Application Note, V1.0, Jul. 2006

AP16095

XC16x

Interfacing the XC16x Microcontroller to a Serial SPI EEPROM

Microcontrollers

Never stop thinking

Edition 2006-07-10

Published by
Infineon Technologies AG
81726 München, Germany

AP16095		
Revision History:	2006-07	V1.0
Previous Version:	none	
V1.0	Initial release	

Table of Contents Page

1 Introduction

Embedded microcontroller applications may require data to be stored and maintained, even in the case of complete power loss. One method for providing this type of non-volatile data storage is through the use of an Electronically Erasable Programmable ROM (EEPROM). To store and retrieve data from the EEPROM IC, a communication bus is required. One commonly used interface is referred to as a Serial Peripheral Interface (SPI). Infineon starter kits for the XC16x series of microcontrollers include a SPI EEPROM, which may be used as a development platform for SPI EEPROM software. This document describes the steps necessary to interface to the SPI EEPROM IC. Example code is provided, which demonstrates read and write operations to the EEPROM, using efficient methods such as the PEC data transfer feature of the XC16x microcontrollers.

1.1 Overview

The overall objective is to provide functionality to read and write data to the EEPROM IC. In order to achieve this, both the command set of the EEPROM IC, as well as the SPI bus interface must be implemented. One possible implementation is shown in the block diagram of Figure 1.

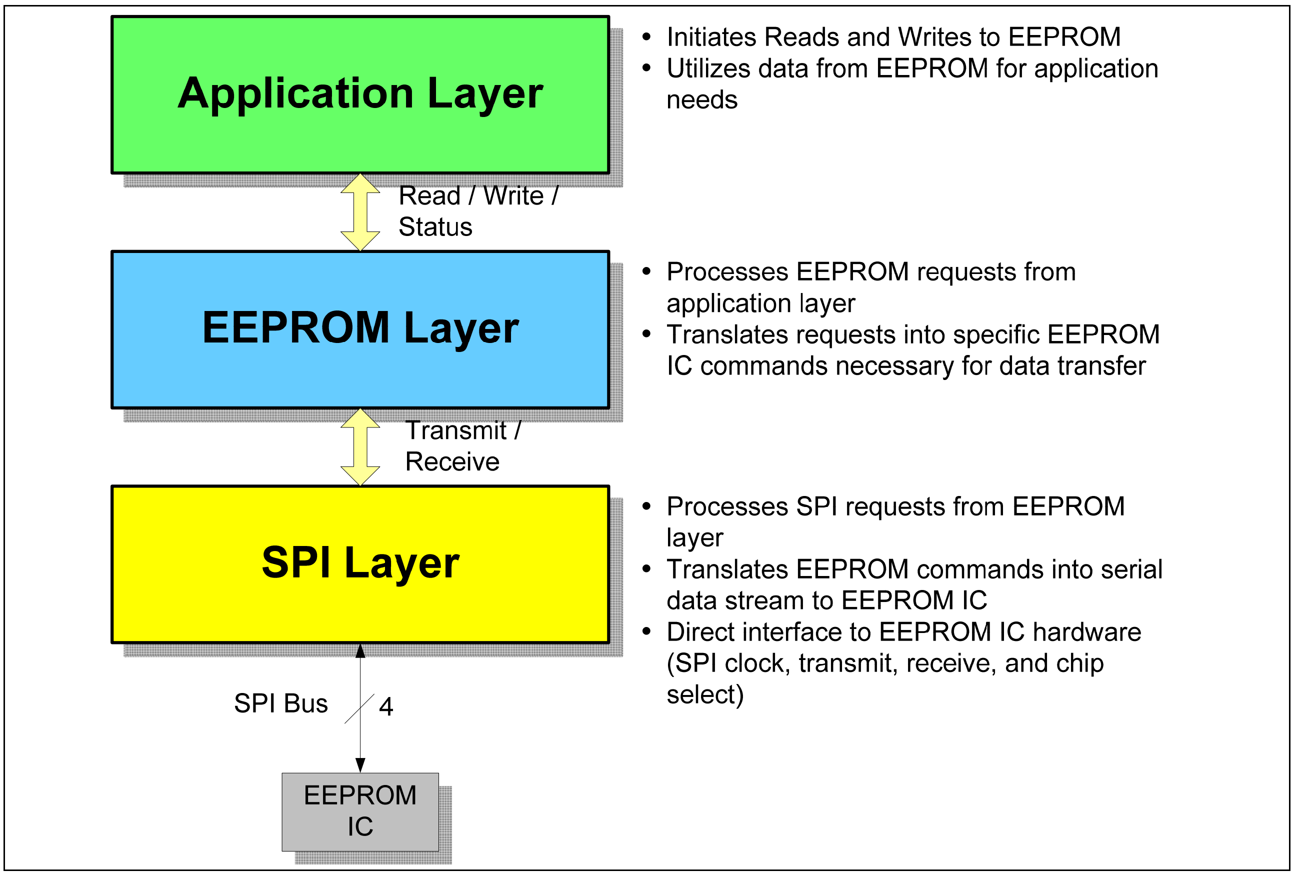

Application Layer
Initiates Reads and Writes to EEPROM
Utilizes data from EEPROM for application needs
Read / Write / Status
EEPROM Layer
Processes EEPROM requests from application layer
Translates requests into specific EEPROM IC commands necessary for data transfer
Transmit / Receive
SPI Layer
Processes SPI requests from EEPROM layer
Translates EEPROM commands into serial data stream to EEPROM IC
Direct interface to EEPROM IC hardware (SPI clock, transmit, receive, and chip select)
SPI Bus
4
EEPROM IC

Figure 1 EEPROM Functional Interface

This implementation uses a layered approach. At the application layer, the software interface provides functionality to read and write data to any arbitrary location in the EEPROM. At the EEPROM layer, the command set of the EEPROM IC is supported. And finally, at the SPI layer, support is provided to transmit and receive data across the SPI bus. The software is layered in such a way to provide flexibility. For example, the EEPROM layer can be modified to support other EEPROM ICs and instruction sets, without the need to modify the other layers. Similarly, the SPI layer can be expanded upon to support other SPI devices (for example, shift registers, power devices, etc.).

2 Hardware Overview

2.1 XC16x Starter Kit and EEPROM Connections

The XC16x starter kit comes with an Atmel AT25128 SPI serial EEPROM installed. The EEPROM is connected to Synchronous Serial Channel 0 (SSC0), which is a SPI compatible peripheral of the XC16x microcontroller. The EEPROM connections are shown in Figure 2.

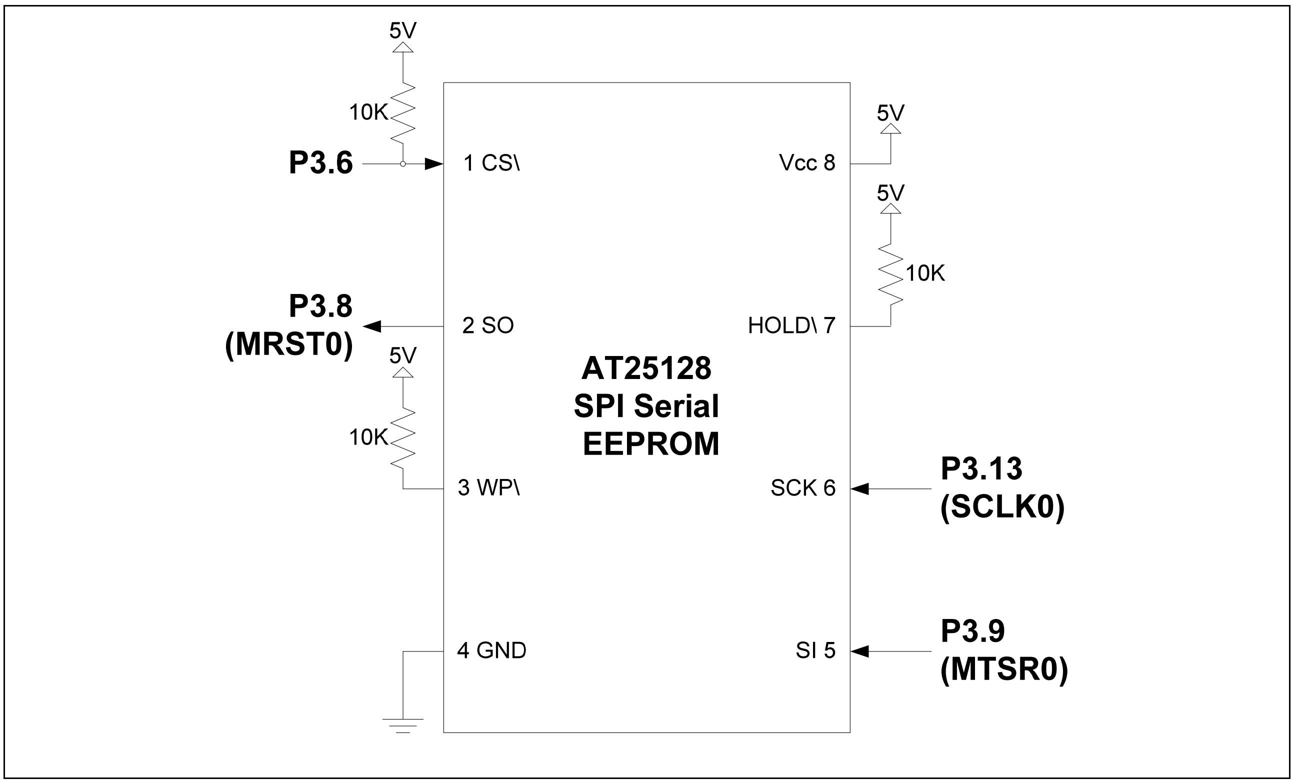


Figure 2 EEPROM IC Connections

Port pin P3.6 is a general purpose I/O pin, which is used to control the chip select signal to the EEPROM IC. Port pins P3.8, P3.9, and P3.13 are associated with the SSC0 peripheral. The naming convention for SPI signals varies, and so the SPI signal names used by the XC16x microcontroller and Atmel EEPROM IC are shown in Table 1.

