Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	CANbus, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, Temp Sensor, WDT
Number of I/O	-
Program Memory Size	64KB (32K x 16)
Program Memory Type	FLASH
EEPROM Size	2K x 8
RAM Size	4K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 11x10b; D/A 1x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 150°C (TA)
Mounting Type	Surface Mount
Package / Case	32-TQFP
Supplier Device Package	32-TQFP (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega64c1-15ad

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- 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 (store program memory) 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 is 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 Atmel ATmega16/32/64/M1/C1 has extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

3.3 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” section for a detailed description.

3.4 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 Reference. 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.

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
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 – I: Global Interrupt Enable**

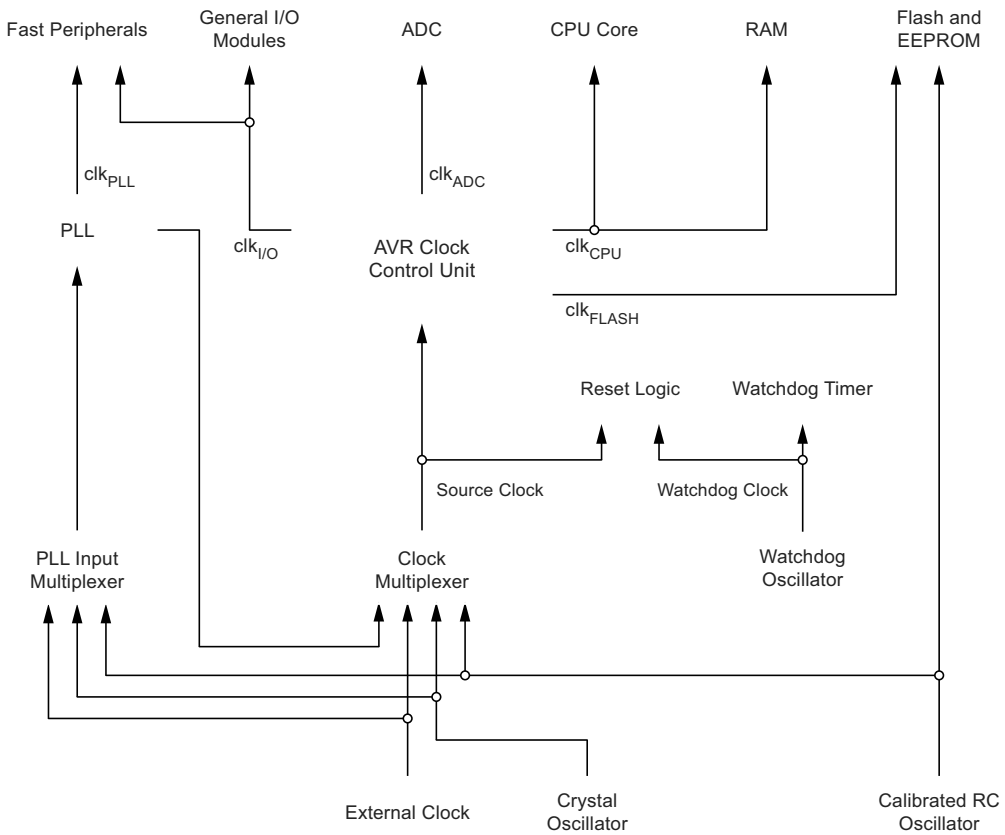
The global interrupt enable bit must be set to enabled the interrupts. The individual interrupt enable control is then performed in separate control registers. If the global interrupt enable register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

5. System Clock

5.1 Clock Systems and their Distribution

Figure 5-1 presents the principal clock systems in the AVR® and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to unused modules can be halted by using different sleep modes, as described in Section 6. “Power Management and Sleep Modes” on page 34. The clock systems are detailed below.

Figure 5-1. Clock Distribution



5.1.1 CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

5.1.2 I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, UART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted.

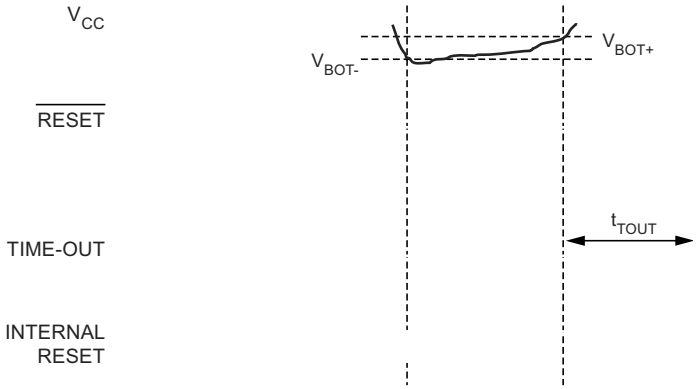
5.1.3 Flash Clock – clk_{FLASH}

The Flash clock controls operation of the Flash interface. The flash clock is usually active simultaneously with the CPU clock.

When the BOD is enabled, and V_{CC} decreases to a value below the trigger level (V_{BOT-} in Figure 7-5 on page 41), the brown-out reset is immediately activated. When V_{CC} increases above the trigger level (V_{BOT+} in Figure 7-5 on page 41), the delay counter starts the MCU after the Time-out period t_{TOUT} has expired.

The BOD circuit will only detect a drop in V_{CC} if the voltage stays below the trigger level for longer than t_{BOD} given in Table 7-3.

