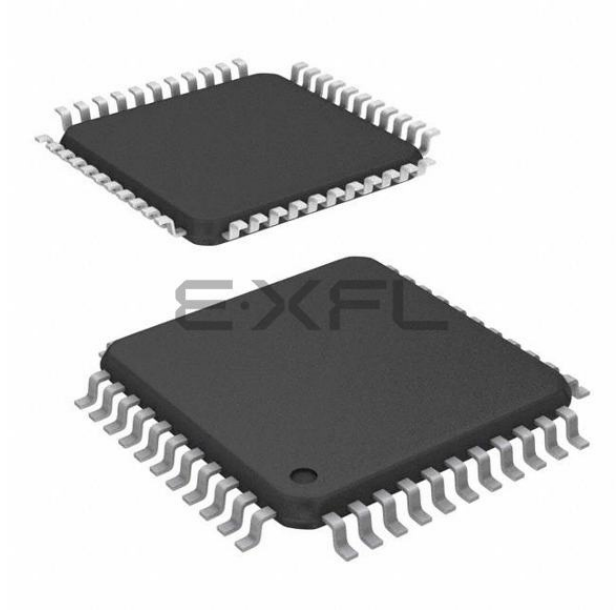


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	32
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	1K x 8
RAM Size	2K 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	44-TQFP
Supplier Device Package	44-TQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega32a-au

10.7. Reset and Interrupt Handling

The Atmel 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* 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* . 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 Interrupt Vector Select (IVSEL) bit in the General Interrupt Control Register (GICR). Refer to *Interrupts* for more information. The Reset Vector can also be moved to the start of the boot Flash section by programming the BOOTRST Fuse, see *Boot Loader Support – Read-While-Write Self-Programming*.

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, EEMWE ; start EEPROM write
sbi EECR, EEWE
out SREG, r16 ; restore SREG value (I-bit)
```

13.2. Idle Mode

When the SM2:0 bits are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing SPI, USART, Analog Comparator, ADC, Two-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ACD bit in the Analog Comparator Control and Status Register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

13.3. ADC Noise Reduction Mode

When the SM2:0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the external interrupts, the Two-wire Serial Interface address watch, Timer/Counter2 and the Watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing the other clocks to run.

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart form the ADC Conversion Complete interrupt, only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an External level interrupt on INT0 or INT1, or an external interrupt on INT2 can wake up the MCU from ADC Noise Reduction mode.

13.4. Power-down Mode

When the SM2:0 bits are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the External Oscillator is stopped, while the External Interrupts, the Two-wire Serial Interface address watch, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brownout Reset, a Two-wire Serial Interface address match interrupt, an External Level Interrupt on INT0 or INT1, or an External Interrupt on INT2 can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. Refer to *External Interrupts* for details.

When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL Fuses that define the Reset Time-out period, as described in *Clock Sources*.

Related Links

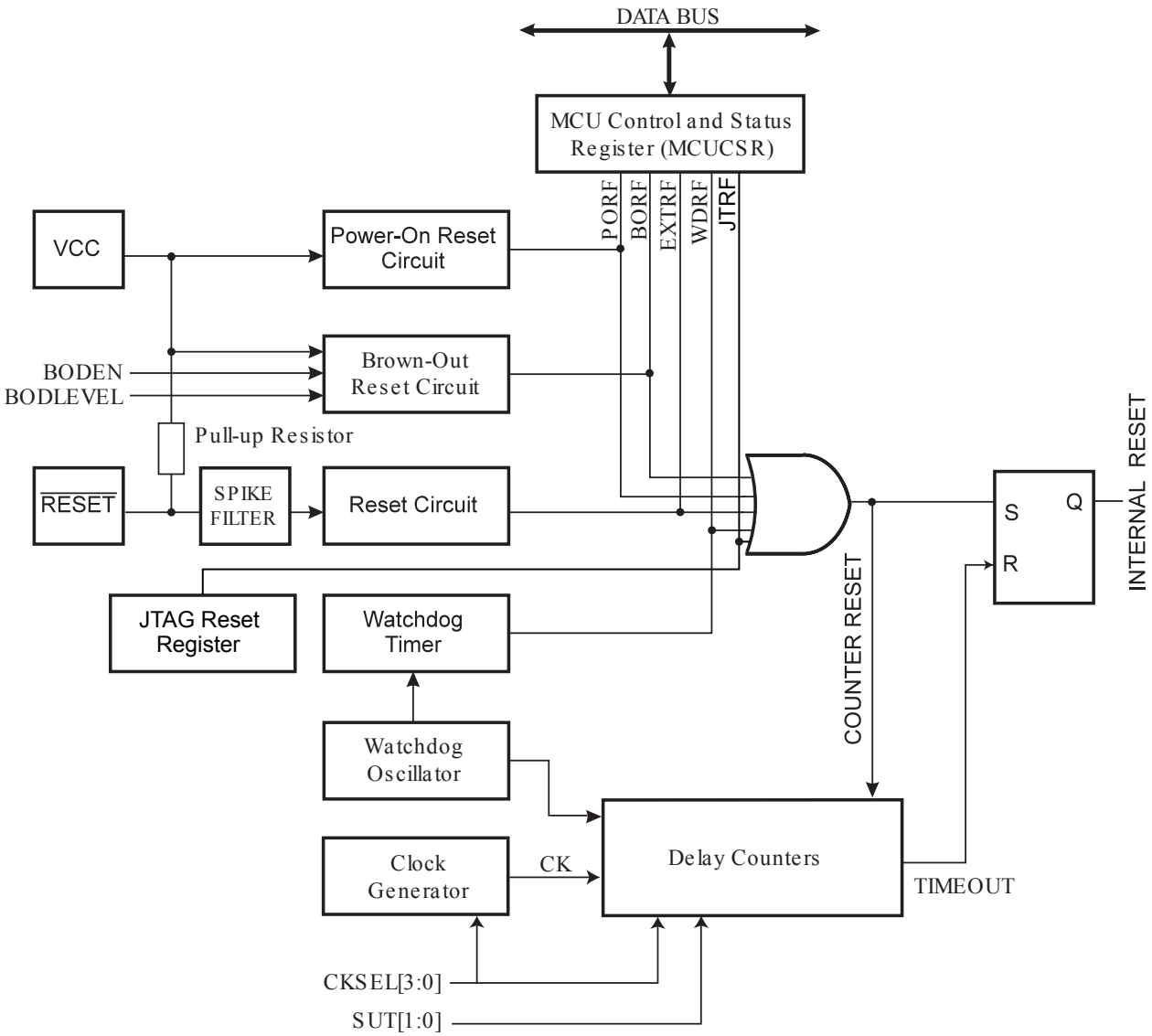
External Interrupts on page 69

Clock Sources on page 40

13.5. Power-save Mode

When the SM2:0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception:

Figure 14-1. Reset Logic



Related Links

IEEE 1149.1 (JTAG) Boundary-scan on page 287

14.2.1. Power-on Reset

A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. The detection level is defined in the table in *System and Reset Characteristics*. The POR is activated whenever V_{CC} is below the detection level. The POR circuit can be used to trigger the Start-up Reset, as well as to detect a failure in supply voltage.