Table 1 SPI Signal Naming

Microcontroller Port Pin	Microcontroller Name	EEPROM IC Name
P3.8	Master Receive Slave Transmit (MRST0)	Serial Output (SO)
P3.9	Master Transmit Slave Receive (MTSR0)	Serial Input (SI)
P3.13	Serial Clock (SCLK0)	Serial Clock (SCK)

2.2 SSC Microcontroller Peripheral

The Synchronous Serial Channel (SSC) peripheral provides a SPI compatible interface. The microcontroller may contain more than one SSC peripheral, and so the SSC peripheral name is followed by a sequential number, beginning with zero. In this particular case, the EEPROM IC is connected to the first SSC peripheral, and so only port pins related to SSC0 are discussed. Other than I/O connections, the features of SSC0 and SSC1 are identical.

The SSC peripheral offers a large number of features, which allows connection to a wide variety of SPI devices:

- Master and Slave Mode operation
  - Full-duplex or half-duplex operation
- Flexible data format
  - Programmable number of data bits: 2 to 16 bits
  - Programmable shift direction: LSB or MSB shift first
  - Programmable clock polarity: idle low or high state for the shift clock
  - Programmable clock/data phase: data shift with leading or trailing edge of the shift clock
- Baudrate generation from 20 Mbit/s to 306.6 bit/s (@ 40 MHz module clock)
- Interrupt generation
  - On a Transmitter-Empty condition
  - On a Receiver-Full condition
  - On an Error condition (receive, phase, baudrate, transmit error)

A block diagram of the SSC is shown in Figure 3.

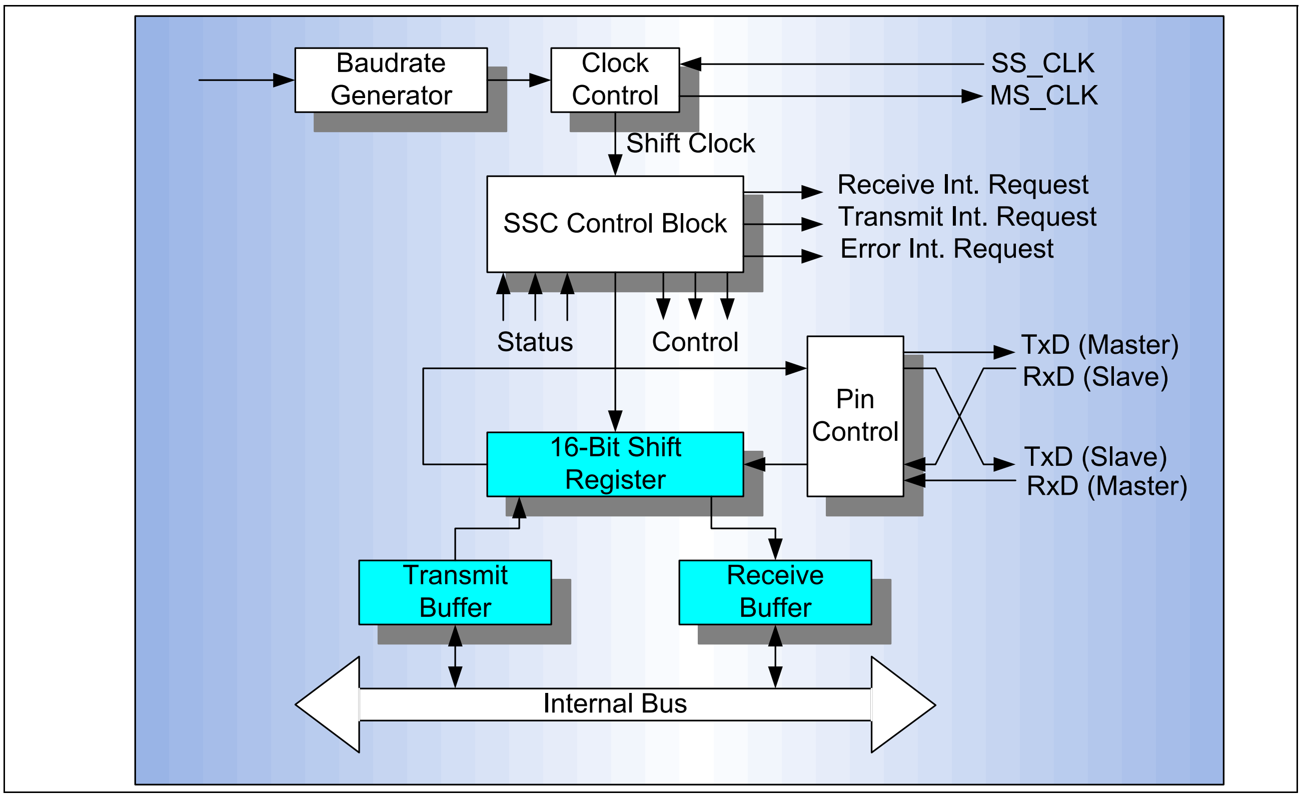


Figure 3 Synchronous Serial Channel (SSC) Block Diagram

For communication with an EEPROM, the microcontroller always acts as the master device on the SPI bus, supplying the serial clock. Commands are shifted out of the microcontroller into the EEPROM. In the case that data is to be received by the microcontroller, the microcontroller must deliver the serial clock pulses to shift the data out of the EEPROM.

For the Atmel AT25128 EEPROM, the following SSC configuration is required:

Table 2 SSC Configuration for EEPROM

SPI Characteristic	SSC Configuration Required for EEPROM
Master/Slave Operating Mode	Master
Data Width	8 bits
Shift Direction	Most Significant Bit (MSB) First
Clock Polarity	Idle LOW
Clock Phase	Latch receive data on leading clock edge, shift on trailing edge
Baud Rate	3MHz maximum

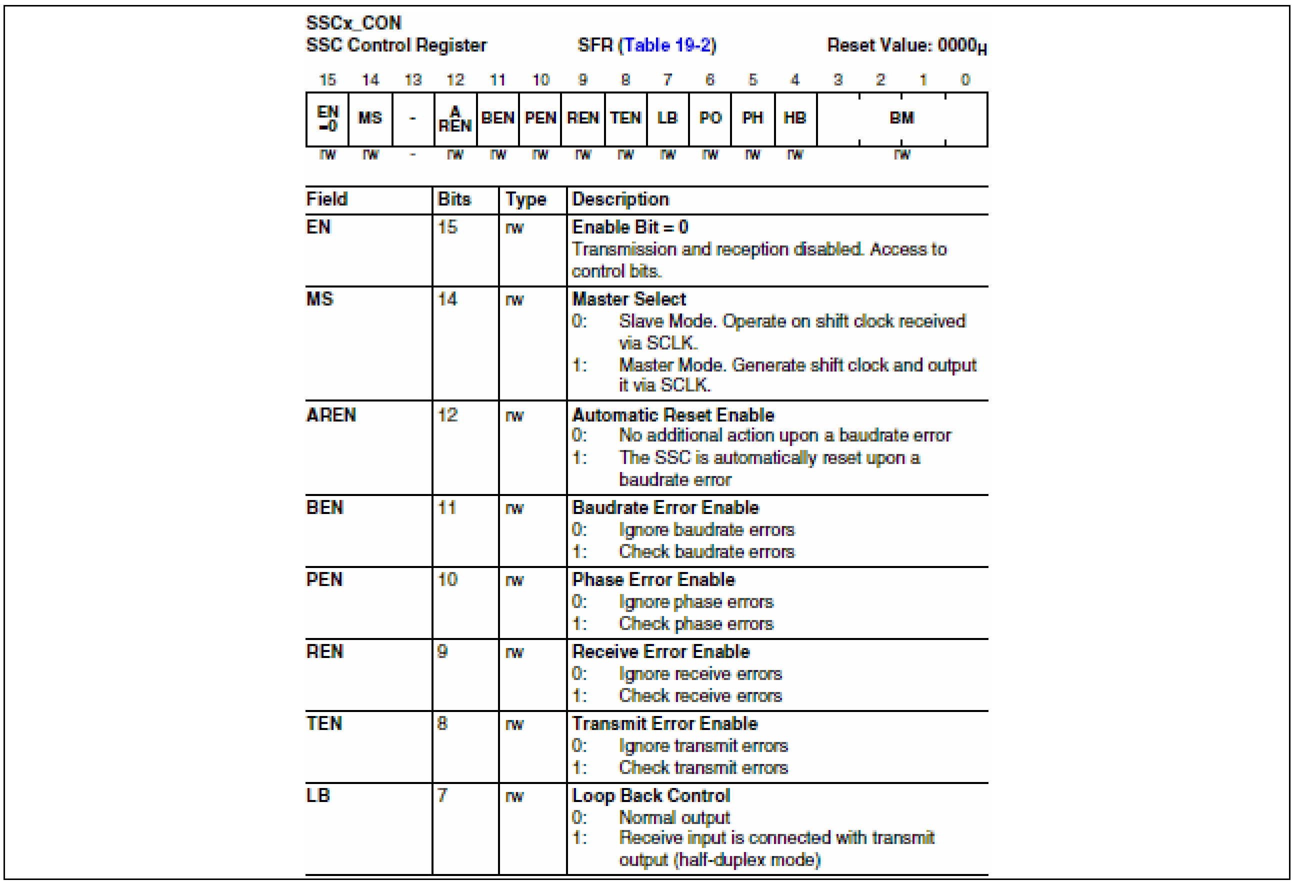
The SSC peripheral is configured through two registers: the control register (SSCx_CON) and the baud rate register (SSCx_BR).