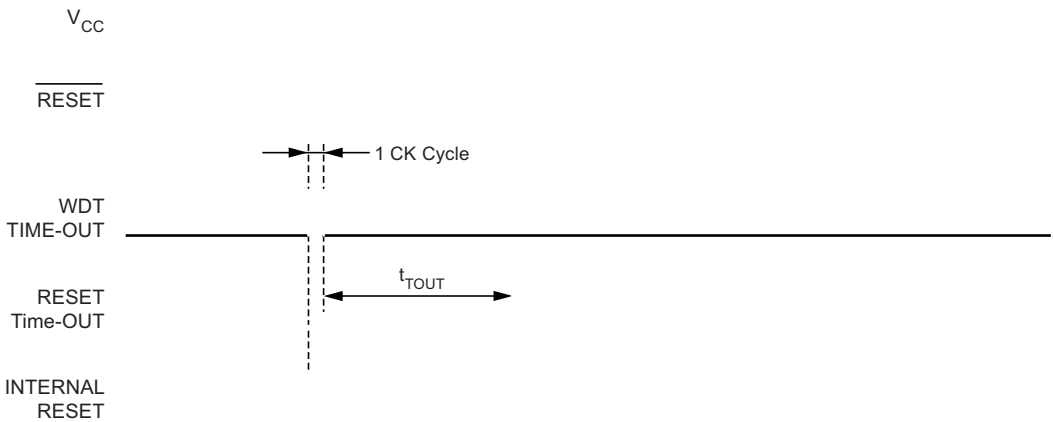
Figure 7-5. Brown-out Reset during Operation



7.2.4 Watchdog Reset

When the watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the time-out period t_{TOUT}. Refer to Section 7.4 “Watchdog Timer” on page 43 for details on operation of the watchdog timer.

Figure 7-6. Watchdog Reset during Operation



13.7.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COMnx1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COMnx1:0 = 0 tells the waveform generator that no action on the OCnx register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 13-1 on page 110. For fast PWM mode refer to Table 13-2 on page 110, and for phase correct and phase and frequency correct PWM refer to Table 13-3 on page 111.

A change of the COMnx1:0 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 FOCnx strobe bits.

13.8 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* (WGMn3:0) and *Compare Output mode* (COMnx1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COMnx1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COMnx1:0 bits control whether the output should be set, cleared or toggle at a compare match (see Section 13.7 “Compare Match Output Unit” on page 101). For detailed timing information refer to Section 13.9 “Timer/Counter Timing Diagrams” on page 108.

13.8.1 Normal Mode

The simplest mode of operation is the *Normal mode* (WGMn3:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the *Timer/Counter Overflow Flag* (TOVn) will be set in the same timer clock cycle as the TCNTn becomes zero. The TOVn flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOVn flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

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

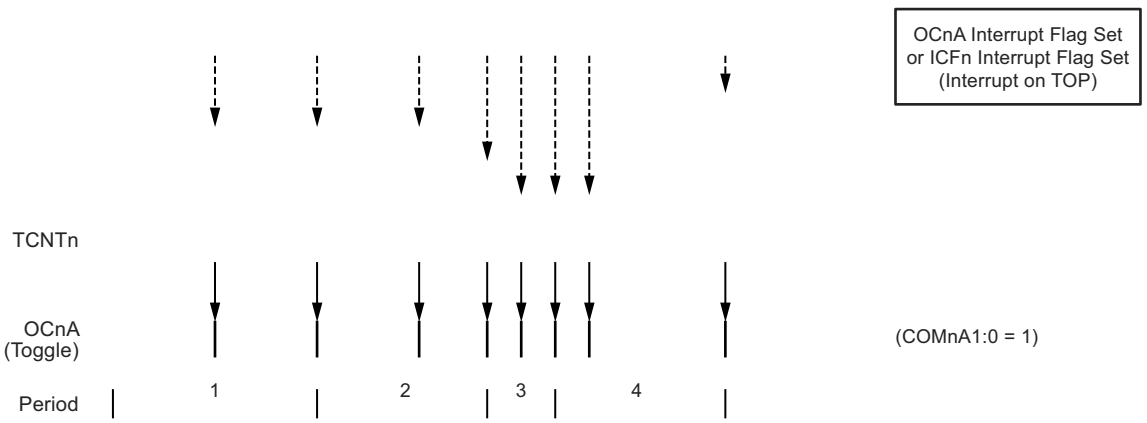
The output compare units can be used to generate interrupts at some given time. Using the output compare to generate waveforms in normal mode is not recommended, since this will occupy too much of the CPU time.

13.8.2 Clear Timer on Compare Match (CTC) Mode

In *Clear Timer on Compare* or CTC mode (WGMn3:0 = 4 or 12), the OCRnA or ICRn register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNTn) matches either the OCRnA (WGMn3:0 = 4) or the ICRn (WGMn3:0 = 12). The OCRnA or ICRn define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 13-6. The counter value (TCNTn) increases until a compare match occurs with either OCRnA or ICRn, and then counter (TCNTn) is cleared.

Figure 13-6. CTC Mode, Timing Diagram



The extreme values for the OCRnx register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

13.9 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{Tn}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set, and when the OCRnx register is updated with the OCRnx buffer value (only for modes utilizing double buffering). Figure 13-10 shows a timing diagram for the setting of OCFnx.

Figure 13-10.Timer/Counter Timing Diagram, Setting of OCFnx, no Prescaling

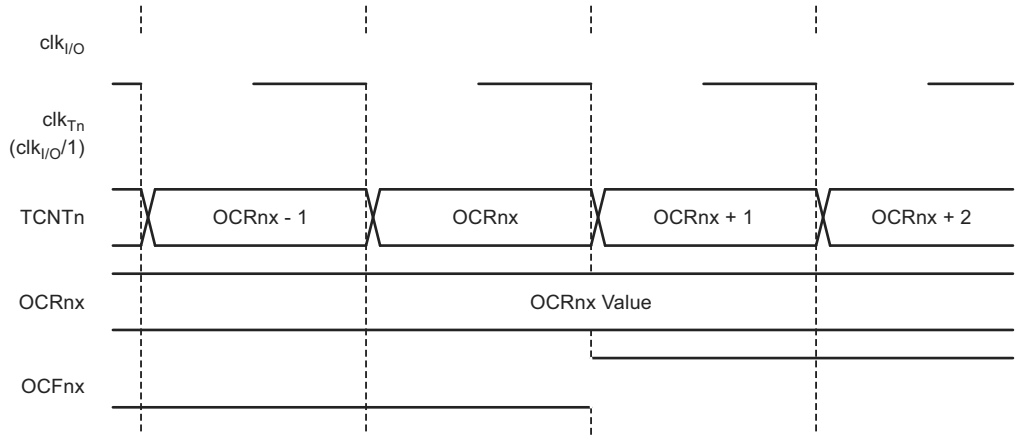


Figure 13-11 shows the same timing data, but with the prescaler enabled.

