Atmel - AT90CAN128-15MT Datasheet

Welcome to E-XFL.COM

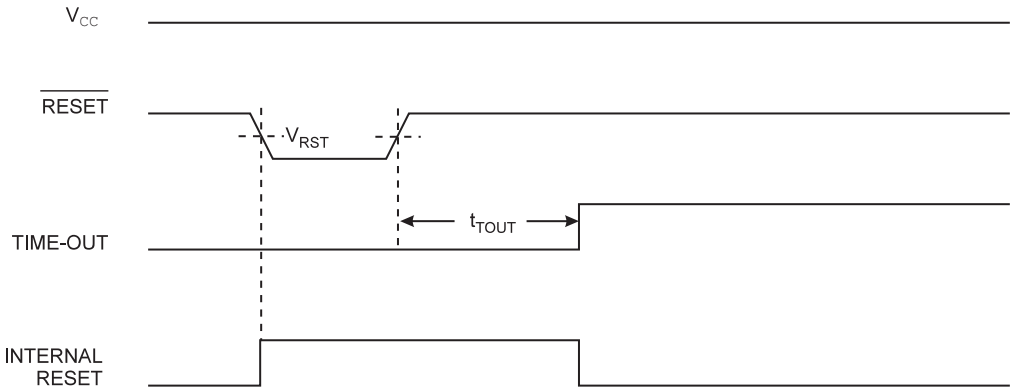
What is "Embedded - Microcontrollers"?

"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, I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	53
Program Memory Size	128KB (128K x 8)
Program Memory Type	FLASH
EEPROM Size	4K x 8
RAM Size	4K 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	64-VFQFN Exposed Pad
Supplier Device Package	64-QFN (9x9)
Purchase URL	https://www.e-xfl.com/product-detail/atmel/at90can128-15mt

Figure 7-4. External Reset During Operation



7.1.5 Brown-out Detection

AT90CAN32/64/128 has an On-chip Brown-out Detection (BOD) circuit for monitoring the V_{CC} level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODLEVEL Fuses. The trigger level has a hysteresis to ensure spike free Brown-out Detection. The hysteresis on the detection level should be interpreted as $V_{BOT+} = V_{BOT} + V_{HYST}/2$ and $V_{BOT-} = V_{BOT} - V_{HYST}/2$.

Table 7-2. BODLEVEL Fuse Coding(1)

BODLEVEL 2..0 Fuses	Min V_{BOT}	Typ V_{BOT}	Max V_{BOT}	Units
111	BOD Disabled			
110	3.8	4.1	4.4	V
101		4.0(2)		V
100		3.9(2)		V
011		3.8(2)		V
010	2.5	2.7	2.9	V
001		2.6(2)		V
000		2.5(2)		V

Notes: 1. V_{BOT} may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to $V_{CC} = V_{BOT}$ during the production test. This guarantees that a Brown-Out Reset will occur before V_{CC} drops to a voltage where correct operation of the microcontroller is no longer guaranteed. The test is performed using BODLEVEL = 010 for Low Operating Voltage and BODLEVEL = 101 for High Operating Voltage .

2. Not tested.

Table 7-3. Brown-out Characteristics

Symbol	Parameter	Min.	Typ.	Max.	Units
V_{HYST}	Brown-out Detector Hysteresis		70		mV
t_{BOD}	Min Pulse Width on Brown-out Reset		2		µs

Table 9-8. Overriding Signals for Alternate Functions in PB3..PB0

Signal Name	PB3/MISO	PB2/MOSI	PB1/SCK	PB0/$\overline{SS}$
PUOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
PUOV	PORTB3 • $\overline{PUD}$	PORTB2 • $\overline{PUD}$	PORTB1 • $\overline{PUD}$	PORTB0 • $\overline{PUD}$
DDOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
DDOV	0	0	0	0
PVOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • MSTR	0
PVOV	SPI SLAVE OUTPUT	SPI MASTER OUTPUT	SCK OUTPUT	0
PTOE	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	SPI MASTER INPUT	SPI SLAVE INPUT • $\overline{RESET}$	SCK INPUT	SPI $\overline{SS}$
AIO	–	–	–	–

9.3.4 Alternate Functions of Port C

The Port C has an alternate function as the address high byte for the External Memory Interface.

The Port C pins with alternate functions are shown in Table 9-9.

Table 9-9. Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	A15/CLKO (External memory interface address 15 or Divided System Clock)
PC6	A14 (External memory interface address 14)
PC5	A13 (External memory interface address 13)
PC4	A12 (External memory interface address 12)
PC3	A11 (External memory interface address 11)
PC2	A10 (External memory interface address 10)
PC1	A9 (External memory interface address 9)
PC0	A8 (External memory interface address 8)

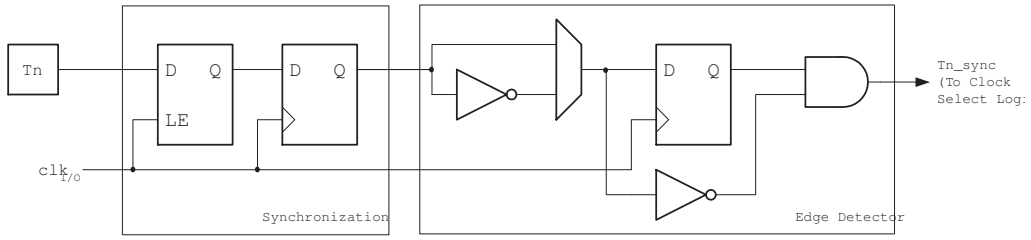
The alternate pin configuration is as follows:

- **A15/CLKO – Port C, Bit 7**

A15, External memory interface address 15.