2.2.1 SSCx_CON Register

The operating mode of the SSC peripheral is determined by the values written to the control register SSCx_CON (x = the sequential number for the SSC peripheral of interest). The control register defines all of the SPI characteristics except for the baud rate.

Attention: Please note that the SSC control register is dual purpose, serving both as control register when the SSC is disabled, and as a status register with the SSC is enabled. Therefore, the user must ensure the SSC is disabled before attempting to modify the control register settings. Once the SSC is enabled, the register provides status flags that should only be read by the user.

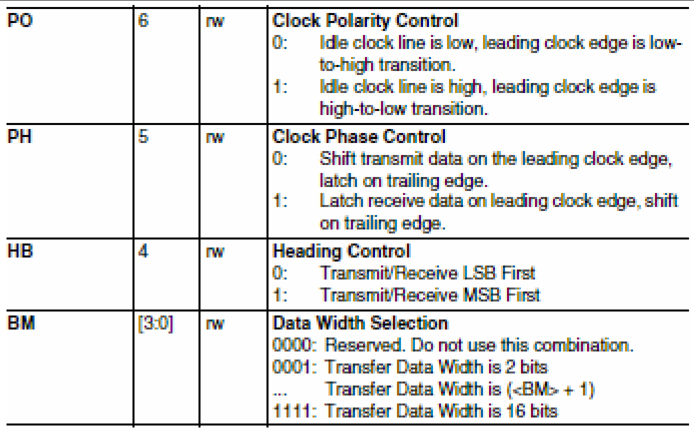
Figure 4 is an excerpt from the XC161 user manual, which describes the bitfields of the SSC control register when the SSC is disabled (enable bitfield EN = 0).



SSCx_CON
SSC Control Register SFR (Table 19-2) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN =0	MS	-	AREN	BEN	PEN	REN	TEN	LB	PO	PH	HB	BM			
rw	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

Field	Bits	Type	Description
EN	15	rw	**Enable Bit = 0** Transmission and reception disabled. Access to control bits.
MS	14	rw	**Master Select** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
AREN	12	rw	**Automatic Reset Enable** 0: No additional action upon a baudrate error 1: The SSC is automatically reset upon a baudrate error
BEN	11	rw	**Baudrate Error Enable** 0: Ignore baudrate errors 1: Check baudrate errors
PEN	10	rw	**Phase Error Enable** 0: Ignore phase errors 1: Check phase errors
REN	9	rw	**Receive Error Enable** 0: Ignore receive errors 1: Check receive errors
TEN	8	rw	**Transmit Error Enable** 0: Ignore transmit errors 1: Check transmit errors
LB	7	rw	**Loop Back Control** 0: Normal output 1: Receive input is connected with transmit output (half-duplex mode)



PO	6	rw	**Clock Polarity Control** 0: Idle clock line is low, leading clock edge is low-to-high transition. 1: Idle clock line is high, leading clock edge is high-to-low transition.
PH	5	rw	**Clock Phase Control** 0: Shift transmit data on the leading clock edge, latch on trailing edge. 1: Latch receive data on leading clock edge, shift on trailing edge.
HB	4	rw	**Heading Control** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
BM	[3:0]	rw	**Data Width Selection** 0000: Reserved. Do not use this combination. 0001: Transfer Data Width is 2 bits ... Transfer Data Width is (<BM> + 1) 1111: Transfer Data Width is 16 bits

Figure 4 SSC Control Register

The clock polarity and clock phase bitfields allow the SPI clock signal to be adjusted to meet the requirements of the device connected to the microcontroller. The data sheets of SPI devices typically list the required settings, but the exact definition of these bitfields may vary between microcontroller manufactures. These settings should be checked carefully to ensure they meet the requirements of the interfacing device. The definition of these bitfields for the SSC peripheral is shown in Figure 5.

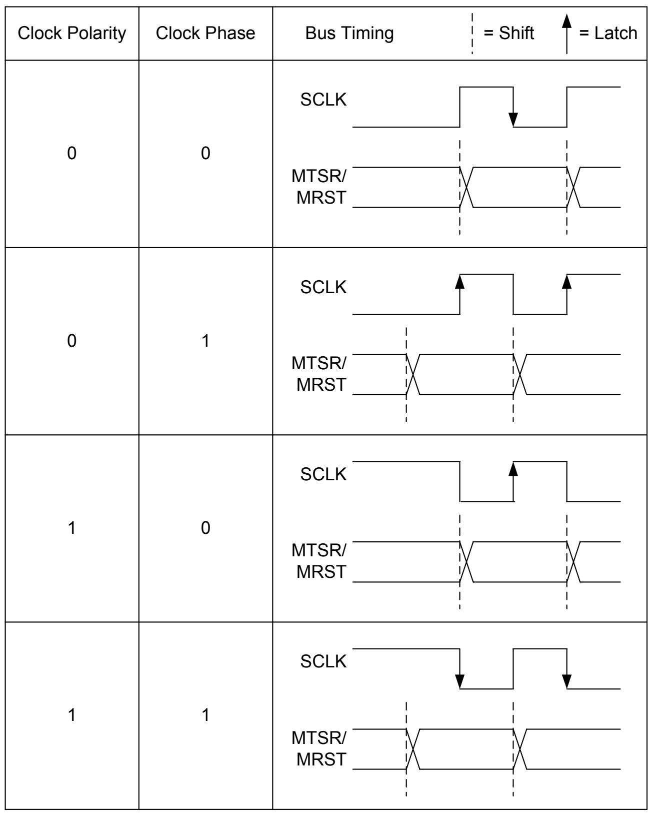


Figure 5 SSC Clock Phase and Polarity Settings

2.2.2 SSC Baud Rate Register

The SSC baud rate register controls the SPI clock rate. The baud rate register is 16-bits wide, and is used as a reload value for a baud rate counter, as shown in Figure 6.

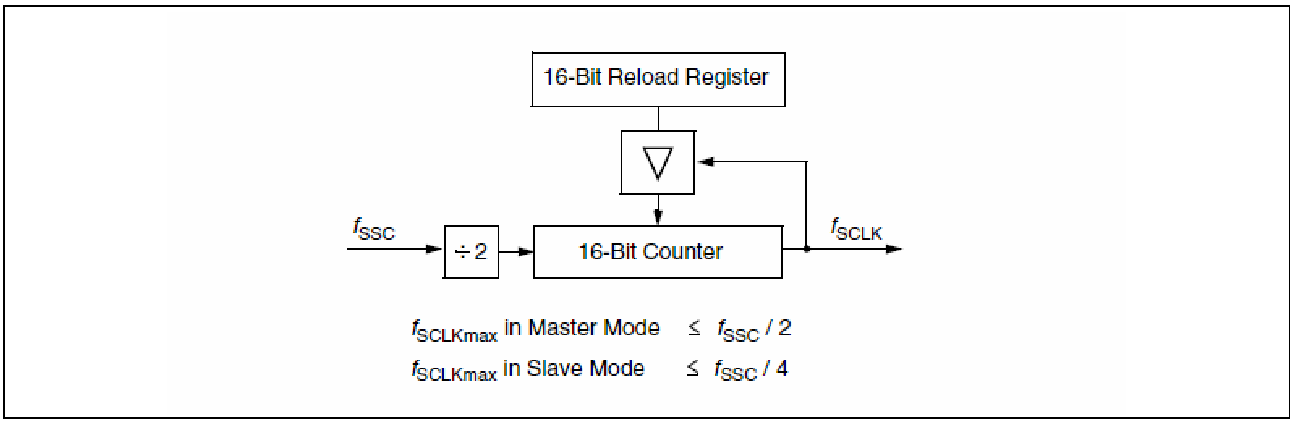

16-Bit Reload Register
f_{SSC}
÷2
16-Bit Counter
f_{SCLK}
$f_{SCLKmax}$ in Master Mode ≤ f_{SSC} / 2
$f_{SCLKmax}$ in Slave Mode ≤ f_{SSC} / 4