Figure 13-11.Timer/Counter Timing Diagram, Setting of OCFnx, with Prescaler ($f_{clk\_I/O}/8$)

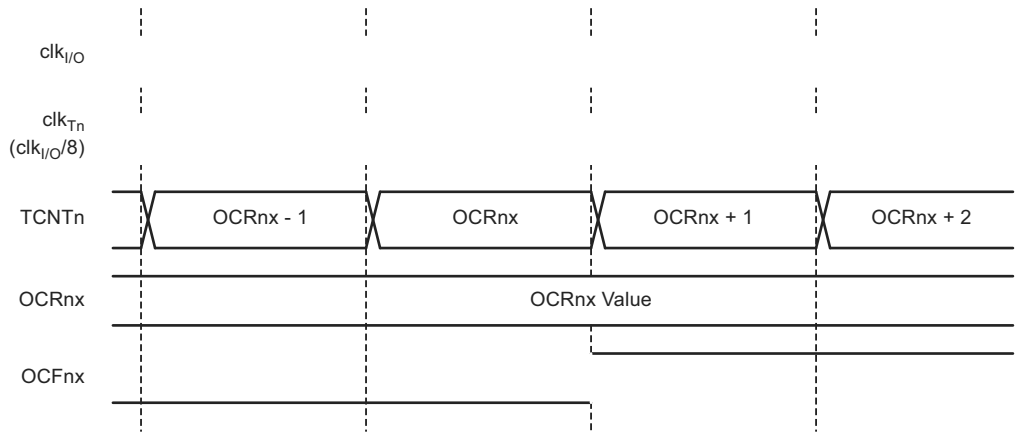


Figure 14-3. Cycle Presentation in Centered Mode

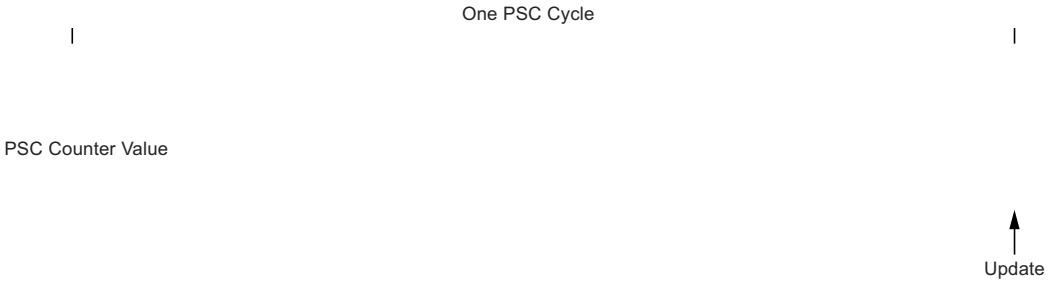


Figure 14-2 on page 118 and Figure 14-3 graphically illustrate the values held in the PSC counter. Centered Mode is like one ramp mode which counts down and then up.

Notice that the update of the waveform generator registers is done regardless of ramp mode at the end of the PSC cycle.

14.5.3 Operation Mode Descriptions

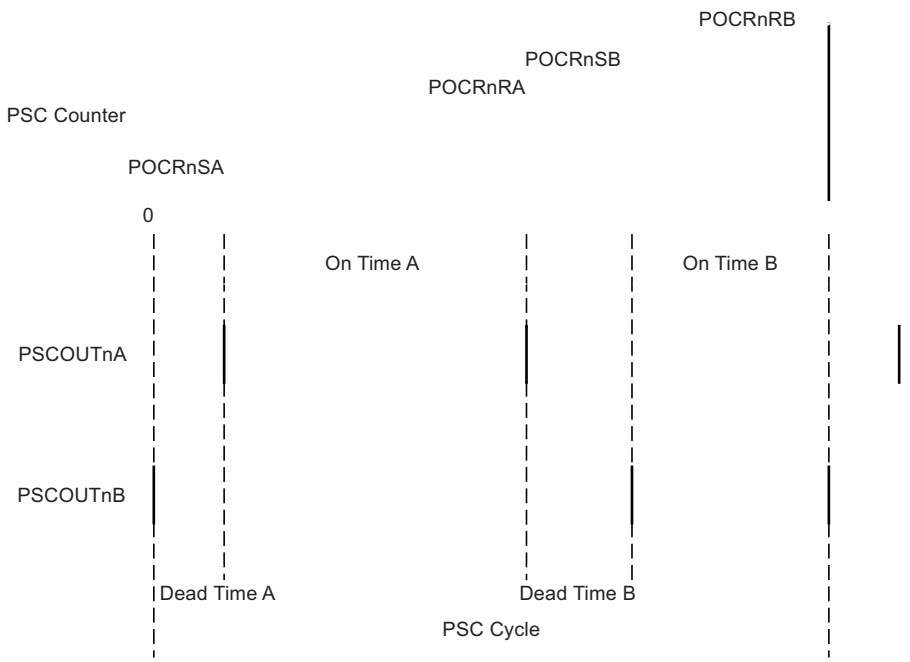
Waveforms and duration of output signals are determined by parameters held in the registers (POCRnSA, POCRnRA, POCRnSB, POCR_RB) and by the running mode. Two modes are possible:

- One ramp mode: In this mode, all the 3 PSCOUTnB outputs are edge-aligned and the 3 PSCOUTnA can be also edge-aligned when setting the same values in the dedicated registers.
  In this mode, the PWM frequency is twice the center aligned mode PWM frequency.
- Center aligned mode: In this mode, all the 6 PSC outputs are aligned at the center of the period. Except when using the same duty cycles on the 3 modules, the edges of the outputs are not aligned. So the PSC outputs do not commute at the same time, thus the system which is driven by these outputs will generate less commutation noise.
  In this mode, the PWM frequency is twice slower than in one ramp mode.

14.5.3.1 One Ramp Mode (Edge-Aligned)

The following figure shows the resultant outputs PSCOUTnA and PSCOUTnB operating in one ramp mode over a PSC cycle.

Figure 14-4. PSCOUTnA and PSCOUTnB Basic Waveforms in One Ramp Mode



14.16.10 PSC Interrupt Mask Register – PIM

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	PEVE2	PEVE1	PEVE0	PEOPE	PIM
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 – not use**

not use.