A Power-on Reset (POR) circuit ensures that the device is reset from Power-on. Reaching the Power-on Reset threshold voltage invokes the delay counter, which determines how long the device is kept in RESET after V_{CC} rise. The RESET signal is activated again, without any delay, when V_{CC} decreases below the detection level.

the TCNTn value can be accessed by the CPU, independent of whether clk_{Tn} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the *Waveform Generation mode* bits (WGMn3:0) located in the *Timer/Counter Control Registers* A and B (TCCRnA and TCCRnB). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare Outputs OCnx. For more details about advanced counting sequences and waveform generation, see Modes of Operation.

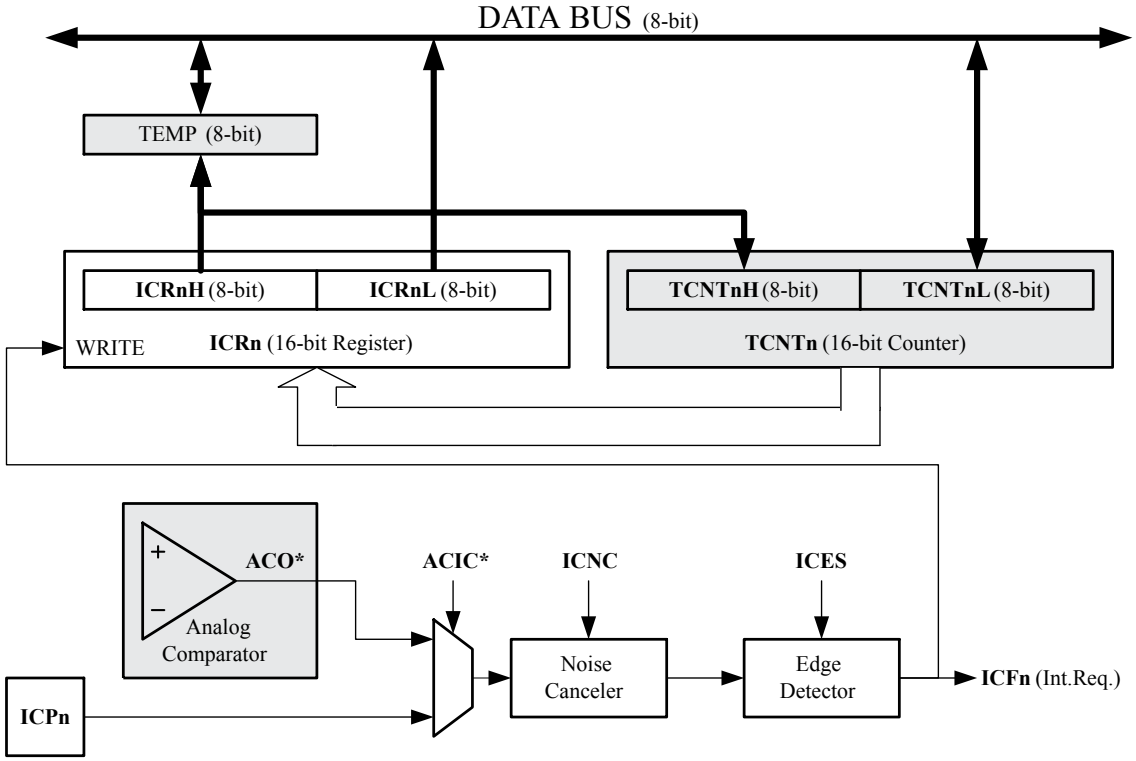
The *Timer/Counter Overflow* (TOVn) flag is set according to the mode of operation selected by the WGMn3:0 bits. TOVn can be used for generating a CPU interrupt.

19.6. Input Capture Unit

The Timer/Counter incorporates an Input Capture unit that can capture external events and give them a timestamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICPn pin or alternatively, via the Analog Comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

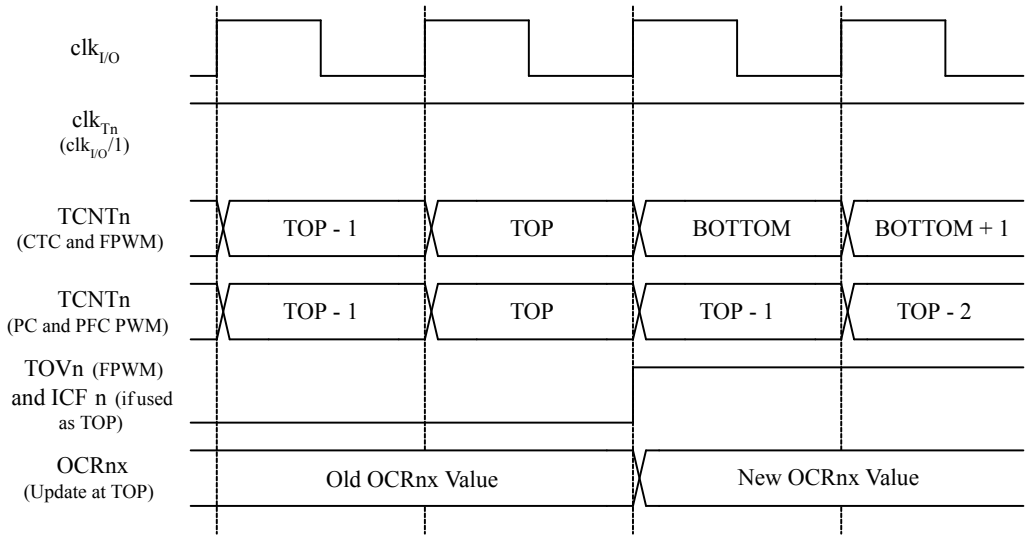
The Input Capture unit is illustrated by the block diagram below. The elements of the block diagram that are not directly a part of the Input Capture unit are gray shaded. The small “n” in register and bit names indicates the Timer/Counter number.