Figure 6 SSC Baud Rate Generation

The baud rate is calculated as:

$$BR = \frac{fssc}{(2 \times BaudRate)} - 1$$

where

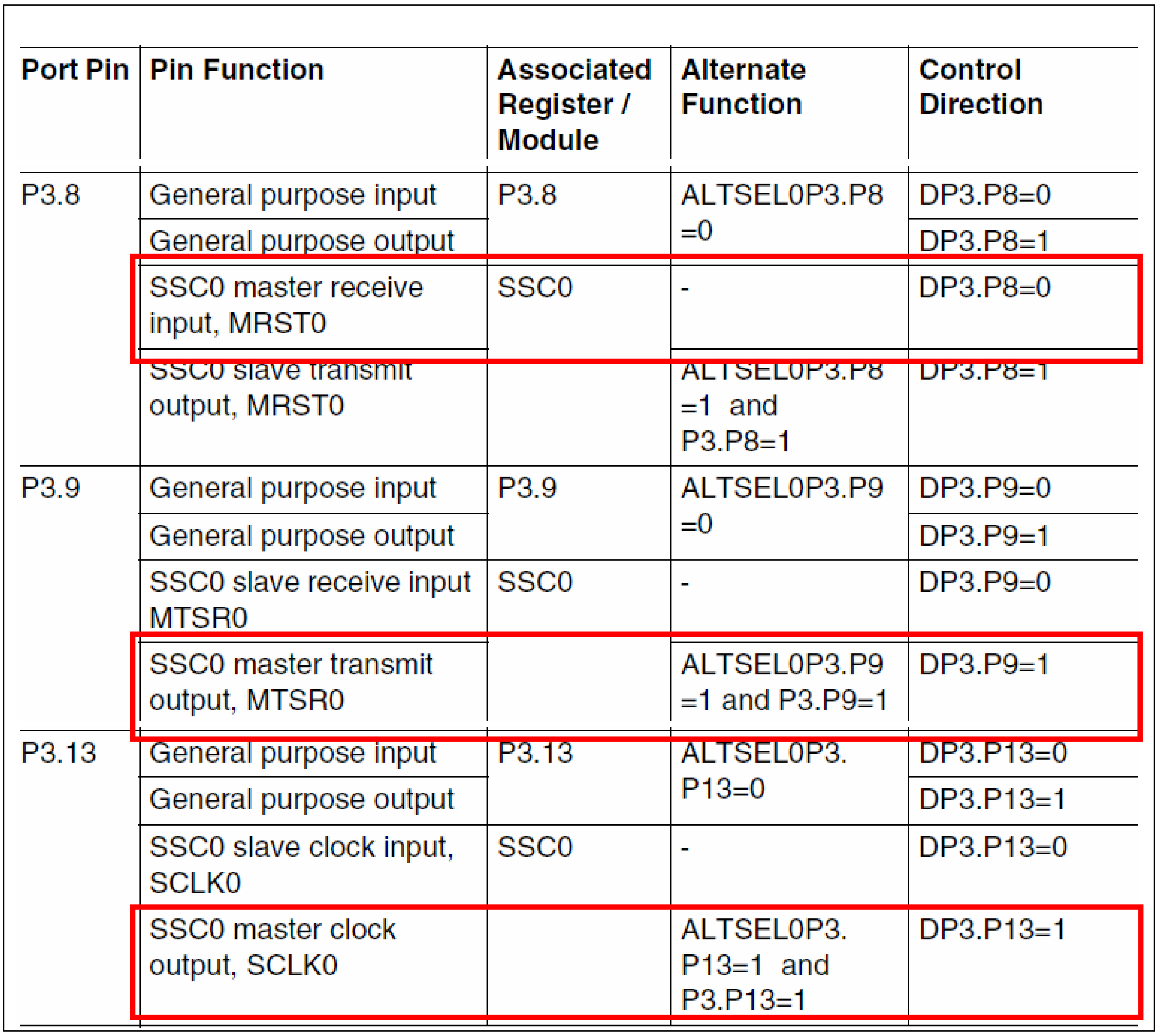
- *BR* represents the contents of the baud rate register, taken as an unsigned 16-bit integer
- f_{SSC} represents the internal frequency of the SSC peripheral (which is equal to f_{SYS})
- *BaudRate* is the desired baud rate

2.2.3 SSC I/O Configuration

The I/O associated with the SSC peripheral must be configured to route the signals through the appropriate microcontroller pins. The master receive slave transmit (MRST) pin must be configured as an input. The chip select (CS), SPI clock (SCLK), and master transmit slave receive (MTSR) pins must all be configured as outputs. The port direction is controlled by register DPx, where x is the port number. Setting a bit to “0” within the port direction register configures it as an input, and “1” configures it as an output.

In addition, the output pins which will be driven directly by the SSC peripheral (SCLK and MTSR) require that the port be configured to use the alternate output functionality, in place of the general purpose I/O. The alternate output functionality is controlled by the alternate select registers ALTSELnPx, where n selects the appropriate control register (0 or 1), and x is the port number. The alternate select registers should be configured as described in the parallel ports section of the user manual. Figure 7 is an excerpt from the parallel ports section of the XC161 user manual, which summarizes the required I/O configuration for using SSC0.

It should also be noted that for SSC0, the outputs are routed to port 3, which has a special output structure for alternate outputs. Alternate outputs of port 3 are logically “ANDed” with the port latch data. Therefore, to achieve the alternate functionality at the port pin, the corresponding bits of the port latch register P3 should be set to “1”.



Port Pin	Pin Function	Associated Register / Module	Alternate Function	Control Direction
P3.8	General purpose input	P3.8	ALTSEL0P3.P8 =0	DP3.P8=0
	General purpose output			DP3.P8=1
	SSC0 master receive input, MRST0	SSC0	-	DP3.P8=0
	SSC0 slave transmit output, MRST0		ALTSEL0P3.P8 =1 and P3.P8=1	DP3.P8=1
P3.9	General purpose input	P3.9	ALTSEL0P3.P9 =0	DP3.P9=0
	General purpose output			DP3.P9=1
	SSC0 slave receive input MTSR0	SSC0	-	DP3.P9=0
	SSC0 master transmit output, MTSR0		ALTSEL0P3.P9 =1 and P3.P9=1	DP3.P9=1
P3.13	General purpose input	P3.13	ALTSEL0P3. P13=0	DP3.P13=0
	General purpose output			DP3.P13=1
	SSC0 slave clock input, SCLK0	SSC0	-	DP3.P13=0
	SSC0 master clock output, SCLK0		ALTSEL0P3. P13=1 and P3.P13=1	DP3.P13=1

Figure 7 SSC0 I/O Configuration

2.2.4 SSC Configuration Sequence

In order to ensure proper SSC configuration, the following sequence is recommended:

1. Disable SSC (SSCx_CON bitfield EN = 0)
2. Program the desired baud rate (SSCx_BR = desired value)
3. Program control register with desired SPI settings, and enable SSC (note that enable bitfield is part of SSCx_CON)
4. Configure the I/O by programming the alternate select, port latch, and port direction registers

2.3 EEPROM IC

As previously mentioned, the XC16x starter kit includes an Atmel AT25128 EEPROM. This EEPROM provides 16KB of byte-addressable data storage. Instructions are given from the microcontroller to the EEPROM through the SPI interface. The microcontroller always initiates the SPI communications, and must provide the SPI clock signal for any data transmitted to or from the EEPROM IC. The complete details of the EEPROM IC are given in the Atmel AT25128 data sheet, but some highlights are provided here.

2.3.1 SPI Timing Diagram

The general format for an SPI transmission to the EEPROM IC is shown below.

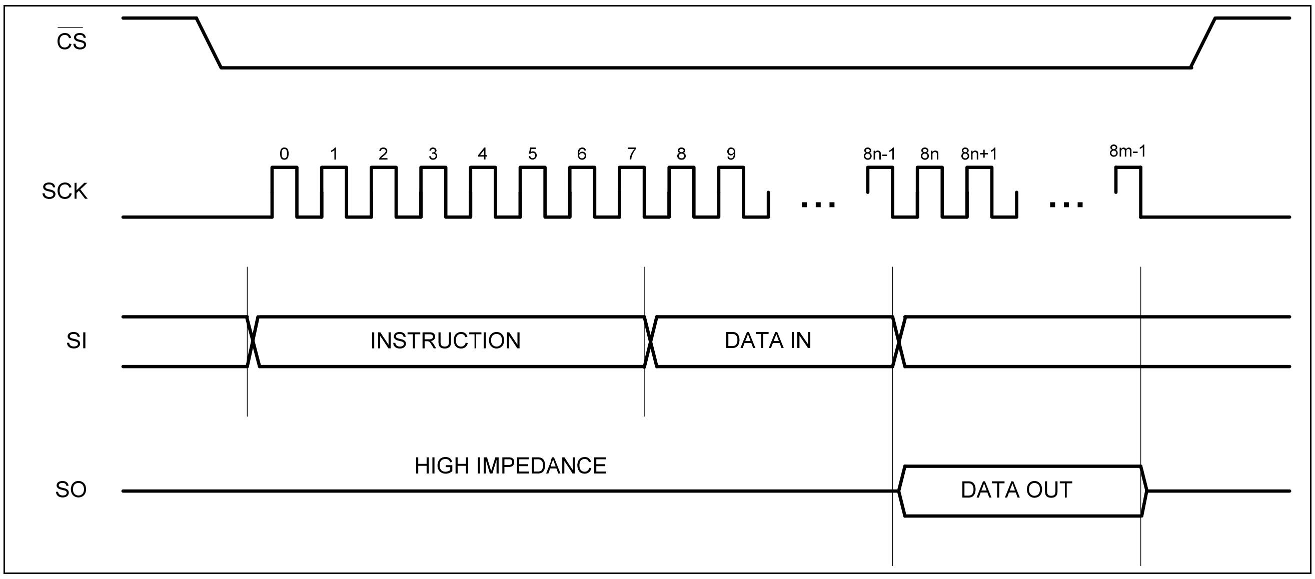


Figure 8 General SPI Timing for EEPROM IC

All communications to the EEPROM IC are started by asserting the chip select line, which is accomplished by bringing the chip select signal to a low level. SPI traffic is ignored by the EEPROM IC while the chip select signal is high. Data at the serial input (SI) to the EEPROM IC is latched with the rising edge of the SPI clock, and therefore, the SI data is presented by the microcontroller a half clock cycle ahead of the rising clock edge.