- **Bit 3 – PEVE2: PSC External Event 2 Interrupt Enable**

When this bit is set, an external event which can generates a a fault on module 2 generates also an interrupt.

- **Bit 2 – PEVE1: PSC External Event 1 Interrupt Enable**

When this bit is set, an external event which can generates a fault on module 1 generates also an interrupt.

- **Bit 1 – PEVE0: PSC External Event 0 Interrupt Enable**

When this bit is set, an external event which can generates a fault on module 0 generates also an interrupt.

- **Bit 0 – PEOPE: PSC End Of Cycle Interrupt Enable**

When this bit is set, an interrupt is generated when PSC reaches the end of the whole cycle.

14.16.11 PSC Interrupt Flag Register – PIFR

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	PEV2	PEV1	PEV0	PEOP	PIFR
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 – not use**

not use.

- **Bit 3 – PEV2: PSC External Event 2 Interrupt**

This bit is set by hardware when an external event which can generates a fault on module 2 occurs.

Must be cleared by software by writing a one to its location.

This bit can be read even if the corresponding interrupt is not enabled (PEVE2 bit = 0).

- **Bit 2 – PEV1: PSC External Event 1 Interrupt**

This bit is set by hardware when an external event which can generates a fault on module 1 occurs.

Must be cleared by software by writing a one to its location.

This bit can be read even if the corresponding interrupt is not enabled (PEVE1 bit = 0).

- **Bit 1 – PEV0: PSC External Event 0 Interrupt**

This bit is set by hardware when an external event which can generates a fault on module 0 occurs.

Must be cleared by software by writing a one to its location.

This bit can be read even if the corresponding interrupt is not enabled (PEVE0 bit = 0).

- **Bit 0 – PEOP: PSC End Of Cycle Interrupt**

This bit is set by hardware when an “end of PSC cycle” occurs.

Must be cleared by software by writing a one to its location.

This bit can be read even if the corresponding interrupt is not enabled (PEOPE bit = 0).

16.6.4 Stamping Message

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

16.7 Error Management

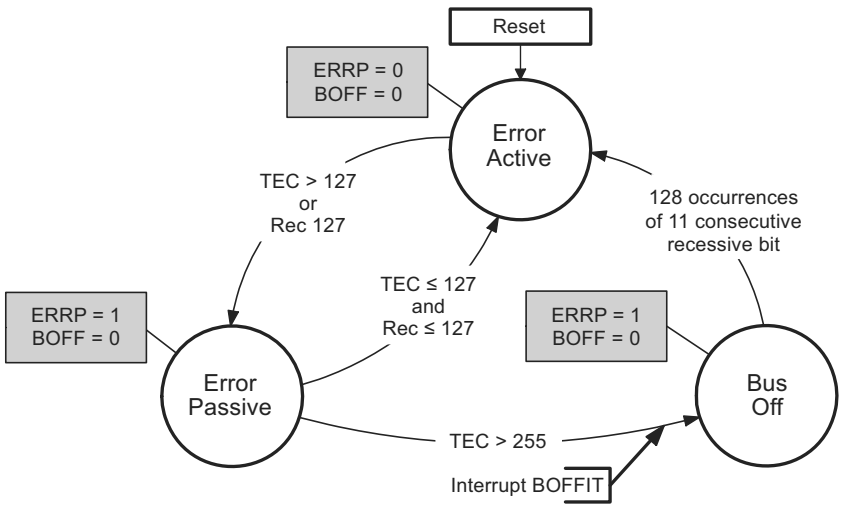
16.7.1 Fault Confinement

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

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

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

Figure 16-12. Line Error Mode



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

- **Bit 6:5 – SJW1:0: Re-Synchronization Jump Width**

To compensate for phase shifts between clock oscillators of different bus controllers, the controller must re-synchronize on any relevant signal edge of the current transmission. The synchronization jump width defines the maximum number of clock cycles. A bit period may be shortened or lengthened by a re-synchronization.

$$Tsjw = Tscl \times (SJW[1:0] + 1)$$

- **Bit 4 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, it must be written to zero when CANBT2 is written.

- **Bit 3:1 – PRS2:0: Propagation Time Segment**

This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal propagation time on the bus line, the input comparator delay and the output driver delay.

$$Tprs = Tscl \times (PRS[2:0] + 1)$$

- **Bit 0 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, it must be written to zero when CANBT2 is written.

16.10.10 CAN Bit Timing Register 3 - CANBT3

Bit	7	6	5	4	3	2	1	0	
	-	**PHS22**	**PHS21**	**PHS20**	**PHS12**	**PHS11**	**PHS10**	**SMP**	**CANBT3**
Read/Write	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	-	0	0	0	0	0	0	0	

- **Bit 7– Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, it must be written to zero when CANBT3 is written.

- **Bit 6:4 – PHS22:0: Phase Segment 2**

This phase is used to compensate for phase edge errors. This segment may be shortened by the re-synchronization jump width. PHS2[2..0] shall be ≥1 and ≤PHS1[2..0] (c.f. Section 16.2.3 “CAN Bit Timing” on page 143 and Section 16.4.3 “Baud Rate” on page 148).

Tphs2 = Tscl × (PHS2[2:0] + 1)

- **Bit 3:1 – PHS12:0: Phase Segment 1**

This phase is used to compensate for phase edge errors. This segment may be lengthened by the re-synchronization jump width.

Tphs1 = Tscl × (PHS1[2:0] + 1)

- **Bit 0 – SMP: Sample Point(s)**

This option allows to filter possible noise on TxCAN input pin.

- 0 - the sampling will occur once at the user configured sampling point - **SP**.
- 1 - with three-point sampling configuration the first sampling will occur two $Tclk_{IO}$ clocks before the user configured sampling point - **SP**, again at one $Tclk_{IO}$ clock before **SP** and finally at **SP**. Then the bit level will be determined by a majority vote of the three samples.

‘SMP=1’ configuration is not compatible with ‘BRP[5:0]=0’ because TQ = $Tclk_{IO}$.
If BRP = 0, SMP must be cleared.