CLKO, Divided System Clock: The divided system clock can be output on the PC7 pin. The divided system clock will be output if the CKOUT Fuse is programmed, regardless of the PORTC7 and DDC7 settings. It will also be output during reset.

Figure 11-1. T3/T1/T0 Pin Sampling



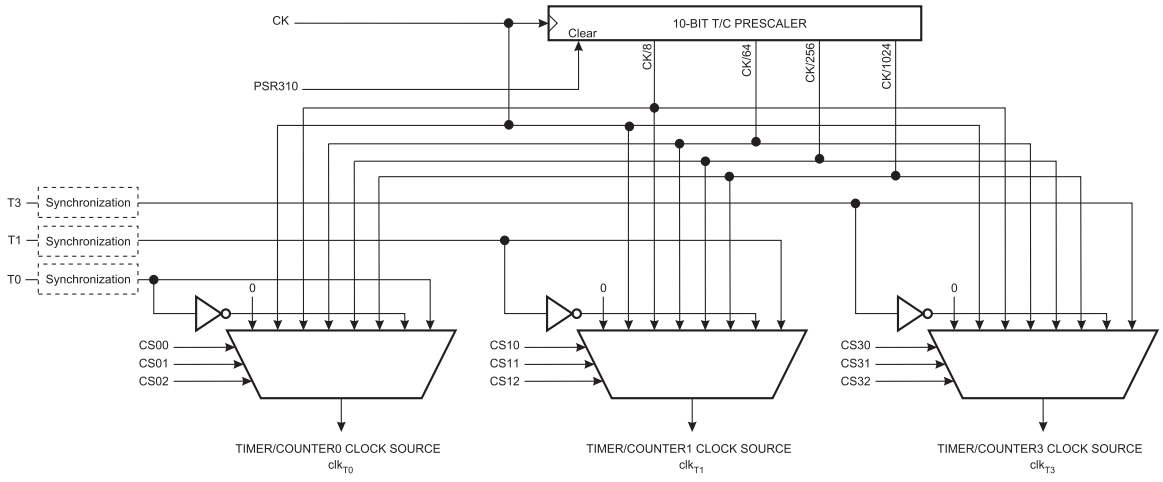
The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the T3/T1/T0 pin to the counter is updated.

Enabling and disabling of the clock input must be done when T3/T1/T0 has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.

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 11-2. Prescaler for Timer/Counter3, Timer/Counter1 and Timer/Counter0 (1)



Note: 1. The synchronization logic on the input pins (T0/T1/T3) is shown in Figure 11-1.

13.3.1 Code Examples

The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCRnx and ICRn Registers. Note that when using “C”, the compiler handles the 16-bit access.

Assembly Code Examples[1]

```
...
; Set TCNTn to 0x01FF
ldi     r17,0x01
ldi     r16,0xFF
sts     TCNTnH,r17
sts     TCNTnL,r16
; Read TCNTn into r17:r16
lds     r16,TCNTnL
lds     r17,TCNTnH
...
```

C Code Examples[1]

```
unsigned int i;
...
/* Set TCNTn to 0x01FF */
TCNTn = 0x1FF;
/* Read TCNTn into i */
i = TCNTn;
...
```

Note: 1. The example code assumes that the part specific header file is included.

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

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit timer registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx Register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

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

The extreme values for the OCRnx Register represent 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 continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

13.9.5 Phase and Frequency Correct PWM Mode

The phase and frequency correct Pulse Width Modulation, or phase and frequency correct PWM mode (WGMn3:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting Compare Output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCRnx Register is updated by the OCRnx Buffer Register, (see Figure 13-8 and Figure 13-9).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICRn (WGMn3:0 = 8), or the value in OCRnA (WGMn3:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on Figure 13-9. The figure shows phase and frequency correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx interrupt flag will be set when a compare match occurs.

- **Bit 2 – OCFnB: Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register B (OCRnB).

Note that a Forced Output Compare (FOCnB) strobe will not set the OCFnB flag.

OCFnB is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCFnB can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCFnA: Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register A (OCRnA).

Note that a Forced Output Compare (FOCnA) strobe will not set the OCFnA flag.

OCFnA is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCFnA can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOVn: Timer/Counter Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOVn flag is set when the timer overflows. Refer to Table 13-4 on page 138 for the TOVn flag behavior when using another WGMn3:0 bit setting.

TOVn is automatically cleared when the Timer/Countern Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.

Figure 14-9. Timer/Counter Timing Diagram, no Prescaling

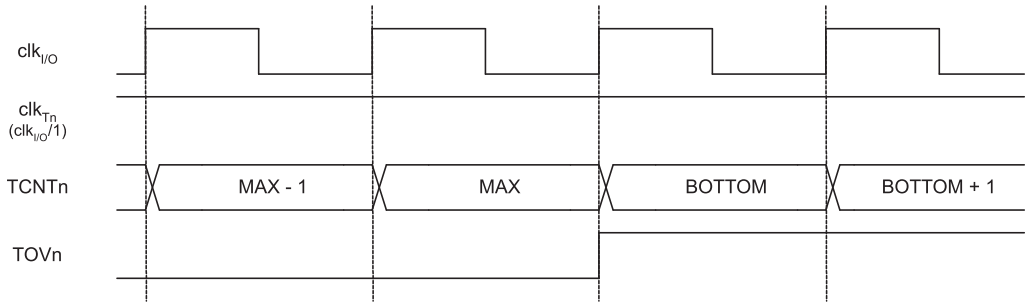


Figure 14-10 shows the same timing data, but with the prescaler enabled.