Following the chip selection, the first SI data clocked into the EEPROM IC is the instruction. The EEPROM IC instructions include READ, WRITE ENABLE, WRITE, and READ STATUS REGISTER, as well as others. The instructions are all 8 bits in length.

The SI data that follows the instruction is dependent on the type of instruction executed. For example, for READ and WRITE instructions, the next two bytes represent an address within the EEPROM. In some cases, there may even be no data following the instruction, such as the WRITE ENABLE instruction.

SO data may be clocked out of the EEPROM IC, again depending upon the type of instruction. To clock the data out of the EEPROM, the microcontroller must deliver the SPI clock pulses. During this time, the SI data bits clocked into the EEPROM IC are “don’t care” values.

Finally, the chip selection is deasserted by bringing signal to a high level. This completes the instruction sequence.

2.3.2 Read Instruction

Data can be read one byte at a time, or for any arbitrary number of bytes as long as the chip select signal remains asserted and SPI clock pulses are delivered to the EEPROM IC. The two data bytes following the READ instruction represent the address to be read. An internal address counter is automatically incremented after each byte is read, allowing multiple bytes to be read with a single READ instruction.

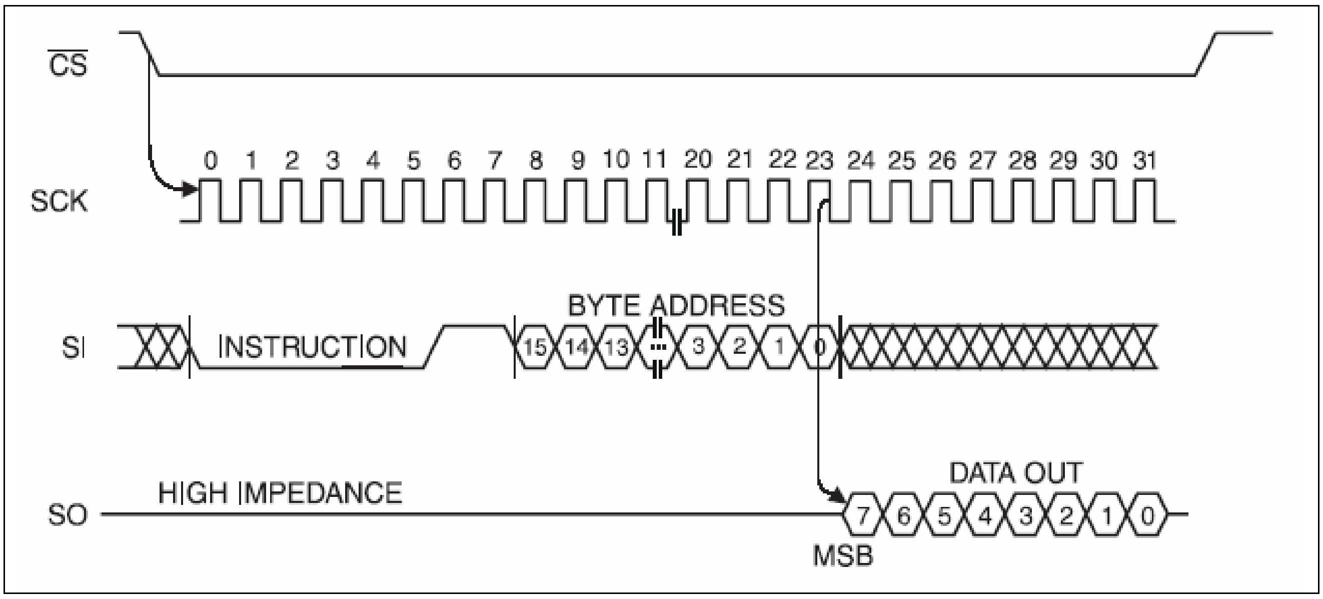


Figure 9 Read instruction Timing

2.3.3 Write Instruction

Data can be written one byte at a time, or up to 64 bytes at a time using page mode. The two data bytes following the WRITE instruction represent the address to be written. Note that when page mode is used, each byte of data that is received causes an internal address counter to be automatically increment by one. The internal address counter is only 6 bits wide, and so the address will “wrap around” when reaching a 64 byte boundary. The user must ensure that this “wrap around” effect is handled properly.

Before writing data to the EEPROM, the EEPROM must be placed in the “write enable” state. The write sequence is therefore typically composed of two instructions in succession: a WRITE ENABLE instruction, followed immediately by a WRITE instruction. The “write enable” state is cleared by the write operation, and a new WRITE ENABLE instruction is required before each write operation.

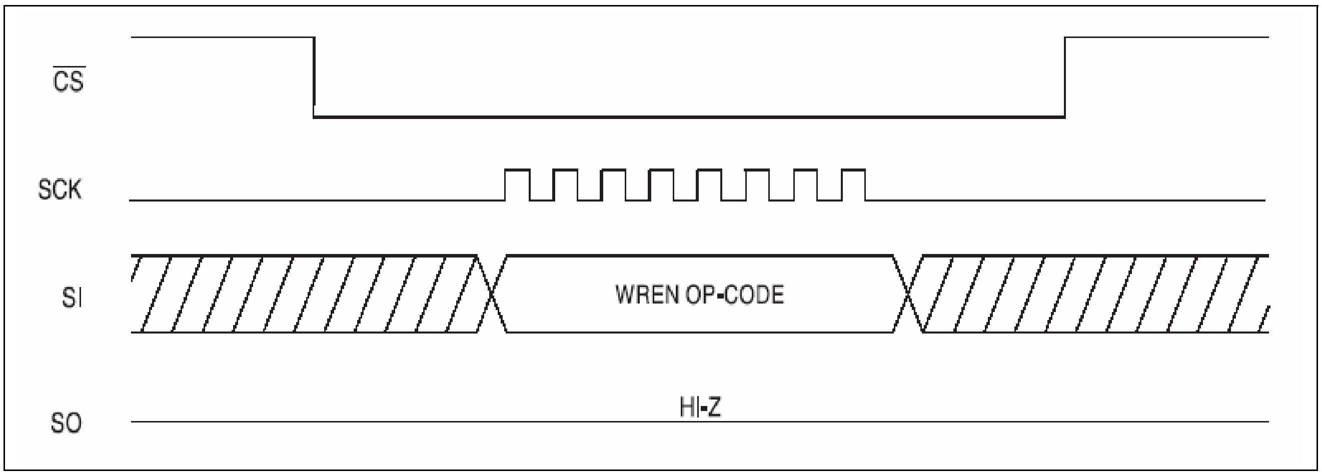


Figure 10 Write Enable instruction timing

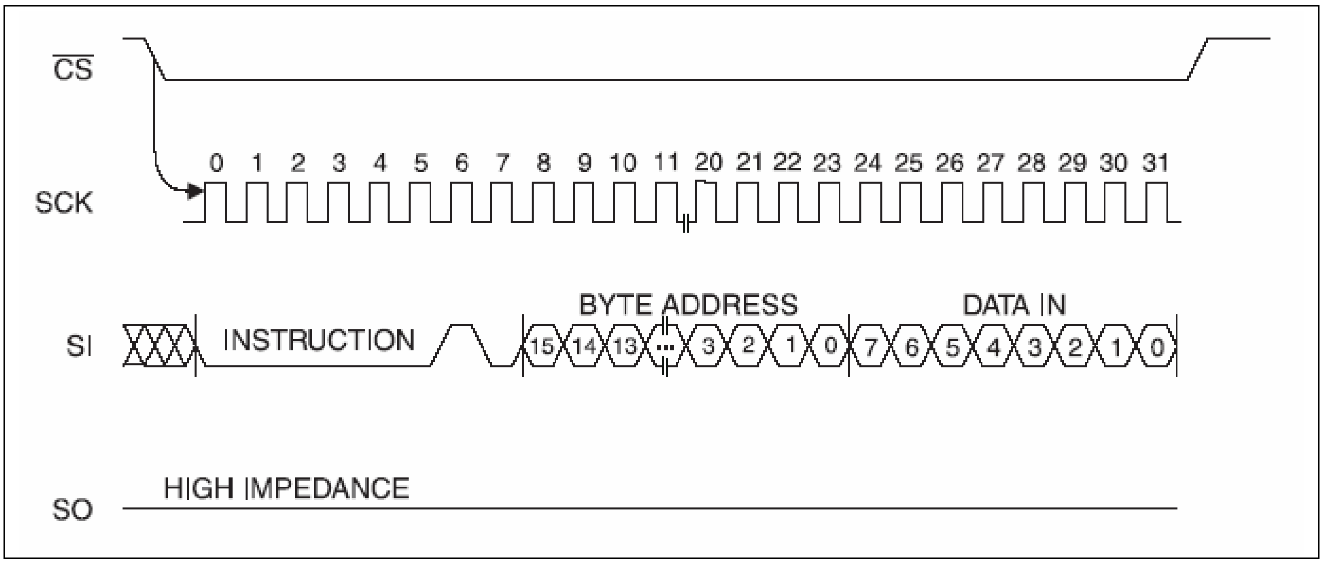


Figure 11 Write Instruction Timing

2.3.4 Read Status Register Instruction

Following the WRITE instruction sequence, the EEPROM begins a self-timed write cycle which commits the data to memory. During the write cycle, only the read status register command is recognized. The user can poll the status register to determine when the write cycle is complete.