16.10.15 CAN Receive Error Counter Register - CANREC

Bit	7	6	5	4	3	2	1	0	
	REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0	CANREC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – REC7:0: Receive Error Count**

CAN receive error counter range 0 to 255.

16.10.16 CAN Highest Priority MOb Register - CANHPMOB

Bit	7	6	5	4	3	2	1	0	
	HPMOB3	HPMOB2	HPMOB1	HPMOB0	CGP3	CGP2	CGP1	CGP0	CANHPMOB
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	0	0	0	0	

- **Bit 7:4 – HPMOB3:0: Highest Priority MOb Number**

MOb having the highest priority in CANSIT registers.
If CANSIT = 0 (no MOb), the return value is 0xF.

Note: Do not confuse “MOb priority” and “Message ID priority”- <Helv>See “Message Objects” on page 149.

- **Bit 3:0 – CGP3:0: CAN General Purpose Bits**

These bits can be pre-programmed to match with the wanted configuration of the CANPAGE register (i.e., $\overline{\text{AINC}}$ and INDX2:0 setting).

16.10.17 CAN Page MOb Register - CANPAGE

Bit	7	6	5	4	3	2	1	0	
	MOBNB3	MOBNB2	MOBNB1	MOBNB0	AINC	INDX2	INDX1	INDX0	CANPAGE
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 7:4 – MOBNB3:0: MOb Number**

Selection of the MOb number, the available numbers are from 0 to 5.

Note: MOBNB3 always must be written to zero for compatibility with all AVR CAN devices.

- **Bit 3 – $\overline{\text{AINC}}$: Auto Increment of the FIFO CAN Data Buffer Index (Active Low)**
  - 0 - auto increment of the index (default value).
  - 1- no auto increment of the index.

- **Bit 2:0 – INDX2:0: FIFO CAN Data Buffer Index**

Byte location of the CAN data byte into the FIFO for the defined MOb.

Table 18-7. ADC Auto Trigger Source Selection

ADTS3	ADTS2	ADTS1	ADTS0	Description
0	0	0	0	Free running mode
0	0	0	1	External interrupt request 0
0	0	1	0	Timer/Counter0 compare match
0	0	1	1	Timer/Counter0 overflow
0	1	0	0	Timer/Counter1 compare match B
0	1	0	1	Timer/Counter1 overflow
0	1	1	0	Timer/Counter1 capture event
0	1	1	1	PSC Module 0 synchronization signal
1	0	0	0	PSC Module 1 synchronization signal
1	0	0	1	PSC Module 2 synchronization signal
1	0	1	0	Analog comparator 0
1	0	1	1	Analog comparator 1
1	1	0	0	Analog comparator 2
1	1	0	1	Analog comparator 3
1	1	1	0	Reserved
1	1	1	1	Reserved

18.9.4 ADC Result Data Registers – ADCH and ADCL

When an ADC conversion is complete, the conversion results are stored in these two result data registers.

When the ADCL register is read, the two ADC result data registers can't be updated until the ADCH register has also been read.
Consequently, in 10-bit configuration, the ADCL register must be read first before the ADCH.
Nevertheless, to work easily with only 8-bit precision, there is the possibility to left adjust the result thanks to the ADLAR bit in the ADCSRA register. Like this, it is sufficient to only read ADCH to have the conversion result.

18.9.4.1 ADLAR = 0

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	-	-	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

18.9.4.2 ADLAR = 1

Bit	7	6	5	4	3	2	1	0	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	-	-	-	-	-	-	ADCL
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

20.3 Use of ADC Amplifiers

Thanks to AMPCMP0 configuration bit, comparator 0 positive input can be connected to amplifier O output. In that case, the clock of comparator 0 is twice the amplifier 0 clock. See Section 18.11.1 “Amplifier 0 control and status register – AMP0CSR” on page 218.

Thanks to AMPCMP1 configuration bit, comparator 1 positive input can be connected to amplifier 1 output. In that case, the clock of comparator 1 is twice the amplifier 1 clock. See Section 18.11.2 “Amplifier 1 Control and Status Register – AMP1CSR” on page 219.

Thanks to AMPCMP2 configuration bit, comparator 2 positive input can be connected to amplifier 2 output. In that case, the clock of comparator 2 is twice the amplifier 2 clock. See Section 18.11.2 “Amplifier 1 Control and Status Register – AMP1CSR” on page 219.

20.4 Analog Comparator Register Description

Each analog comparator has its own control register.

A dedicated register has been designed to consign the outputs and the flags of the 4 analog comparators.

20.4.1 Analog Comparator 0 Control Register – AC0CON

Bit	7	6	5	4	3	2	1	0	
	AC0EN	**AC0IE**	**AC0IS1**	**AC0IS0**	**ACCKSEL**	**AC0M2**	**AC0M1**	**AC0M0**	**AC0CON**
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– AC0EN: analog comparator 0 Enable Bit**

Set this bit to enable the analog comparator 0.
Clear this bit to disable the analog comparator 0.

- **Bit 6– AC0IE: analog comparator 0 Interrupt Enable bit**

Set this bit to enable the analog comparator 0 interrupt.
Clear this bit to disable the analog comparator 0 interrupt.

- **Bit 5, 4– AC0IS1, AC0IS0: analog comparator 0 Interrupt Select bit**

These 2 bits determine the sensitivity of the interrupt trigger.
The different setting are shown in Table 18-7.

Table 20-1. Interrupt Sensitivity Selection

AC0IS1	AC0IS0	Description
0	0	Comparator interrupt on output toggle
0	1	Reserved
1	0	Comparator interrupt on output falling edge
1	1	Comparator interrupt on output rising edge

- **Bit 3 – ACCKSEL: Analog Comparator Clock Select**

Set this bit to use the 16MHz PLL output as comparator clock. Clear this bit to use the CLK_{IO} as comparator clock.

- **Bit 2, 1, 0– AC0M2, AC0M1, AC0M0: Analog Comparator 0 Multiplexer Register**

These 3 bits determine the input of the negative input of the analog comparator.
The different setting are shown in Table 20-2 on page 228.