Figure 19-3. Input Capture Unit Block Diagram



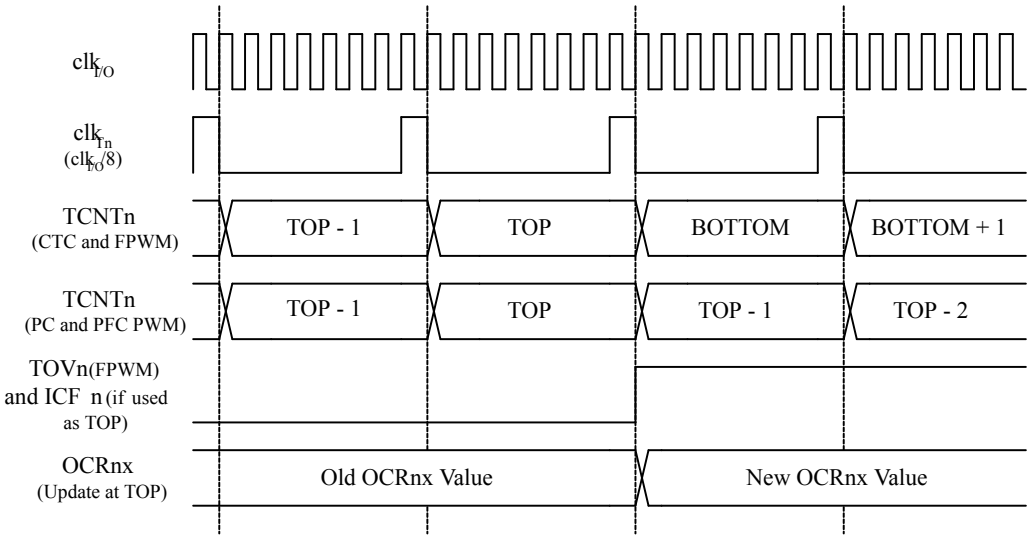
When a change of the logic level (an event) occurs on the *Input Capture Pin* (ICPn), alternatively on the *Analog Comparator Output* (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNTn) is written to the *Input Capture Register* (ICRn). The *Input Capture Flag* (ICFn) is set at the same system clock as the TCNTn value is copied into ICRn Register. If enabled (TICIEn = 1), the Input Capture Flag generates an

Figure 19-12. Timer/Counter Timing Diagram, no Prescaling.



The next figure shows the same timing data, but with the prescaler enabled.

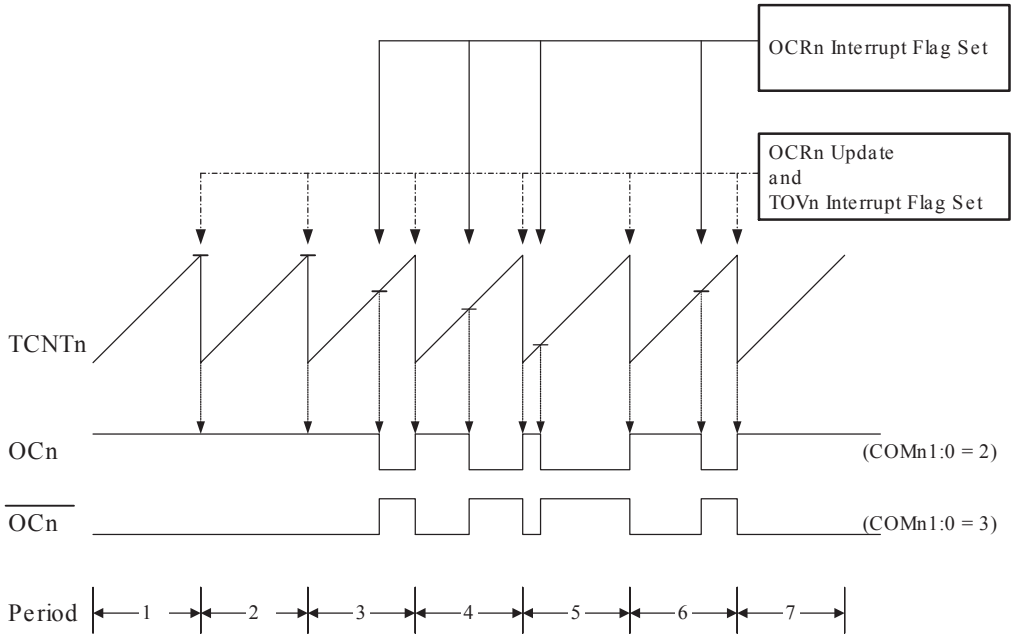
Figure 19-13. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)



19.11. Register Description

small horizontal line marks on the TCNT2 slopes represent compare matches between OCR2 and TCNT2.