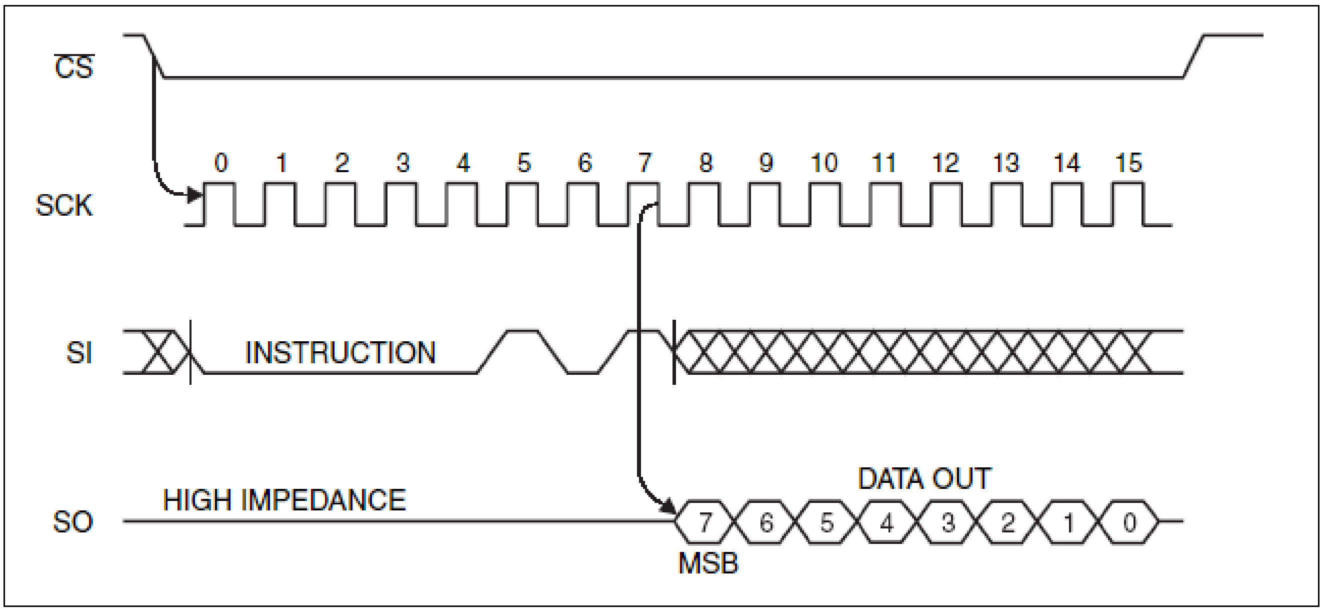


Figure 12 Read Status Register Instruction Timing

3 Software Implementation

This application note includes example software which demonstrates an SPI EEPROM interface. The example software was developed for use with the Altium Tasking C166 v8.6r1 compiler, and an XC16x starter kit. The starter kit includes an Atmel 25128 SPI EEPROM, which is exercised by the example software.

3.1 Application Layer

The application layer makes use of the EEPROM data, and must manage the initialization and requests made to the EEPROM layer. In the example code included with this application note, a RAM copy of the EEPROM data is maintained by the application layer. During initialization, data is read from the EEPROM and copied to RAM. A special initialization value (0xAA55) is stored in the EEPROM to indicate the EEPROM has been initialized. If a read of the EEPROM indicates this value is not present, it is assumed that this is the first time the program has been run, and all EEPROM values are initialized by writing default values.

For the purposes of an example, the application layer which has been implemented is very simplistic. After initializing the EEPROM data, the application code simply increments a cycle counter, and writes it back to the EEPROM. After this operation, the code enters an endless loop.

As mentioned previously, a layered approach was used for the software implementation. The application layer interacts with the EEPROM layer, but not directly with the SPI layer. From the point of view of the application layer, function calls are made to the EEPROM layer to perform read, write, and status operations, without any knowledge of how the lower layers perform these operations.

3.2 EEPROM Layer

The EEPROM layer is responsible for processing EEPROM requests from the application layer. The EEPROM layer translates these application requests into the required instruction sequences required by the EEPROM IC. The EEPROM instruction set is contained within this layer. The EEPROM layer relies upon the SPI layer to perform the actual SPI transmit and receive operations.

3.2.1 EEPROM Layer Interface

The EEPROM layer provides the following interface functions for communicating with the application layer.

3.2.1.1 InitEEPROM

Function name:	InitEEPROM
Syntax:	void InitEEPROM(void)
Parameters (in):	None
Parameters (out):	None
Description:	Performs any initialization required by EEPROM

3.2.1.2 WriteEEPROM

Function name:	WriteEEPROM	
Syntax:	void WriteEEPROM(U16 address, U16 numBytes, U8 *src)	
Parameters (in):	address	EEPROM address to be written
	numBytes	the number of bytes to be written
	src	pointer to address of data to be written

Parameters (out):	None
Description:	Writes a number of bytes to EEPROM IMPORTANT NOTE - a page-write operation is used. The EEPROM address will wrap-around, using only the lower 6 bits of the address. It is the user's responsibility to ensure that this wrap-around addressing mode of the EEPROM is used correctly.

3.2.1.3 ReadEEPROM

Function name:	ReadEEPROM	
Syntax:	void ReadEEPROM(U16 address, U16 numBytes, U8 *dst)	
Parameters (in):	address	EEPROM address to be read
	numBytes	the number of bytes to be read
	dst	pointer to address where data will be copied to
Parameters (out):	None	
Description:	Reads a number of bytes from EEPROM	

3.2.1.4 ReadEEPROMStatus

Function name:	ReadEEPROMStatus	
Syntax:	U8 ReadEEPROMStatus(void)	
Parameters (in):	None	
Parameters (out):	Status	Returns EEPROM status byte
Description:	Reads the EEPROM status register	

3.2.1.5 PollEEPROMBusy

Function name:	PollEEPROMBusy	
Syntax:	U8 PollEEPROMBusy(void)	
Parameters (in):	None	
Parameters (out):	Status	Returns EEPROM busy status
Description:	Reads the EEPROM status register, and returns "true" if EEPROM is busy	

3.3 SPI Layer

The SPI layer is responsible for processing SPI requests from higher layers. In the example code, only the EEPROM layer makes requests of the SPI layer, but the software has been written in a generalized way to support additional SPI devices. Requests could potentially come from other higher level layers. The SPI layer translates the SPI requests into the required SPI bus signals, including the chip select of SPI devices, as well as the SCLK, MTSR, and MRST signals. The example code uses the SSC0 peripheral to implement these SPI features. To make efficient data transfers, the peripheral event controller (PEC) feature of the XC16x microcontroller is used.

3.3.1 SSC Configuration

The SPI layer performs the necessary SSC configuration to match the SPI characteristics of the SPI devices connected to the microcontroller. The example code includes macro definitions to assist in the configuration. The user simply enters the desired SSC values, and the macros convert this into the required control register settings. The user must also specify the rate of the hardware clock used in the system (8MHz in the example), along with the desire SPI baud rate (1MHz in the example). The user may also adapt the functions which are called before and after an SPI transfer (function names pre_transfer_config and post_transfer_config). These functions perform any necessary I/O configuration needed for the SPI transfer, such as the assertion and deassertion of chip select signals.

3.3.2 Peripheral Event Controller (PEC)

To provide efficient data transfers, the peripheral event controller (PEC) is used. PEC performs a single word or byte data transfer between any two locations within the microcontroller memory. The XC16x microcontroller provides 8 PEC channels. PEC transfers are triggered by an interrupt request, and perform a data transfer in place of calling an interrupt service routine. PEC transfers are quicker than an interrupt service routine, because they "steal" cycles from the CPU to make the data move. PEC transfers do not require the overhead of an interrupt routine, such as jumping and returning, pushing and popping registers, or switching the context of the register set.

Each PEC channels has a set of control registers, which select how the channel will operate. The most commonly used features are described below.