23.4 Software Break Points

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

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

23.5 Limitations of debugWIRE

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

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

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

23.6 debugWIRE Related Register in I/O Memory

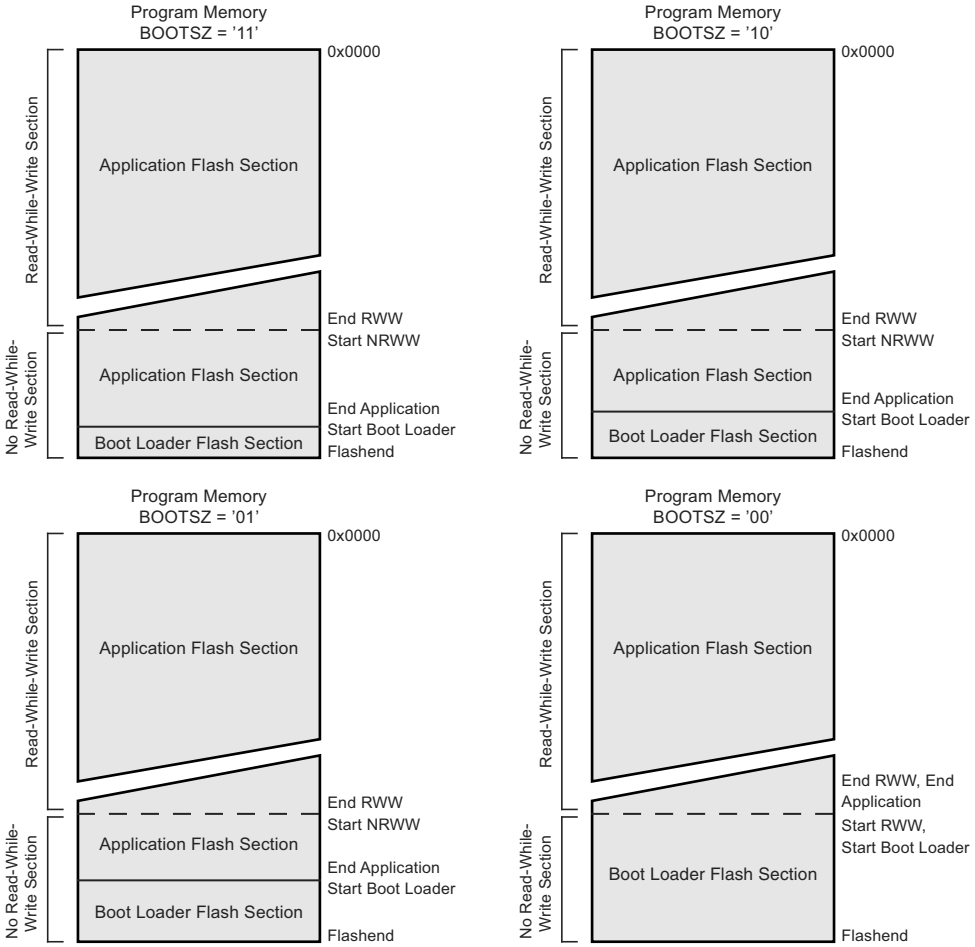
The following section describes the registers used with the debugWire.

23.6.1 debugWire Data Register – DWDR

Bit	7	6	5	4	3	2	1	0	
	DWDR[7:0]								DWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

Figure 24-2. Memory Sections



Note: 1. The parameters in the figure above are given in Table 24-7 on page 251.

24.4 Boot Loader Lock Bits

If no boot loader capability is needed, the entire flash is available for application code. 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.

The user can select:

- To protect the entire flash from a software update by the MCU.
- To protect only the boot loader flash section from a software update by the MCU.
- To protect only the application flash section from a software update by the MCU.
- Allow software update in the entire flash.

See Table 24-2 and Table 24-3 on page 244 for further details. The boot lock bits can be set in software and in serial or parallel programming mode, but they can be cleared by a chip erase command only. The general write lock (lock bit mode 2) does not control the programming of the flash memory by SPM instruction. Similarly, the general Read/Write lock (lock bit mode 1) does not control reading nor writing by LPM/SPM, if it is attempted.

Table 24-2. Boot Lock Bit0 Protection Modes (Application Section)(1)

BLB0 Mode	BLB02	BLB01	Protection
1	1	1	No restrictions for SPM or LPM accessing the application section.
2	1	0	SPM is not allowed to write to the application section.
3	0	0	SPM is not allowed to write to the application section, and LPM executing from the boot loader section is not allowed to read from the application section. If interrupt vectors are placed in the boot loader section, interrupts are disabled while executing from the application section.
4	0	1	LPM executing from the boot loader section is not allowed to read from the application section. If interrupt vectors are placed in the boot loader section, interrupts are disabled while executing from the application section.

Note: 1. “1” means unprogrammed, “0” means programmed.

Table 24-3. Boot Lock Bit1 Protection Modes (Boot Loader Section)(1)

BLB1 Mode	BLB12	BLB11	Protection
1	1	1	No restrictions for SPM or LPM accessing the boot loader section.
2	1	0	SPM is not allowed to write to the boot loader section.
3	0	0	SPM is not allowed to write to the boot loader section, and LPM executing from the application section is not allowed to read from the boot loader section. If Interrupt vectors are placed in the application section, interrupts are disabled while executing from the boot loader section.
4	0	1	LPM executing from the application section is not allowed to read from the boot loader section. If interrupt vectors are placed in the application section, interrupts are disabled while executing from the boot loader section.

Note: “1” means unprogrammed, “0” means programmed

24.5 Entering the Boot Loader Program

Entering the boot loader takes place by a jump or call from the application program. This may be initiated by a trigger such as a command received via UART, or SPI interface. Alternatively, the boot reset fuse can be programmed so that the reset vector is pointing to the boot flash start address after a reset. In this case, the boot loader is started after a reset. After the application code is loaded, the program can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the boot reset fuse is programmed, the reset vector will always point to the boot loader reset and the fuse can only be changed through the serial or parallel programming interface.

Table 24-4. Boot Reset Fuse(1)

BOOTRST	Reset Address
1	Reset vector = Application reset (address 0x0000)
0	Reset vector = Boot loader Reset (see Table 24-7 on page 251)

Note: 1. “1” means unprogrammed, “0” means programmed

24.5.1 Store Program Memory Control and Status Register – SPMCSR

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	
	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCSR
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

25.4 Signature Bytes

All Atmel® microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space.