Figure 14-10. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

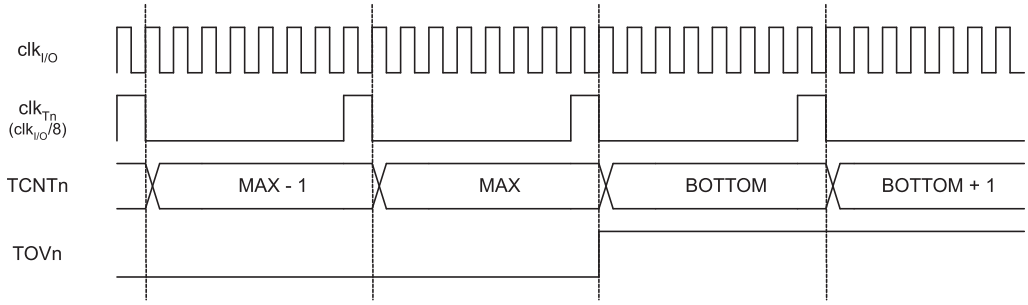
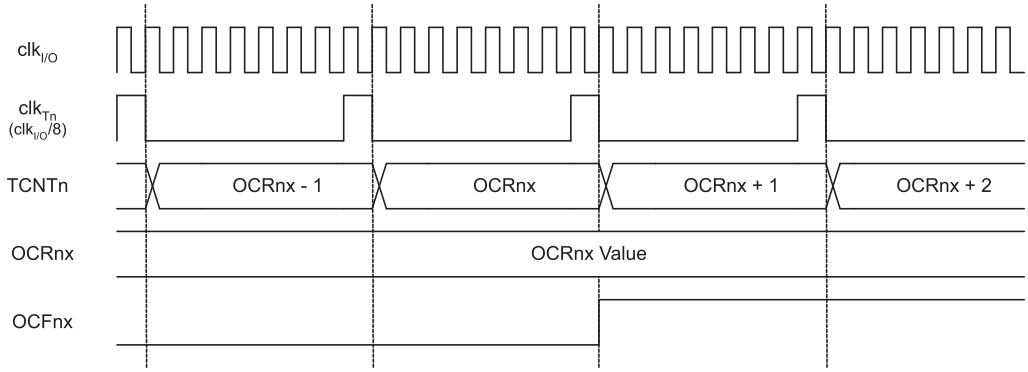


Figure 14-11 shows the setting of OCF2A in all modes except CTC mode.

Figure 14-11. Timer/Counter Timing Diagram, Setting of OCF2A, with Prescaler ($f_{clk\_I/O}/8$)



15. Output Compare Modulator - OCM

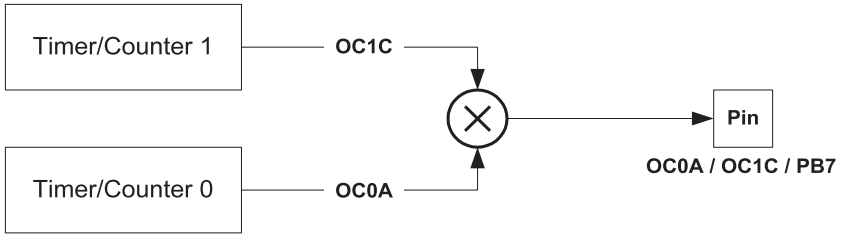
15.1 Overview

Many register and bit references in this section are written in general form.

- A lower case “n” replaces the Timer/Counter number, in this case 0 and 1. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.
- A lower case “x” replaces the Output Compare unit channel, in this case A or C. However, when using the register or bit defines in a program, the precise form must be used, i.e., OCR0A for accessing Timer/Counter0 output compare channel A value and so on.

The Output Compare Modulator (OCM) allows generation of waveforms modulated with a carrier frequency. The modulator uses the outputs from the Output Compare Unit C of the 16-bit Timer/Counter1 and the Output Compare Unit of the 8-bit Timer/Counter0. For more details about these Timer/Counters see “16-bit Timer/Counter (Timer/Counter1 and Timer/Counter3)” on page 113 and “8-bit Timer/Counter0 with PWM” on page 99.

Figure 15-1. Output Compare Modulator, Block Diagram



When the modulator is enabled, the two output compare channels are modulated together as shown in the block diagram (Figure 15-1).

15.2 Description

The Output Compare unit 1C and Output Compare unit 0A shares the PB7 port pin for output. The outputs of the Output Compare units (OC1C and OC0A) overrides the normal PORTB7 Register when one of them is enabled (i.e., when COMnx1:0 is not equal to zero). When both OC1C and OC0A are enabled at the same time, the modulator is automatically enabled.

When the modulator is enabled the type of modulation (logical AND or OR) can be selected by the PORTB7 Register. Note that the DDRB7 controls the direction of the port independent of the COMnx1:0 bit setting.

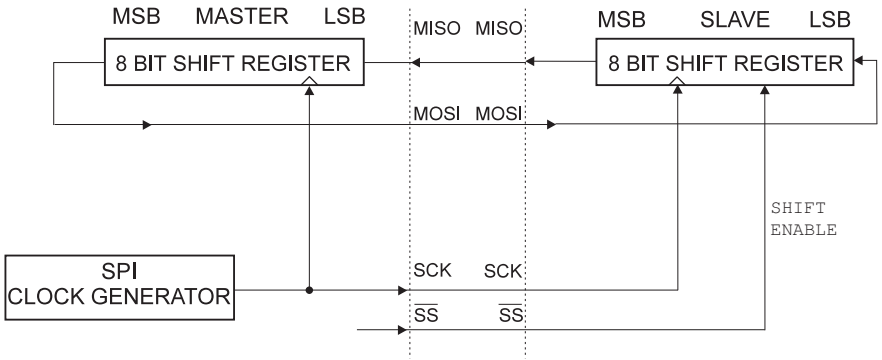
The functional equivalent schematic of the modulator is shown on Figure 15-2. The schematic includes part of the Timer/Counter units and the port B pin 7 output driver circuit.