Figure 20-6. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC2 pin. Setting the COM21:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM21:0 to 3. The actual OC2 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 setting (or clearing) the OC2 Register at the Compare Match between OCR2 and TCNT2, and clearing (or setting) the OC2 Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{\mathrm{OCnPWM}} = \frac{f_{\mathrm{clk\_I/O}}}{N \cdot 256}$$

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

The extreme values for the OCR2 Register represent special cases when generating a PWM waveform output in the fast PWM mode. If the OCR2 is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR2 equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM21:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC2 to toggle its logical level on each Compare Match (COM21:0 = 1). The waveform generated will have a maximum frequency of $f_{oc2} = f_{clk\_I/O}/2$ when OCR2 is set to zero. This feature is similar to the OC2 toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

20.7.4. Phase Correct PWM Mode

The phase correct PWM mode (WGM21:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output

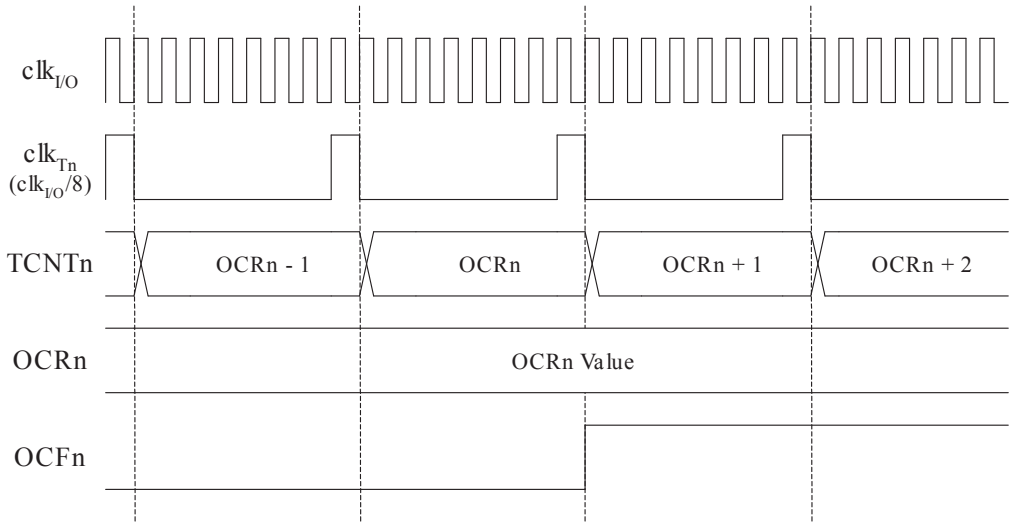
Bits 2:0 – CS2n: Clock Select [n = 2:0]

The three Clock Select bits select the clock source to be used by the Timer/Counter.

Table 20-6. Clock Select Bit Description

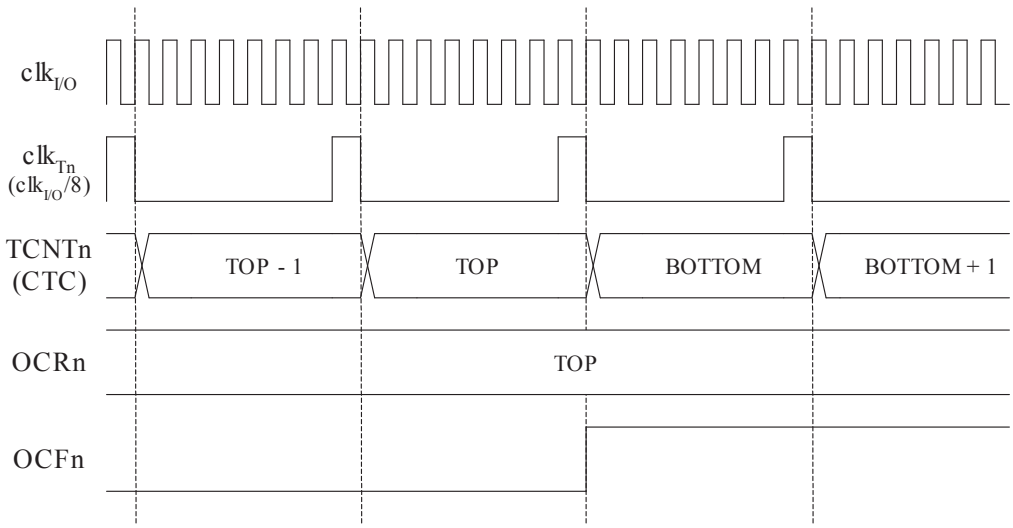
CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/32 (From prescaler)
1	0	0	clkI/O/64 (From prescaler)
1	0	1	$clk_{I/O}$/128 (From prescaler)
1	1	0	$clk_{I/O}$/256 (From prescaler)
1	1	1	$clk_{I/O}$/1024 (From prescaler)

Figure 21-10. Timer/Counter Timing Diagram, Setting of OCF0, with Prescaler ($f_{clk\_I/O}/8$)



The next figure shows the setting of OCF0 and the clearing of TCNT0 in CTC mode.

Figure 21-11. Timer/Counter Timing Diagram, Clear Timer on Compare Match Mode, with Prescaler ($f_{clk\_I/O}/8$)



21.9. Register Description

```
    ldi r16,(1<<URSEL) | (1<<USBS) | (1<<UCSZ1)
    out UCSRC,r16
    :.
```

C Code Example(1)

```
:.
   /* Set UBRRH to 2 */
   UBRRH = 0x02;
   :.
   /* Set the USBS and the UCSZ1 bit to one, and */
   /* the remaining bits to zero. */
   UCSRC = (1<<URSEL) | (1<<USBS) | (1<<UCSZ1);
   :.
```

Note: 1. See *About Code Examples.*

As the code examples illustrate, write accesses of the two registers are relatively unaffected of the sharing of I/O location.

Related Links

About Code Examples on page 19

23.10.2. Read Access

Doing a read access to the UBRRH or the UCSRC Register is a more complex operation. However, in most applications, it is rarely necessary to read any of these registers.

The read access is controlled by a timed sequence. Reading the I/O location once returns the UBRRH Register contents. If the register location was read in previous system clock cycle, reading the register in the current clock cycle will return the UCSRC contents. Note that the timed sequence for reading the UCSRC is an atomic operation. Interrupts must therefore be controlled (e.g., by disabling interrupts globally) during the read operation.

The following code example shows how to read the UCSRC Register contents.

Assembly Code Example(1)

```
USART_ReadUCSRC:
   ; Read UCSRC
   in r16,UBRRH
   in r16,UCSRC
   ret
```

C Code Example(1)

```
unsigned char USART_ReadUCSRC( void )
{
   unsigned char ucsrc;
   /* Read UCSRC */
   ucsrc = UBRRH;
   ucsrc = UCSRC;
   return ucsrc;
}
```

Note: 1. See *About Code Examples.*

The assembly code example returns the UCSRC value in r16.

Reading the UBRRH contents is not an atomic operation and therefore it can be read as an ordinary register, as long as the previous instruction did not access the register location.