25.4.1 Signature Bytes

For the **ATmega16M1** the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x94 (indicates 16kB flash memory).
3. 0x002: 0x84 (indicates ATmega16M1 device when 0x001 is 0x94).

For the **ATmega32M1** the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x95 (indicates 32kB flash memory).
3. 0x002: 0x84 (indicates ATmega32M1 device when 0x001 is 0x95).

For the **ATmega64M1** the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x96 (indicates 64kB flash memory).
3. 0x002: 0x84 (indicates ATmega64M1 device when 0x001 is 0x96).

For the **ATmega32C1** the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x95 (indicates 32kB flash memory).
3. 0x002: 0x86 (indicates ATmega32C1 device when 0x001 is 0x95).

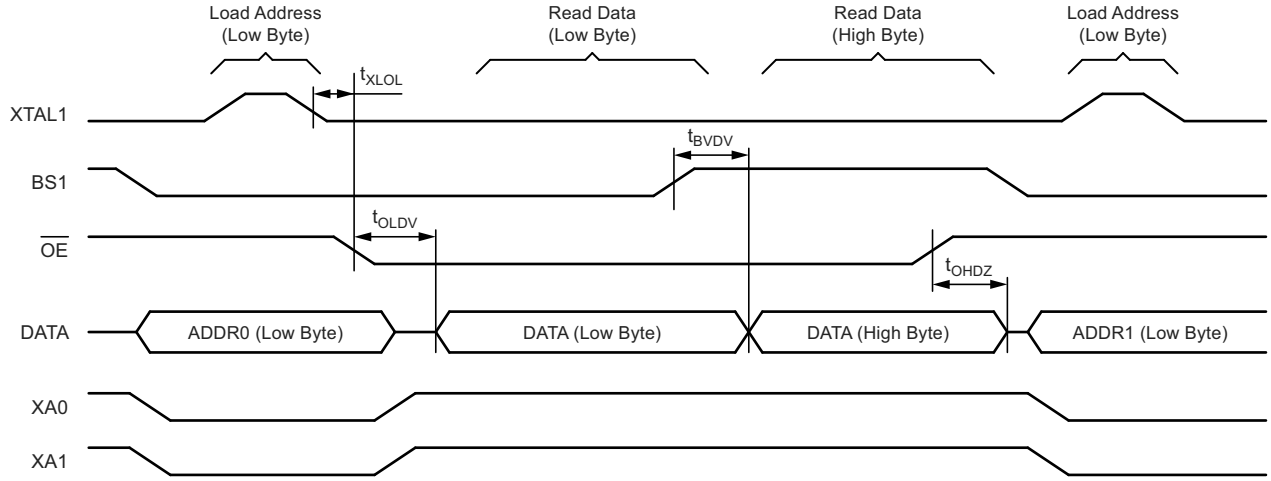
For the **ATmega64C1** the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x96 (indicates 32kB flash memory).
3. 0x002: 0x86 (indicates ATmega64C1 device when 0x001 is 0x96).

25.5 Calibration Byte

The ATmega16/32/64/M1/C1 has a byte calibration value for the internal RC oscillator. This byte resides in the high byte of address 0x000 in the signature address space. during reset, this byte is automatically written into the OSCCAL register to ensure correct frequency of the calibrated RC oscillator.

Figure 25-9. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 25-7 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

Table 25-15. Parallel Programming Characteristics, V_{CC} = 5V ±10%

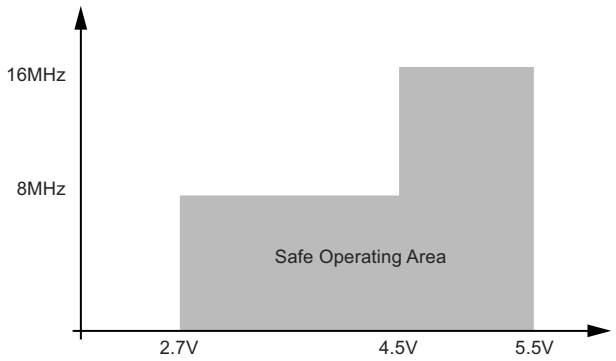
Parameter	Symbol	Min	Typ	Max	Unit
Programming enable voltage	V_{PP}	11.5		12.5	V
Programming enable current	I_{PP}			250	µA
Data and control valid before XTAL1 high	t_{DVXH}	67			ns
XTAL1 low to XTAL1 high	t_{XLXH}	200			ns
XTAL1 pulse width high	t_{XHXL}	150			ns
Data and control hold after XTAL1 low	t_{XLDX}	67			ns
XTAL1 low to $\overline{WR}$ low	t_{XLWL}	0			ns
XTAL1 low to PAGEL high	t_{XLPH}	0			ns
PAGEL low to XTAL1 high	t_{PLXH}	150			ns
BS1 valid before PAGEL high	t_{BVPH}	67			ns
PAGEL pulse width high	t_{PHPL}	150			ns
BS1 hold after PAGEL low	t_{PLBX}	67			ns
BS2/1 hold after $\overline{WR}$ low	t_{WLBX}	67			ns
PAGEL low to $\overline{WR}$ low	t_{PLWL}	67			ns
BS1 valid to $\overline{WR}$ low	t_{BVWL}	67			ns
$\overline{WR}$ pulse width low	t_{WLWH}	150			ns
$\overline{WR}$ low to RDY/$\overline{BSY}$ low	t_{WLRL}	0		1	µs
$\overline{WR}$ low to RDY/$\overline{BSY}$ high(1)	t_{WLRH}	3.7		4.5	ms
$\overline{WR}$ low to RDY/$\overline{BSY}$ high for chip erase(2)	t_{WLRH_CE}	7.5		9	ms
XTAL1 low to $\overline{OE}$ low	t_{XLOL}	0			ns
BS1 valid to DATA valid	t_{BVDV}	0		250	ns
$\overline{OE}$ low to DATA valid	t_{OLDV}			250	ns
$\overline{OE}$ high to DATA tri-stated	t_{OHDZ}			250	ns

Notes: 1. t_{WLRH} is valid for the write flash, write EEPROM, write fuse bits and write lock bits commands.
2. t_{WLRH_CE} is valid for the chip erase command.