The interconnection between Master and Slave CPUs with SPI is shown in Figure 16-2. The system consists of two shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select $\overline{SS}$ pin of the desired Slave. Master and Slave prepare the data to be sent in their respective shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, $\overline{SS}$, line.

When configured as a Master, the SPI interface has no automatic control of the $\overline{SS}$ line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, $\overline{SS}$ line. The last incoming byte will be kept in the Buffer Register for later use.

When configured as a Slave, the SPI interface will remain sleeping with MISO tri-stated as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. As one byte has been completely shifted, the end of transmission flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR Register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the Buffer Register for later use.

Figure 16-2. SPI Master-slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed $f_{clkio}/4$.

and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 16-3 and Figure 16-4 for an example. The CPOL functionality is summarized below:

Table 16-2. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 16-3 and Figure 16-4 for an example. The CPOL functionality is summarized below:

Table 16-3. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the clk_{IO} frequency f_{clkio} is shown in the following table:

Table 16-4. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{clkio}/4$
0	0	1	$f_{clkio}/16$
0	1	0	$f_{clkio}/64$
0	1	1	$f_{clkio}/128$
1	0	0	$f_{clkio}/2$
1	0	1	$f_{clkio}/8$
1	1	0	$f_{clkio}/32$
1	1	1	$f_{clkio}/64$

check that there are no unread data in the receive buffer. Note that the TXCn flag must be cleared before each transmission (before UDRn is written) if it is used for this purpose.

The following simple USART0 initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 Registers.

Assembly Code Example [1]

```
USART0_Init:
  ; Set baud rate
  sts    UBRR0H, r17
  sts    UBRR0L, r16
  ; Set frame format: 8data, no parity & 2 stop bits
  ldi    r16, (0<<UMSEL0)|(0<<UPM0)|(1<<USBS0)|(3<<UCSZ0)
  sts    UCSR0C, r16
  ; Enable receiver and transmitter
  ldi    r16, (1<<RXEN0)|(1<<TXEN0)
  sts    UCSR0B, r16
  ret
```

C Code Example [1]

```
void USART0_Init (unsigned int baud)
{
  /* Set baud rate */
  UBRR0H = (unsigned char) (baud>>8);
  UBRR0L = (unsigned char) baud;
  /* Set frame format: 8data, no parity & 2 stop bits */
  UCSR0C = (0<<UMSEL0) | (0<<UPM0) | (1<<USBS0) | (3<<UCSZ0);
  /* Enable receiver and transmitter */
  UCSR0B = (1<<RXEN0) | (1<<TXEN0);
}
```

Note: 1. The example code assumes that the part specific header file is included.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

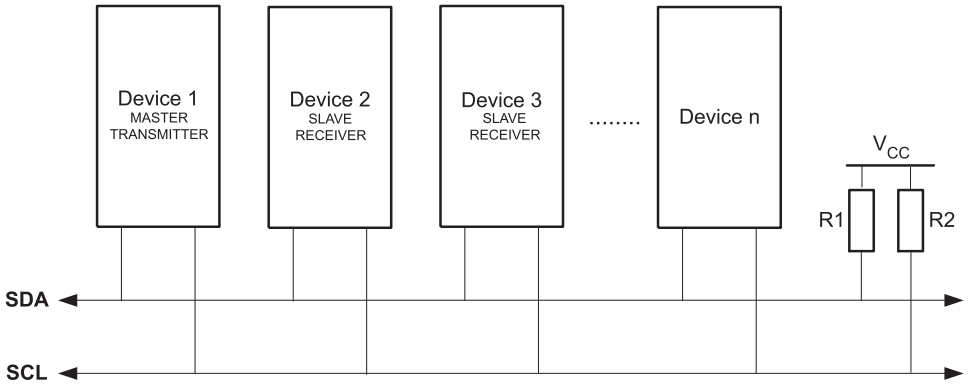
17.7 Data Transmission – USART Transmitter

The USARTn Transmitter is enabled by setting the Transmit Enable (TXENn) bit in the UCSRnB Register. When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USARTn and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If syn-

18.9 Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a slave receiver.

Figure 18-20. An Arbitration Example



Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same slave. In this case, neither the slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Losing masters will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to slave mode to check if they are being addressed by the winning master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

19.11 MOb Registers

The MOb registers has **no** initial (default) value after RESET.

19.11.1 CAN MOb Status Register - CANSTMOB

Bit	7	6	5	4	3	2	1	0	
	DLCW	TXOK	RXOK	BERR	SERR	CERR	FERR	AERR	CANSTMOB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	-	-	-	-	-	-	-	-	

- **Bit 7 – DLCW: Data Length Code Warning**

The incoming message does not have the DLC expected. Whatever the frame type, the DLC field of the CANCDMOB register is updated by the received DLC.

- **Bit 6 – TXOK: Transmit OK**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The communication enabled by transmission is completed. TxOK rises at the end of EOF field. When the controller is ready to send a frame, if two or more message objects are enabled as producers, the lower MOb index (0 to 14) is supplied first.

- **Bit 5 – RXOK: Receive OK**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The communication enabled by reception is completed. RxOK rises at the end of the 6th bit of EOF field. In case of two or more message object reception hits, the lower MOb index (0 to 14) is updated first.

- **Bit 4 – BERR: Bit Error (Only in Transmission)**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The bit value monitored is different from the bit value sent.