Related Links

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or Read data byte or Read data byte or Read data byte	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	Switched to the not addressed Slave mode; no recognition of own SLA or GCA Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave	No action	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	Switched to the not addressed Slave mode; no recognition of own SLA or GCA Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

Figure 26-4. ADC Timing Diagram, First Conversion (Single Conversion Mode)

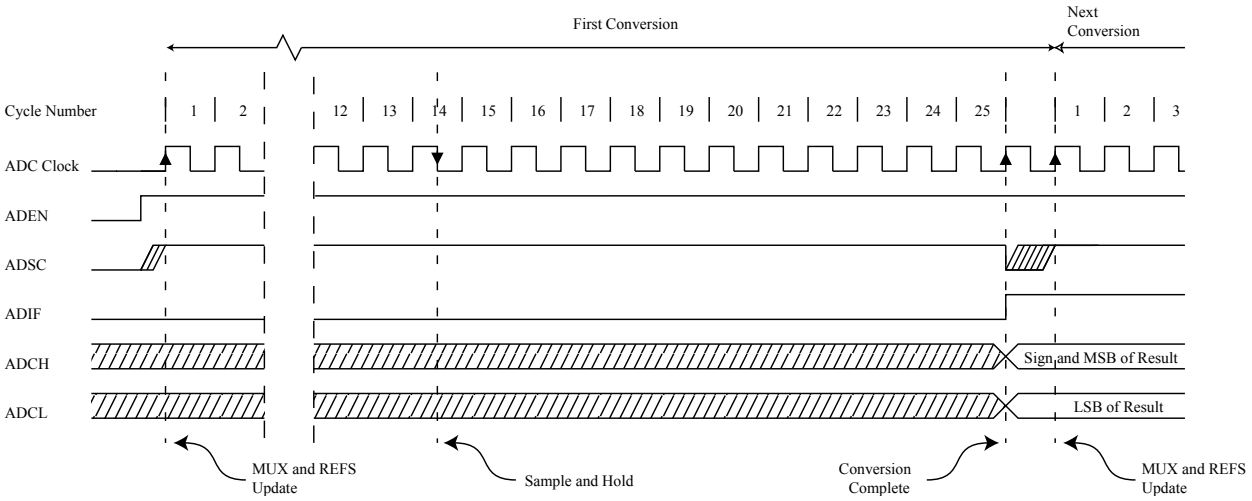


Figure 26-5. ADC Timing Diagram, Single Conversion

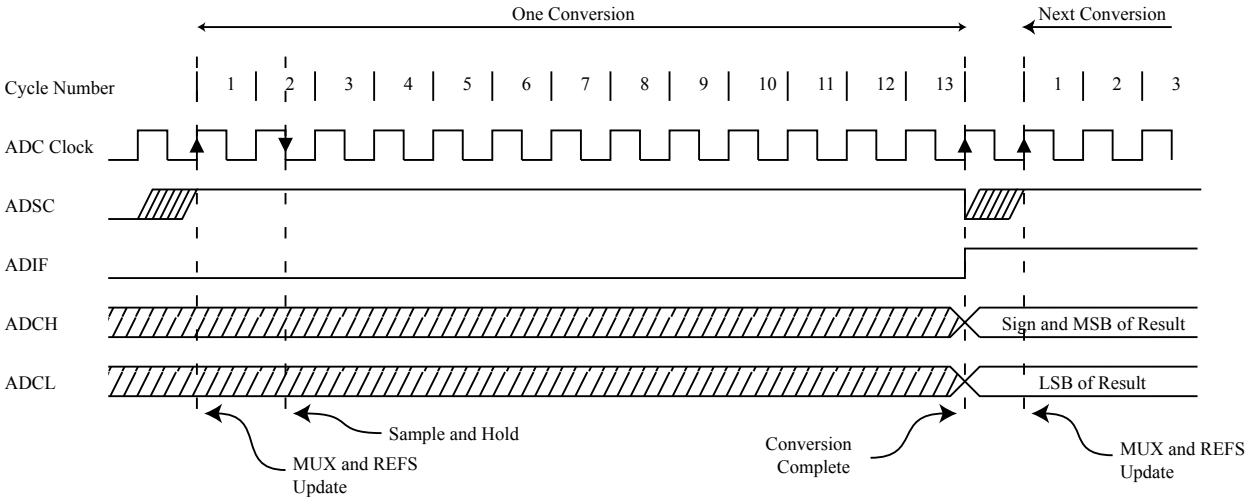


Figure 26-6. ADC Timing Diagram, Auto Triggered Conversion

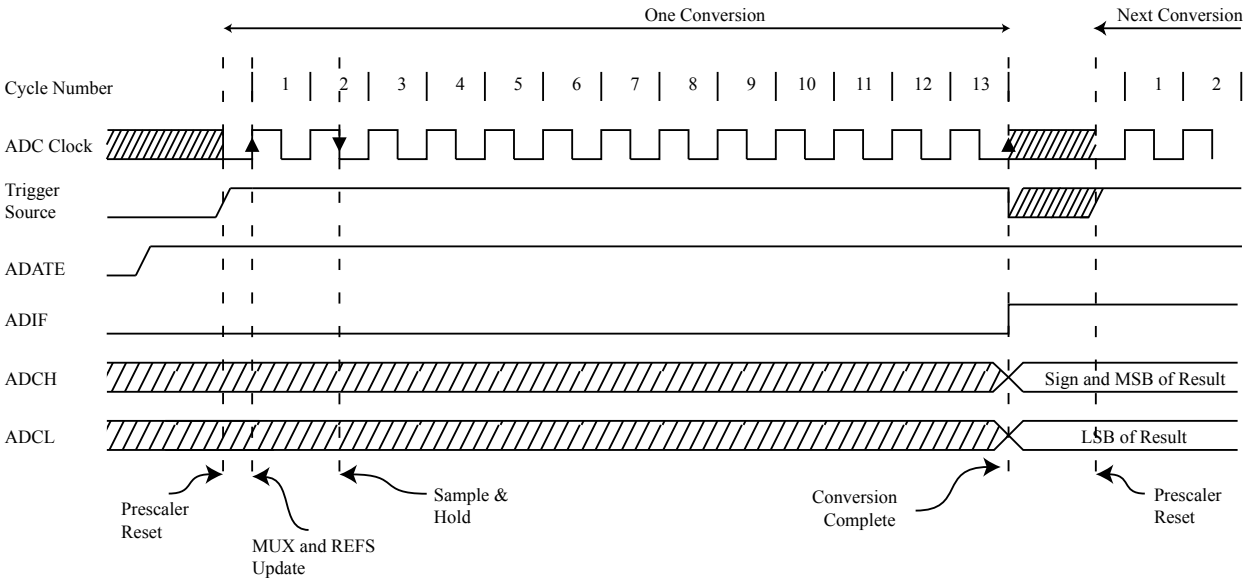


Figure 26-7. ADC Timing Diagram, Free Running Conversion

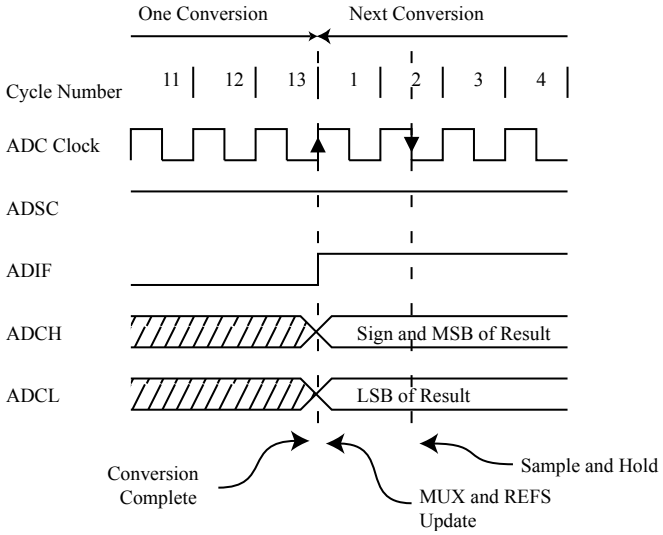


Table 26-1. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5	13/14

26.4.1. Differential Gain Channels

When using differential gain channels, certain aspects of the conversion need to be taken into consideration.

Differential conversions are synchronized to the internal clock CK_{ADC2} equal to half the ADC clock. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific edge of CK_{ADC2}. A conversion initiated by the user (that is, all single conversions, and the first free running conversion) when CK_{ADC2} is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when CK_{ADC2} is high will take 14 ADC clock cycles due to the synchronization mechanism. In free running mode, a new conversion is initiated immediately after the previous conversion completes, and since CK_{ADC2} is high at this time, all automatically started (that is, all but the first) free running conversions will take 14 ADC clock cycles.

The gain stage is optimized for a bandwidth of 4kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. For example the ADC clock period may be 6µs, allowing a channel to be sampled at 12kSPS, regardless of the bandwidth of this channel.

If differential gain channels are used and conversions are started by Auto Triggering, the ADC must be switched off between conversions. When Auto Triggering is used, the ADC prescaler is reset before the conversion is started. Since the gain stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion

(writing ADEN in ADCSRA to “0” then to “1”), only extended conversions are performed. The result from the extended conversions will be valid. Refer to Prescaling and Conversion Timing for timing details.

26.5. Changing Channel or Reference Selection

The MUXn and REFS1:0 bits in the ADMUX Register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

1. When ADATE or ADEN is cleared.
2. During conversion, minimum one ADC clock cycle after the trigger event.
3. After a conversion, before the interrupt flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the gain stage may take as much as 125µs to stabilize to the new value. Thus conversions should not be started within the first 125µs after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).

26.5.1. ADC Input Channels

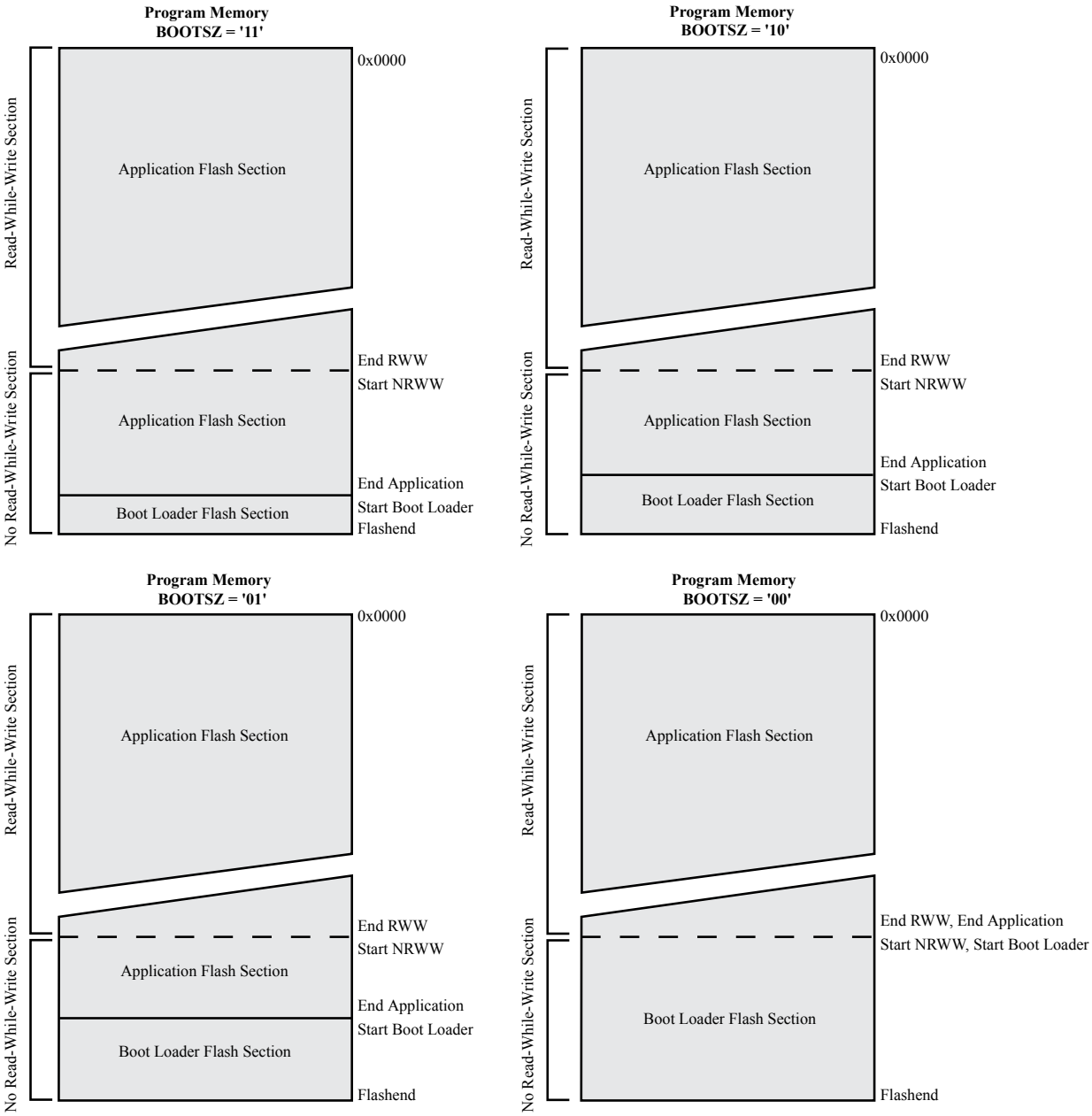
When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

- In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.
- In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

When switching to a differential gain channel, the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. The user should preferably disregard the first conversion result.

Bit Number	Signal Name	Module
66	EXTCLK (XTAL1)	Clock input and Oscillators for the main clock (Observe-only)
65	OSCCK	
64	RCCK	
63	OSC32CK	
62	TWIEN	TWI
61	PD0.Data	Port D
60	PD0.Control	
59	PD0.Pullup_Enable	
58	PD1.Data	
57	PD1.Control	
56	PD1.Pullup_Enable	
55	PD2.Data	
54	PD2.Control	
53	PD2.Pullup_Enable	
52	PD3.Data	
51	PD3.Control	
50	PD3.Pullup_Enable	
49	PD4.Data	
48	PD4.Control	
47	PD4.Pullup_Enable	
46	PD5.Data	
45	PD5.Control	
44	PD5.Pullup_Enable	
43	PD6.Data	
42	PD6.Control	
41	PD6.Pullup_Enable	
40	PD7.Data	
39	PD7.Control	
38	PD7.Pullup_Enable	

Figure 28-2. Memory Sections



Related Links

ATmega32A Boot Loader Parameters on page 323

28.5. 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

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

The algorithm for reading the Fuse Low bits is similar to the one described above for reading the Lock Bits. To read the Fuse Low bits, load the Z-pointer with 0x0000 and set the BLBSET and SPMEN bits in SPMCR. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCR, the value of the Fuse Low bits (FLB) will be loaded in the destination register as shown below. Refer to table *Fuse Low Byte* in section *Fuse Bits* for a detailed description and mapping of the fuse low bits.

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

Similarly, when reading the Fuse High bits, load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCR, the value of the Fuse High bits (FHB) will be loaded in the destination register as shown below. Refer to table *Fuse High Byte* in section *Fuse Bits* for detailed description and mapping of the fuse high bits.

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 read as '0'. Fuse and Lock bits that are unprogrammed, will read as '1'.

28.8.10. 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 SPMCR Register and thus the Flash from unintentional writes.

28.8.11. Programming Time for Flash when Using SPM

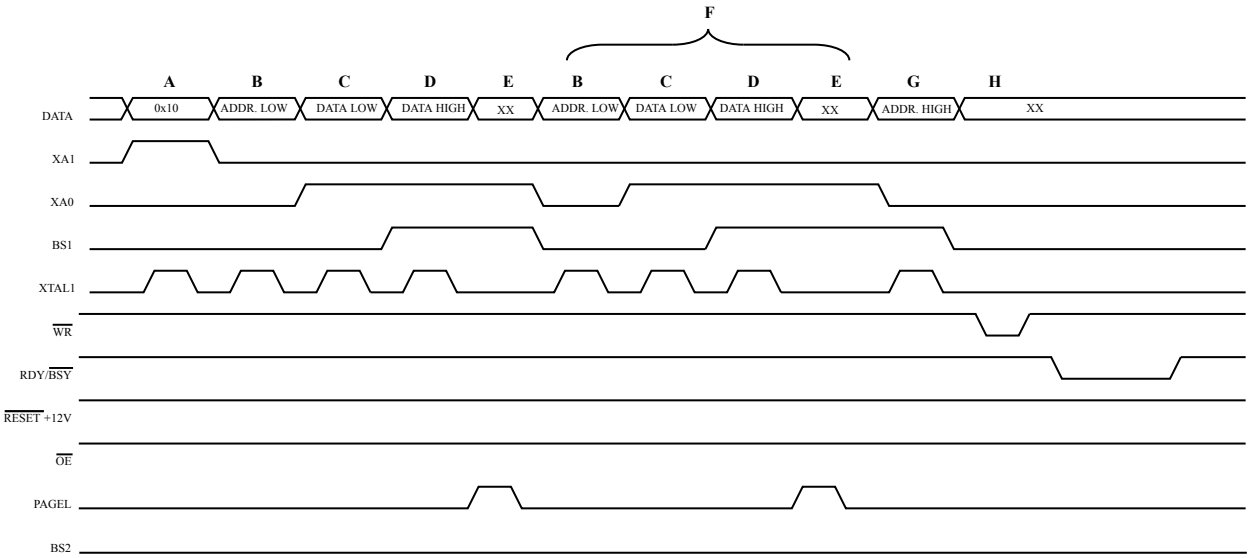
The calibrated RC Oscillator is used to time Flash accesses. The following table shows the typical programming time for Flash accesses from the CPU.

Table 28-5. SPM Programming Time(1)

Symbol	Min. Programming Time	Max. Programming Time
Flash write (Page Erase, Page Write, and write Lock bits by SPM)	3.7ms	4.5ms

Note: 1. Minimum and maximum programming time is per individual operation.

Figure 29-3. Programming the Flash Waveforms



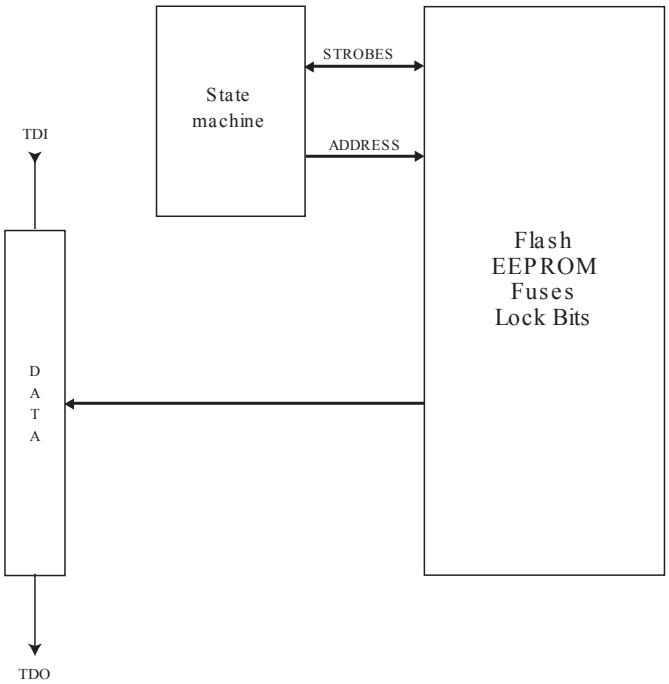
Note: “XX” is don’t care. The letters refer to the programming description above.

29.7.5. Programming the EEPROM

The EEPROM is organized in pages, see Table 29-12, in the Page Size section. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (For details on Command, Address and Data loading, refer to Programming the Flash):

1. Step A: Load Command “0001 0001”.
2. Step G: Load Address High Byte (0x00 - 0xFF).
3. Step B: Load Address Low Byte (0x00 - 0xFF).
4. Step C: Load Data (0x00 - 0xFF).
5. Step E: Latch data (give PAGEL a positive pulse).
6. Step K:Repeat 3 through 5 until the entire buffer is filled.
7. Step L: Program EEPROM page
   - 7.1. Set BS1 to “0”.
   - 7.2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
   - 7.3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page. Refer to the figure below for signal waveforms.

Figure 29-17. Virtual Flash Page Read Register



29.10.13. Programming Algorithm

All references below of type “1a”, “1b”, and so on, refer to Table 29-17.

29.10.14. Entering Programming Mode

1. Enter JTAG instruction AVR_RESET and shift 1 in the Reset Register.
2. Enter instruction PROG_ENABLE and shift 1010_0011_0111_0000 in the Programming Enable Register.

29.10.15. Leaving Programming Mode

1. Enter JTAG instruction PROG_COMMANDS.
2. Disable all programming instructions by using no operation instruction 11a.
3. Enter instruction PROG_ENABLE and shift 0000_0000_0000_0000 in the programming Enable Register.
4. Enter JTAG instruction AVR_RESET and shift 0 in the Reset Register.

29.10.16. Performing Chip Erase

1. Enter JTAG instruction PROG_COMMANDS.
2. Start chip erase using programming instruction 1a.
3. Poll for chip erase complete using programming instruction 1b, or wait for t_{WLRH_CE} (refer to table *Command Byte Bit Coding* in section *Parallel Programming Parameters, Pin Mapping, and Commands*).

Related Links

Parallel Programming Characteristics on page 339

29.10.17. Programming the Flash

Before programming the Flash a Chip Erase must be performed. See Performing Chip Erase.

Related Links

Parallel Programming Characteristics on page 339

29.10.22. Programming the Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Lock bit write using programming instruction 7a.
3. Load data using programming instructions 7b. A bit value of “0” will program the corresponding lock bit, a “1” will leave the lock bit unchanged.
4. Write Lock bits using programming instruction 7c.
5. Poll for Lock bit write complete using programming instruction 7d, or wait for t_{WLRH} (refer to table *Parallel Programming Characteristics, VCC = 5V ±10%* in chapter *Parallel Programming Characteristics*).

Related Links

Parallel Programming Characteristics on page 339

29.10.23. Reading the Fuses and Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse/Lock bit read using programming instruction 8a.
3. – To read all Fuses and Lock bits, use programming instruction 8e.
   – To only read Fuse high byte, use programming instruction 8b.
   – To only read Fuse low byte, use programming instruction 8c.
   – To only read Lock bits, use programming instruction 8d.

29.10.24. Reading the Signature Bytes

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Signature byte read using programming instruction 9a.
3. Load address 0x00 using programming instruction 9b.
4. Read first signature byte using programming instruction 9c.
5. Repeat steps 3 and 4 with address 0x01 and address 0x02 to read the second and third signature bytes, respectively.

29.10.25. Reading the Calibration Byte

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Calibration byte read using programming instruction 10a.
3. Load address 0x00 using programming instruction 10b.
4. Read the calibration byte using programming instruction 10c.

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{OL}	Output Low Voltage[3] (Ports A,B,C,D)	I_{OL} = 20mA, V_{CC} = 5V I_{OL} = 10mA, V_{CC} = 3V			0.7 0.5	V V
V_{OH}	Output High Voltage[4] (Ports A,B,C,D)	I_{OH} = -20mA, V_{CC} = 5V I_{OH} = -10mA, V_{CC} = 3V	4.2 2.2			V V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset Pull-up Resistor		30	60	85	kΩ
R_{pu}	I/O Pin Pull-up Resistor		20		50	kΩ
I_{CC}	Power Supply Current	Active 1MHz, V_{CC} = 3V		0.6		mA
		Active 4MHz, V_{CC} = 3V		2.1	5	mA
		Active 8MHz, V_{CC} = 5V		7.5	15	mA
		Idle 1MHz, V_{CC} = 3V		0.2		mA
		Idle 4MHz, V_{CC} = 3V		0.6	2.5	mA
		Idle 8MHz, V_{CC} = 5V		2.8	8	mA
	Power-down mode[5]	WDT enabled, V_{CC} = 3V		<10	20	µA
		WDT disabled, V_{CC} = 3V		<1	10	µA
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2			40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACPD}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Note:

1. “Max” means the highest value where the pin is guaranteed to be read as low
2. “Min” means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:

   PDIP Package:

   3.1. The sum of all IOL, for all ports, should not exceed 200mA.

   3.2. The sum of all IOL, for ports A0 - A7 should not exceed 100mA.