26.5 Maximum Speed versus V_{CC}

Maximum frequency is depending on $V_{CC.}$ As shown in Figure 26-2, the maximum frequency equals 8MHz when V_{CC} is between 2.7V and 4.5V and equals 16MHz when V_{CC} is between 4.5V and 5.5V.

Figure 26-2. Maximum Frequency versus V_{CC}, ATmega16/32/64/M1/C1



26.6 PLL Characteristics

Table 26-3. PLL Characteristics - V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Parameter	Symbol	Min.	Typ.	Max.	Unit
Input Frequency	PLL_{IF}	0.5	1	2	MHz
PLL Factor	PLL_{F}		64		
Lock-in Time	PLL_{LT}			80	µS

Note: While connected to external clock or external oscillator, PLL input frequency must be selected to provide outputs with frequency in accordance with driven parts of the circuit (CPU core, PSC...)

29. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xB9)(5)	PMIC1	POVEN1	PISEL1	PELEV1	PFLTE1	PAOC1	PRFM12	PRFM11	PRFM10	131
(0xB8)(5)	PMIC0	POVEN0	PISEL0	PELEV0	PFLTE0	PAOC0	PRFM02	PRFM01	PRFM00	131
(0xB7)(5)	PCTL	PPRE1	PPRE0	PCLKSEL	–	–	–	PCCYC	PRUN	130
(0xB6)(5)	POC	–	–	POEN2B	POEN2A	POEN1B	POEN1A	POEN0B	POEN0A	33
(0xB5)(5)	PCNF	–	–	PULOCK	PMODE	POPB	POPA	–	–	130
(0xB4)(5)	PSYNC	–	–	PSYNC21	PSYNC20	PSYNC11	PSYNC10	PSYNC01	PSYNC00	128
(0xB3)(5)	POCR_RBH	–	–	–	–	POCR_RB11	POCR_RB10	POCR_RB9	POCR_RB8	129
(0xB2)(5)	POCR_RBL	POCR_RB7	POCR_RB6	POCR_RB5	POCR_RB4	POCR_RB3	POCR_RB2	POCR_RB1	POCR_RB0	129
(0xB1)(5)	POCR2SBH	–	–	–	–	POCR2SB11	POCR2SB10	POCR2SB9	POCR2SB8	129
(0xB0)(5)	POCR2SBL	POCR2SB7	POCR2SB6	POCR2SB5	POCR2SB4	POCR2SB3	POCR2SB2	POCR2SB1	POCR2SB0	129
(0xAF)(5)	POCR2RAH	–	–	–	–	POCR2RA11	POCR2RA10	POCR2RA9	POCR2RA8	129
(0xAE)(5)	POCR2RAL	POCR2RA7	POCR2RA6	POCR2RA5	POCR2RA4	POCR2RA3	POCR2RA2	POCR2RA1	POCR2RA0	129
(0xAD)(5)	POCR2SAH	–	–	–	–	POCR2SA11	POCR2SA10	POCR2SA9	POCR2SA8	129
(0xAC)(5)	POCR2SAL	POCR2SA7	POCR2SA6	POCR2SA5	POCR2SA4	POCR2SA3	POCR2SA2	POCR2SA1	POCR2SA0	129
(0xAB)(5)	POCR1SBH	–	–	–	–	POCR1SB11	POCR1SB10	POCR1SB9	POCR1SB8	129
(0xAA)(5)	POCR1SBL	POCR1SB7	POCR1SB6	POCR1SB5	POCR1SB4	POCR1SB3	POCR1SB2	POCR1SB1	POCR1SB0	129
(0xA9)(5)	POCR1RAH	–	–	–	–	POCR1RA11	POCR1RA10	POCR1RA9	POCR1RA8	129
(0xA8)(5)	POCR1RAL	POCR1RA7	POCR1RA6	POCR1RA5	POCR1RA4	POCR1RA3	POCR1RA2	POCR1RA1	POCR1RA0	129
(0xA7)(5)	POCR1SAH	–	–	–	–	POCR1SA11	POCR1SA10	POCR1SA9	POCR1SA8	129
(0xA6)(5)	POCR1SAL	POCR1SA7	POCR1SA6	POCR1SA5	POCR1SA4	POCR1SA3	POCR1SA2	POCR1SA1	POCR1SA0	129
(0xA5)(5)	POCR0SBH	–	–	–	–	POCR0SB11	POCR0SB10	POCR0SB9	POCR0SB8	129
(0xA4)(5)	POCR0SBL	POCR0SB7	POCR0SB6	POCR0SB5	POCR0SB4	POCR0SB3	POCR0SB2	POCR0SB1	POCR0SB0	129
(0xA3)(5)	POCR0RAH	–	–	–	–	POCR0RA11	POCR0RA10	POCR0RA9	POCR0RA8	129
(0xA2)(5)	POCR0RAL	POCR0RA7	POCR0RA6	POCR0RA5	POCR0RA4	POCR0RA3	POCR0RA2	POCR0RA1	POCR0RA0	129
(0xA1)(5)	POCR0SAH	–	–	–	–	POCR0SA11	POCR0SA10	POCR0SA9	POCR0SA8	129
(0xA0)(5)	POCR0SAL	POCR0SA7	POCR0SA6	POCR0SA5	POCR0SA4	POCR0SA3	POCR0SA2	POCR0SA1	POCR0SA0	129
(0x9F)	Reserved	–	–	–	–	–	–	–	–	
(0x9E)	Reserved	–	–	–	–	–	–	–	–	
(0x9D)	Reserved	–	–	–	–	–	–	–	–	
(0x9C)	Reserved	–	–	–	–	–	–	–	–	
(0x9B)	Reserved	–	–	–	–	–	–	–	–	
(0x9A)	Reserved	–	–	–	–	–	–	–	–	
(0x99)	Reserved	–	–	–	–	–	–	–	–	
(0x98)	Reserved	–	–	–	–	–	–	–	–	
(0x97)	AC3CON	AC3EN	AC3IE	AC3IS1	AC3IS0	–	AC3M2	AC3M1	AC3M0	229

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

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**