Exceptions: the monitored recessive bit sent as a dominant bit during the arbitration field and the acknowledge slot detecting a dominant bit during the sending of an error frame.

- **Bit 3 – SERR: Stuff Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

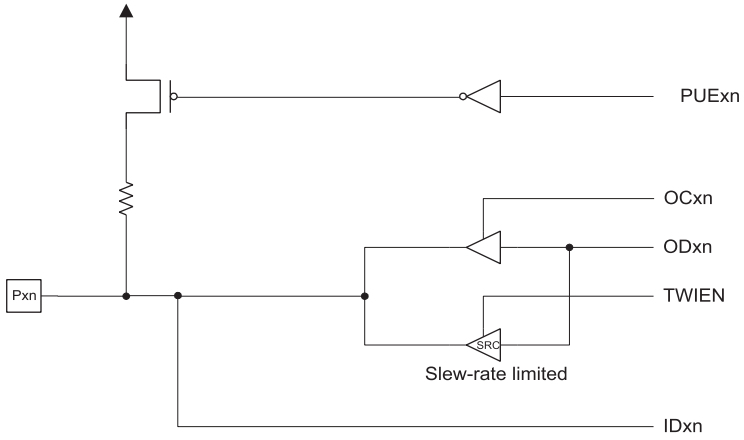
Detection of more than five consecutive bits with the same polarity. This flag can generate an interrupt.

- **Bit 2 – CERR: CRC Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The receiver performs a CRC check on every de-stuffed received message from the start of frame up to the data field. If this checking does not match with the de-stuffed CRC field, a CRC error is set.

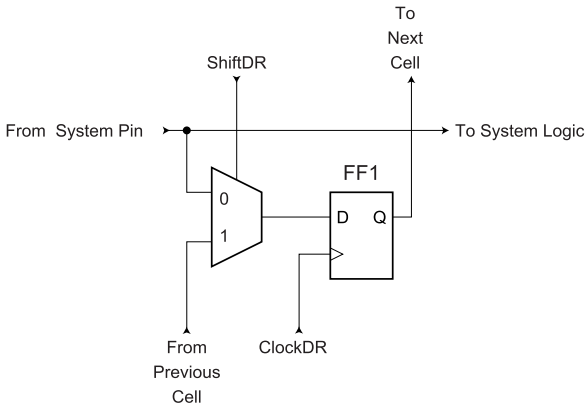
Figure 23-5. Additional Scan Signal for the Two-wire Interface



23.6.3 Scanning the RESET Pin

The RESET pin accepts 3V or 5V active low logic for standard reset operation, and 12V active high logic for High Voltage Parallel programming. An observe-only cell as shown in Figure 23-6 is inserted both for the 3V or 5V reset signal - RSTT, and the 12V reset signal - RSTHV.

Figure 23-6. Observe-only Cell for RESET pin



23.6.4 Scanning the Clock Pins

The AVR devices have many clock options selectable by fuses. These are: Internal RC Oscillator, External Clock, (High Frequency) Crystal Oscillator, Low-frequency Crystal Oscillator, and Ceramic Resonator.

Figure 23-7 shows how each oscillator with external connection is supported in the scan chain. The Enable signal is supported with a general Boundary-scan cell, while the Oscillator/clock output is attached to an observe-only cell. In addition to the main clock, the Timer2 Oscillator is scanned in the same way. The output from the internal RC Oscillator is not scanned, as this oscillator does not have external connections.