Table 3 PEC Configuration Features

PEC Feature	Description
Source Pointer	24-bit address indicating source of PEC data move
Destination Pointer	24-bit address indicating destination of PEC data move
Count	Number of transfers to be performed
Byte/Word Transfer	Width of data to be transferred (8-bit or 16-bit)
Increment Control	Selects automatic increment of source pointer, destination pointer, both, or neither after each transfer

Each PEC channel is also associated with a specific interrupt priority level and group. The default setting associates PEC channels 0-3 with interrupt level 14, groups 0-3, and PEC channels 4-7 with interrupt level 15, groups 0-3. Assuming that the PEC channel has been configured, an interrupt request occurring at one of these interrupt levels will trigger the corresponding PEC transfer, and clear the interrupt request. When performing the data transfer, the PEC count is decremented by one. When the last data transfer occurs, the count decrements from 1 to 0, and not only does the data transfer occur, but additionally, the interrupt service routine is called. This provides a convenient way to take any necessary actions after all data has been transferred, such as reconfiguration of a peripheral or modifying software flags.

The operation of a PEC channels can be understood better with the following example. Consider that a PEC channel will be used to transfer bytes received over SPI. It is desired to move the received bytes from the SSC receive buffer to a buffer in RAM. In this example, the source pointer would be configured to point to the SSC receive buffer register. The destination pointer would be configured to point to the buffer in RAM. The PEC count would be configured for the number of bytes which will be received. The byte/word transfer setting would be configured for byte-width in this case, since we are interested in byte-oriented SPI transfers. Finally, the increment control would be set to increment only the destination pointer, so that received bytes would always be pulled from the same address of the receive buffer, but then written sequentially into the RAM buffer.

Each time a byte has been received, it generates an SSC receive buffer interrupt request, which triggers the PEC data transfer. As the bytes are received, the PEC counter decrements. The counter will eventually transition from 1 to 0, causing the SSC receive interrupt routine to be called.

3.3.3 SPI Transmit and Receive Operations

Since the microcontroller is always the master in the example code, all SPI communications will always include at least a transmit operation. In the case that data must be read from a device, the SPI communications will require both a transmit and receive operation. These two operations must be handled slightly differently in software, to accommodate the additionally received data in the second case.

The example software has been implemented using two PEC channels. The first PEC channel is dedicated to reloading the SSC transmit buffer every time it is emptied by a byte being transmitted across the SPI bus. The second PEC channel is dedicated to unloading the SSC receive buffer every time it is filled by a byte being received over the SPI bus. To facilitate chip select control, the receive PEC channel is used in all cases (even when the received data is not needed), in order to generate an interrupt request, and call to an interrupt service routine which deasserts the chip select signal.

3.3.3.1 SPI Transmit Only Operations

For transmit only operations, one PEC channel is configured to load the SSC transmit buffer. The source pointer is set to a user defined RAM buffer which holds the data to be transmitted, and the destination pointer is set to the SSC transmit buffer register. Each time the transmit buffer is emptied, the PEC channel refills it with a new byte. In this way, data is always ready to be transmitted, eliminating gaps between transmitted bytes, and producing nearly 100% SPI bus utilization. The source pointer for the PEC channel is incremented after each byte transfer to step through all bytes of the RAM buffer. After all bytes have been loaded by PEC into the transmit buffer, a transmit interrupt service routine is called. In this case, there is no special action required, so the interrupt service routine is simply an empty function. It merely provides a mechanism to clear the interrupt request, which is cleared automatically in hardware when the interrupt service routine is called.

A second PEC channel is configured to unload the SSC receive buffer. Although there is no meaningful data to be received in this case, the receive PEC channel is used to count the bytes transferred across the SPI bus, and will generate a call to a receive interrupt service routine after the last byte has been received. The source pointer is set to the SSC receive buffer register. The destination pointer is set to a “dummy” register (in the example, the ZEROS register), which effectively discards the received data. Once the last byte has been received, the receive interrupt service routine is called. Within the receive interrupt service routine, the chip select signal to the SPI device is deasserted. The transmit interrupt service routine can not be used for this purpose, because this interrupt service routine occurs at the time when the last transmit buffer byte has been loaded, which is before the last byte has actually been transferred across the SPI bus.

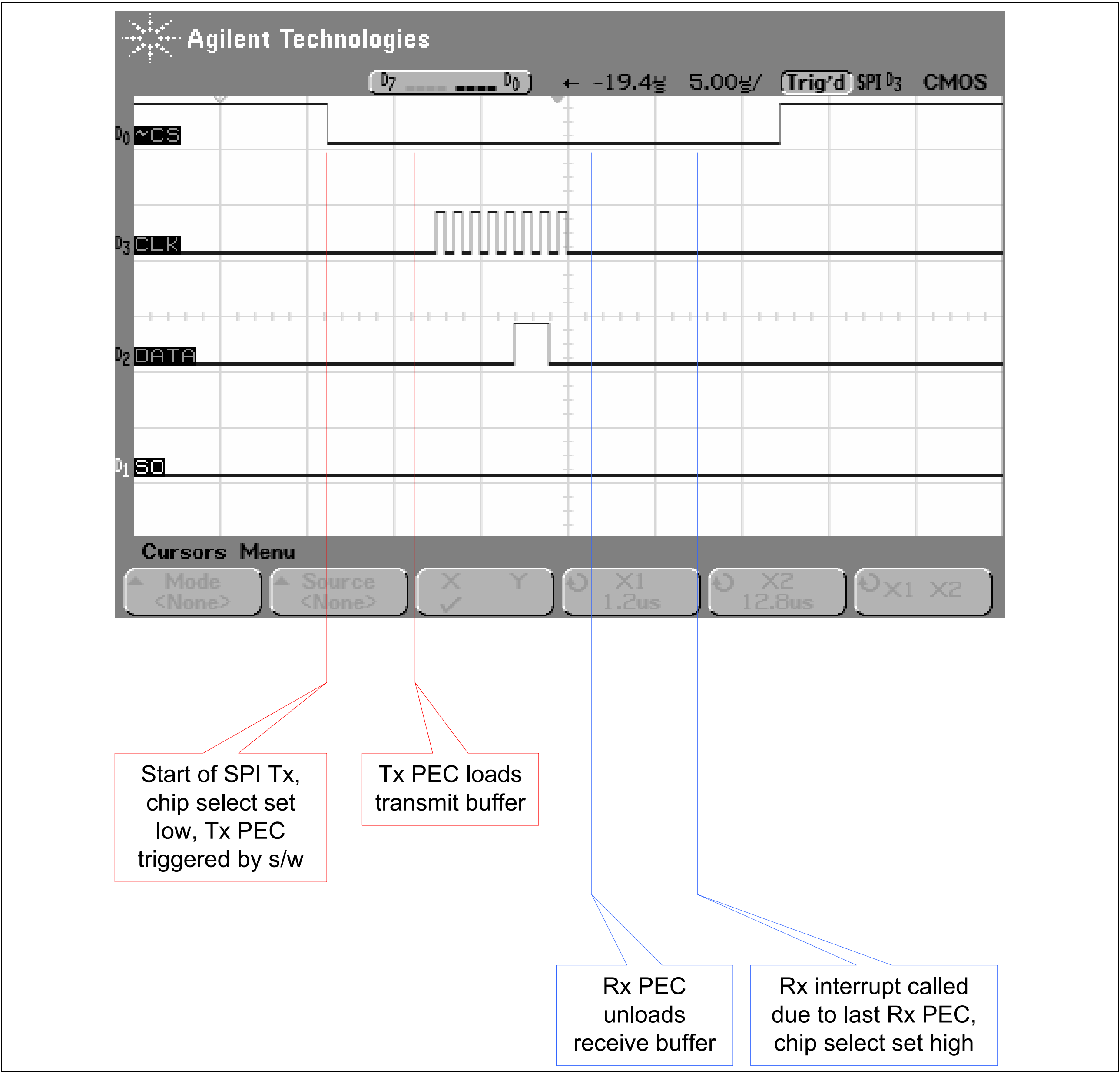

Agilent Technologies
D7 D0 ← -19.4us 5.00us/ Trig'd SPI D3 CMOS
D0 ~CS
D3 CLK
D2 DATA
D1 SO
Cursors Menu
Mode <None>
Source <None>
X Y
X1 1.2us
X2 12.8us
X1 X2
Start of SPI Tx, chip select set low, Tx PEC triggered by s/w
Tx PEC loads transmit buffer
Rx PEC unloads receive buffer
Rx interrupt called due to last Rx PEC, chip select set high

Figure 13 SPI Transmit Only Timing

3.3.3.2 SPI Transmit and Receive Operations

Transmit and receive operations are handled in two halves: a transmit operation, followed by a receive operation. The transmit operation occurs as described above in the proceeding section regarding SPI transmit only operations. One PEC channel reloads the SSC transmit buffer, while another PEC channel unloads (and discards) received bytes. At the completion of the last transmitted byte, the receive interrupt routine is called. But, in this case, the receive interrupt routine reconfigures the PEC channels to prepare for the second half of the operation.

The receive half of the operation begins in the receive interrupt service routine. The PEC channels are reconfigured, and take on a new role. The first PEC channel is reconfigured to reload the SSC transmit buffer register with “dummy” data (in the example, zeros), simply to generate the necessary SPI clock pulses to shift data out of the SPI device. The second PEC channel is reconfigured to again unload the SSC receive buffer register, but this time, the data is no longer discarded. The PEC destination pointer is now set to a user defined RAM buffer which will hold the received data.

The receive half of the operation continues until all bytes have been received. As before, the reception of the last bytes causes the receive interrupt service routine is called. A software state variable is used to track the fact that this is the second entry into the receive interrupt service routine, and therefore, the entire transfer is complete, and the chip select signal to the SPI device is deasserted.

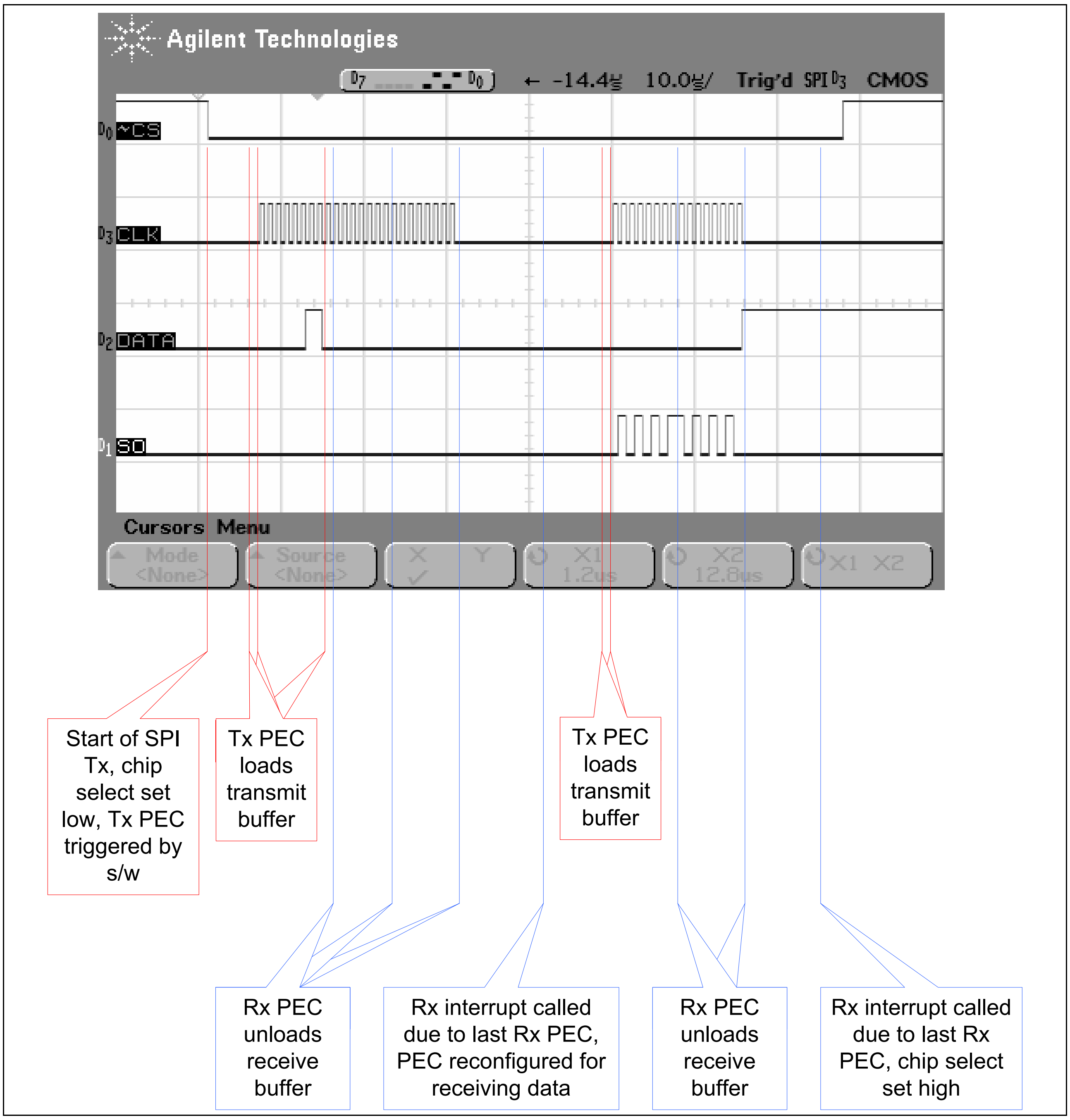

Agilent Technologies
D7 D0
← -14.4us 10.0us/ Trig'd SPI D3 CMOS
D0 ~CS
D3 CLK
D2 DATA
D1 SO
Cursors Menu
Mode <None>
Source <None>
X Y
X1 1.2us
X2 12.8us
X1 X2
Start of SPI Tx, chip select set low, Tx PEC triggered by s/w
Tx PEC loads transmit buffer
Tx PEC loads transmit buffer
Rx PEC unloads receive buffer
Rx interrupt called due to last Rx PEC, PEC reconfigured for receiving data
Rx PEC unloads receive buffer
Rx interrupt called due to last Rx PEC, chip select set high

Figure 14 SPI Transmit and Receive Timing

3.3.4 SPI Layer Interface

The SPI layer provides the following interface functions.

3.3.4.1 InitSPI

Function name:	InitSPI	
Syntax:	void InitSPI(void)	
Parameters (in):	None	
Parameters (out):	None	
Description:	Performs initialization for SPI	

3.3.4.2 Tx_SPI

Function name:	Tx_SPI	
Syntax:	void Tx_SPI(SPI_DEVICE_TYPE dev, U8 TxCount, U8* src)	
Parameters (in):	dev	SPI device to be selected
	TxCount	number of bytes to transmit
	src	pointer to source data
Parameters (out):	None	
Description:	Transmits a number of bytes over SPI bus NOTE: Function initiates transmit, and continues transmission in the background.	

3.3.4.3 Tx_Rx_SPI

Function name:	Tx_Rx_SPI	
Syntax:	void Tx_Rx_SPI(SPI_DEVICE_TYPE dev, U8 TxCount, U8* src, U8 RxCount, U8* dst)	
Parameters (in):	dev	SPI device to be selected
	TxCount	number of bytes to transmit
	src	pointer to source data
	RxCount	number of bytes to receiver (after transmit)
	dst	pointer to address where received data will be stored
Parameters (out):	None	
Description:	Transmits and receives a number of bytes over SPI bus NOTE: Function initiates transfer, and continues transfer in the background	

3.3.4.4 SPI_Dev_Busy

Function name:	SPI_Dev_Busy	
Syntax:	U8 SPI_Dev_Busy(SPI_DEVICE_TYPE dev)	
Parameters (in):	None	
Parameters (out):	Status	Returns busy status
Description:	Provides indication if SPI device is busy (SPI transfer in progress)	

4 Additional Application Considerations

Although the example code provided with this application note was constructed in a generalized way, it obviously can not meet the needs of every application. This section provides some comments on areas that should be considered when integrating similar code into an application.

4.1 EEPROM IC AC Characteristics

The EEPROM IC AC characteristics should be reviewed when implementing an SPI interface into an application. These characteristics include such things as maximum allowed SPI clock frequency, and setup and hold times. The example code was implemented assuming an 8MHz operating frequency for the microcontroller, and 1MHz SPI clock frequency. The EEPROM IC timing requirements can be met when operating at these frequencies. But, when operating at higher frequencies, particular attention should be paid to the signal timing. In particular, the chip select setup and hold timing values are directly influenced by the microcontroller operating frequency. Adjustments may be required to remain within the EEPROM IC specifications.

4.2 Microcontroller Interrupt Suppression

The example code has been constructed assuming that PEC data transfers will occur without significant delays. But, in the case that an application globally disables interrupts, or uses high priority interrupts (above the PEC interrupt level), it is possible that PEC transfers could be delayed for a significant amount of time. In the case that PEC transfers are delayed for more than one SPI byte transfer time, it is possible to overrun the SSC receive buffer. In such a case, although the SSC transmit buffer is no longer being reloaded by PEC transfers, the SSC hardware will continue to transmit the byte within the transmit shift register. In addition, if another byte has already been loaded into the transmit buffer, it will also be transferred to the shift register and transmitted. In this way, it is possible to transmit two bytes while PEC transfers have been delayed. The SSC receive hardware can only buffer one receive byte, and so the second byte would be lost in this scenario.

In case the application has interrupt suppression that exceeds one SPI byte transfer time, it is recommended to implement an SSC receive interrupt routine without the use of PEC channels. The receive interrupt routine should then process received bytes before loading the next byte into the transmit buffer. Using this approach, it is not possible to overrun the receive buffer, even with very long interrupt suppression times.

4.3 Memory model and pointers used in example code

The example code uses a small memory model, and assumes that all pointer references can be made using near addressing (a 16-bit address pointer). The XC16x microcontroller is capable of using longer addresses (such as 24-bit or 32-bit) for pointers and PEC transfers, but this is not accommodated in the example code. Adjustments to the example code would be required if larger addresses are required.

5 Conclusion

This application note described the steps necessary to interface the XC16x series of microcontrollers to an SPI EEPROM IC. The example code demonstrates how this EEPROM interface can be implemented on the Infineon XC16x starter kits, which include an SPI EEPROM. The example code also demonstrates the use of the PEC data transfer mechanism, providing an efficient data transfer with minimal CPU overhead and maximum SPI bus utilization. The flexible SSC peripheral, along with efficient PEC transfers, provide a convenient, yet powerful, interface to SPI devices.