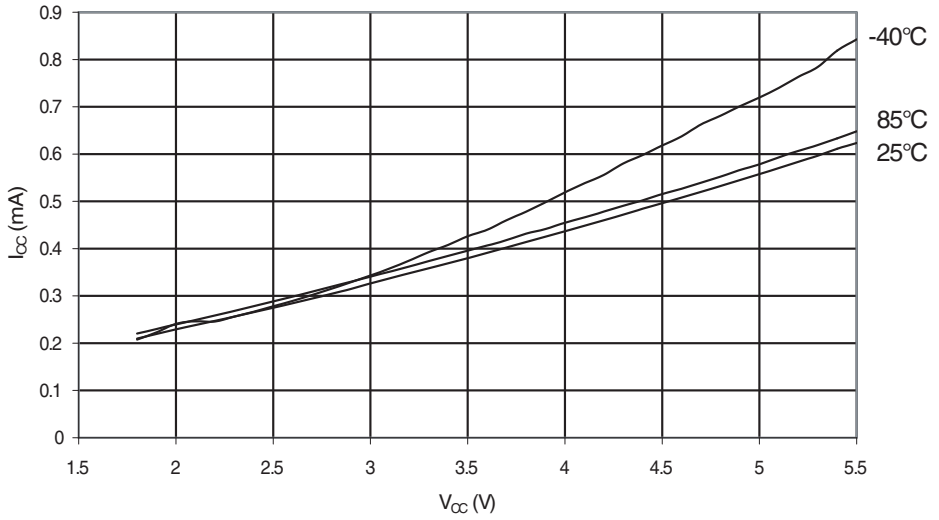


Figure 32-9. Idle Supply Current vs. V_{CC} (Internal RC Oscillator, 1MHz)



38. Datasheet Revision History

Note that the referring page numbers in this section are referring to this document. The referring revisions in this section are referring to the document revision.

38.1 Rev. 2549Q-02/2014

1. Updated the “Reset Sources” on page 57. Brown-out Reset: The MCU is reset when the supply voltage AVcc is below the Brown-out Reset threshold (VBOT) and the Brown-out Detector is enabled.
2. Updated the Figure 12-1 on page 58. Power-on reset is now connected to AVcc and not to Vcc.
3. Updated the content in “Brown-out Detection” on page 59. Replaced Vcc by AVcc throughout the section.
4. Updated the Figure 12-5 on page 60. Replaced Vcc by AVcc.
5. Updated “External Interrupts” on page 109. Removed the text “Note that recognition of falling or rising edge.....”.
6. Updated the description of “PCMSK1 – Pin Change Mask Register 1” on page 113. The description mentions "PCIE1 bit in EIMSK". This has been changed to “PCIE1 bit in PCICR”.
7. Updated “Ordering Information” in “ATmega2561” on page 411.
8. Removed Errata “Inaccurate ADC conversion in differential mode with 200× gain” from “ATmega1280 rev. B” on page 416 and from “ATmega1281 rev. B” on page 417
9. Updated “Errata” in “ATmega2560 rev. F” on page 417 and in “ATmega2561 rev. F” on page 419.
10. Updated the datasheet with new Atmel brand (new logo and addresses).

38.2 Rev. 2549P-10/2012

1. Replaced drawing of “64M2” on page 415.
2. Former page 439 has been deleted as the content of this page did not belong there (same page as the last page).
3. Some small correction made in the setup.

38.3 Rev. 2549O-05/2012

1. The datasheet changed status from Preliminary to Complete. Removed “Preliminary” from the front page.
2. Replaced Figure 10-3 on page 44 by a new one.
3. Updated the last page to include the new address for Atmel Japan site.

38.4 Rev. 2549N-05/2011

1. Added Atmel QTouch Library Support and QTouch Sensing Capablity Features.
2. Updated Cross-reference in “Bit 5, 2:0 - WDP3:0: Watchdog Timer Prescaler 3, 2, 1 and 0” on page 65.
3. Updated Assembly codes in section “USART Initialization” on page 205.
4. Added “Standard Power-On Reset” on page 360.