Figure 23-8. Analog Comparator

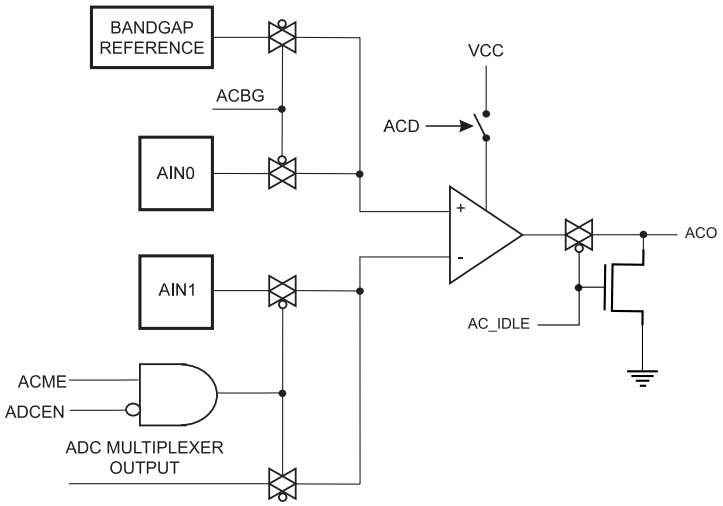


Figure 23-9. General Boundary-scan cell Used for Signals for Comparator and ADC

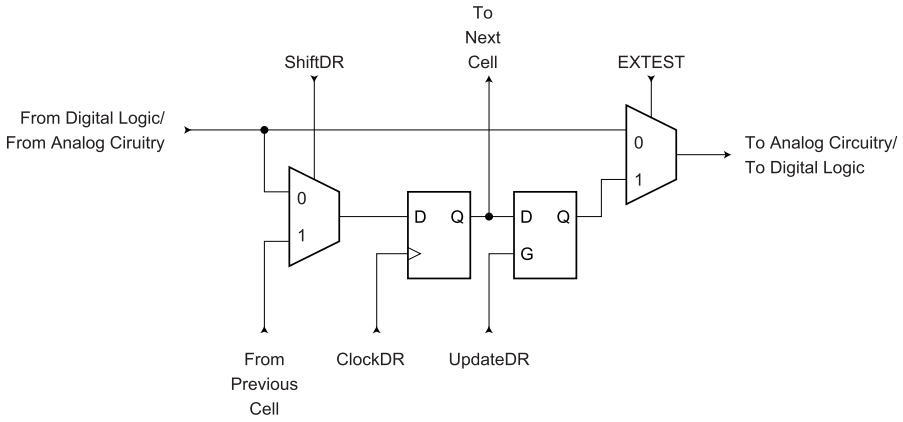


Table 23-6. Boundary-scan Signals for the Analog Comparator

Signal Name	Direction as Seen from the Comparator	Description	Recommended Input when Not in Use	Output Values when Recommended Inputs are Used
AC_IDLE	input	Turns off Analog Comparator when true	1	Depends upon µC code being executed
ACO	output	Analog Comparator Output	Will become input to µC code being executed	0
ACME	input	Uses output signal from ADC mux when true	0	Depends upon µC code being executed
ACBG	input	Bandgap Reference enable	0	Depends upon µC code being executed

29.3 Power-down Supply Current

Figure 29-12. Power-down Supply Current vs. Vcc (Watchdog Timer Disabled) - Temp. = 25°C

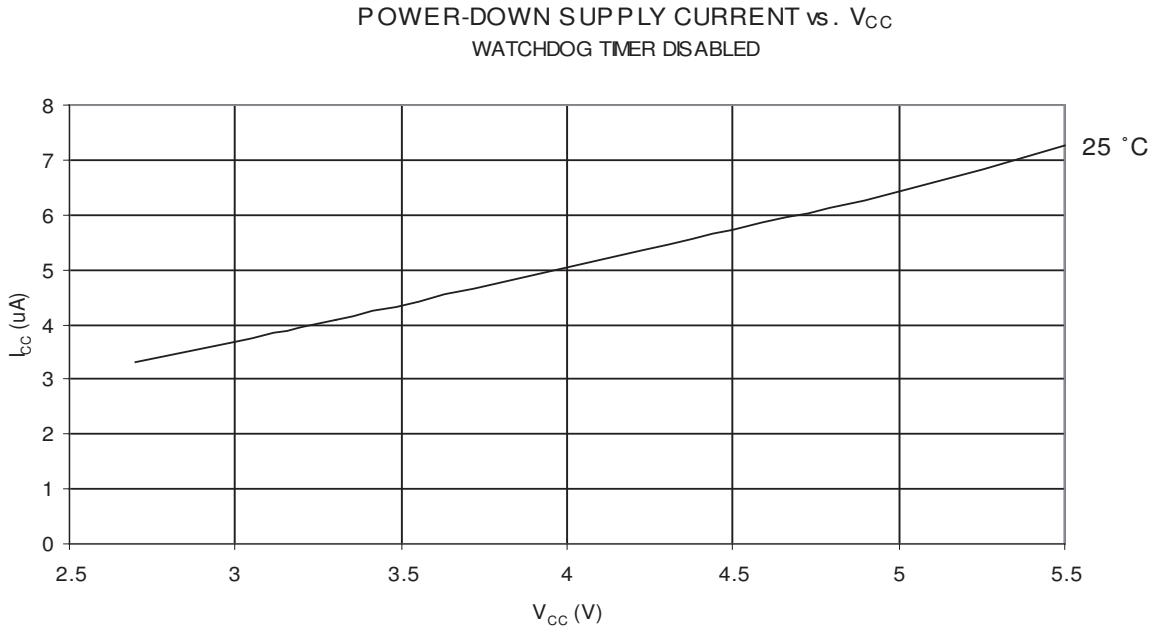
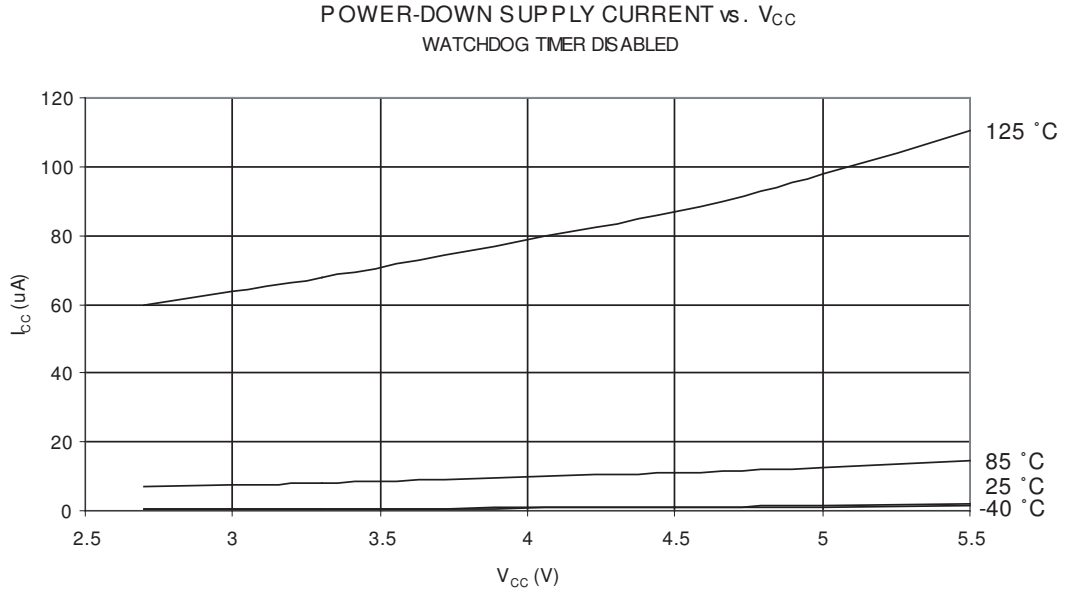


Figure 29-13. Power-down Supply Current vs. Vcc (Watchdog Timer Disabled) - Temp.= 125°C



29.10 Current Consumption of Peripheral Units

Figure 29-33. Brownout Detector Current vs. Operating Voltage

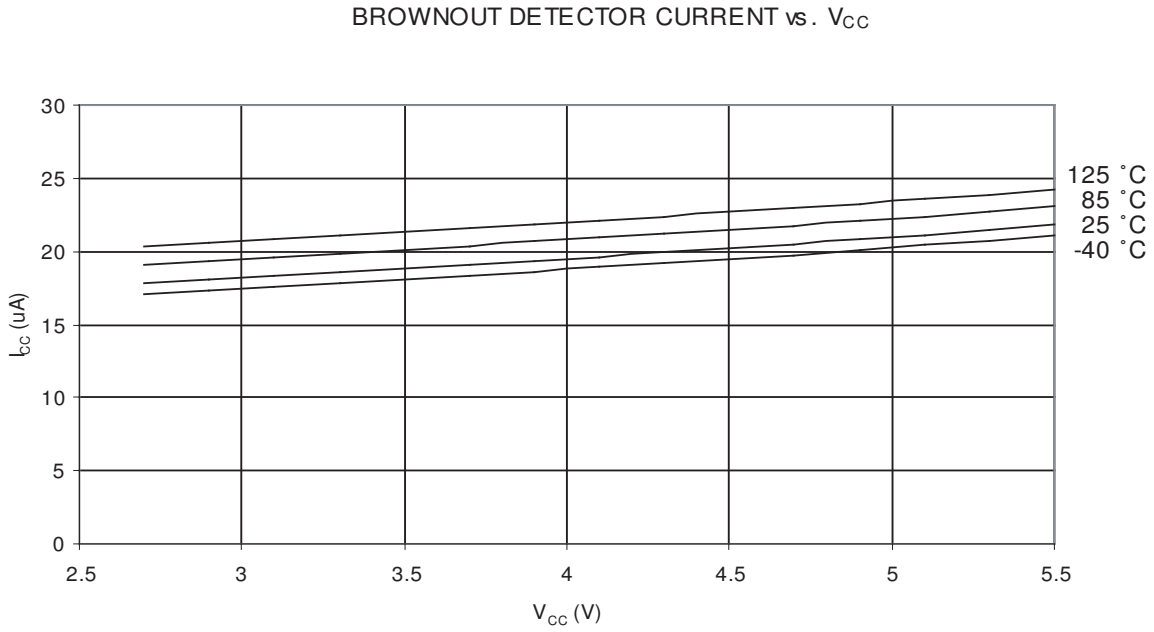


Figure 29-34. AREF External Reference Current vs. Operating Voltage

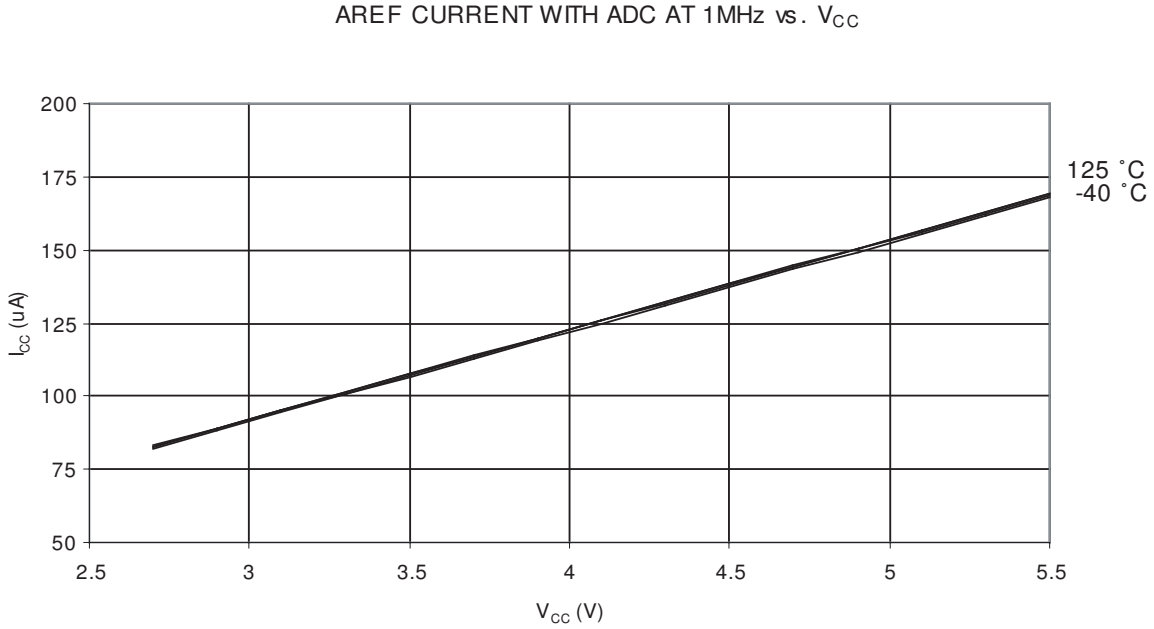
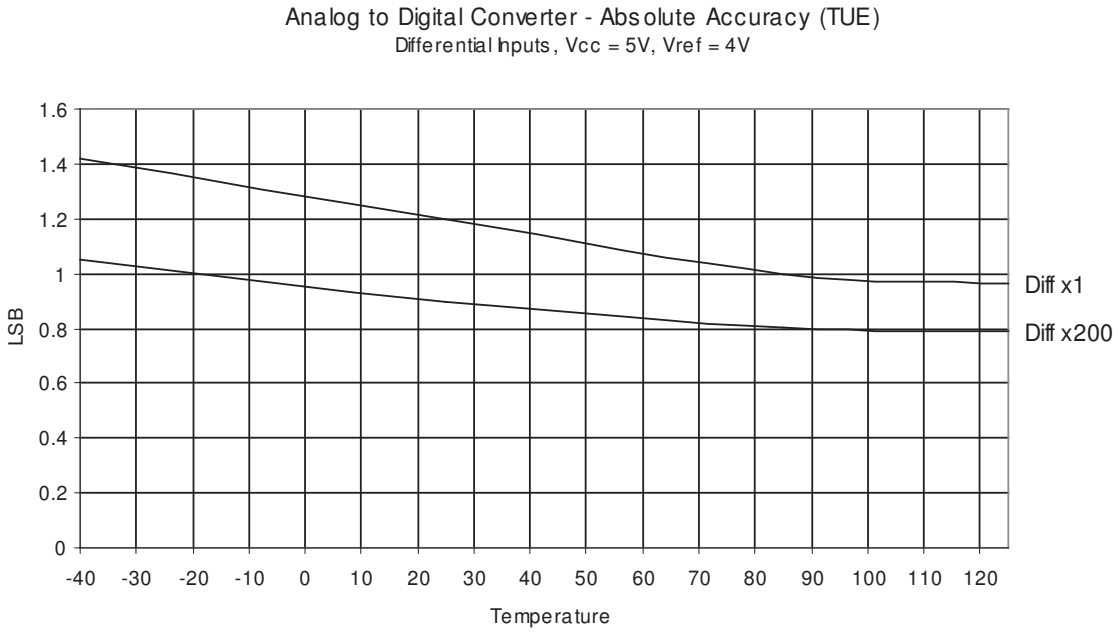


Figure 29-49. Absolute Accuracy (TUE), Differential Inputs



4. CAN acknowledge error in 3-sample mode with prescaler =1

Some acknowledge errors can occur when the clock prescaler = 1 (BRP[5..0] = 0 in CANBTR1 register) and the SMP bit is set (CANBTR3[0] = 1 in CANBTR3 register). That can result in a reduction of the maximum length of the CAN bus.

Problem fix / workaround

If BRP[5..0]=0 use SMP=0.

3. CAN transmission after 3-bit intermission

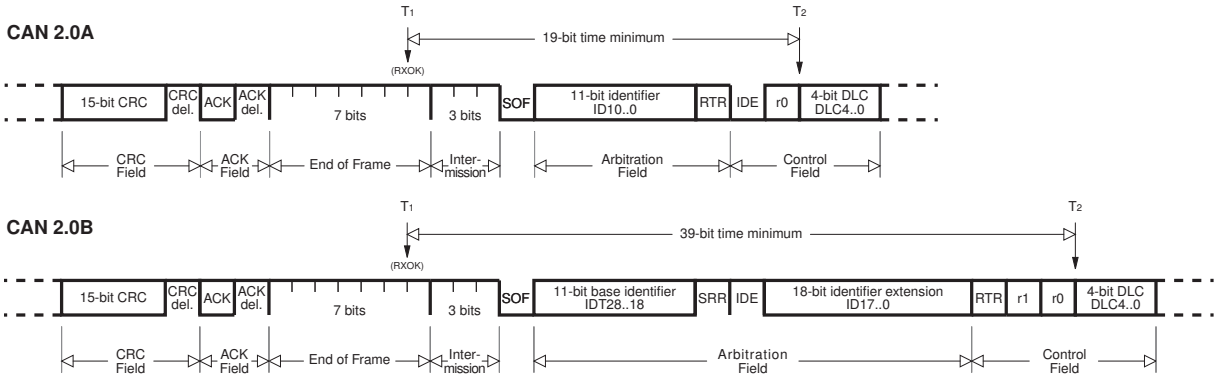
If a Transmit Message Object (MOb) is enabled while the CAN bus is busy with an on going message, the transmitter will wait for the 3-bit intermission before starting its transmission. This is in full agreement with the CAN recommendation.

If the transmitter lost arbitration against another node, two conditions can occur:

- At least one receive MOb of the chip are programmed to accept the incoming message. In this case, the transmitter will wait for the next 3-bit intermission to retry its transmission.
- No receive MOb of the chip are programmed to accept the incoming message. In this case the transmitter will wait for a 4-bit intermission to retry its transmission. In this case, any other CAN nodes ready to transmit after a 3-bit intermission will start transmit before the chip transmitter, even if their messages have lower priority IDs.

Problem fix / workaround

Always have a receive MOb enabled ready to accept any incoming messages. Thanks to the implementation of the CAN interface, a receive MOb must be enable at latest, before the 1st bit of the DLC field. The receive MOb status register is written (RXOK if message OK) immediately after the 6th bit of the End of Frame field. This will leave in CAN2.0A mode a minimum 19-bit time delay to respond to the end of message interrupt (RXOK) and re-enable the receive MOb before the start of the DLC field of the next incoming message. This minimum delay will be 39-bit time in CAN2.0B. See CAN2.0A CAN2.0B frame timings below.



Workaround implementation

The workaround is to have the last MOb (MOb14) as "spy" enabled all the time; it is the MOb of lowest priority. If a MOb other than MOb14 is programmed in receive mode and its acceptance filter matches with the incoming message ID, this MOb will take the message. MOb14 will only take messages than no other MObs will have accepted. MOb14 will need to be re-enabled fast enough to manage back to back frames. The deadline to do this is the beginning of DLC slot of incoming frames as explained above.

Minimum code to insert in CAN interrupt routine: