Inter-Integrated Circuit (I^2C)

HIGHLIGHTS

This section of the manual contains the following major topics:

1.0 Introduction 2
2.0 I^2C Bus Characteristics 4
3.0 Control and Status Registers 8
4.0 Enabling I^2C Operation 18
5.0 Communicating as a Master in a Single Master Environment 20
6.0 Communicating as a Master in a Multi-Master Environment 34
7.0 Communicating as a Slave 37
8.0 Connection Considerations for I^2C Bus 61
9.0 Operation in Power-Saving Modes 63
10.0 Peripheral Module Disable (PMDx) Registers 63
11.0 Effects of a Reset 63
12.0 Constant-Current Source 64
13.0 Register Maps 66
14.0 Design Tips 67
15.0 Related Application Notes 68
16.0 Revision History 69

© 2007-2015 Microchip Technology Inc.

Note: This family reference manual section is meant to serve as a complement to device data sheets. Depending on the device variant, this manual section may not apply to all PIC24 and dsPIC33 devices.

Please consult the note at the beginning of the **"Inter-Integrated Circuit (I^2C)"** chapter in the current device data sheet to check whether this document supports the device you are using.

Device data sheets and family reference manual sections are available for download from the Microchip Worldwide Web site at:
http://www.microchip.com

This document supersedes the following PIC24 and dsPIC® DSC Family Reference Manual sections:

DS Number	Section Number	Title
DS70195	19	dsPIC33F/PIC24H Family Reference Manual
DS70330	19	dsPIC33E/PIC24E Family Reference Manual
DS39702	24	PIC24F Family Reference Manual
DS70235	19	PIC24H Family Reference Manual
DS70068	21	dsPIC30F Family Reference Manual

1.0 INTRODUCTION

The Inter-Integrated Circuit (I^2C) module is a serial interface useful for communicating with other peripheral or microcontroller (MCU) devices. The external peripheral devices may be serial EEPROMs, display drivers, Analog-to-Digital Converters (ADC) and so on.

The I^2C module can operate as any one of the following in the I^2C system:

- Slave device
- Master device in a single master system (slave may be active)
- Master or slave device in a multi-master system (bus collision detection and arbitration are available)

The I^2C module contains an independent I^2C master logic and a I^2C slave logic, which generates interrupts based on their events. In the multi-master systems, the user software is simply partitioned into the master controller and the slave controller.

When the I^2C master logic is active, the slave logic also remains active, detecting the state of the bus and potentially receiving messages from itself in a single master system or from the other masters in a multi-master system. No messages are lost during the multi-master bus arbitration.

In a multi-master system, the bus collision conflicts with the other masters in the system when detected, and the module provides a method to terminate and then restart the message.

The I^2C module contains a Baud Rate Generator (BRG). The I^2C BRG does not consume other timer resources in the device. Figure 1-1 illustrates the I^2C module block diagram.

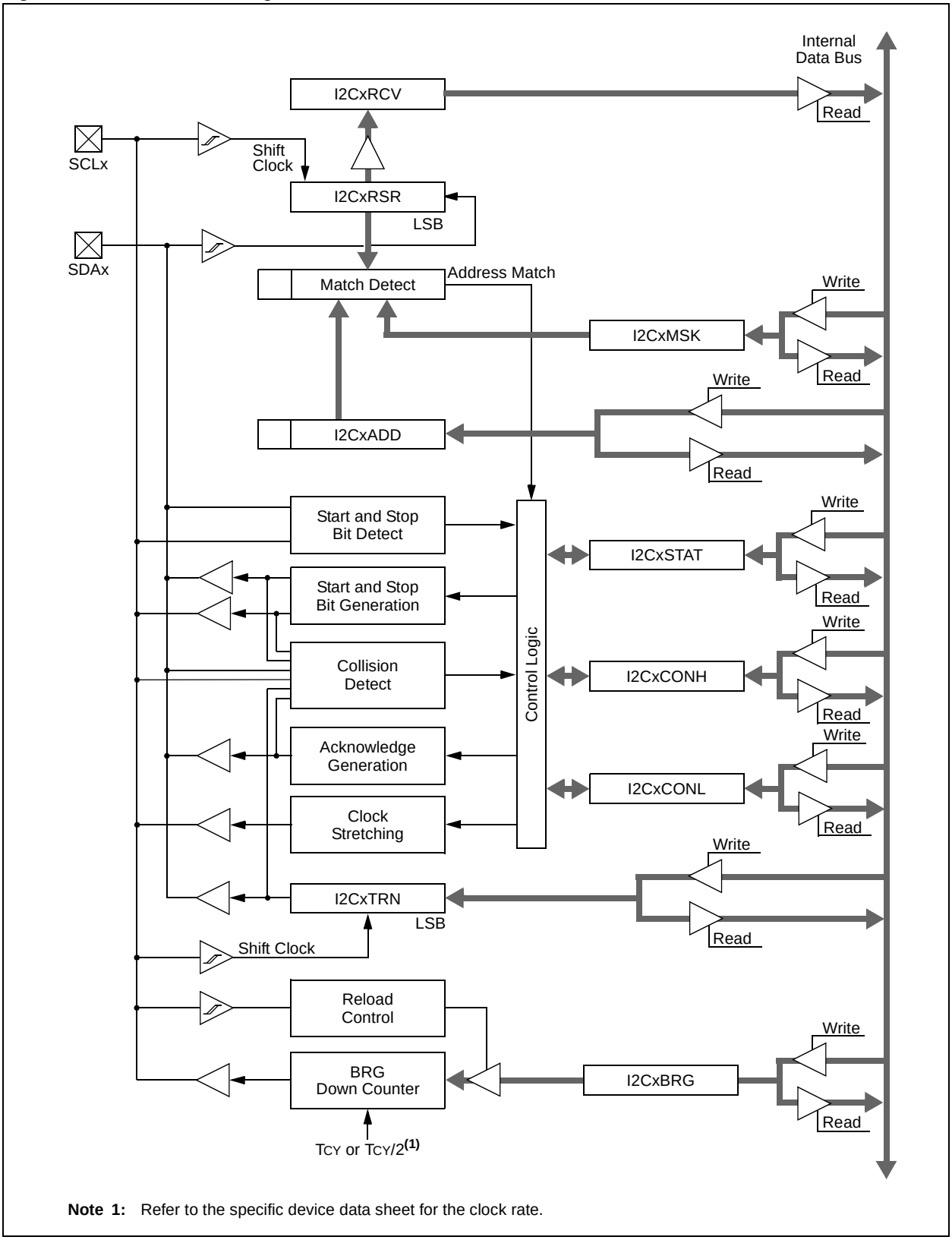
Key features of the I^2C module include the following:

- Independent master and slave logic
- Multi-master support which prevents message losses in arbitration
- Detects 7-bit and 10-bit device addresses with configurable address masking in Slave mode
- Detects general call addresses as defined in the I^2C protocol
- Bus Repeater mode, allowing the module to accept all messages as a slave, irrespective of the address
- Automatic SCLx clock stretching provides delays for the processor to respond to a slave data request
- Supports 100 kHz and 400 kHz bus specifications
- Supports the Intelligent Platform Management Interface (IPMI) standard
- Supports SDAx hold time for SMBus (300 nS or 150 nS) in Slave mode

Note: For more information, refer to the SDAHT bit description in the specific device data sheet.

© 2007-2015 Microchip Technology Inc.

Figure 1-1: I^2C Block Diagram



Note 1: Refer to the specific device data sheet for the clock rate.

2.0 I²C BUS CHARACTERISTICS

The I²C bus is a 2-wire serial interface. Figure 2-1 illustrates the schematic of an I²C connection between a dsPIC33/PIC24 device and a 24LC256 I²C serial EEPROM, which is a typical example for any I²C interface.

The I²C interface uses a comprehensive protocol to ensure reliable transmission and reception of the data. When communicating, one device acts as the “master” and it initiates transfer on the bus, and generates the clock signals to permit that transfer, while the other devices act as the “slave” responding to the transfer. The clock line, SCLx, is output from the master and input to the slave, although occasionally the slave drives the SCLx line. The data line, SDAx, may be output and input from both the master and slave.

Because the SDAx and SCLx lines are bidirectional, the output stages of the devices driving the SDAx and SCLx lines must have an open-drain in order to perform the wired-AND function of the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down.

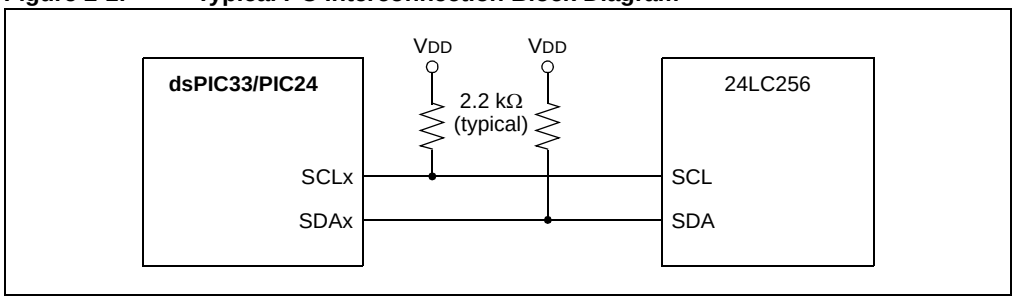
Note: SCLx and SDAx must be configured as digital.

In the I²C interface protocol, each device has an address. When a master needs to initiate a data transfer, it first transmits the address of the device that it wants to “communicate”. All of the devices “listen” to see if this is their address. Within this address, bit 0 specifies whether the master wants to read from or write to the slave device. The master and slave are always in opposite modes (Transmitter or Receiver) of operation during a data transfer. That is, they operate in either of the following two relationships:

- Master-Transmitter and Slave-Receiver
- Slave-Transmitter and Master-Receiver

In both cases, the master originates the SCLx clock signal.

Figure 2-1: Typical I²C Interconnection Block Diagram



© 2007-2015 Microchip Technology Inc.

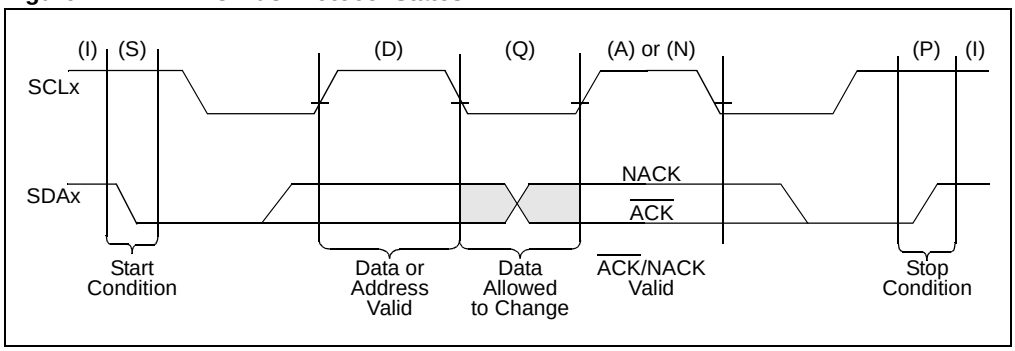
2.1 Bus Protocol

The following I^2C bus protocol has been defined:

- The data transfer may be initiated only when the bus is not busy.
- During the data transfer, the data line must remain stable whenever the SCLx clock line is high. Any changes in the data line, while the SCLx clock line is high, will be interpreted as a Start or Stop condition.

Accordingly, the bus conditions are defined as illustrated in Figure 2-2.

Figure 2-2: I^2C Bus Protocol States



2.1.1 START DATA TRANSFER (S)

After a bus Idle state, a high-to-low transition of the SDAx line while the clock (SCLx) is high determines a Start condition. All data transfers must be preceded by a Start condition.

2.1.2 STOP DATA TRANSFER (P)

A low-to-high transition of the SDAx line while the clock (SCLx) is high determines a Stop condition. All data transfers must end with a Stop condition.

2.1.3 REPEATED START (R)

After a Wait state, a high-to-low transition of the SDAx line while the clock (SCLx) is high determines a Repeated Start condition. Repeated Starts allow a master to change bus direction or address a slave device without relinquishing control of the bus.

2.1.4 DATA VALID (D)

After a Start condition, the state of the SDAx line represents valid data when the SDAx line is stable for the duration of the high period of the clock signal. There is one bit of data per SCLx clock.

2.1.5 ACKNOWLEDGE (A) OR NOT ACKNOWLEDGE (N)

All data byte transmissions must be Acknowledged ($\overline{ACK}$) or Not Acknowledged (NACK) by the receiver. The receiver will pull the SDAx line low for an $\overline{ACK}$ or release the SDAx line for a NACK. The Acknowledge is a 1-bit period using one SCLx clock.

2.1.6 WAIT/DATA INVALID (Q)

The data on the line must be changed during the low period of the clock signal. The devices may also stretch the clock low time by asserting a low on the SCLx line, causing a Wait on the bus.

2.1.7 BUS IDLE (I)

Both data and clock lines remain high after a Stop condition and before a Start condition.

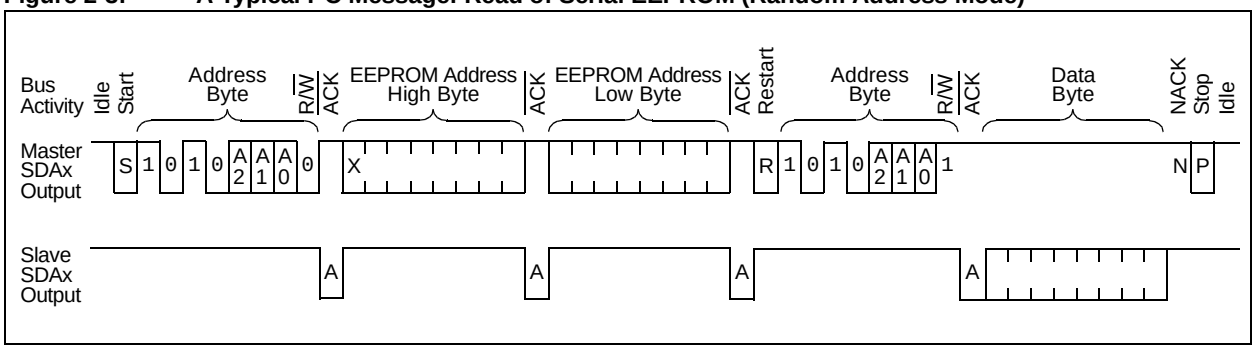
© 2007-2015 Microchip Technology Inc.

2.2 Message Protocol

A typical I^2C message is illustrated in Figure 2-3. In this example, the message will read a specified byte from a 24LC256 I^2C serial EEPROM. The dsPIC33/PIC24 device will act as the master and the 24LC256 device will act as the slave.

Figure 2-3 illustrates the data as driven by the master device and the slave device, taking into account that the combined SDAx line is a wired-AND of the master and slave data. The master device controls and sequences the protocol. The slave device will only drive the bus at specifically determined times.

Figure 2-3: A Typical I^2C Message: Read of Serial EEPROM (Random Address Mode)



2.2.1 START MESSAGE

Each message is initiated with a Start condition and terminated with a Stop condition. The number of data bytes transferred between the Start and Stop conditions is determined by the master device. As defined by the system protocol, the bytes of the message may have special meaning, such as the device address byte or the data byte.

2.2.2 ADDRESS SLAVE

In Figure 2-3, the first byte is the device address byte, which must be the first part of any I^2C message. It contains a device address and a R/$\overline{W}$ status bit. Note that R/$\overline{W}$ = 0 for this first address byte, indicating that the master will be a transmitter and the slave will be a receiver.

2.2.3 SLAVE ACKNOWLEDGE

The receiving device is obliged to generate an Acknowledge signal, $\overline{ACK}$, after the reception of each byte. The master device must generate an extra SCLx clock, which is associated with this Acknowledge bit.

2.2.4 MASTER TRANSMIT

The next two bytes, sent by the master to the slave, are data bytes that contain the location of the requested EEPROM data byte. The slave must Acknowledge each of the data bytes.

2.2.5 REPEATED START

The slave EEPROM has the required address information that is required to return the requested data byte to the master. However, the R/$\overline{W}$ status bit from the first device address byte specifies the master transmission and the slave reception. The direction of the bus must be reversed for the slave to send data to the master.

To perform this function without ending the message, the master sends a Repeated Start. The Repeated Start is followed with a device address byte containing the same device address as before, and with R/$\overline{W}$ = 1, to indicate the slave transmission and the master reception.

© 2007-2015 Microchip Technology Inc.

2.2.6 SLAVE REPLY

The slave transmits the data byte by driving the SDAx line, while the master continues to originate clocks but releases its SDAx drive.

2.2.7 MASTER ACKNOWLEDGE

During reads, a master must terminate data requests to the slave by generating a NACK on the last byte of the message.

2.2.8 STOP MESSAGE

The master sends a Stop signal to terminate the message and returns the bus to an Idle state.

3.0 CONTROL AND STATUS REGISTERS

The I^{2}C module has registers for operation that are accessible by the user application. All registers are accessible in either Byte or Word mode. The registers are as follows:

- **I2CxCON: I2Cx Control Register** or **I2CxCONL: I2Cx Control Register Low** and **I2CxCONH: I2Cx Control Register High**

  These registers allow control of the module's operation.

- **I2CxSTAT: I2Cx Status Register**

  This register contains status flags indicating the module's state during operation.

- **I2CxMSK: I2Cx Slave Mode Address Mask Register**

  This register designates which bit positions in the I2CxADD register can be ignored, which allows for multiple address support.

- **ISRCCON: I2Cx Current Source Control Register(1)**

  This register allows control of the current source module.

- **I2CxRCV: I2Cx Receive Buffer Register**

  This is the buffer register from which data bytes can be read. The I2CxRCV register is a read-only register.

- **I2CxTRN: I2Cx Transmit Register**

  This is the transmit register. The bytes are written to this register during a transmit operation. The I2CxTRN register is a read/write register.

- **I2CxADD: I2Cx Address Register**

  This register holds the slave device address.

- **I2CxBRG: I2Cx Baud Rate Generator Reload Register**

  This register holds the BRG reload value for the I^{2}C module BRG.

Note 1:	The I2CxCONL, I2CxCONH and ISRCCON registers are not available on all devices. Refer to the specific device data sheet for availability.

The transmit data is written to the I2CxTRN register. This register is used when the module operates as a master transmitting data to the slave or when it operates as a slave sending reply data to the master. As the message progresses, the I2CxTRN register shifts out the individual bits. Therefore, the I2CxTRN register cannot be written to unless the bus is Idle.

The data being received by either the master or the slave is shifted into a non-accessible shift register, I2CxRSR. When a complete byte is received, the byte transfers to the I2CxRCV register. In receive operations, the I2CxRSR and I2CxRCV registers create a double-buffered receiver. This allows reception of the next byte to begin before reading the current byte of the received data.

If the module receives another complete byte before the user software reads the previous byte from the I2CxRCV register, a receiver overflow occurs and sets the I2COV bit (I2CxSTAT<6>). The byte in the I2CxRSR register is lost if BOEN = 0. Further reception and clock stretching are disabled until the I^{2}C module sees a Start/Repeated, Start/Stop condition on the bus. If the I2COV flag has been cleared, the reception can proceed normally. If the I2COV flag is not cleared, the module will receive the next byte correctly, but will send a NACK. It will then be unable to receive further bytes or stretch the clock until it detects a Start/Repeated and Start/Stop condition.

The I2CxADD register holds the slave device address. In 10-Bit Addressing mode, all bits are relevant. In 7-Bit Addressing mode, only the I2CxADD<6:0> bits are relevant. The I2CxADD<6:0> bits correspond to the upper 7 bits in the address byte. The Read/$\overline{\text{Write}}$ bit (R/$\overline{\text{W}}$) is not included in the value in this register. The A10M bit (I2CxCON<10> or I2CxCONL<10>) specifies the expected mode of the slave address. By using the I2CxMSK register with the I2CxADD register in Slave Addressing mode, one or more bit positions can be removed from the exact address matching, allowing the module, in Slave mode, to respond to multiple addresses.

© 2007-2015 Microchip Technology Inc.

Register 3-1: I2CxCON: I2Cx Control Register

R/W-0	U-0	R/W-0	R/W-1, HC	R/W-0	R/W-0	R/W-0	R/W-0
I2CEN	—	I2CSIDL	SCLREL	IPMIEN(1)	A10M	DISSLW	SMEN
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0, HC	R/W-0, HC	R/W-0, HC	R/W-0, HC	R/W-0, HC
GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

Legend:	HC = Hardware Clearable bit		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at Reset	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15 **I2CEN:** I2Cx Enable bit
1 = Enables the I2Cx module and configures the SDAx and SCLx pins as serial port pins
0 = Disables the I2Cx module; all the I^2C pins are controlled by port functions

bit 14 **Unimplemented:** Read as '0'

bit 13 **I2CSIDL:** I2Cx Stop in Idle Mode bit
1 = Discontinues the module operation when a device enters the Idle mode
0 = Continues the module operation in the Idle mode

bit 12 **SCLREL:** SCLx Release Control bit (when operating as I^2C slave)
1 = Releases the SCLx clock
0 = Holds the SCLx clock low (clock stretch)
If STREN = 1:
User software may write '0' to initiate a clock stretch and write '1' to release the clock. Hardware clears at the beginning of every slave data byte transmission. Hardware clears at the end of every slave address byte reception. Hardware clears at the end of every slave data byte reception.
If STREN = 0:
User software may only write '1' to release the clock. Hardware clears at the beginning of every slave data byte transmission. Hardware clears at the end of every slave address byte reception.

bit 11 **IPMIEN:** IPMI Enable bit(1)
1 = IPMI Support mode is enabled, all addresses are Acknowledged
0 = IPMI Support mode is disabled

bit 10 **A10M:** 10-Bit Slave Address bit
1 = I2CxADD register is a 10-bit slave address
0 = I2CxADD register is a 7-bit slave address

bit 9 **DISSLW:** Disable Slew Rate Control bit
1 = Slew rate control is disabled
0 = Slew rate control is enabled

bit 8 **SMEN:** SMBus Input Levels bit
1 = Enables the I/O pin thresholds compliant with the SMBus specification
0 = Disables the SMBus input thresholds

bit 7 **GCEN:** General Call Enable bit (when operating as I^2C slave)
1 = Enables the interrupt when a general call address is received in the I2CxRSR register (module is enabled for reception)
0 = Disables the general call address

bit 6 **STREN:** SCLx Clock Stretch Enable bit (I^2C Slave mode only; used in conjunction with the SCLREL bit)
1 = Enables the user software or the receive clock stretching
0 = Disables the user software or the receive clock stretching

Note 1: The IPMIEN bit should not be set when the I^2C module is operating as a master.

Register 3-1: I2CxCON: I2Cx Control Register (Continued)

bit 5 **ACKDT:** Acknowledge Data bit (I^2C Master mode; receive operation only)
Value that will be transmitted when the user software initiates an Acknowledge sequence.
1 = Sends a NACK during an Acknowledge
0 = Sends an ACK during an Acknowledge

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (I^2C Master mode receive operation)
1 = Initiates the Acknowledge sequence on the SDAx and SCLx pins and transmits the ACKDT data bit (hardware clears at the end of the master Acknowledge sequence)
0 = Acknowledge sequence is not in progress

bit 3 **RCEN:** Receive Enable bit (I^2C Master mode)
1 = Enables Receive mode for I^2C (hardware clears at the end of eighth bit of master receive data byte)
0 = Receive sequence is not in progress

bit 2 **PEN:** Stop Condition Enable bit (I^2C Master mode)
1 = Initiates the Stop condition on the SDAx and SCLx pins (hardware clears at the end of master Stop sequence)
0 = Stop condition is not in progress

bit 1 **RSEN:** Repeated Start Condition Enable bit (I^2C Master mode)
1 = Initiates the Repeated Start condition on the SDAx and SCLx pins (hardware clears at the end of master Repeated Start sequence)
0 = Repeated Start condition is not in progress

bit 0 **SEN:** Start Condition Enable bit (I^2C Master mode)
1 = Initiates the Start condition on the SDAx and SCLx pins (hardware clears at the end of master Start sequence)
0 = Start condition is not in progress

Note 1: The IPMIEN bit should not be set when the I^2C module is operating as a master.

© 2007-2015 Microchip Technology Inc.

Register 3-2: I2CxCONL: I2Cx Control Register Low

R/W-0	U-0	R/W-0, HC	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
I2CEN	—	I2CSIDL	SCLREL(1)	STRICT	A10M	DISSLW	SMEN
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0, HC	R/W-0, HC	R/W-0, HC	R/W-0, HC	R/W-0, HC
GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

Legend:	HC = Hardware Clearable bit		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15 **I2CEN:** I2Cx Enable bit
1 = Enables the I^2C module and configures the SDAx and SCLx pins as serial port pins
0 = Disables the I^2C module; all the I^2C pins are controlled by port functions

bit 14 **Unimplemented:** Read as '0'

bit 13 **I2CSIDL:** I2Cx Stop in Idle Mode bit
1 = Discontinues the module operation when a device enters Idle mode
0 = Continues the module operation in the Idle mode

bit 12 **SCLREL:** SCLx Release Control bit (I^2C Slave mode only)(1)
Module resets and (I2CEN = 0) sets SCLREL = 1.
If STREN = 0:(2)
1 = Releases the clock
0 = Forces clock low (clock stretch)
If STREN = 1:
1 = Releases the clock
0 = Holds clock low (clock stretch); the user may program this bit to '0', clock stretch at next SCLx low

bit 11 **STRICT:** Strict I^2C Reserved Address Rule Enable bit
1 = Strict reserved addressing is enforced; for reserved addresses
In a Slave mode, the device does not respond to reserved address space and the addresses falling in that category are NACKed.
In a Master mode, the device is allowed to generate addresses with the reserved address space.
0 = Reserved addressing would be Acknowledged
In a Slave mode, the device will respond to an address falling in the reserved address space. When there is a match with any of the reserved addresses, the device will generate an ACK.
In a Master mode, it is reserved.

bit 10 **A10M:** 10-Bit Slave Address Flag bit
1 = I2CxADD is a 10-bit slave address
0 = I2CxADD is a 7-bit slave address

bit 9 **DISSLW:** Slew Rate Control Disable bit
1 = Slew rate control is disabled for Standard Speed mode (100 kHz, disabled for 1 MHz mode)
0 = Slew rate control is enabled for High-Speed mode (400 kHz)

bit 8 **SMEN:** SMBus Input Levels Enable bit
1 = Enables the input logic; therefore, thresholds are compliant with the SMBus specification
0 = Disables the SMBus protocol-specific inputs

Note 1: Automatically cleared to '0' at the beginning of the slave transmission; automatically cleared to '0' at the end of the slave reception.
2: Automatically cleared to '0' at the beginning of the slave transmission.

Register 3-2: I2CxCONL: I2Cx Control Register Low (Continued)

bit 7 **GCEN:** General Call Enable bit (I^2C Slave mode only)
1 = Enables the interrupt when a general call address is received in I2CxRSR; the module is enabled for reception
0 = General call address is disabled

bit 6 **STREN:** SCLx Clock Stretch Enable bit
In I^2C Slave mode only; used in conjunction with the SCLREL bit.
1 = Enables clock stretching
0 = Disables clock stretching

bit 5 **ACKDT:** Acknowledge Data bit
In I^2C Master mode during Master Receive mode. The value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.
In I^2C Slave mode when AHEN = 1 or DHEN = 1. The value that the slave will transmit when it initiates an Acknowledge sequence at the end of an address or data reception.
1 = NACK is sent
0 = ACK is sent

bit 4 **ACKEN:** Acknowledge Sequence Enable bit
In I^2C Master mode only; applicable during Master Receive mode.
1 = Initiates the Acknowledge sequence on the SDAx and SCLx pins, and transmits the ACKDT data bit
0 = Acknowledge sequence is in Idle mode

bit 3 **RCEN:** Receive Enable bit (I^2C Master mode only)
1 = Enables Receive mode for I^2C; automatically cleared by hardware at the end of an 8-bit receive data byte
0 = Receive sequence is not in progress

bit 2 **PEN:** Stop Condition Enable bit (I^2C Master mode only)
1 = Initiates the Stop condition on the SDAx and SCLx pins
0 = Stop condition is in Idle mode

bit 1 **RSEN:** Restart Condition Enable bit (I^2C Master mode only)
1 = Initiates the Restart condition on the SDAx and SCLx pins
0 = Restart condition is in Idle mode

bit 0 **SEN:** Start Condition Enable bit (I^2C Master mode only)
1 = Initiates the Start condition on the SDAx and SCLx pins
0 = Start condition is in Idle mode

Note 1: Automatically cleared to '0' at the beginning of the slave transmission; automatically cleared to '0' at the end of the slave reception.

2: Automatically cleared to '0' at the beginning of the slave transmission.

© 2007-2015 Microchip Technology Inc.

Register 3-3: I2CxCONH: I2Cx Control Register High

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	PCIE	SCIE	BOEN	SDAHT	SBCDE	AHEN	DHEN
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-7 **Unimplemented:** Read as '0'

bit 6 **PCIE:** Stop Condition Interrupt Enable bit (I^2C Slave mode only)
1 = Enables the interrupt on detection of a Stop condition
0 = Stop detection interrupts are disabled

bit 5 **SCIE:** Start Condition Interrupt Enable bit (I^2C Slave mode only)
1 = Enables the interrupt on detection of a Start or Restart condition
0 = Start detection interrupts are disabled

bit 4 **BOEN:** Buffer Overwrite Enable bit (I^2C Slave mode only)
1 = The I2CxRCV register is updated and an $\overline{ACK}$ is generated for a received address or data byte, ignoring the state of the I2COV bit only if the RBF bit = 0
0 = The I2CxRCV register is only updated when the I2COV bit is clear

bit 3 **SDAHT:** SDAx Hold Time Selection bit(1)
1 = Minimum of 300 ns hold time on SDAx after the falling edge of the SCLx clock
0 = Minimum of 100 ns hold time on SDAx after the falling edge the of SCLx clock

bit 2 **SBCDE:** Slave Mode Bus Collision Detect Enable bit (I^2C Slave mode only)
If, on the rising edge of the SCLx, SDAx is sampled low when the module is outputting a high state, the BCL bit is set and the bus goes into Idle mode. This Detection mode is valid only during the data and ACK transmit sequences.
1 = Enables the slave bus collision interrupts
0 = Disables the slave bus collision interrupts

bit 1 **AHEN:** Address Hold Enable bit (I^2C Slave mode only)
1 = Following the falling edge of the eighth SCLx clock for a matching received address byte; the SCLREL bit (I2CxCONL<12>) will be cleared and the SCLx will be held low
0 = Address holding is disabled

bit 0 **DHEN:** Data Hold Enable bit (I^2C Slave mode only)
1 = Following the eighth falling edge of the SCLx clock for a received data byte; slave hardware clears the SCLREL bit (I2CxCONL<12>) and SCLx is held low
0 = Data holding is disabled

Note 1: This bit must be set to '0' for 1 MHz operation.

Register 3-4: I2CxSTAT: I2Cx Status Register

R-0, HSC	R-0, HSC	R-0, HSC	U-0	U-0	R/C-0, HS	R-0, HSC	R-0, HSC
ACKSTAT	TRSTAT	ACKTIM(1)	—	—	BCL	GCSTAT	ADD10
bit 15							bit 8

R/C-0, HS	R/C-0, HS	R-0, HSC	R-0, HSC	R-0, HSC	R-0, HSC	R-0, HSC	R-0, HSC
IWCOL	I2COV	D/$\overline{A}$	P	S	R/$\overline{W}$	RBF	TBF
bit 7							bit 0

Legend:	C = Clearable bit	HSC = Hardware Settable/Clearable bit	
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	HS = Hardware Settable bit

bit 15 **ACKSTAT:** Acknowledge Status bit
1 = NACK received from slave
0 = ACK received from slave
Hardware sets or clears at the end of slave or master Acknowledge.

bit 14 **TRSTAT:** Transmit Status bit (I^2C Master mode transmit operation)
1 = Master transmit is in progress (8 bits + ACK)
0 = Master transmit is not in progress
Hardware sets at the beginning of master transmission; hardware clears at the end of slave Acknowledge.

bit 13 **ACKTIM:** Acknowledge Time Status bit (valid in I^2C Slave mode only)(1)
1 = Indicates that the I^2C bus is in an Acknowledge sequence; set on the falling edge of the eighth SCLx clock
0 = Not an Acknowledge sequence, cleared on ninth rising edge of the SCLx clock

bit 12-11 **Unimplemented:** Read as '0'

bit 10 **BCL:** Bus Collision Detect bit (Master and Slave modes)
1 = A bus collision has been detected during a master or slave operation
0 = No collision
Hardware sets at detection of a bus collision; clears when I^2C module is disabled, I2CEN = 0.

bit 9 **GCSTAT:** General Call Status bit
1 = General call address was received
0 = General call address was not received
Hardware sets when address matches general call address; hardware clears at Stop detection.

bit 8 **ADD10:** 10-Bit Address Status bit
1 = 10-bit address was matched
0 = 10-bit address was not matched
Hardware sets at match of second byte of matched 10-bit address; hardware clears at Stop detection.

bit 7 **IWCOL:** I2Cx Write Collision Detect bit
1 = An attempt to write to the I2CxTRN register failed because the I^2C module is busy
0 = No collision
Hardware sets at occurrence of a write to the I2CxTRN register while busy (cleared by software).

bit 6 **I2COV:** I2Cx Receive Overflow Flag bit
1 = A byte is received while the I2CxRCV register is still holding the previous byte
0 = No overflow
Hardware sets at attempt to transfer the I2CxRSR register to the I2CxRCV register (cleared by software).

Note 1: Refer to the specific device data sheet for availability of the ACKTIM bit.

© 2007-2015 Microchip Technology Inc.

Register 3-4: I2CxSTAT: I2Cx Status Register (Continued)

bit 5 **D/$\overline{A}$:** Data/$\overline{Address}$ bit (I^2C Slave mode)
1 = Indicates that the last byte received was data
0 = Indicates that the last byte received was a device address
Hardware clears at device address match; hardware sets by reception of a slave byte or sets after the transmission is complete and the TBF flag is cleared.

bit 4 **P:** Stop bit
1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last
Hardware sets or clears when Start, Repeated Start or Stop is detected.

bit 3 **S:** Start bit
1 = Indicates that a Start (or Repeated Start) bit has been detected last
0 = Start bit was not detected last
Hardware sets or clears when Start, Repeated Start or Stop is detected.

bit 2 **R/$\overline{W}$:** Read/$\overline{Write}$ Information bit (when operating as I^2C slave)
1 = Read, data transfer is an output from the slave
0 = Write, data transfer is an input to the slave
Hardware sets or clears after reception of an I^2C device address byte.

bit 1 **RBF:** Receive Buffer Full Status bit
1 = Receive completes; the I2CxRCV register is full
0 = Receive is not complete; the I2CxRCV register is empty
Hardware sets when the I2CxRCV register is written with a received byte; hardware clears when user software reads the I2CxRCV register.

bit 0 **TBF:** Transmit Buffer Full Status bit
1 = Transmit is in progress; the I2CxTRN register is full
0 = Transmit completes; the I2CxTRN register is empty
Hardware sets when user software writes to the I2CxTRN register; hardware clears at completion of the data transmission.

Note 1: Refer to the specific device data sheet for availability of the ACKTIM bit.

Register 3-5: I2CxMSK: I2Cx Slave Mode Address Mask Register

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0
—	—	—	—	—	—	AMSK<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
AMSK<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at Reset	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-10 **Unimplemented:** Read as '0'

bit 9-0 **AMSK<9:0>:** Mask for Address Bit x Select bits

For 10-Bit Address:

1 = Enables masking for bit Ax of the incoming message address; bit match is not required in this position

0 = Disables masking for bit Ax; bit match is required in this position

For 7-Bit Address (I2CxMSK<6:0> only):

1 = Enables masking for bit Ax + 1 of the incoming message address; bit match is not required in this position

0 = Disables masking for bit Ax + 1; bit match is required in this position

© 2007-2015 Microchip Technology Inc.

Register 3-6: ISRCCON: I2Cx Current Source Control Register(1)

R/W-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
ISRCEN	—	—	—	—	OUTSEL2	OUTSEL1	OUTSEL0
bit 15							bit 8

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	ISRCCAL5(2)	ISRCCAL4(2)	ISRCCAL3(2)	ISRCCAL2(2)	ISRCCAL1(2)	ISRCCAL0(2)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15 **ISRCEN:** I2Cx Current Source Enable bit
1 = Current source is enabled
0 = Current source is disabled

bit 14-11 **Unimplemented:** Read as '0'

bit 10-8 **OUTSEL<2:0>:** Output Select for Current bits
111 = Reserved
110 = Reserved
101 = Reserved
100 = Selected input pin is ISRC4 (AN4)
011 = Selected input pin is ISRC3 (AN5)
010 = Selected input pin is ISRC2 (AN6)
001 = Selected input pin is ISRC1 (AN7)
000 = No output is selected

bit 7-6 **Unimplemented:** Read as '0'

bit 5-0 **ISRCCAL<5:0>:** I2Cx Current Source Calibration bits(2)

Note 1: This register is not available on all devices. Refer to the specific device data sheet for availability.
2: The calibration value must be retrieved from the Flash memory and stored in this location at start-up time.

4.0 ENABLING I2C OPERATION

The I2C module is enabled by setting the I2CEN bit (I2CxCON<15> or I2CxCONL<15>). The I2C module fully implements all master and slave functions. When the module is enabled, the master and slave functions are active simultaneously and will respond according to the user software or bus events.

When initially enabled, the module will release the SDAx and SCLx pins, putting the bus into an Idle state. The master functions will remain in an Idle state unless the user software sets the SEN control bit and the data is loaded into the I2CxTRN register. These two actions initiate a master event.

When the master logic is active, the slave logic also remains active. Therefore, the slave functions will begin to monitor the bus. If the slave logic detects a Start event and a valid address on the bus, the slave logic will begin a slave transaction.

4.1 I2C I/O Pins

Two pins are used for the bus operation. These are the SCLx pin, which is the clock, and the SDAx pin, which is the data. When the module is enabled, assuming no other module with higher priority has control, the module will assume control of the SDAx and SCLx pins. The user software need not be concerned with the state of the port I/O of the pins as the module overrides the port state and direction. At initialization, the pins are tri-stated (released).

4.2 I2C Interrupts

The I2C module generates three interrupts: MI2CxIF, SI2CxIF and I2CxBCIF. The MI2CxIF interrupt is assigned to the master events, the SI2CxIF interrupt is assigned to the slave events and the I2CxBCIF is assigned for the bus collision interrupt. These interrupts set a corresponding interrupt flag bit and interrupt the user software process if the corresponding interrupt enable bit is set, and the corresponding interrupt priority is higher than the CPU interrupt priority.

The MI2CxIF interrupt is generated on completion of the following master message events:

- Start condition
- Stop condition
- Data transfer byte transmitted or received
- Acknowledge transmit
- Repeated Start
- Detection of a bus collision event.

Note: In some devices, the bus collision interrupt is not tied with the MI2CxIF interrupt.

The SI2CxIF interrupt is generated on detection of a message directed to the slave, including the following events:

- Detection of a Start condition (see **Note 1**)
- Detection of a Stop condition (see **Note 1**)
- Detection of a Repeated Start condition (see **Note 1**)
- Detection of a valid device address (including general call) during data reception
- Request to transmit the data ($\overline{ACK}$) or to stop the data transmission (NACK)
- Reception of data

The I2CxBCIF interrupt is generated on a bus collision event in master/slave transmit operation:

- Start condition (master)
- Stop condition (master)
- Repeated Start (master)
- Data (master and slave)
- Acknowledge transmit (master and slave)

Note 1: These interrupts may not be present on all devices. Refer to the specific device data sheet for availability.

© 2007-2015 Microchip Technology Inc.

4.3 Setting Baud Rate When Operating as a Bus Master

When operating as an I^2C master, the module must generate the system SCLx clock. Generally, the I^2C system clocks are specified to be either 100 kHz, 400 kHz or 1 MHz. The system clock rate is specified as the minimum SCLx low time, plus the minimum SCLx high time. In most cases, that is defined by two BRG periods (TBRG).

The reload value for the BRG is the I2CxBRG register, as illustrated in Figure 4-1. When the BRG is loaded with this value, the generator counts down to zero and stops until another reload has taken place. The BRG is reloaded automatically on baud rate restart. For example, if clock synchronization is taking place, the BRG will be reloaded when the SCLx pin is sampled high.

Note: The I2CxBRG register values that are less than two are not supported.

Equation 4-1 shows the formula for computing the BRG reload value.

Equation 4-1: BRG Reload Value Calculation

$$I2CxBRG = \left(\left(\frac{1}{F_{SCL}} - Delay\right) \times \frac{F_{CY}}{2}\right) - 2$$ (See **Note 1**)

or

$$I2CxBRG = \left(\left(\frac{1}{F_{SCL}} - Delay\right) \times F_{CY}\right) - 2$$ Default (See **Note 1** and **Note 2**)

Where:

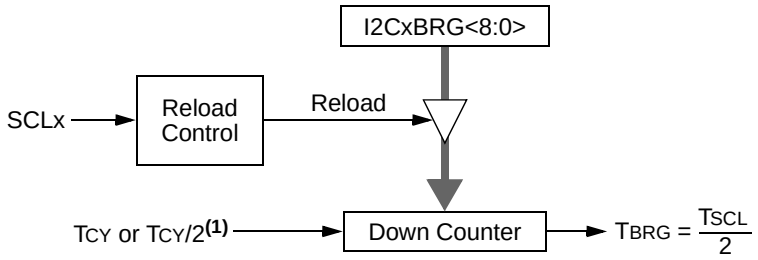
Typical value of delay varies from 110 ns to 130 ns.

Note 1: Refer to the specific device data sheet for BRG reload value calculation.

2: If there is no calculation mentioned in the data sheet, then the default BRG reload value calculation should be considered.

Note: Equation 4-1 is only for a design guideline. Due to system-dependent parameters, the actual baud rate may differ slightly. Testing is required to confirm that the actual baud rate meets the system requirements; otherwise, the value of the I2CxBRG register has to be adjusted.

Figure 4-1: Baud Rate Generator Block Diagram



Note 1: Refer to the specific device data sheet for the clock rate.

5.0 COMMUNICATING AS A MASTER IN A SINGLE MASTER ENVIRONMENT

The I²C module's typical operation in a system is using the I²C to communicate with an I²C peripheral, such as an I²C serial memory. In an I²C system, the master controls the sequence of all data communication on the bus. In this example, the dsPIC33/PIC24 device and its I²C module have the role of the single master in the system. As the single master, it is responsible for generating the SCLx clock and controlling the message protocol.

Note: The IPMIEN bit (I2CxCON<11>) should not be set when operating as a master.

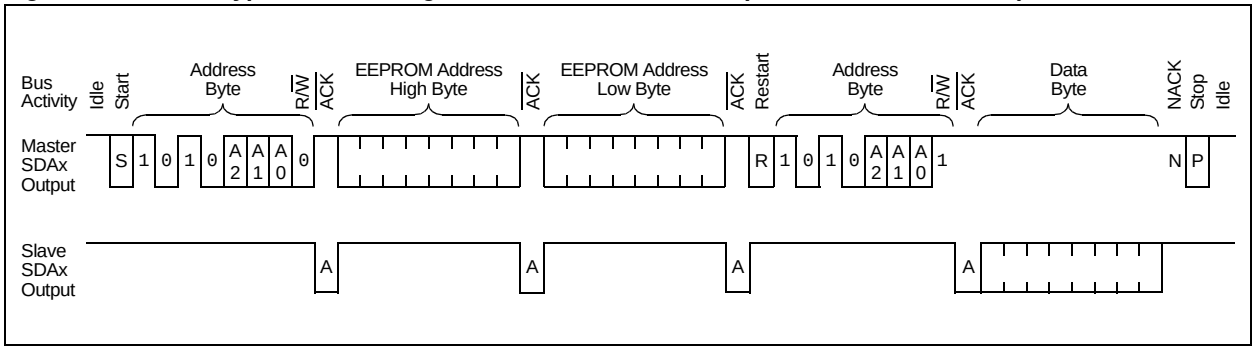
The I²C module controls individual portions of the I²C message protocol; however, sequencing of the components of the protocol to construct a complete message is performed by the user software.

For example, a typical operation in a single master environment is to read a byte from an I²C serial EEPROM. Figure 5-1 illustrates the example message.

To accomplish this message, the user software will sequence through the following steps:

1. Assert a Start condition on SDAx and SCLx.
2. Send the I²C device address byte to the slave with a write indication.
3. Wait for and verify an Acknowledge from the slave.
4. Send the serial memory address high byte to the slave.
5. Wait for and verify an Acknowledge from the slave.
6. Send the serial memory address low byte to the slave.
7. Wait for and verify an Acknowledge from the slave.
8. Assert a Repeated Start condition on SDAx and SCLx.
9. Send the device address byte to the slave with a read indication.
10. Wait for and verify an Acknowledge from the slave.
11. Enable the master reception to receive serial memory data.
12. Generate an $\overline{ACK}$ or NACK condition at the end of a received byte of data.
13. Generate a Stop condition on SDAx and SCLx.

Figure 5-1: A Typical I²C Message: Read of Serial EEPROM (Random Address Mode)



The I²C module supports Master mode communication with the inclusion of the Start and Stop generators, data byte transmission, data byte reception, Acknowledge generator and a BRG. Generally, the user software will write to a control register to start a particular step, then wait for an interrupt or poll status to wait for completion. These operations are discussed in the subsequent sections.

Note: The I²C module does not allow queuing of events. For example, the user software is not allowed to initiate a Start condition and immediately write the I2CxTRN register to initiate transmission before the Start condition is complete. In this case, the I2CxTRN register will not be written to and the IWCOL status bit (I2CxSTAT<7>) will be set, indicating that this write to the I2CxTRN register did not occur.

© 2007-2015 Microchip Technology Inc.

5.1 Generating Start Bus Event

To initiate a Start event, the user software sets the SEN bit (I2CxCON<0> or I2CxCONL<0>). Prior to setting the Start bit, the user software can check the P status bit (I2CxSTAT<4>) to ensure that the bus is in an Idle state.

Note:	A delay (typically 150 nS) should be given between the enabling of the I^2C module and the Start bus event.

Figure 5-2 illustrates the timing of the Start condition.

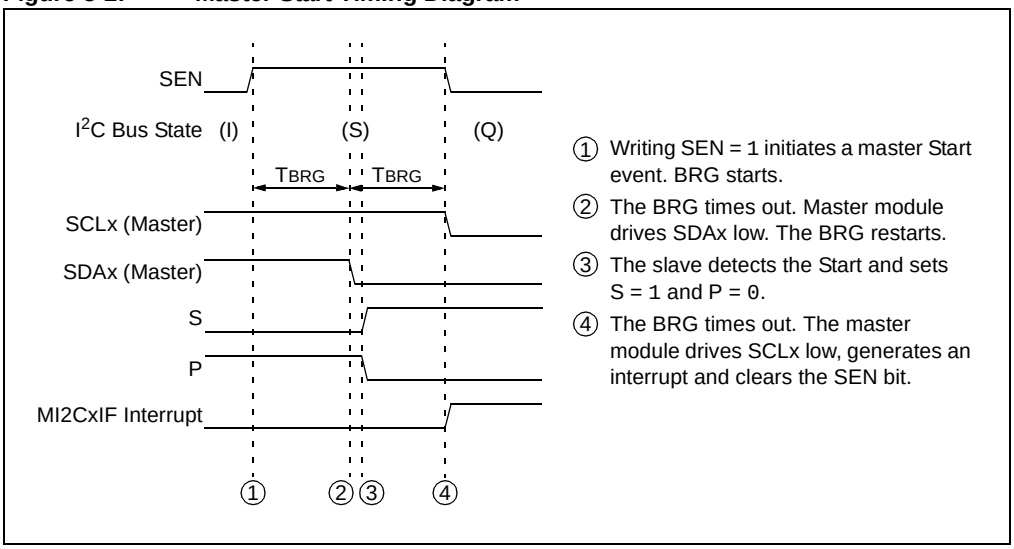
- Slave logic detects the Start condition, sets the S status bit (I2CxSTAT<3>) and clears the P status bit (I2CxSTAT<4>)
- The SEN bit is automatically cleared at completion of the Start condition
- The MI2CxIF interrupt is generated at completion of the Start condition
- After the Start condition, the SDAx line and SCLx lines are left low (Q state)

5.1.1 IWCOL STATUS FLAG

If the user software writes to the I2CxTRN register when a Start sequence is in progress, the IWCOL status bit (I2CxSTAT<7>) is set and the contents of the transmit buffer are unchanged (the write does not occur).

Note:	As the queuing of events is not allowed, writing to the lower 5 bits of the I2CxCON or I2CxCONL register is disabled until the Start condition is complete.

Figure 5-2: Master Start Timing Diagram



5.2 Sending Data to a Slave Device

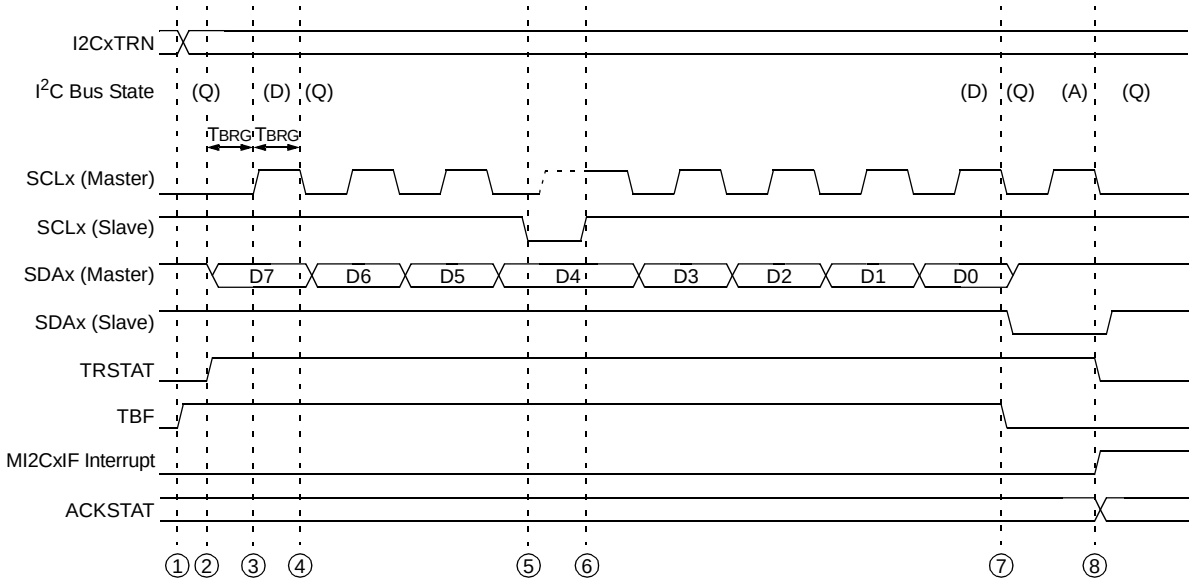
The transmission of a data byte, a 7-bit device address byte or the second byte of a 10-bit address is accomplished by writing the appropriate value to the I2CxTRN register. Loading this register will start the following process:

1. The user software loads the I2CxTRN register with the data byte to transmit.
2. Writing to the I2CxTRN register sets the TBF bit (I2CxSTAT<0>).
3. The data byte is shifted out through the SDAx pin until all 8 bits are transmitted. Each bit of address or data will be shifted out onto the SDAx pin after the falling edge of SCLx.
4. On the ninth SCLx clock, the module shifts in the $\overline{ACK}$ bit from the slave device and writes its value into the ACKSTAT status bit (I2CxSTAT<15>).
5. The module generates the MI2CxIF interrupt at the end of the ninth SCLx clock cycle.

The module does not generate or validate the data bytes. The contents and usage of the bytes are dependent on the state of the message protocol maintained by the user software.

The sequence of events that occur during master transmission and master reception are provided in Figure 5-3.

Figure 5-3: Master Transmission Timing Diagram



(1) Writing to the I2CxTRN register will start a master transmission event. The TBF status bit is set.

(2) The BRG starts. The Most Significant Byte (MSB) of the I2CxTRN register drives SDAx. SCLx remains low. The TRSTAT status bit is set.

(3) The BRG times out. SCLx is released and the BRG restarts.

(4) The BRG times out. SCLx is driven low. After SCLx is detected low, the next bit of the I2CxTRN register drives SDAx.

(5) While SCLx is low, the slave can also pull SCLx low to initiate a Wait (clock stretch).

(6) Master has already released SCLx and slave can release to end the Wait. The BRG restarts.

(7) At the falling edge of the eighth SCLx clock, the master releases SDAx. The TBF status bit is cleared. The slave drives an $\overline{ACK}$/NACK.

(8) At the falling edge of the ninth SCLx clock, the master generates the interrupt. SCLx remains low until the next event. The slave releases SDAx and the TRSTAT status bit is clear.

© 2007-2015 Microchip Technology Inc.

5.2.1 SENDING A 7-BIT ADDRESS TO THE SLAVE

Sending a 7-bit device address involves sending one byte to the slave. A 7-bit address byte must contain the 7 bits of the I^2C device address and a R/$\overline{W}$ status bit that defines whether the message will be a write to the slave (master transmission and slave reception) or a read from the slave (slave transmission and master reception).

Note:	In a 7-Bit Addressing mode, each node using the I^2C protocol should be configured with a unique address that is stored in the I2CxADD register. While transmitting the address byte, the master must shift the address bits<7:0>, left by 1 bit, and configure bit 0 as the R/$\overline{W}$ bit.

5.2.2 STRICT SUPPORT IN MASTER MODE

The master device is allowed to generate an address that falls in the reserved address space if the STRICT bit (I2CxCONL<11>) is set. For more information on the reserved address, refer to Table 7-2.

5.2.3 SENDING A 10-BIT ADDRESS TO THE SLAVE

Sending a 10-bit device address involves sending two bytes to the slave. The first byte contains 5 bits of the I^2C device address reserved for 10-Bit Addressing modes and 2 bits of the 10-bit address. As the next byte, which contains the remaining 8 bits of the 10-bit address, must be received by the slave, the R/$\overline{W}$ status bit in the first byte must be '0', indicating master transmission and slave reception. If the message data is also directed toward the slave, the master can continue sending data. However, if the master expects a reply from the slave, a Repeated Start sequence with the R/$\overline{W}$ status bit at '1' will change the R/$\overline{W}$ state of the message to a read of the slave.

Note:	In a 10-Bit Addressing mode, each node using the I^2C protocol should be configured with a unique address that is stored in the I2CxADD register. While transmitting the first address byte, the master must shift the address bits<9:8>, left by one bit, and configure bit 0 as the R/$\overline{W}$ bit.

5.2.4 RECEIVING ACKNOWLEDGE FROM THE SLAVE

On the falling edge of the eighth SCLx clock, the TBF status bit is cleared and the master will deassert the SDAx pin, allowing the slave to respond with an Acknowledge. The master will then generate a ninth SCLx clock.

This allows the slave device being addressed to respond with an $\overline{ACK}$ bit during the ninth bit time if an address match occurs or data was received properly. A slave sends an Acknowledge when it has recognized its device address (including a general call) or when the slave has properly received its data.

The status of $\overline{ACK}$ is written into the ACKSTAT bit (I2CxSTAT<15>) on the falling edge of the ninth SCLx clock. After the ninth SCLx clock, the module generates the MI2CxIF interrupt and enters into the Idle state until the next data byte is loaded into the I2CxTRN register.

5.2.5 ACKSTAT STATUS FLAG

The ACKSTAT bit (I2CxSTAT<15>) is cleared when the slave has sent an Acknowledge ($\overline{ACK}$ = 0) and is set when the slave does not Acknowledge ($\overline{ACK}$ = 1).

5.2.6 TBF STATUS FLAG

When transmitting, the TBF status bit (I2CxSTAT<0>) is set when the CPU writes to the I2CxTRN register and is cleared when all 8 bits are shifted out.

5.2.7 IWCOL STATUS FLAG

If the user software attempts to write to the I2CxTRN register when a transmit is already in progress (that is, the module is still shifting a data byte), the IWCOL status bit (I2CxSTAT<7>) is set and the contents of the buffer are unchanged (the write does not occur). The IWCOL status bit must be cleared in the user software.

Note:	Because queuing of events is not allowed, writing to the lower 5 bits of the I2CxCON or I2CxCONL register is disabled until the transmit condition is complete.

5.3 Receiving Data from a Slave Device

The master can receive data from the slave device after the master has transmitted the slave address with an R/$\overline{W}$ status bit value of '1'. This is enabled by setting the RCEN bit (I2CxCON<3> or I2CxCONL<3>). The master logic begins to generate clocks, and before each falling edge of the SCLx, the SDAx line is sampled and data is shifted into the I2CxRSR register.

Note:	The lower 5 bits of the I2CxCON or I2CxCONL register must be '0' before attempting to set the RCEN bit. This ensures that the master logic is inactive.

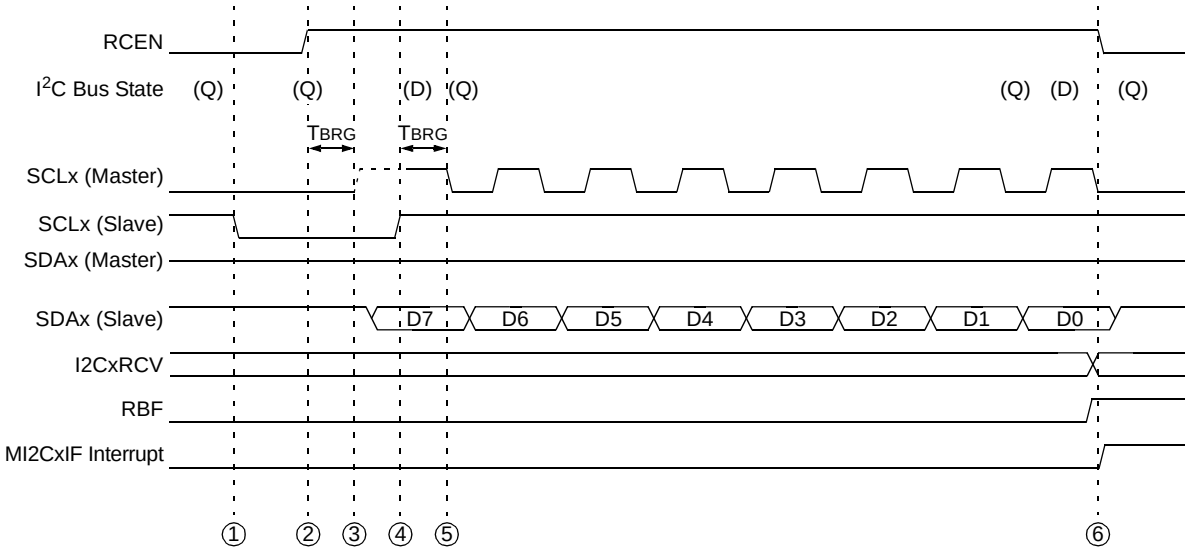
After the falling edge of the eighth SCLx clock, the following events occur:

- The RCEN bit is automatically cleared
- The contents of the I2CxRSR register transfer into the I2CxRCV register
- The RBF status bit (I2CxSTAT<1>) is set
- The I^2C module generates the MI2CxIF interrupt

When the CPU reads the receive buffer (I2CxRCV), the RBF status bit is automatically cleared. The user software can process the data and then execute an Acknowledge sequence.

The sequence of events that occurs during master transmission and master reception is illustrated in Figure 5-4.

Figure 5-4: Master Reception Timing Diagram



① Typically, the slave can pull SCLx low (clock stretch) to request a Wait to prepare the data response. The slave will drive the MSB of the data response on SDAx when ready.

② Writing the RCEN bit will start a master reception event. The BRG starts. SCLx remains low.

③ The BRG times out. The master attempts to release SCLx.

④ When the slave releases SCLx, the BRG restarts.

⑤ The BRG times out. The MSB of the response is shifted to the I2CxRSR register. SCLx is driven low for the next baud interval.

⑥ At the falling edge of the eighth SCLx clock, the I2CxRSR register is transferred to the I2CxRCV register. The module clears the RCEN bit. The RBF status bit is set. The master generates the interrupt.

© 2007-2015 Microchip Technology Inc.

5.3.1 RBF STATUS FLAG

When receiving data, the RBF status bit (I2CxSTAT<1>) is set when a device address or data byte is loaded into the I2CxRCV register from the I2CxRSR register. It is cleared when the user software reads the I2CxRCV register.

5.3.2 I2COV STATUS FLAG

If another byte is received in the I2CxRSR register while the RBF status bit remains set, and the previous byte remains in the I2CxRCV register, the I2COV status bit (I2CxSTAT<6>) is set and the data in the I2CxRSR register is lost.

Leaving the I2COV status bit set does not inhibit further reception. If the RBF status bit is cleared by reading the I2CxRCV register, and the I2CxRSR register receives another byte, that byte will be transferred to the I2CxRCV register.

5.3.3 IWCOL STATUS FLAG

If the user software writes to the I2CxTRN register when a receive is already in progress (that is, the I2CxRSR register is still shifting in a data byte), the IWCOL status bit (I2CxSTAT<7>) is set and the contents of the buffer are unchanged (the write does not occur).

Note:	Because queuing of events is not allowed, writing to the lower 5 bits of the I2CxCON register is disabled until the data reception condition is complete.

5.4 Acknowledge Generation

Setting the ACKEN bit (I2CxCON<4> or I2CxCONL<4>) enables the generation of a master Acknowledge sequence.

Note:	The lower 5 bits of the I2CxCON or I2CxCONL register must be '0' (master logic inactive) before attempting to set the ACKEN bit.

Figure 5-5 illustrates an $\overline{ACK}$ sequence and Figure 5-6 illustrates a NACK sequence. The ACKDT bit (I2CxCON<5> or I2CxCONL<5>) specifies an $\overline{ACK}$ or NACK sequence.

After two baud periods, the ACKEN bit is automatically cleared and the module generates the MI2CxIF interrupt.

5.4.1 IWCOL STATUS FLAG

If the user software writes the I2CxTRN register when an Acknowledge sequence is in progress, the IWCOL status bit (I2CxSTAT<7>) is set and the contents of the buffer are unchanged (the write does not occur).

Note:	Because queuing of events is not allowed, writing to the lower 5 bits of the I2CxCON or I2CxCONL register is disabled until the Acknowledge condition is complete.

Figure 5-5: Master Acknowledge ($\overline{ACK}$) Timing Diagram

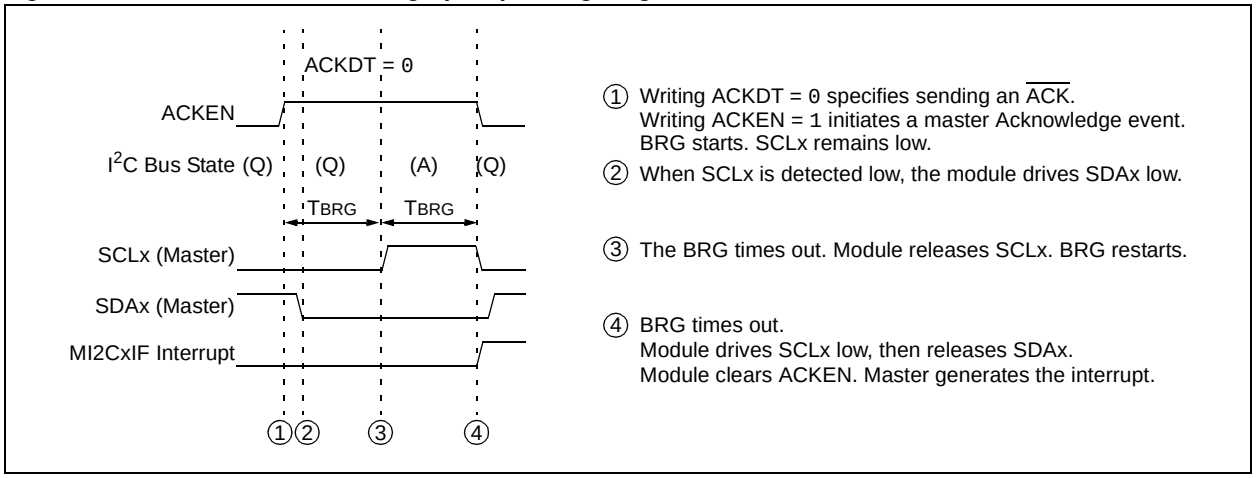
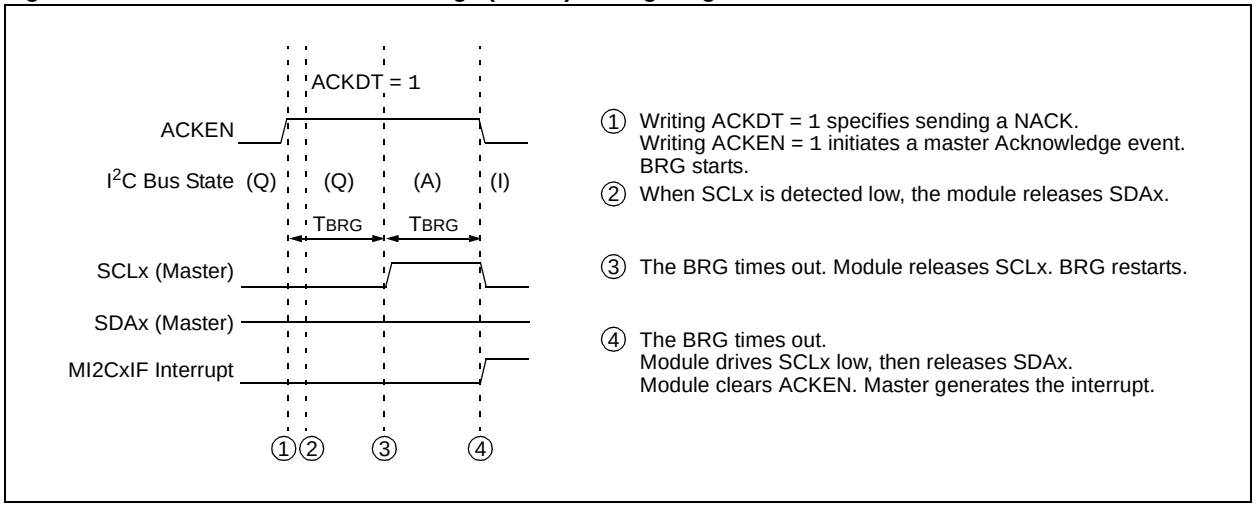


Figure 5-6: Master Not Acknowledge (NACK) Timing Diagram



© 2007-2015 Microchip Technology Inc.

5.5 Generating a Stop Bus Event

Setting the PEN bit (I2CxCON<2> or I2CxCONL<2>) enables the generation of a master Stop sequence.

Note:	The lower 5 bits of the I2CxCON or I2CxCONL register must be '0' (master logic inactive) before attempting to set the PEN bit.

When the PEN bit is set, the master generates the Stop sequence, as illustrated in Figure 5-7.

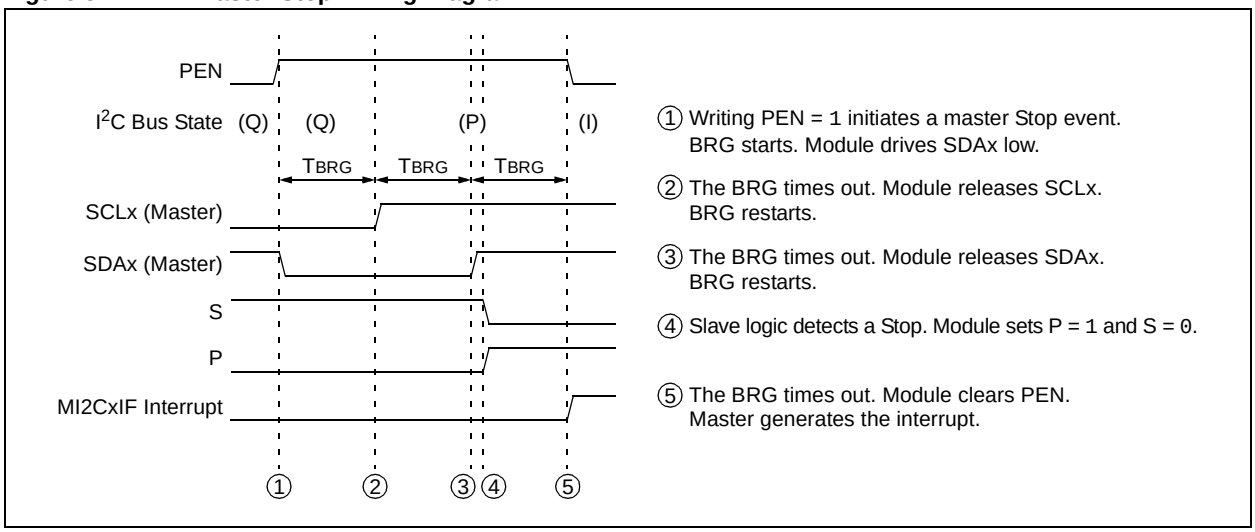
- The slave detects the Stop condition, sets the P status bit (I2CxSTAT<4>) and clears the S status bit (I2CxSTAT<3>)
- The PEN bit is automatically cleared
- The module generates the MI2CxIF interrupt

5.5.1 IWCOL STATUS FLAG

If the user software writes the I2CxTRN register when a Stop sequence is in progress, the IWCOL status bit (I2CxSTAT<7>) is set and the contents of the buffer are unchanged (the write does not occur).

Note:	Because queuing of events is not allowed, writing to the lower 5 bits of the I2CxCON or I2CxCONL register is disabled until the Stop condition is complete.

Figure 5-7: Master Stop Timing Diagram



5.6 Generating a Repeated Start Bus Event

Setting the RSEN bit (I2CxCON<1> or I2CxCONL<1>) enables the generation of a master Repeated Start sequence, as illustrated in Figure 5-8.

Note:	The lower 5 bits of the I2CxCON or I2CxCONL register must be '0' (master logic inactive) before attempting to set the RSEN bit.

To generate a Repeated Start condition, the user software sets the RSEN bit. The master module asserts the SCLx pin low. When the module samples the SCLx pin low, the module releases the SDAx pin for 1 TBRG. When the BRG times out and the module samples SDAx high, the module deasserts the SCLx pin. When the module samples the SCLx pin high, the BRG reloads and begins counting. SDAx and SCLx must be sampled high for 1 TBRG. This action is then followed by assertion of the SDAx pin low for 1 TBRG while SCLx is high.

The following is the Repeated Start sequence:

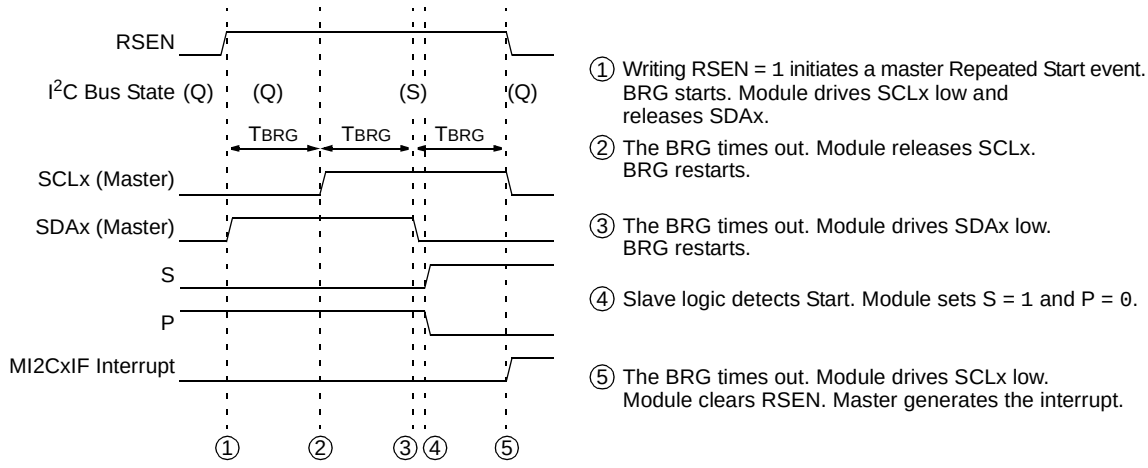
1. The slave detects the Start condition, sets the S status bit (I2CxSTAT<3>) and clears the P status bit (I2CxSTAT<4>).
2. The RSEN bit is automatically cleared.
3. The I^2C module generates the MI2CxIF interrupt.

5.6.1 IWCOL STATUS FLAG

If the user software writes the I2CxTRN register when a Repeated Start sequence is in progress, the IWCOL status bit (I2CxSTAT<7>) is set and the contents of the buffer are not changed (the write does not occur).

Note:	Because queuing of events is not allowed, writing of the lower 5 bits of the I2CxCON or I2CxCONL register is disabled until the Repeated Start condition is complete.

Figure 5-8: Master Repeated Start Timing Diagram



© 2007-2015 Microchip Technology Inc.

5.7 Building Complete Master Messages

As described in **Section 5.0 "Communicating as a Master in a Single Master Environment"**, the user software is responsible for constructing messages with the correct message protocol. The module controls individual portions of the I^2C message protocol; however, sequencing of the components of the protocol to construct a complete message is performed by the user software.

The user software can use polling or interrupt methods while using the module. The timing diagrams shown in this document use interrupts for detecting various events.

The user software can use the SEN, RSEN, PEN, RCEN and ACKEN bits (Least Significant 5 bits of the I2CxCON or I2CxCONL register), and the TRSTAT status bit as a 'state' flag when progressing through a message. For example, Table 5-1 shows some example state numbers associated with bus states.

Table 5-1: Master Message Protocol States

Example State Number(1)	I2CxCON<4:0> or I2CxCONL<4:0>	TRSTAT (I2CxSTAT<14>)	State
0	00000	0	Bus Idle or Wait
1	00001	N/A	Sending Start Event
2	00000	1	Master Transmitting
3	00010	N/A	Sending Repeated Start Event
4	00100	N/A	Sending Stop Event
5	01000	N/A	Master Reception
6	10000	N/A	Master Acknowledgment

Note 1: The example state numbers are for reference only. The user software can assign the state numbers as desired.

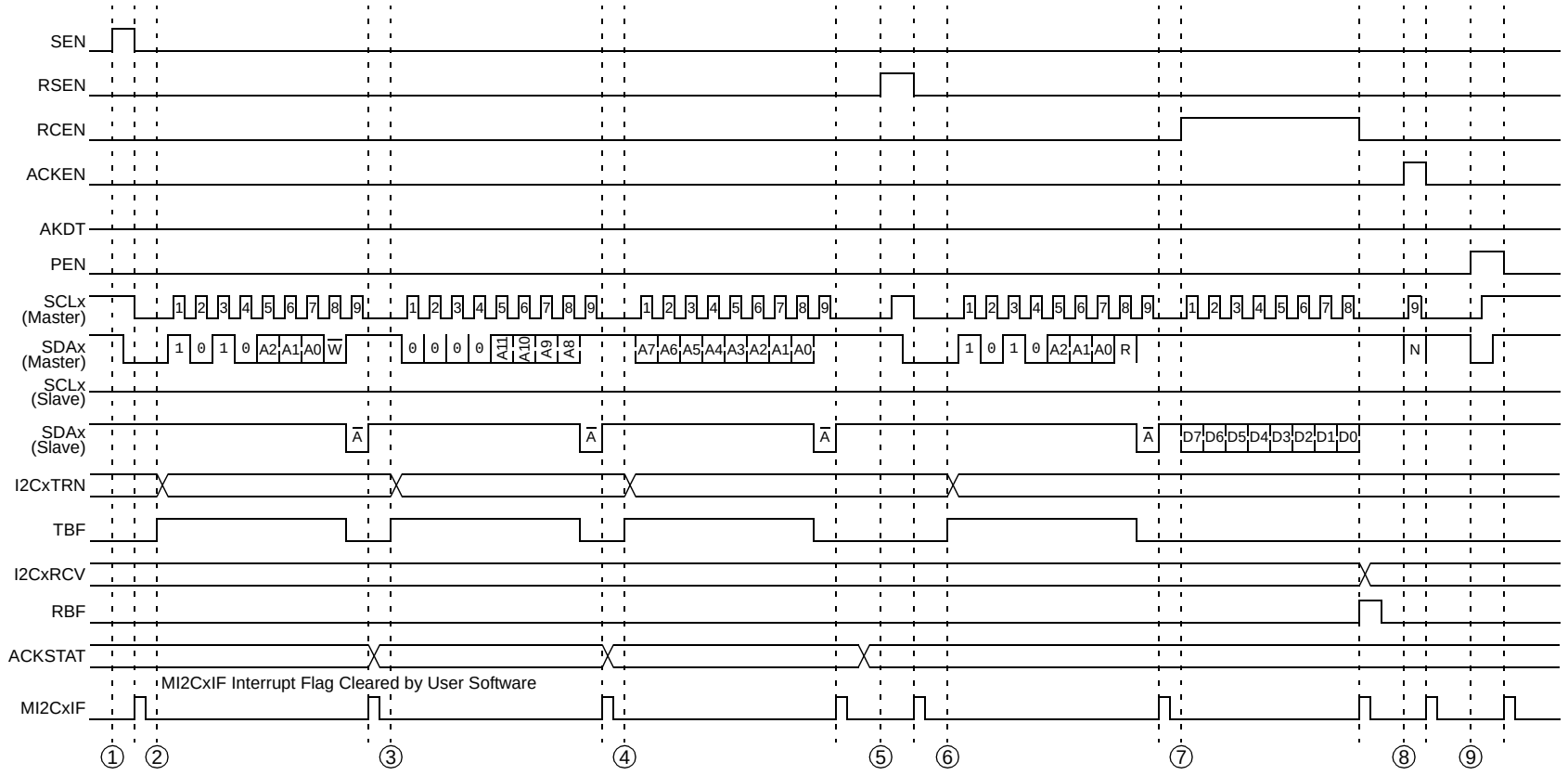
The user software will begin a message by issuing a Start condition. The user software will record the state number corresponding to the Start.

As each event completes and generates an interrupt, the interrupt handler may check the state number. Therefore, for a Start state, the interrupt handler will confirm execution of the Start sequence and then start a master transmission event to send the I^2C device address, changing the state number to correspond to the master transmission.

On the next interrupt, the interrupt handler will again check the state, determining that a master transmission just completed. The interrupt handler will confirm successful transmission of the data, then move on to the next event, depending on the contents of the message. In this manner, on each interrupt, the interrupt handler will progress through the message protocol until the complete message is sent.

Figure 5-9 provides a detailed examination of the same message sequence as shown in Figure 5-1. Figure 5-10 provides a few simple examples of the messages using a 7-bit addressing format. Figure 5-11 provides an example of a 10-bit addressing format message sending data to a slave. Figure 5-12 provides an example of a 10-bit addressing format message receiving data from a slave.

Figure 5-9: Master Message (Typical I^2C Message: Read of Serial EEPROM)



① Setting the SEN bit begins a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the serial EEPROM device address byte, with the R/W status bit clear, indicating a write.

③ Writing the I2CxTRN register starts a master transmission. The data is the first byte of the EEPROM data address.

④ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the EEPROM data address.

⑤ Setting the RSEN bit starts a Repeated Start event.

⑥ Writing the I2CxTRN register starts a master transmission. The data is a resend of the serial EEPROM device address byte, but with R/W status bit set, indicating a read.

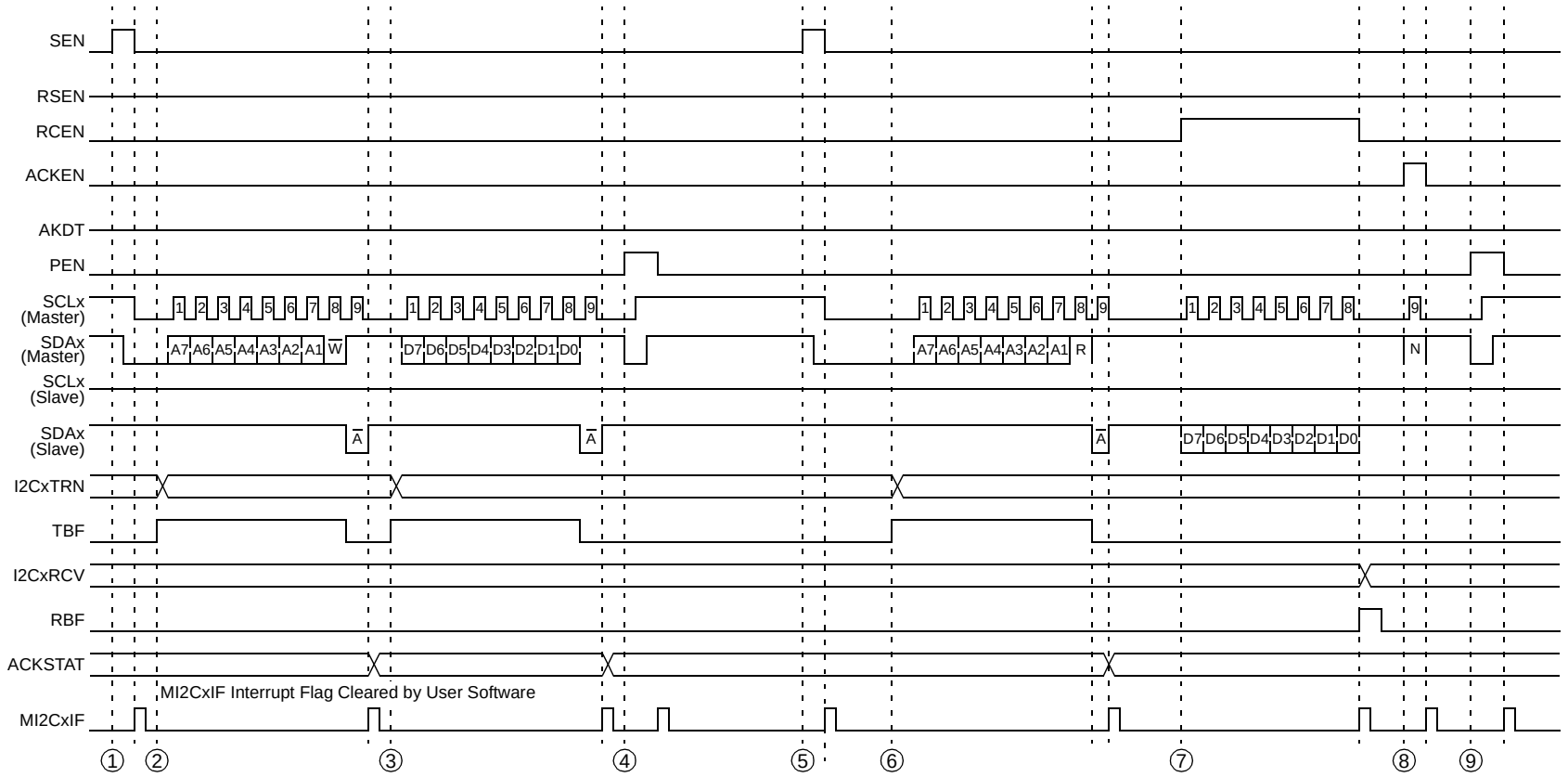
⑦ Setting the RCEN bit starts a master reception. On interrupt, the user software reads the I2CxRCV register, which clears the RBF status bit.

⑧ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 1 to send a NACK.

⑨ Setting the PEN bit starts a master Stop event.

© 2007-2015 Microchip Technology Inc.

Figure 5-10: Master Message (7-Bit Address: Transmission and Reception)



① Setting the SEN bit begins a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the address byte with the R/$\overline{W}$ status bit clear.

③ Writing the I2CxTRN register starts a master transmission. The data is the message byte.

④ Setting the PEN bit starts a master Stop event.

⑤ Setting the SEN bit begins a Start event. An interrupt is generated on completion of the Start event.

⑥ Writing the I2CxTRN register starts a master transmission. The data is the address byte with the R/$\overline{W}$ status bit set.

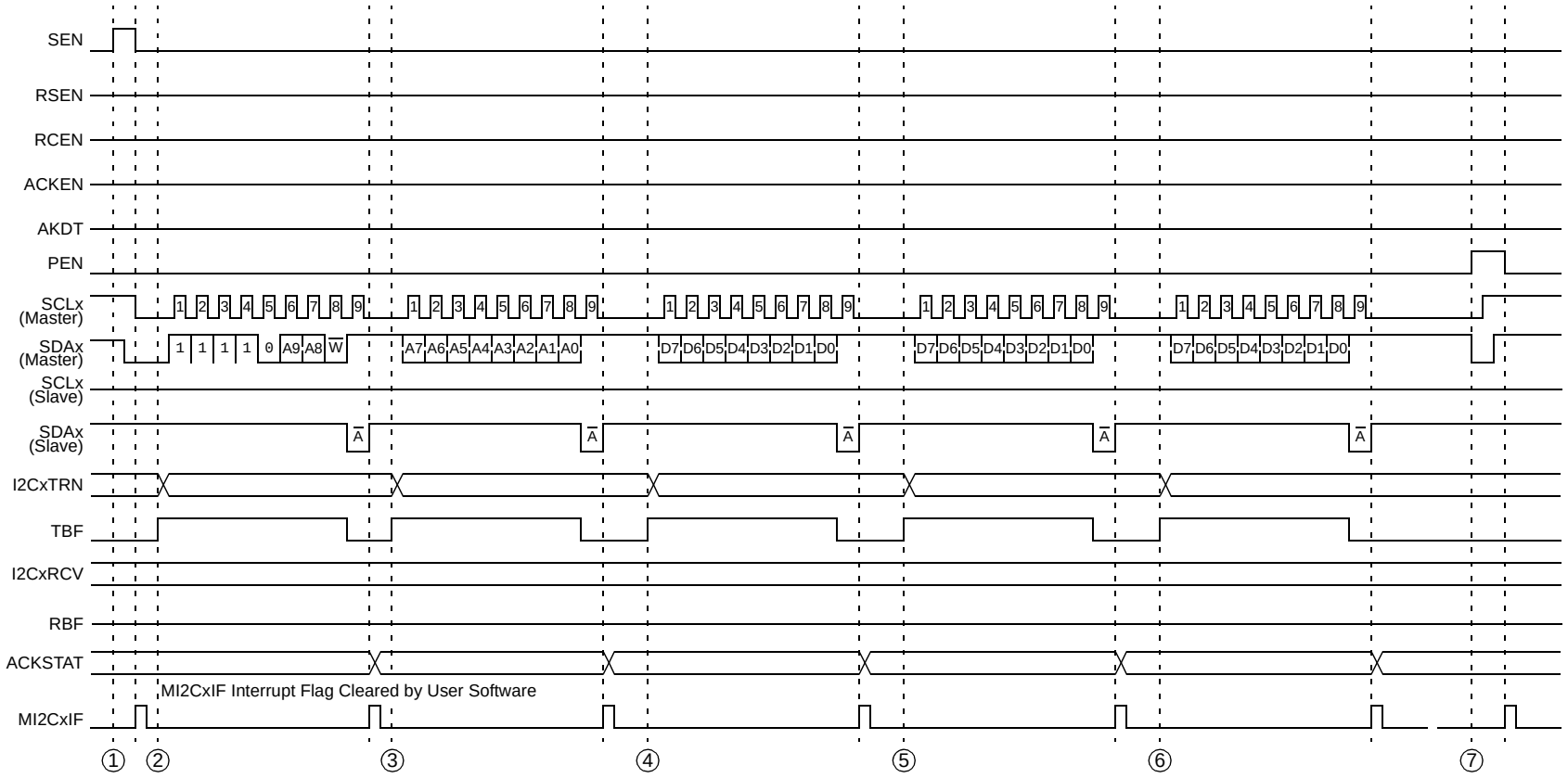
⑦ Setting the RCEN bit starts a master reception.

⑧ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 1 to send a NACK.

⑨ Setting the PEN bit starts a master Stop event.

© 2007-2015 Microchip Technology Inc.

Figure 5-11: Master Message (10-Bit Transmission)



① Setting the SEN bit begins a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the first byte of the address.

③ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the address.

④ Writing the I2CxTRN register starts a master transmission. The data is the first byte of the message data.

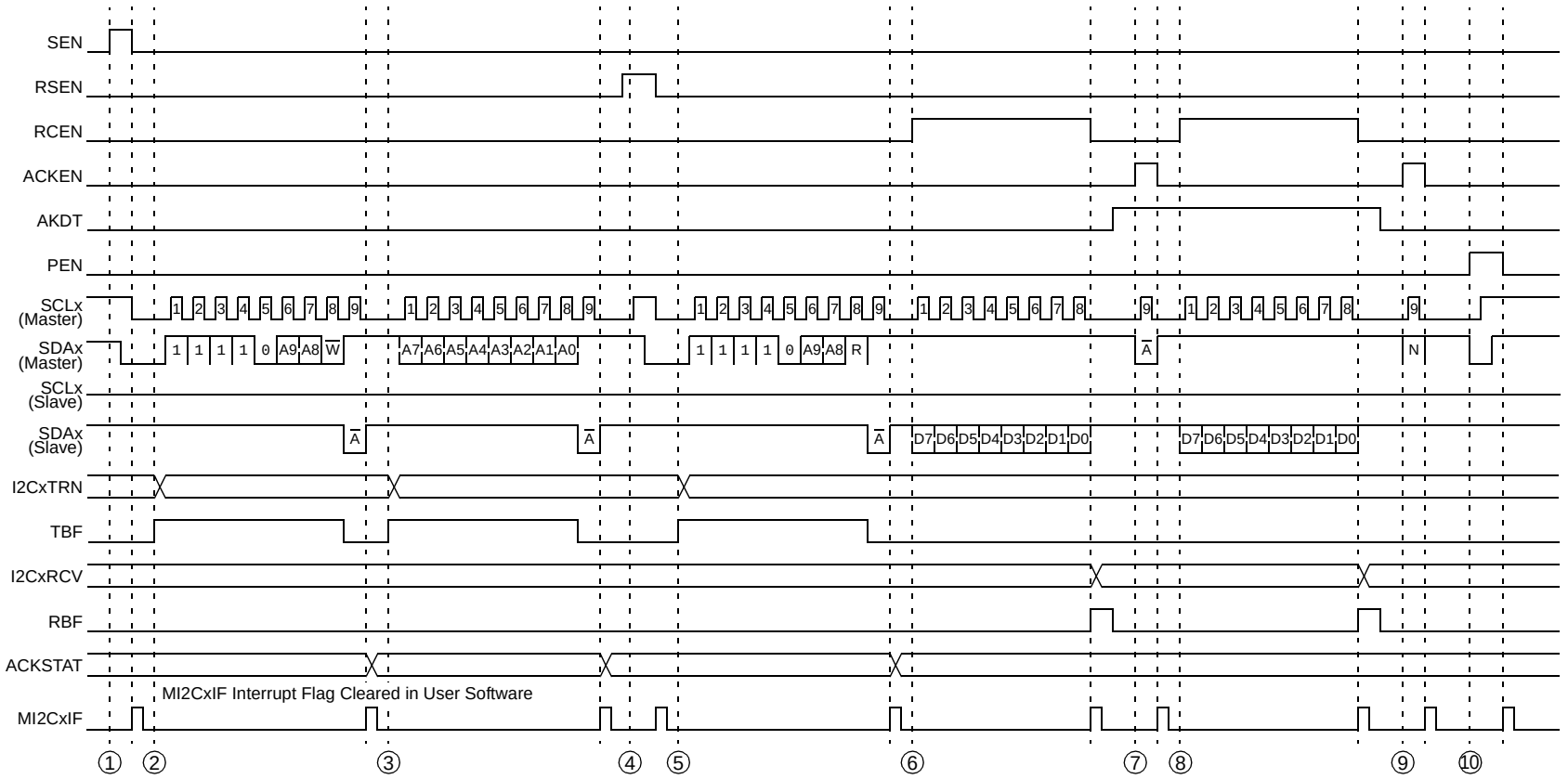
⑤ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the message data.

⑥ Writing the I2CxTRN register starts a master transmission. The data is the third byte of the message data.

⑦ Setting the PEN bit starts a master Stop event.

© 2007-2015 Microchip Technology Inc.

Figure 5-12: Master Message (10-Bit Reception)



① Setting the SEN bit begins a Start event.

② Writing the I2CxTRN register starts a master transmission. The data is the first byte of the address with the R/W status bit cleared.

③ Writing the I2CxTRN register starts a master transmission. The data is the second byte of the address.

④ Setting the RSEN bit starts a master Restart event.

⑤ Writing the I2CxTRN register starts a master transmission. The data is a resend of the first byte with the R/W status bit set.

⑥ Setting the RCEN bit starts a master reception. On interrupt, the user software reads the I2CxRCV register, which clears the RBF status bit.

⑦ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 0 to send $\overline{\text{ACK}}$.

⑧ Setting the RCEN bit starts a master reception.

⑨ Setting the ACKEN bit starts an Acknowledge event. ACKDT = 1 to send NACK.

⑩ Setting the PEN bit starts a master Stop event.

© 2007-2015 Microchip Technology Inc.

6.0 COMMUNICATING AS A MASTER IN A MULTI-MASTER ENVIRONMENT

The I^2C protocol allows more than one master to be attached to a system bus. Taking into account that a master can initiate message transactions and generate clocks for the bus, the protocol has methods to account for situations where more than one master is attempting to control the bus. The clock synchronization ensures that multiple nodes can synchronize their SCLx clocks to result in one common clock on the SCLx line. The bus arbitration ensures that if more than one node attempts a message transaction, only one node will be successful in completing the message. The other nodes lose bus arbitration and are left with a bus collision.

Note: The IPMIEN bit (I2CxCON<11>) should not be set when operating as a master.

6.1 Multi-Master Operation

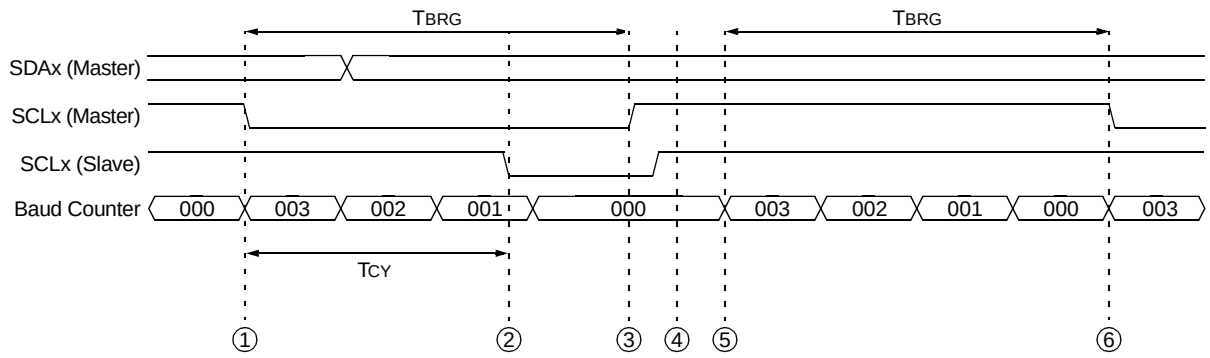
The master module has no special settings to enable the multi-master operation. The module performs the clock synchronization and bus arbitration at all times. If the module is used in a single master environment, clock synchronization only occurs between the master and slaves, and bus arbitration does not occur.

6.2 Master Clock Synchronization

In a multi-master system, different masters can have different baud rates. The clock synchronization ensures that when these masters are attempting to arbitrate the bus, their clocks will be coordinated.

The clock synchronization occurs when the master deasserts the SCLx pin (SCLx intended to float high). When the SCLx pin is released, the BRG is suspended from counting until the SCLx pin is actually sampled high. When the SCLx pin is sampled high, the BRG is reloaded with the contents of I2CxBRG<8:0> and begins counting. This ensures that the SCLx high time will always be at least one BRG rollover count in the event that the clock is held low by an external device, as illustrated in Figure 6-1.

Figure 6-1: Baud Rate Generator Timing with Clock Synchronization



① The baud counter decrements twice per TCY. On rollover, the master SCLx will transition.

② The slave has pulled SCLx low to initiate a Wait.

③ At what would be the master baud counter rollover, detecting SCLx low holds the counter.

④ Logic samples SCLx once per TCY. Logic detects SCLx high.

⑤ The baud counter rollover occurs on the next cycle.

⑥ On the next rollover, the master SCLx will transition.

© 2007-2015 Microchip Technology Inc.

6.3 Bus Arbitration and Bus Collision

The bus arbitration supports the multi-master system operation. The wired-AND nature of the SDAx line permits arbitration. Arbitration takes place when the first master outputs '1' on SDAx by letting the SDAx float high, and simultaneously, the second master outputs '0' on SDAx by pulling SDAx low. The SDAx signal will go low. In this case, the second master has won bus arbitration. The first master has lost bus arbitration, and thus, has a bus collision.

For the first master, the expected data on SDAx is '1', still the data sampled on SDAx is '0'. This is the definition of a bus collision.

The first master will set the BCL bit (I2CxSTAT<10>) and generates a master (MI2CxIF) or a bus collision (I2CxBCIF) interrupt. The Master module will reset the I^2C port to its Idle state.

In multi-master operation, the SDAx line must be monitored for arbitration to see if the signal level is the expected output level. This check is performed by the master logic, with the result placed in the BCL status bit.

The states where arbitration can be lost are:

- Start condition
- Repeated Start condition
- Address, Data or Acknowledge bit
- Stop condition

Note:	The bus collision interrupt is not available on all devices. Refer to the specific device data sheet for availability.

6.4 Detecting Bus Collisions and Resending Messages

When a bus collision occurs, the master module sets the BCL status bit and generates a master (MI2CxIF) or a bus collision (I2CxBCIF) interrupt. If a bus collision occurs during a byte transmission, the transmission is stopped, the TBF status bit is cleared, and the SDAx and SCLx pins are deasserted. If a bus collision occurs during a Start, Repeated Start, Stop or Acknowledge condition, the condition is aborted, the respective control bits in the I2CxCON register are cleared, and the SDAx and SCLx lines are deasserted.

If the user software is expecting an interrupt at the completion of the master event, the user software can check the BCL status bit to determine if the master event completed successfully or a bus collision occurred (or it may branch to a bus collision interrupt on bus collision), or a master interrupt occurred in case of a successful master event. If a bus collision occurs, the user software must abort sending the rest of the pending message and prepare to resend the entire message sequence, beginning with the Start condition, after the bus returns to the Idle state. The user software can monitor the S and P status bits to wait for an Idle bus. When the user software executes the master Interrupt Service Routine (ISR) and the I^2C bus is free, the user software can resume communication by asserting a Start condition.

Note:	The bus collision interrupt is not available on all devices. Refer to the specific device data sheet for availability.

6.5 Bus Collision During a Start Condition

Before issuing a Start condition, the user software should verify an Idle state of the bus using the S and P status bits. Two masters may attempt to initiate a message at a similar point in time. Typically, the masters will synchronize clocks and continue arbitration into the message until one loses arbitration. Any of the following conditions can cause a bus collision to occur during a Start:

- If the SDAx and SCLx pins are at a low logic state at the beginning of the Start condition
- If the SCLx line is at a low logic state before the SDAx line is driven low

In either case, the master that loses arbitration during the Start condition generates a bus collision interrupt.

6.6 Bus Collision During a Repeated Start Condition

When the two masters do not collide throughout an address byte, a bus collision can occur when one master attempts to assert a Repeated Start while another transmits data. In this case, the master generating the Repeated Start loses arbitration and generates a bus collision interrupt.

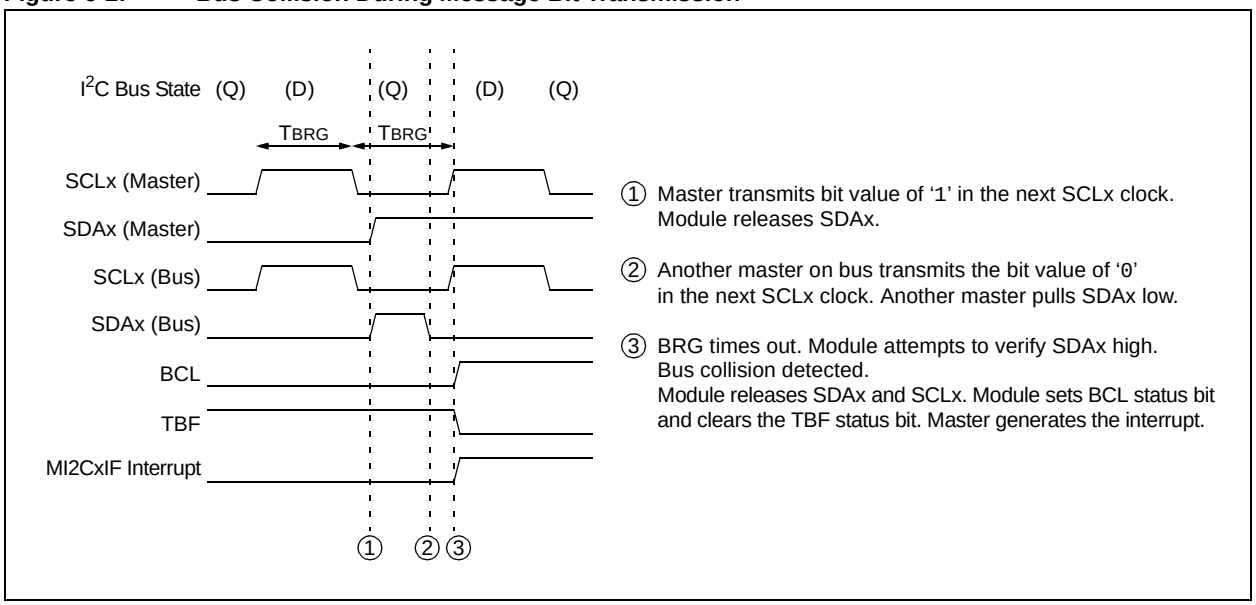
Note:	The bus collision interrupt is not available on all devices. Refer to the specific device data sheet for availability.

6.7 Bus Collision During Message Bit Transmission

The most typical case of data collision occurs while the master is attempting to transmit the device address byte, a data byte or an Acknowledge bit.

If the user software is properly checking the bus state, it is unlikely that a bus collision will occur on a Start condition. However, because another master can, at the same time, check the bus and initiate its own Start condition, it is likely that SDAx arbitration will occur and synchronize the Start of two masters. In this condition, both masters begin and continue to transmit their messages until one master loses arbitration on a message bit. The SCLx clock synchronization keeps the two masters synchronized until one loses arbitration. Figure 6-2 illustrates an example of the message bit arbitration.

Figure 6-2: Bus Collision During Message Bit Transmission



6.8 Bus Collision During a Stop Condition

If the master software loses track of the state of the I2C bus, many existing conditions can cause a bus collision during a Stop condition. In this case, the master generating the Stop condition will lose arbitration and generate a bus collision interrupt.

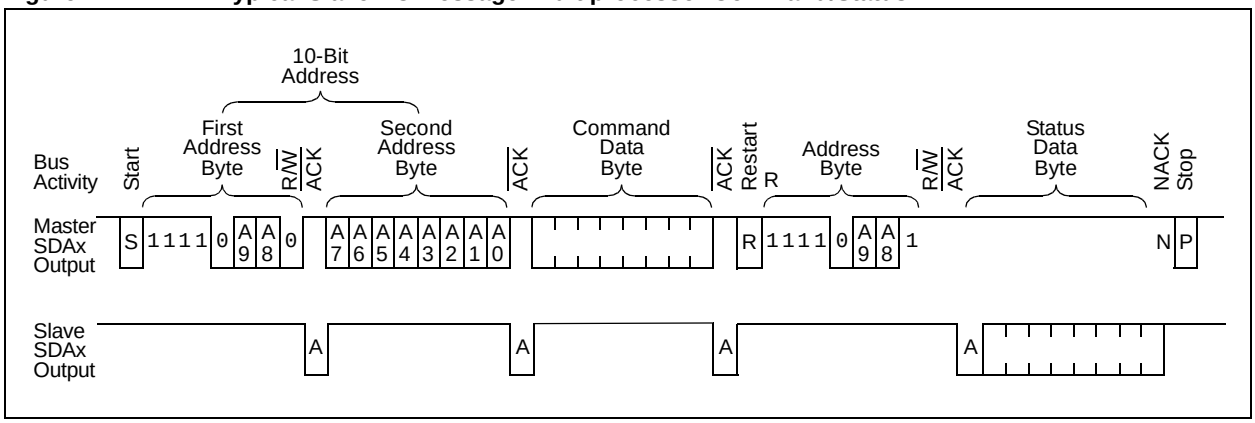
© 2007-2015 Microchip Technology Inc.

7.0 COMMUNICATING AS A SLAVE

In some systems, particularly where multiple processors communicate with each other, the dsPIC33/PIC24 device can communicate as a slave, as illustrated in Figure 7-1. When the I^2C module is enabled, the slave is active. The slave cannot initiate a message; it can only respond to a message sequence initiated by a master. The master requests a response from a particular slave as defined by the device address byte in the I^2C protocol. The slave replies to the master at the appropriate times as defined by the protocol.

As with the master module, sequencing the components of the protocol for the reply is a user software task. However, the slave detects when the device address matches the address specified by the user software for that slave.

Figure 7-1: A Typical Slave I^2C Message: Multiprocessor Command/Status



After a Start condition, the slave receives and checks the device address. The slave can specify either a 7-bit address or a 10-bit address. When a device address is matched, the module will generate an interrupt to notify the user software that its device is selected. Based on the R/$\overline{W}$ status bit sent by the master, the slave either receives or transmits data. If the slave is to receive data, the slave automatically generates the Acknowledge ($\overline{ACK}$), loads the I2CxRCV register with the received value currently in the I2CxRSR register and notifies the user software through an interrupt. If the slave is to transmit data, the user software must load the I2CxTRN register.

7.1 Sampling Receive Data

All the incoming bits are sampled with the rising edge of the clock (SCLx) line.

7.2 Detecting Start and Stop Conditions

The slave detects the Start and Stop conditions on the bus and indicates that status on the S status bit (I2CxSTAT<3>) and P status bit (I2CxSTAT<4>). The Start (S) and Stop (P) status bits are cleared when a Reset occurs or when the module is disabled. After detection of a Start or Repeated Start event, the S status bit is set and the P status bit is cleared. After detection of a Stop event, the P status bit is set and the S status bit is cleared.

7.2.1 INTERRUPT ON START/REPEATED START AND STOP CONDITIONS (SLAVE MODE)

The user software is notified through a slave interrupt if the SCIE bit (I2CxCONH<5>) is set for a Start/Repeated Start condition or if the PCIE bit (I2CxCONH<6>) is set for a Stop condition.

Note:	The PCIE and the SCIE bits are not available on all devices. Refer to the specific device data sheet for availability.

7.3 Detecting the Address

Once the module has been enabled, the slave waits for a Start condition to occur. After a Start, depending on the A10M bit (I2CxCON<10> or I2CxCONL<10>), the slave attempts to detect a 7-bit or 10-bit address. The slave compares one received byte for a 7-bit address or two received bytes for a 10-bit address. A 7-bit address also contains an R/$\overline{W}$ status bit that specifies the direction of the data transfer after the address. If R/$\overline{W}$ = 0, a write is specified and the slave receives data from the master. If R/$\overline{W}$ = 1, a read is specified and the slave sends data to the master. The 10-bit address contains an R/$\overline{W}$ status bit; however, by definition, it is always R/$\overline{W}$ = 0 because the slave must receive the second byte of the 10-bit address.

7.3.1 SLAVE ADDRESS MASKING

The I2CxMSK register masks the address bit positions, designating them as “don’t care” bits for both 10-Bit and 7-Bit Addressing modes. When a bit in the I2CxMSK register is set (= 1), the slave responds when the bit in the corresponding location of the address is a ‘0’ or ‘1’. For example, in 7-Bit Slave mode with I2CxMSK = 0100000, the slave module Acknowledges addresses, ‘0000000’ and ’0100000’, as valid.

To enable address masking, the IPMI must be disabled by clearing the IPMIEN bit (I2CxCON<11>).

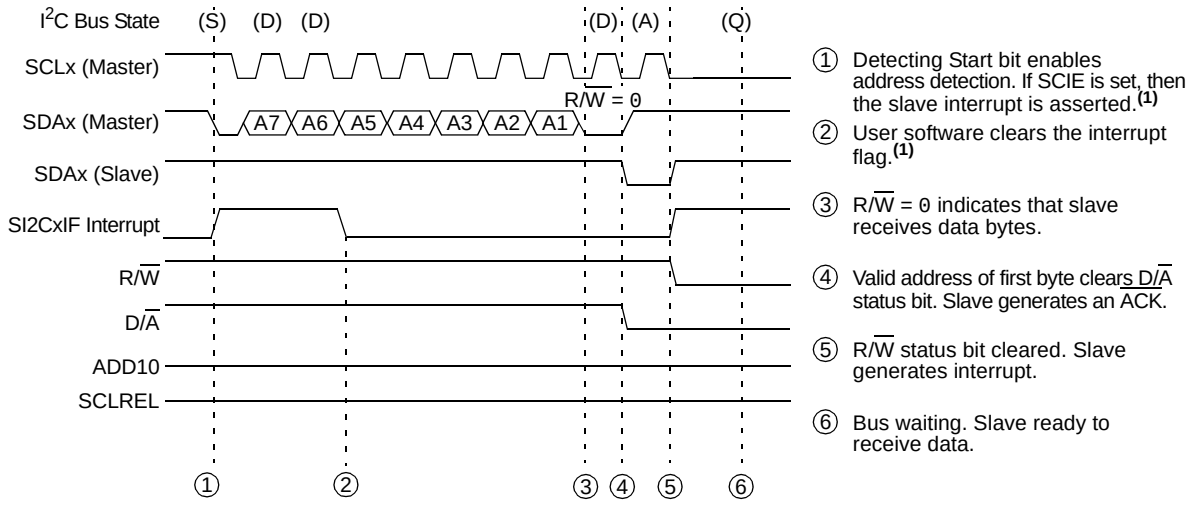
7.3.2 7-BIT ADDRESS AND SLAVE WRITE

After the Start condition, the module shifts 8 bits into the I2CxRSR register, as illustrated in Figure 7-2. The value of the I2CxRSR register is evaluated against that of the I2CxADD and I2CxMSK registers on the falling edge of the eighth clock (SCLx). If the address is valid (that is, an exact match between unmasked bit positions), the following events occur:

- An $\overline{ACK}$ is generated if the AHEN bit is clear
- The D/$\overline{A}$ and R/$\overline{W}$ status bits are cleared
- The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock
- The module waits for the master to send data

Note:	The AHEN bit may not be available on all devices. Refer to the specific device data sheet for availability. If this bit is not present, then the device will generate an ($\overline{ACK}$) on an address match.

Figure 7-2: Slave Write 7-Bit Address Detection Timing Diagram



Note 1: The SCIE bit may not be available on all devices. Refer to the specific device data sheet for availability.

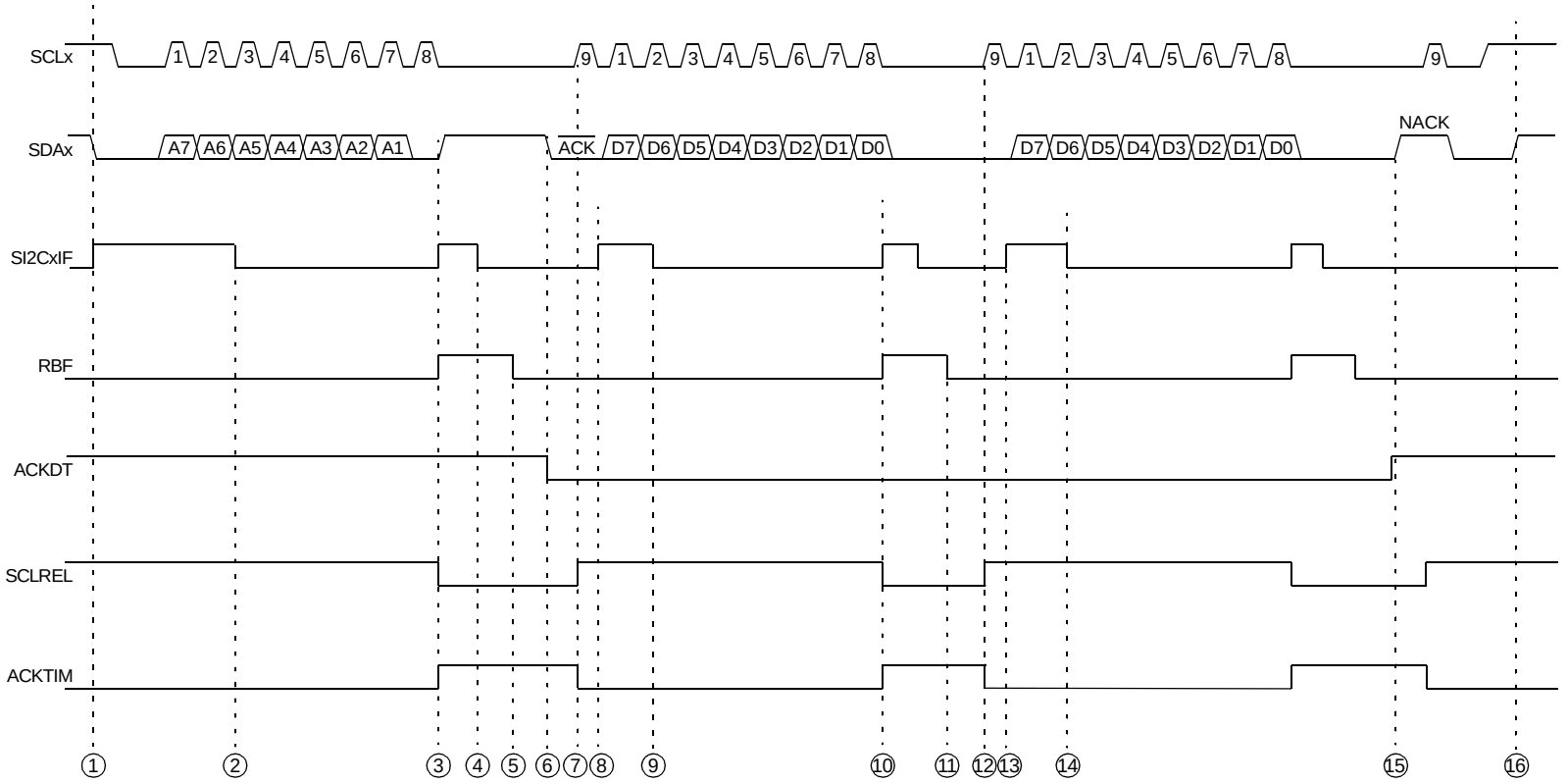
© 2007-2015 Microchip Technology Inc.

7.3.3 7-BIT ADDRESS AND SLAVE WRITE WITH THE AHEN AND DHEN BITS

The slave device reception, with the AHEN and DHEN bits set, operates with extra interrupts and clock stretching added after the eighth falling edge of SCLx. These additional interrupts allow the slave software to decide whether it wants to ACK the receive address or data byte, rather than the hardware. This functionality adds support for the PMBus™ that was not present on previous versions of this module.

Note:	The SI2CxIF interrupt is still set after the ninth falling edge of the SCLx clock, even if there is no clock stretching and the RBF bit has been cleared. The SI2CxIF interrupt is not asserted if a NACK is sent to the master.

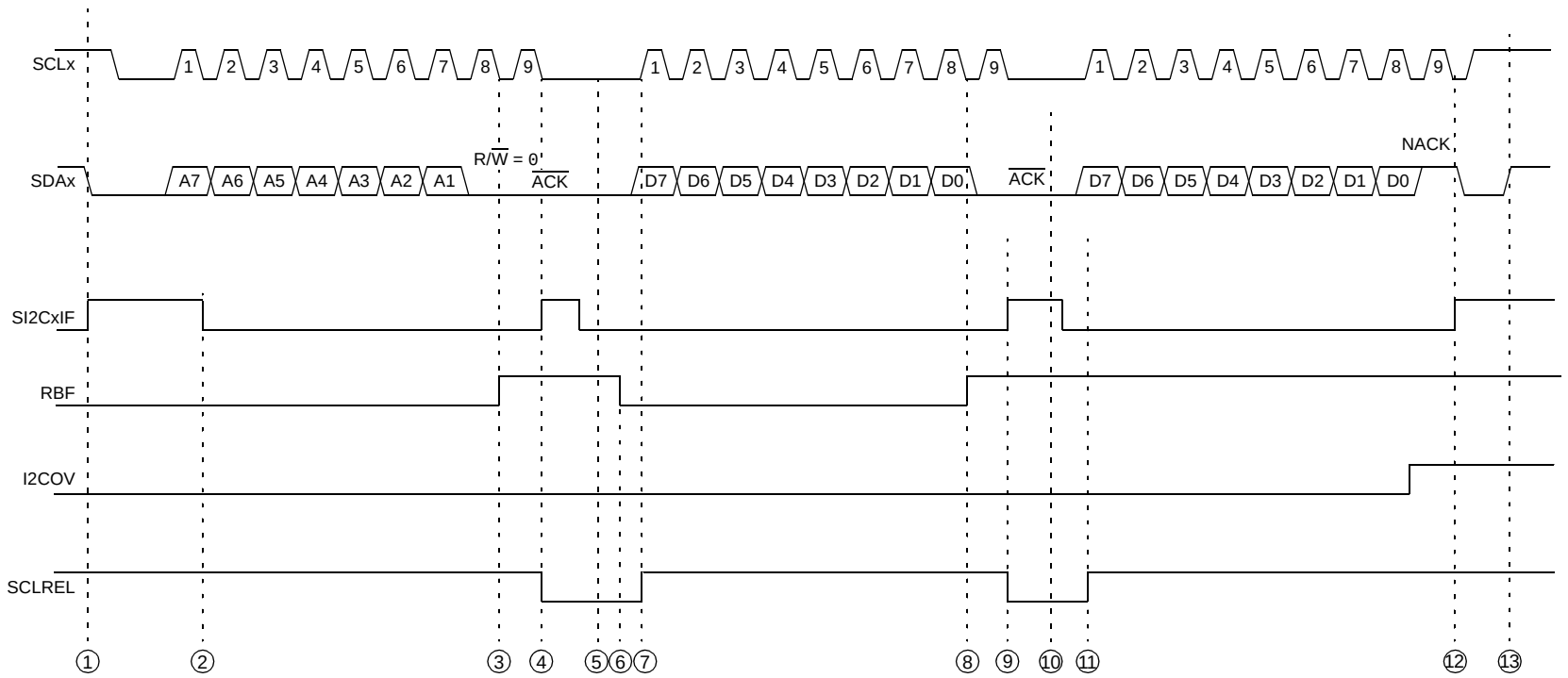
Figure 7-3: I²C Slave, 7-Bit Address, Reception (STREN = 0, AHEN = 1, DHEN = 1)



1. Detecting Start bit enable address detection, interrupt flag is set if SCEN is set.
2. User software clears the interrupt flag.
3. Slave receives the address byte with R/$\overline{W}$ = 0. Hardware clears SCLREL. Interrupt flag is asserted. ACKTIM is asserted. I2CxRCV is loaded with I2CxRSR and RBF is asserted.
4. User software clears the interrupt flag.
5. User software reads I2CxRCV, that clears the RBF flag.
6. ACKDT is written with $\overline{ACK}$ by user software.
7. User software sets SCLREL bit to release clock, ACKTIM is cleared by hardware.
8. Interrupt flag is set (not set if NACK is received).
9. User software clears the interrupt flag.
10. If DHEN = 1, hardware clears the SCLREL bit. I2CxRCV is loaded with I2CxRSR; ACKTIM is asserted at the end of 8th falling edge of SCLx by hardware.
11. User software reads I2CxRCV; clears the RBF flag.
12. User software releases the SCLREL bit, ACKTIM is cleared by hardware.
13. Interrupt flag is set.
14. User software clears the interrupt flag.
15. NACK.
16. Slave recognizes the Stop event.

© 2007-2015 Microchip Technology Inc.

Figure 7-4: I^2C Slave, 7-Bit Address, Reception (STREN = 1, AHEN = 0, DHEN = 0)



① Detecting Start bit, enables address detection, interrupt is set if SCEN is set.

② User software clears the interrupt flag.

③ RBF is set on the 8th falling clock, address is loaded into I2CxRCV. RBF is asserted.

④ Interrupt is asserted.

⑤ SCLx is stretched low until SCLREL is set.

⑥ User software reads the I2CxRCV buffer, that clears the RBF flag.

⑦ User software releases the SCLx line by writing SCLREL to '1'.

⑧ Data is loaded into I2CxRCV. RBR flag is asserted.

⑨ On the 9th falling clock edge, interrupt is asserted.

⑩ SCLx is stretched and held low until SCLREL is set.

⑪ User software releases SCLx line by writing SCLREL to '1'.

⑫ NACK is received (SCLx is not stretched to low).

⑬ Slave recognizes the Stop event.

© 2007-2015 Microchip Technology Inc.

7.3.4 7-BIT ADDRESS AND SLAVE READ

When a slave read is specified by having R/$\overline{W}$ = 1 in a 7-bit address byte, the process of detecting the device address is similar to that of a slave write, as illustrated in Figure 7-5. If the addresses match, the following events occur:

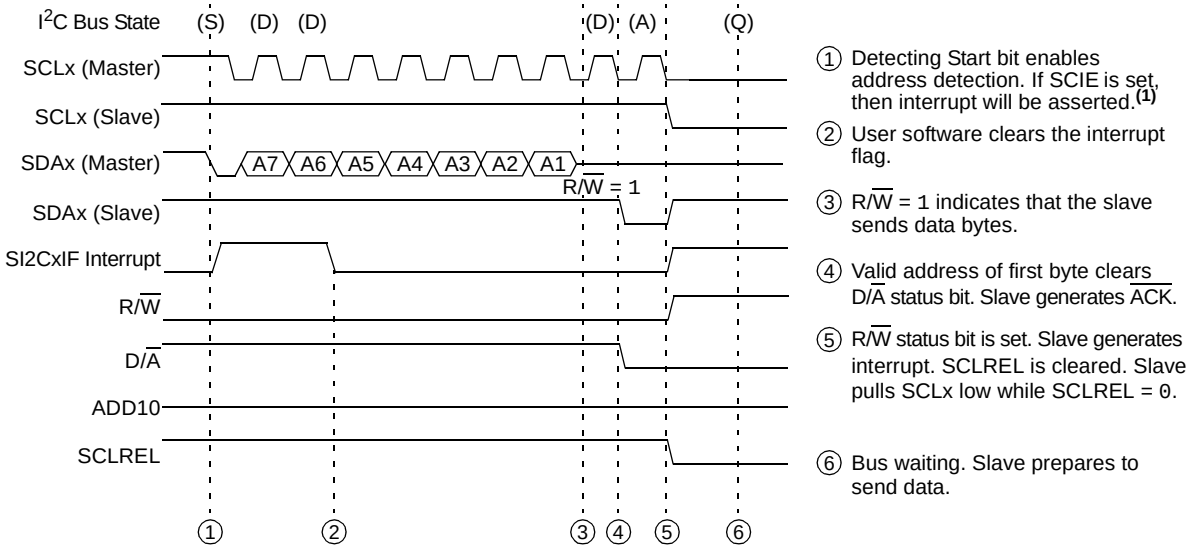
- An $\overline{ACK}$ is generated if the AHEN bit is clear
- The D/$\overline{A}$ status bit is cleared and the R/$\overline{W}$ status bit is set
- The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock

Because the slave is expected to reply with data at this point, it is necessary to suspend the operation of the I^2C bus to allow the user software to prepare a response. This is done automatically when the module clears the SCLREL bit. With SCLREL low, the slave will pull down the SCLx clock line, causing a Wait on the I^2C bus. The slave and the I^2C bus remain in this state until the user software writes the I2CxTRN register with the response data and sets the SCLREL bit.

Note: For more information on the AHEN and DHEN bits, refer to section **Section 7.3.3 "7-Bit Address and Slave Write with the AHEN and DHEN Bits"**.

The SCLREL bit will automatically clear after detecting the slave read address, irrespective of the state of the STREN bit.

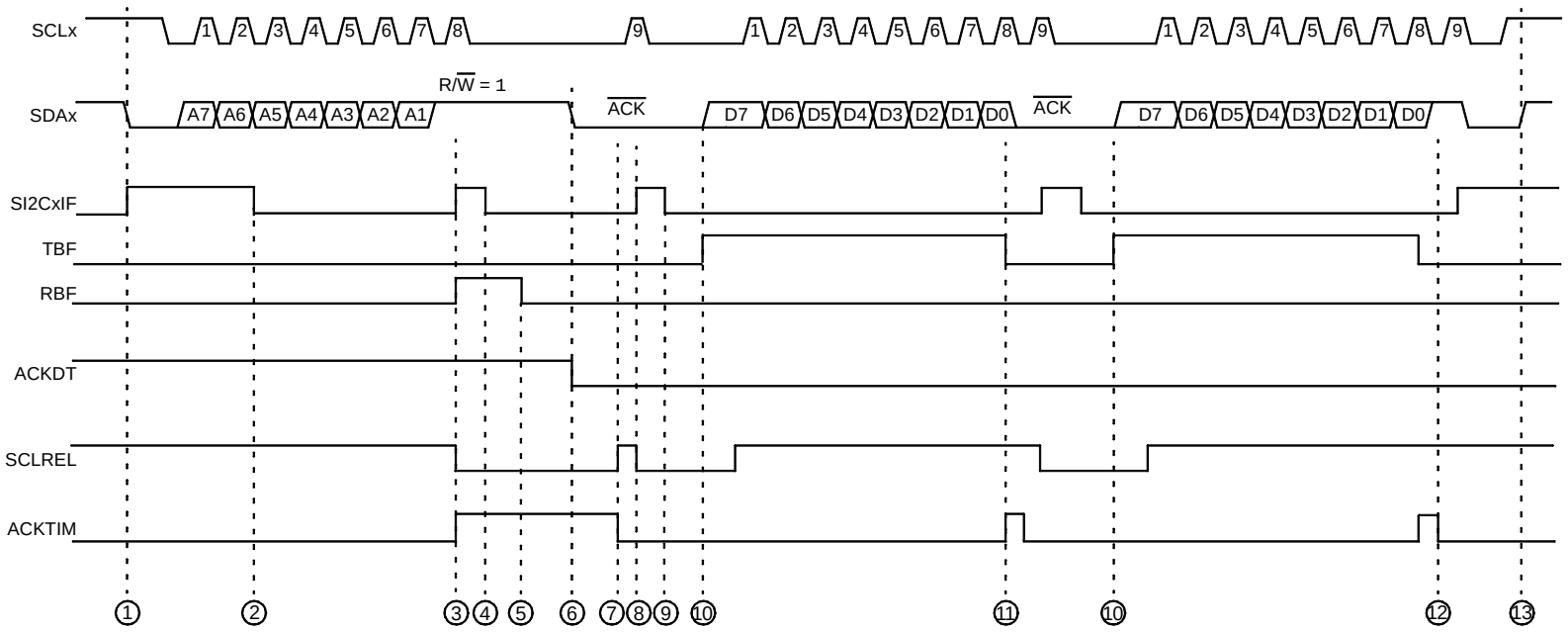
Figure 7-5: Slave Read 7-Bit Address Detection Timing Diagram (AHEN = 0)



Note 1: The SCIE bit may not be available on all devices. Refer to the specific device data sheet for availability.

© 2007-2015 Microchip Technology Inc.

Figure 7-6: I^2C Slave, 7-Bit Address, Transmission (AHEN = 1)



1. Detecting Start bit enables address detection, interrupt is set if the SCIE bit is set.
2. User software clears the interrupt flag.
3. Slave receives the address byte with R/$\overline{W}$ = 1. Hardware clears SCLREL to suspend master clock. ACKTIM and interrupt flag are asserted.
4. User software clears the interrupt flag.
5. Software reads the I2CxRV register, that clears the RBF flag.
6. ACKDT is written with $\overline{ACK}$.
7. User software sets SCLREL to release clock hold. Master clocks in the Acknowledgment sequence. ACKTIM is cleared by hardware.
8. Hardware clears SCLREL to suspend master clock if R/$\overline{W}$ = 1.
9. User software clears the interrupt flag.
10. User software loads the I2CxTRN register with response data. TBF = 1 indicates that the buffer is full.
11. After last bit, module clears TBF bit, indicating buffer is available for next byte.
12. At the end of ninth clock, if master sent NACK, no more data is expected. Module does not suspend the clock.
13. Module recognizes Stop event.

© 2007-2015 Microchip Technology Inc.

7.3.5 10-BIT ADDRESSING MODE

In 10-Bit Addressing mode, the slave must receive two device address bytes, as illustrated in Figure 7-7. The 5 Most Significant bits (MSbs) of the first address byte specify a 10-bit address. The R/$\overline{W}$ status bit of the address must specify a write, causing the slave device to receive the second address byte. For a 10-bit address, the first byte would equal, '11110 A9 A8 0', where A9 and A8 are the 2 MSbs of the address.

The I2CxMSK register can mask any bit position in a 10-bit address. The 2 MSbs of the I2CxMSK register are used to mask the MSbs of the incoming address received in the first byte. The remaining byte of the register is then used to mask the lower byte of the address received in the second byte.

Following the Start condition, the module shifts eight bits into the I2CxRSR register. The value of the I2CxRSR<2:1> bits is evaluated against the value of the I2CxADD<9:8> and I2CxMSK<9:8> bits, while the value of the I2CxRSR<7:3> bits is compared to '11110'. Address evaluation occurs on the falling edge of the eighth SCLx clock. For the address to be valid, the I2CxRSR<7:3> bits must be equal to '11110', while the I2CxRSR<2:1> bits must exactly match any unmasked bits in the I2CxADD<9:8> bits (if both bits are masked, a match is not needed). If the address is valid, the following events occur:

- An $\overline{ACK}$ is generated
- The D/$\overline{A}$ and R/$\overline{W}$ status bits are cleared
- The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock

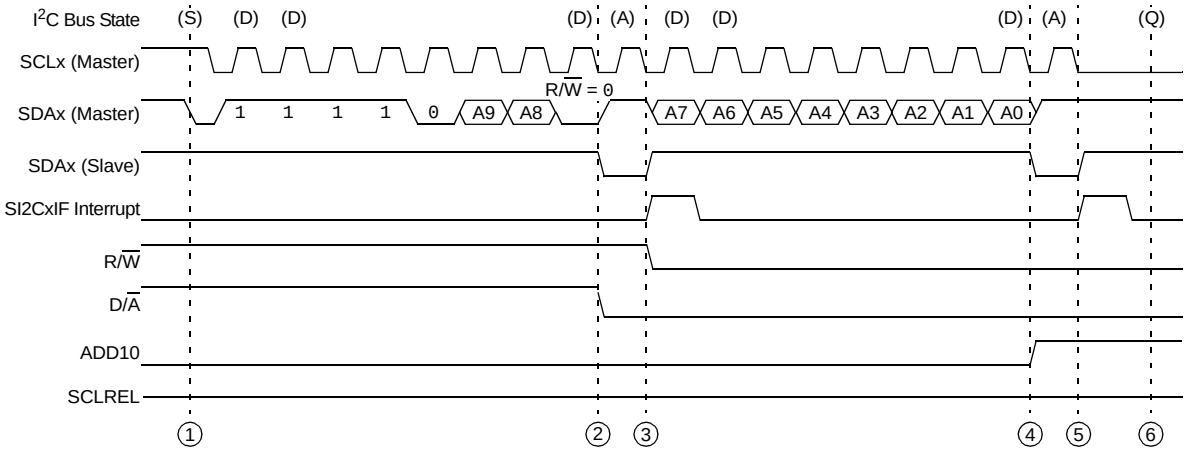
The module does generate an interrupt after the reception of the first byte of a 10-bit address; however, this interrupt is of little use.

The module will continue to receive the second byte into the I2CxRSR register. This time, the I2CxRSR<7:0> bits are evaluated against the I2CxADD<7:0> and I2CxMSK<7:0> bits. If the lower byte of the address is valid, as previously described, the following events occur:

- An $\overline{ACK}$ is generated
- The ADD10 status bit is set
- The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock
- The module will wait for the master to send data or initiate a Repeated Start condition

Note: Following a Repeated Start condition in 10-Bit Addressing mode, the slave only matches the first 7-bit address, '11110 A9 A8 0'.

Figure 7-7: 10-Bit Address Detection Timing Diagram (AHEN = 0)



(1) Detecting the Start bit enables address detection.

(2) Address match of first byte clears the D/$\overline{A}$ status bit and causes slave logic to generate an $\overline{ACK}$.

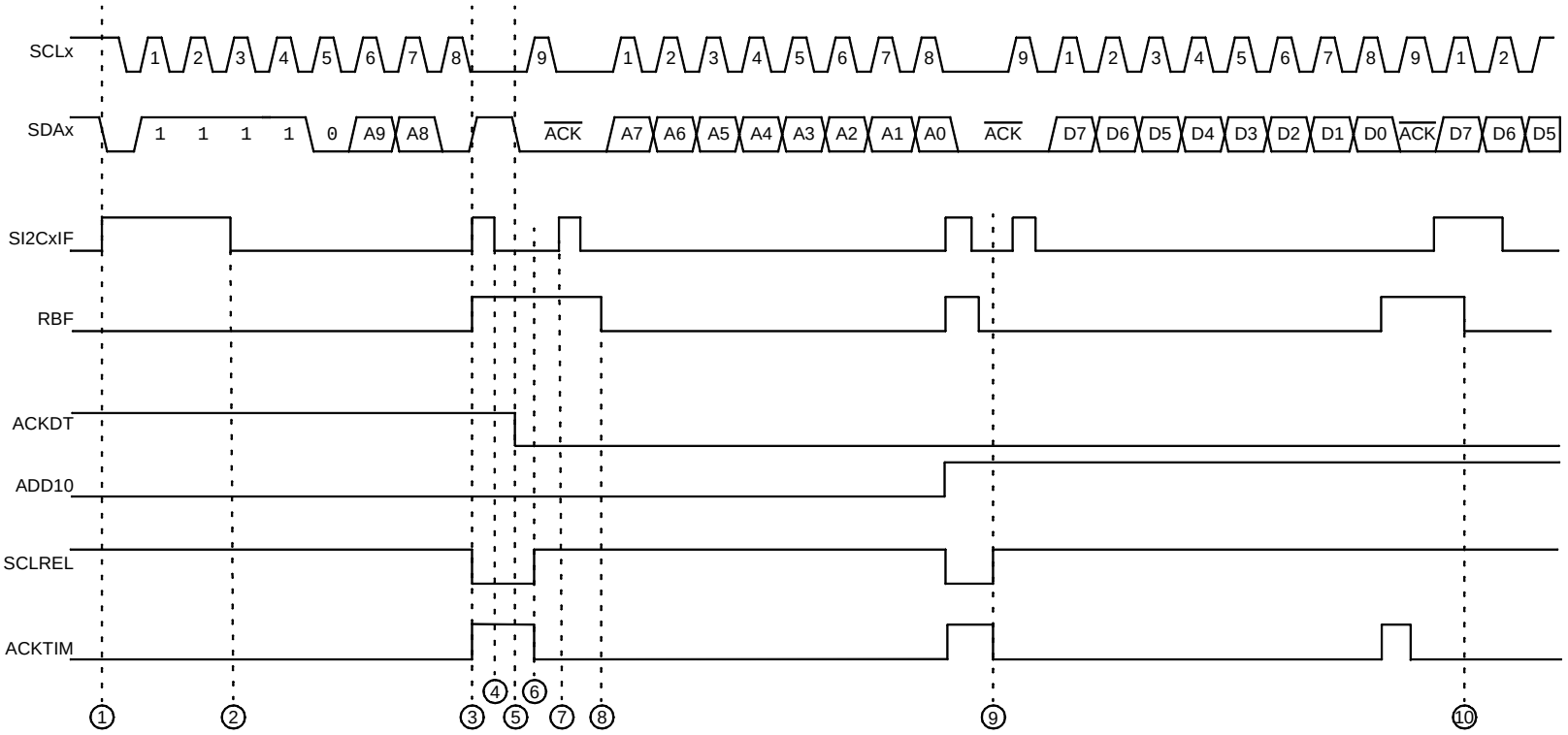
(3) Reception of first byte clears the R/$\overline{W}$ status bit. Slave logic generates an interrupt.

(4) Address match of first and second byte sets the ADD10 status bit and causes slave logic to generate an $\overline{ACK}$.

(5) Reception of second byte completes the 10-bit address. Slave logic generates an interrupt.

© 2007-2015 Microchip Technology Inc.

Figure 7-8: I^2C Slave, 10-Bit Address, Reception (STREN = 0, AHEN = 1, DHEN = 0)



1. Detecting the Start bit enables address detection; interrupt is set if the SCEN bit is set.
2. User software clears the interrupt flag.
3. Slave receives first address byte. Write indicated. Interrupt flag is asserted. ACKTIM is asserted. If AHEN = 1, slave suspends clock. SCLREL is cleared by hardware.
4. User software clears the interrupt flag.
5. ACKDT is written with $\overline{ACK}$ by user software.
6. User software sets SCLREL to release clock hold.
7. Slave interrupt is asserted.
8. User software reads I2CxRCV buffer, that clears RBF flag.
9. Slave Acknowledges the second address byte.
10. User software reads data from the I2CxRCV register.

© 2007-2015 Microchip Technology Inc.

7.3.6 SLAVE MODE BUS COLLISION

On a read request from the master, the slave begins shifting data out on the SDAx line. If a bus collision is detected and the SBCDE bit (I2CxCONH<2>) is set, then the I2CxBCIF bit will be set. After detecting the bus collision, the slave goes into Idle mode and waits to be addressed again. User software can use the I2CxBCIF bit or vectors to the bus collision interrupt to handle a slave bus collision.

Note:	The SBCDE and I2CxBCIF bits may not be available on all the devices. Refer to the specific device data sheet for availability.

7.3.7 GENERAL CALL OPERATION

The addressing procedure for the I^2C bus is such that the first byte after a Start condition usually determines which slave device the master is addressing. The exception is the general call address, which can address all devices. When this address is used, all the enabled devices respond with an Acknowledge. The general call address is one of the eight addresses reserved for specific purposes by the I^2C protocol. It consists of all '0's with R/$\overline{W}$ = 0. The general call is always a slave write operation.

The general call address is recognized when the General Call Enable bit, GCEN (I2CxCON<7> or I2CxCONL<7>), is set, as illustrated in Figure 7-9. Following a Start bit detect, 8 bits are shifted into the I2CxRSR register, and the address is compared against the I2CxADD register and the general call address.

If the general call address matches, the following events occur:

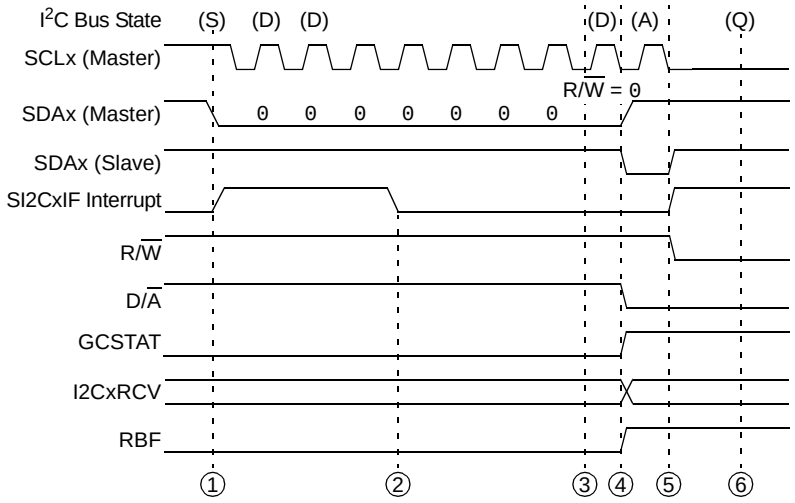
- An $\overline{ACK}$ is generated
- The slave will set the GCSTAT status bit (I2CxSTAT<9>)
- The D/$\overline{A}$ and R/$\overline{W}$ status bits are cleared
- The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock
- The I2CxRSR register is transferred to the I2CxRCV register and the RBF status bit (I2CxSTAT<1>) is set (during the eighth bit)
- The module waits for the master to send data

When the interrupt is serviced, the cause for the interrupt can be checked by reading the contents of the GCSTAT status bit to determine if the device address was device-specific or a general call address.

Note 1:	General call addresses are 7-bit addresses. If configuring the slave for 10-bit addresses and the A10M and GCEN bits are set, the slave will continue to detect the 7-bit general call address.
2:	The slave will Acknowledge the general call address (7-bit address, 0x00) only if GCEN is set, and independent of the STRICT and A10M bits.

© 2007-2015 Microchip Technology Inc.

Figure 7-9: General Call Address Detection Timing Diagram (GCEN = 1, AHEN = 0)



1. Detecting Start bit enables address detection. If SCIE is set, then interrupt will be asserted.(1)
2. User software clears the interrupt flag.(1)
3. All '0's and R/W = 0 indicates a general call.
4. Valid address clears D/A status bit and sets GCSTAT status bit. Slave generates ACK. Address is loaded into I2CxRCV register.
5. R/W status bit cleared. Slave generates an interrupt.
6. Bus waiting. Slave is ready to receive data.

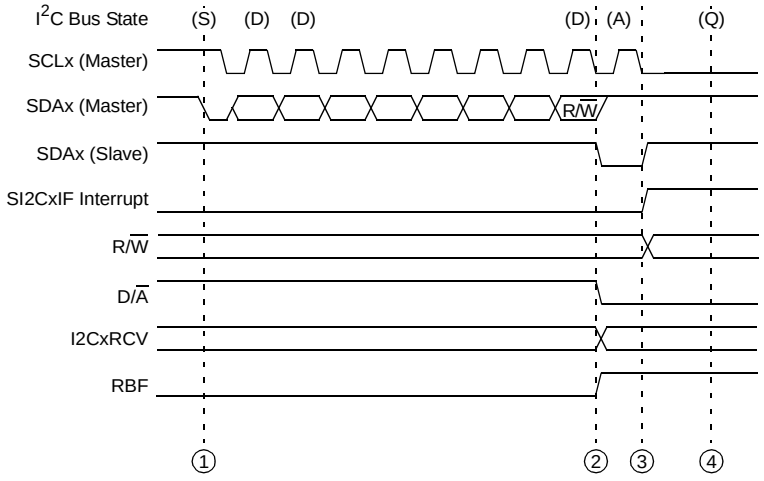
Note 1: The SCIE bit may not be available on all devices. Refer to the specific device data sheet for availability.

7.3.8 RECEIVING ALL ADDRESSES (IPMI OPERATION)

Some I^2C system protocols require a slave to act upon all messages on the bus. For example, the IPMI bus uses the I^2C nodes as message repeaters in a distributed network. To allow a node to repeat all messages, the slave must accept all messages, regardless of the device address.

To enable the IPMI mode, set the IPMIEN bit (I2CxCON<11>) as illustrated in Figure 7-10. Regardless of the state of the A10M and GCEN bits or the value loaded in the I2CxADD register, all addresses are accepted. This includes all valid 7-bit addresses, general call, Start byte, C-Bus, reserved and HS modes, and 10-bit address preambles.

Figure 7-10: IPMI Address Detection Timing Diagram (IPMIEN = 1)



1. Detecting Start bit enables address detection.
2. Regardless of contents, byte address is matched. Address match clears D/A status bit. Slave generates an ACK. Address is loaded into the I2CxRCV register.
3. R/W status bit is set/clear. Slave generates interrupt.
4. Bus is waiting.

Note: The user application must clear the IPMIEN bit (I2CxCON<11>) during an I^2C master operation and set this bit while acting as an IPMI slave.

7.3.9 STRICT SUPPORT

The slave module Acknowledges all the addresses, including the reserved addresses, when STRICT reserved addressing is not enforced (STRICT = 0). The slave device does not Acknowledge the reserved address space if the STRICT bit (I2CxCONL<11>) is set.

Table 7-1: Slave Response to Reserved Addresses

STRICT Bit	I2CxADD Slave Address	Received Address into I2CxRSR	Slave Acknowledge
x	0x1F	0x1F	$\overline{\text{ACK}}$
1	0x1F	C-Bus Address	NACK
1	C-Bus Address	C-Bus Address	NACK
0	C-Bus Address	C-Bus Address	$\overline{\text{ACK}}$
0	C-Bus Address	0x1F	NACK
0	0x1F	C-Bus Address	NACK

Note: When the STRICT bit is cleared, the $\overline{\text{ACK}}$ signal is generated only if the address is matched, even for reserved addresses. The slave device does not generate an $\overline{\text{ACK}}$ if there is an address mismatch, even if the address is a reserved address. Irrespective of the STRICT bit setting, and the address is reserved or not, an $\overline{\text{ACK}}$ signal is generated for a proper address match.

7.3.10 WHEN AN ADDRESS IS INVALID

If a 7-bit address does not match the contents of the I2CxADD<6:0> bits, the slave will return to an Idle state and ignore any activity on the I^2C bus until after the Stop condition.

If the first byte of a 10-bit address does not match the contents of the I2CxADD<9:8> bits, the slave will return to an Idle state and ignore all bus activity until after the Stop condition.

If the first byte of a 10-bit address matches the contents of the I2CxADD<9:8> bits, but the second byte of the 10-bit address does not match the I2CxADD<7:0> bits, the slave will return to an Idle state and ignore all bus activity until after the Stop condition.

© 2007-2015 Microchip Technology Inc.

7.3.11 ADDRESSES RESERVED FROM MASKING

Even when enabled, there are several addresses that are ignored by the I^2C module. For these addresses, an Acknowledge will not be issued independent of the mask setting. These addresses are listed in Table 7-2.

Table 7-2: Reserved I^2C Bus Addresses(3)

7-Bit Address Mode		
Slave Address	**R/$\overline{W}$ Bit**	**Description**
0000 000	0	General Call Address(1)
0000 000	1	Start Byte
0000 001	x	C-Bus Address
0000 010	x	Reserved
0000 011	x	Reserved
0000 1xx	x	HS Mode Master Code
1111 1xx	x	Reserved
1111 0xx	x	10-Bit Slave Upper Byte(2)

Note 1: Address will be Acknowledged only if GCEN = 1.

2: A match on this address can only occur as the upper byte in 10-Bit Addressing mode.

3: These addresses will not be Acknowledged, independent of mask settings and STRICT = 1.

7.4 Receiving Data from a Master Device

When the R/$\overline{W}$ status bit of the device address byte is '0' and an address match occurs, the R/$\overline{W}$ status bit (I2CxSTAT<2>) is cleared. The slave enters a state waiting for data to be sent by the master. After the device address byte, the contents of the data byte are defined by the system protocol and are only received by the slave.

The slave shifts 8 bits into the I2CxRSR register. On the falling edge of the eighth clock (SCLx), the following events occur:

- The module begins to generate an $\overline{ACK}$ or NACK.
- The RBF status bit (I2CxSTAT<1>) is set to indicate received data.
- The I2CxRSR register byte is transferred to the I2CxRCV register for access by the user software.
- The D/$\overline{A}$ status bit is set.
- A slave interrupt is generated. User software can check the status of the I2CxSTAT register to determine the cause of the event and then clear the SI2CxIF interrupt flag.
- The module waits for the next data byte.

7.4.1 ACKNOWLEDGE GENERATION

Normally, the slave Acknowledges all the received bytes by sending an $\overline{\text{ACK}}$ on the ninth SCLx clock. If the receive buffer is overrun, the slave does not generate this $\overline{\text{ACK}}$. The overrun is indicated if either (or both) of the following occur:

- The Receive Buffer Full bit, RBF (I2CxSTAT<1>), was set before the transfer was received
- The Receive Overflow bit, I2COV (I2CxSTAT<6>), was set before the transfer was received

Table 7-3 shows what happens when a data transfer byte is received, given the status of the RBF and I2COV status bits. If the RBF status bit is already set when the slave attempts to transfer to the I2CxRCV register, the transfer does not occur, but the interrupt is generated and the I2COV status bit is set. If both the RBF and I2COV status bits are set, the slave acts similarly. The shaded cells show the condition where user software did not properly clear the overflow condition.

Reading the I2CxRCV register clears the RBF status bit. The I2COV status bit is cleared by writing to a '0' through user software.

Note:	If the BOEN bit (I2CxCONH<4>) is set, then the I2COV bit (I2CxSTAT<6>) is ignored and only the RBF bit (I2CxSTAT<1>) determines whether the module will Acknowledge the message or not.

7.4.2 RECEIVE BUFFER OVERWRITE (I^2C SLAVE MODE ONLY)

If the BOEN bit (I2CxCONH<4>) is set, then the I2COV bit (I2CxSTAT<6>) is ignored. If the RBF bit (I2CxSTAT<1>) is not set, then the $\overline{\text{ACK}}$ is generated for the receive address or data; the I2CxRCV buffer is updated with I2CxRSR.

Table 7-3: Data Transfer Received Byte Actions

Status Bits as Data Byte Received			Transfer I2CxRSR to I2CxRCV	Generate $\overline{\text{ACK}}$	Generate SI2CxIF Interrupt (interrupt occurs if enabled)	Set RBF	Set I2COV
BOEN(1)	RBF	I2COV					
x	0	0	Yes	Yes	Yes	Yes	No change
x	1	0	No	No	Yes	No change	Yes
x	1	1	No	No	Yes	No change	Yes
0	0	1	Yes	No	Yes	Yes	No change
1	0	1	Yes	Yes	Yes	Yes	No change

Legend: Shaded cells show states where the user software did not properly clear the overflow condition.

Note 1: BOEN is not available on all devices. Refer to the specific device data sheet for availability.

7.4.3 WAIT STATES DURING SLAVE RECEPTIONS

When the slave receives a data byte, the master can potentially begin sending the next byte immediately. This allows the user software controlling the nine slave SCLx clock periods to process the previously received byte. If this is not enough time, the slave software may want to generate a bus Wait period.

The STREN bit (I2CxCON<6> or I2CxCONL<6>) enables a bus Wait to occur on slave receptions. When STREN = 1 at the falling edge of the ninth SCLx clock of a received byte, the slave clears the SCLREL bit. Clearing the SCLREL bit causes the slave to pull the SCLx line low, initiating a Wait. The SCLx clock of the master and slave will synchronize, as provided in **Section 6.2 "Master Clock Synchronization"**.

When the user software is ready to resume reception, the user software sets the SCLREL bit. This causes the slave to release the SCLx line and the master resumes clocking.

© 2007-2015 Microchip Technology Inc.

7.4.4 EXAMPLE MESSAGES OF SLAVE RECEPTION

Receiving a slave message is an automatic process. The user software handling the slave protocol uses the slave interrupt to synchronize to the events.

When the slave detects the valid address, the associated interrupt will notify the user software to expect a message. On receive data, as each data byte transfers to the I2CxRCV register, an interrupt notifies the user software to unload the buffer.

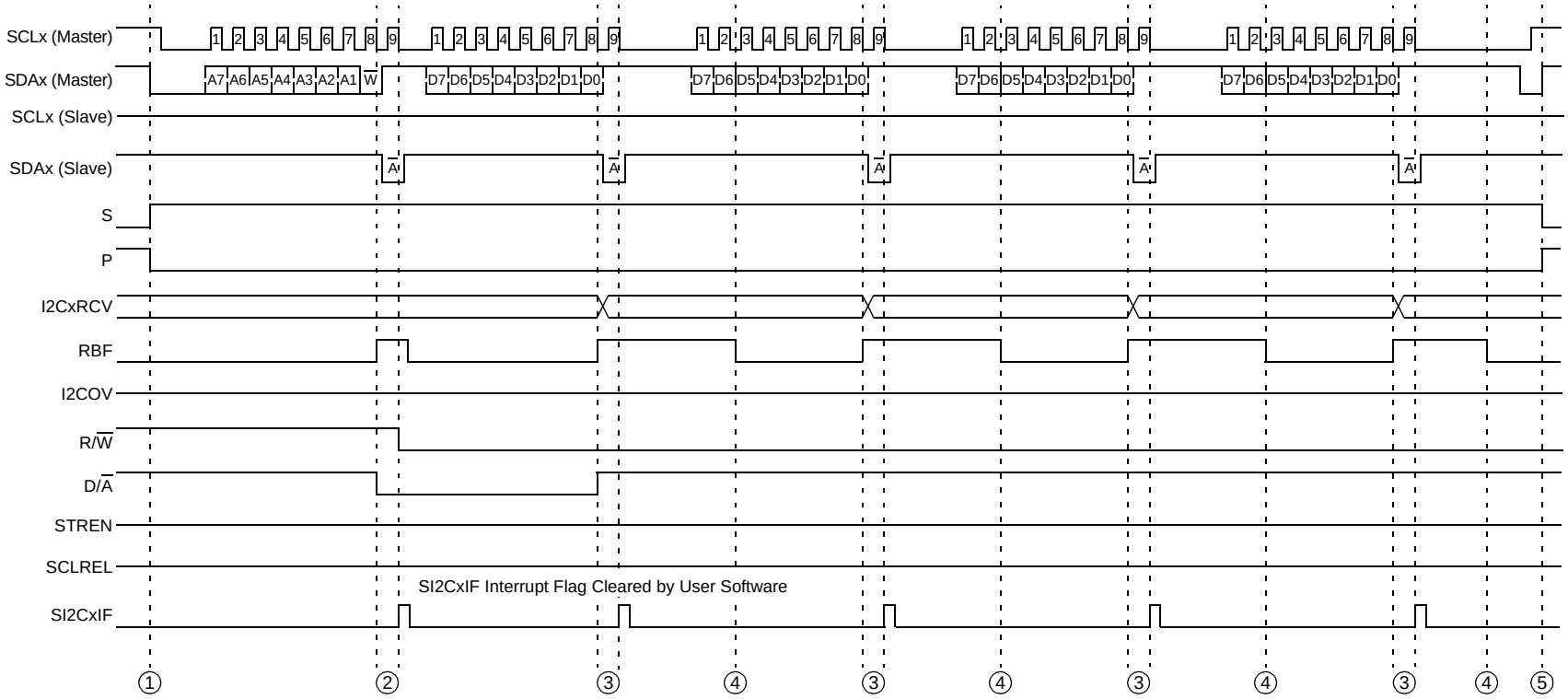
Figure 7-11 illustrates a simple receive message. Because it is a 7-bit address message, only one interrupt occurs for the address bytes. Then, interrupts occur for each of four data bytes. At an interrupt, the user software may monitor the status bits, RBF (I2CxSTAT<1>), D/$\overline{A}$ (I2CxSTAT<5>) and R/$\overline{W}$ (IC2xSTAT<2>), to determine the condition of the byte received.

Figure 7-12 illustrates a similar message using a 10-bit address. In this case, two bytes are required for the address.

Figure 7-13 illustrates a case where the user software does not respond to the received byte and the buffer overruns. On reception of the second byte, the module will automatically NACK the master transmission. Generally, this causes the master to resend the previous byte. The I2COV status bit (I2CxSTAT<6>) indicates that the buffer has overrun. The I2CxRCV register buffer retains the contents of the first byte. On reception of the third byte, the buffer is still full, and again, the module will NACK the master. After this, the user software finally reads the buffer. Reading the buffer will clear the RBF status bit; however, the I2COV status bit remains set. The user software must clear the I2COV status bit (I2CxSTAT<6>). The next received byte is moved to the I2CxRCV register buffer and the module responds with an $\overline{ACK}$.

Figure 7-14 highlights clock stretching while receiving data. In the previous examples, the STREN bit (I2CxCON<6> or I2CxCONL<6>) is equal to '0', which disables clock stretching on receive messages. In this example, the user software sets STREN to enable clock stretching. When STREN = 1, the module will automatically clock stretch after each received data byte, allowing the user software more time to move the data from the buffer. If RBF = 1 at the falling edge of the ninth clock, the module automatically clears the SCLREL bit (I2CxCON<12> or I2CxCONL<12>) and pulls the SCLx bus line low. As shown with the second received data byte, if the user software can read the buffer and clear the RBF status bit before the falling edge of the ninth clock, the clock stretching will not occur. The user software can also suspend the bus at any time. By clearing the SCLREL bit, the module pulls the SCLx line low after it detects the bus SCLx low. The SCLx line remains low, suspending transactions on the bus until the SCLREL bit is set.

Figure 7-11: Slave Message (Write Data to Slave: 7-Bit Address; Address Matches; A10M = 0, GCEN = 0, IPMIEN = 0, AHEN = 0, DHEN = 0, STRICT = 0 and BOEN = 0)



① Slave recognizes Start event; S and P status bits set/clear accordingly.

② Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt. Address byte is moved to the I2CxRCV register and must be read by user software to prevent buffer overflow.

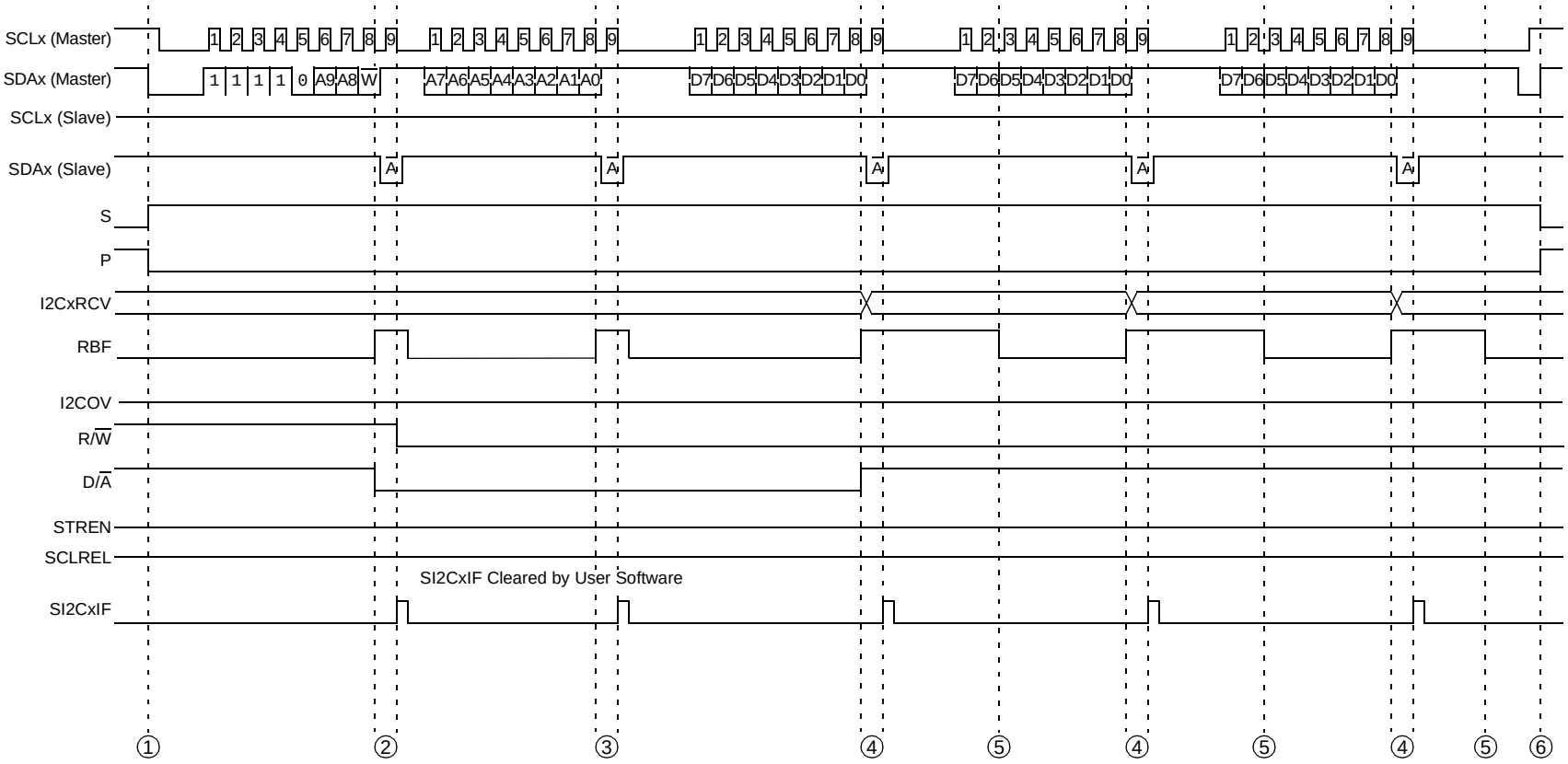
③ Next received byte is message data. The byte moved to the I2CxRCV register sets the RBF status bit. Slave generates interrupt. Slave Acknowledges reception.

④ User software reads the I2CxRCV register. RBF status bit clears.

⑤ Slave recognizes Stop event; S and P status bits set/clear accordingly.

© 2007-2015 Microchip Technology Inc.

Figure 7-12: Slave Message (Write Data to Slave: 10-Bit Address; Address Matches; A10M = 1, GCEN = 0, IPMIEN = 0, AHEN = 0, DHEN = 0, STRICT = 0 and BOEN = 0)



① Slave recognizes Start event, S and P bits set/clear accordingly.

② Slave receives address byte. High-order address matches. Slave Acknowledges and generates interrupt. Address byte is moved to the I2CxRCV register and is read by user software to prevent buffer overflow.

③ Slave receives address byte. Low-order address matches. Slave Acknowledges and generates interrupt. Address byte is moved to the I2CxRCV register and is read by user software to prevent buffer overflow.

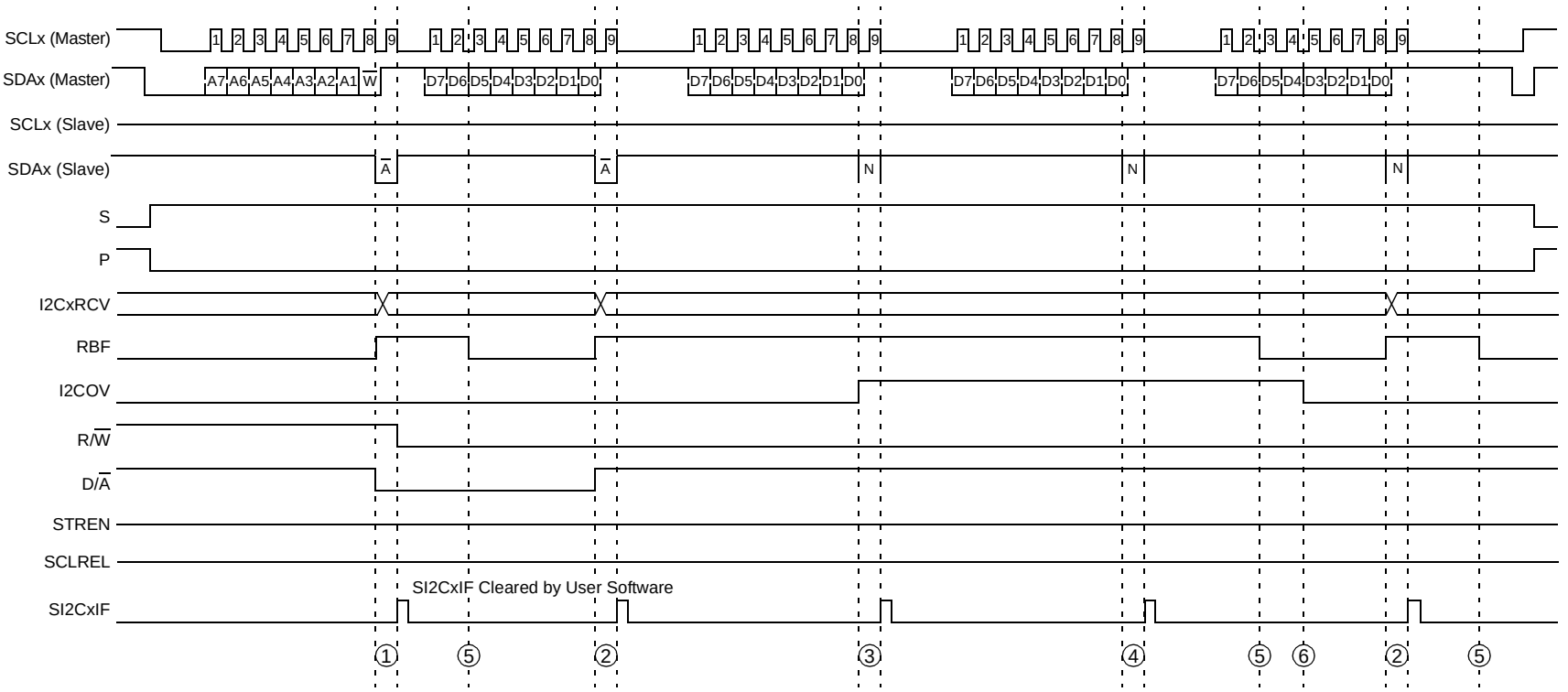
④ Next received byte is message data. Byte moved to the I2CxRCV register, sets RBF. Slave Acknowledges and generates interrupt.

⑤ User software reads the I2CxRCV register. RBF bit clears.

⑥ Slave recognizes Stop event, S and P bits set/clear accordingly.

© 2007-2015 Microchip Technology Inc.

Figure 7-13: Slave Message (Write Data to Slave: 7-Bit Address; Buffer Overrun; A10M = 0, GCEN = 0, IPMIEN = 0, AHEN = 0, DHEN = 0, STRICT = 0 and BOEN = 0)



① Slave receives address byte. Address matches. Slave generates interrupt. Address byte is moved to the I2CxRCV register and must be read by user software to prevent buffer overflow.

② Next received byte is message data. The byte is moved to the I2CxRCV register, sets RBF. Slave generates interrupt. Slave Acknowledges reception.

③ Next byte received before I2CxRCV is read by software. I2CxRCV register is unchanged. I2COV overflow bit is set. Slave generates interrupt. Slave sends NACK for reception.

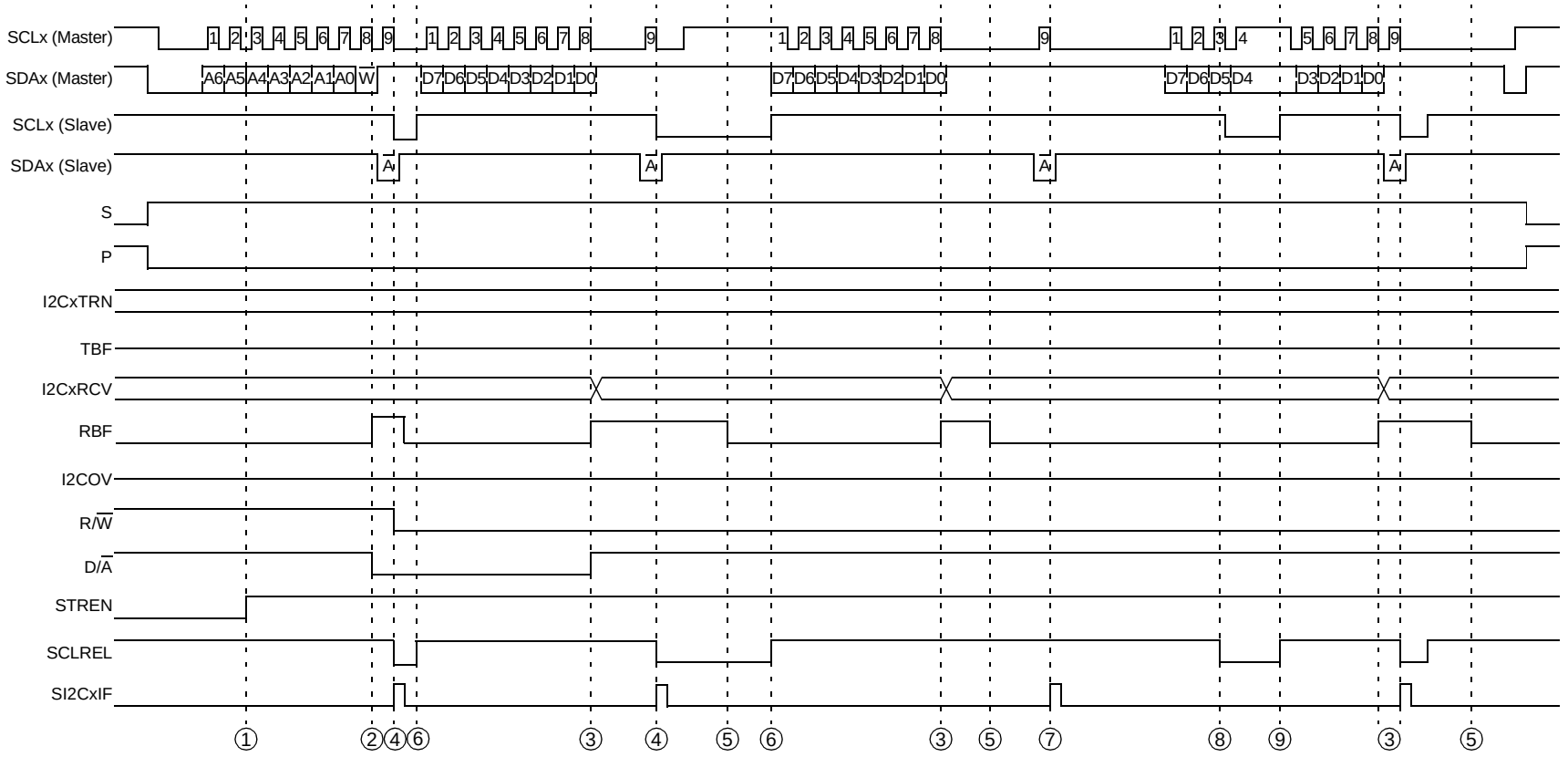
④ Next byte also received before I2CxRCV is read by software. I2CxRCV register is unchanged. Slave generates interrupt. Slave sends NACK for reception. The master state machine should not be programmed to send another byte after receiving a NACK in this manner. Instead, it should abort the transmission with a Stop condition or send a Repeated Start condition and attempt to retransmit the data.

⑤ User software reads the I2CxRCV register. RBF bit clears.

⑥ User software clears the I2COV bit. Reception will still not be able to proceed normally until the module sees a Stop/Repeated Start bit. If neither of these conditions is met, an additional transmission will be received correctly, but sends a NACK and sets the I2COV bit again.

© 2007-2015 Microchip Technology Inc.

Figure 7-14: Slave Message (Write Data to Slave: 7-Bit Address; Clock Stretching Enabled; A10M = 0, GCEN = 0, IPMIEN = 0, AHEN = 0, DHEN = 0, STRICT = 0 and BOEN = 0)



① User software sets the STREN bit to enable clock stretching.

② Slave receives address byte. I2CxRCV register is read by user software to prevent buffer overflow.

③ Next received byte is message data. The byte moves to the I2CxRCV register, sets RBF.

④ Because RBF = 1 at ninth clock, automatic clock stretch begins. Slave clears SCLREL bit. Slave pulls SCLx line low to stretch clock.

⑤ User software reads I2CxRCV register. RBF bit clears.

⑥ User software sets SCLREL bit to release clock.

⑦ Slave does not clear SCLREL because RBF = 0 at this time.

⑧ User software may clear SCLREL to cause a clock hold. Module must detect SCLx low before asserting SCLx low.

⑨ User software may set SCLREL to release a clock hold.

© 2007-2015 Microchip Technology Inc.

7.5 Sending Data to a Master Device

When the R/$\overline{W}$ status bit of the incoming device address byte is '1' and an address match occurs, the R/$\overline{W}$ status bit (I2CxSTAT<2>) is set. At this point, the master device is expecting the slave to respond by sending a byte of data. The contents of the byte are defined by the system protocol and are only transmitted by the slave.

Note:	When the IPMIEN bit (I2CxCON<11>) is equal to '1' (IPMI mode), the I^2C module assumes that the R/$\overline{W}$ bit is '0'. Therefore, the slave transmission function is disabled. If the R/$\overline{W}$ bit is '1', the I^2C module will trigger an interrupt. This interrupt should be ignored (that is, the I^2C interrupt flags should be cleared) and the I^2C slave transmission event should be aborted.

When the interrupt from the address detection occurs, the user software can write a byte to the I2CxTRN register to start the data transmission.

The slave sets the TBF status bit (I2CxSTAT<0>). The eight data bits are shifted out on the falling edge of the SCLx input. This ensures that the SDAx signal is valid during the SCLx high time. When all 8 bits have been shifted out, the TBF status bit is cleared.

The slave detects the Acknowledge from the master-receiver on the rising edge of the ninth SCLx clock.

If the SDAx line is low, indicating an $\overline{ACK}$, the master is expecting more data and the message is not complete. The module generates a slave interrupt and the ACKSTAT status bit (I2CxSTAT<15>) can be inspected to determine whether more data is being requested.

A slave interrupt is generated on the falling edge of the ninth SCLx clock. User software must check the status of the I2CxSTAT register and clear the SI2CxIF interrupt flag.

If the SDAx line is high, indicating a NACK, the data transfer is complete. User software should not write further data to the I2CxTRN register. The slave resets and generates an interrupt, and it waits for detection of the next Start bit.

7.5.1 WAIT STATES DURING SLAVE TRANSMISSIONS

During a slave transmission message, the master expects return data immediately after detection of the valid address with R/$\overline{W}$ = 1. Because of this, the slave automatically generates a bus Wait whenever the slave returns data.

The automatic Wait occurs at the falling edge of the ninth SCLx clock of a valid device address byte, or transmitted byte, Acknowledged by the master, indicating expectation of more transmit data.

The slave clears the SCLREL bit (I2CxCON<12> or I2CxCONL<12>). Clearing the SCLREL bit causes the slave to pull the SCLx line low, initiating a Wait. The SCLx clock of the master and slave will synchronize, as shown in **Section 6.2 "Master Clock Synchronization"**.

When the user software loads the I2CxTRN register and is ready to resume transmission, the user software sets the SCLREL bit. This causes the slave to release the SCLx line and the master resumes clocking.

Note:	The user software must provide a delay between writing to the transmit buffer and setting the SCLREL bit. This delay must be greater than the minimum setup time for slave transmissions, as specified in the **"Electrical Characteristics"** chapter of the specific device data sheet.

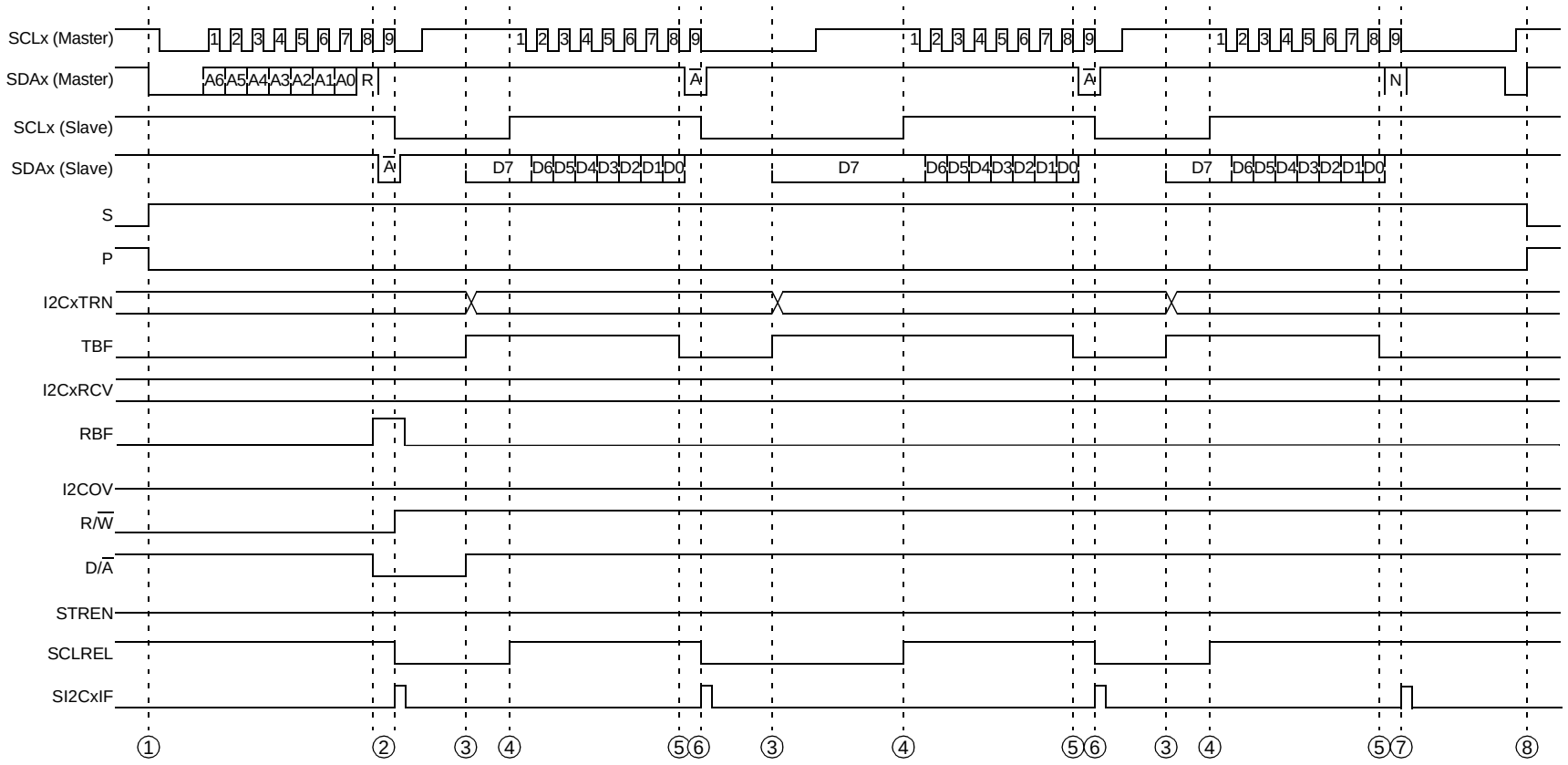
© 2007-2015 Microchip Technology Inc.

7.5.2 EXAMPLE MESSAGES OF SLAVE TRANSMISSION

The slave transmissions for 7-bit address messages are illustrated in Figure 7-15. When the address matches and the R/$\overline{W}$ status bit of the address indicates a slave transmission, the module automatically initiates clock stretching by clearing the SCLREL bit and generates an interrupt to indicate a response byte is required. The user software writes the response byte into the I2CxTRN register. As the transmission completes, the master responds with an $\overline{ACK}$. If the master replies with an $\overline{ACK}$, the master expects more data, and the module again clears the SCLREL bit and generates another interrupt. If the master responds with a NACK, no more data is required and the module will not stretch the clock, but will generate an interrupt.

The slave transmissions for 10-bit address messages require the slave to first recognize a 10-bit address. Because the master must send two bytes for the address, the R/$\overline{W}$ status bit in the first byte of the address specifies a write. To change the message to a read, the master sends a Repeated Start and repeats the first byte of the address with the R/$\overline{W}$ status bit, specifying a read. At this point, the slave transmission begins as illustrated in Figure 7-16.

Figure 7-15: Slave Message (Read Data from Slave: 7-Bit Address)



① Slave recognizes Start event, S and P bits set/clear accordingly.

② Slave receives address byte. Address matches. Slave generates interrupt. Address byte is moved to I2CxRCV register and is read by user software to prevent buffer overflow. R/$\overline{W}$ = 1 to indicate read from slave. SCLREL = 0 to suspend master clock.

③ User software writes I2CxTRN with response data. TBF = 1 indicates that buffer is full. Writing I2CxTRN sets D/$\overline{A}$, indicating a data byte.

④ User software sets SCLREL to release clock hold. Master resumes clocking and slave transmits data byte.

⑤ After last bit, module clears TBF bit, indicating buffer is available for next byte.

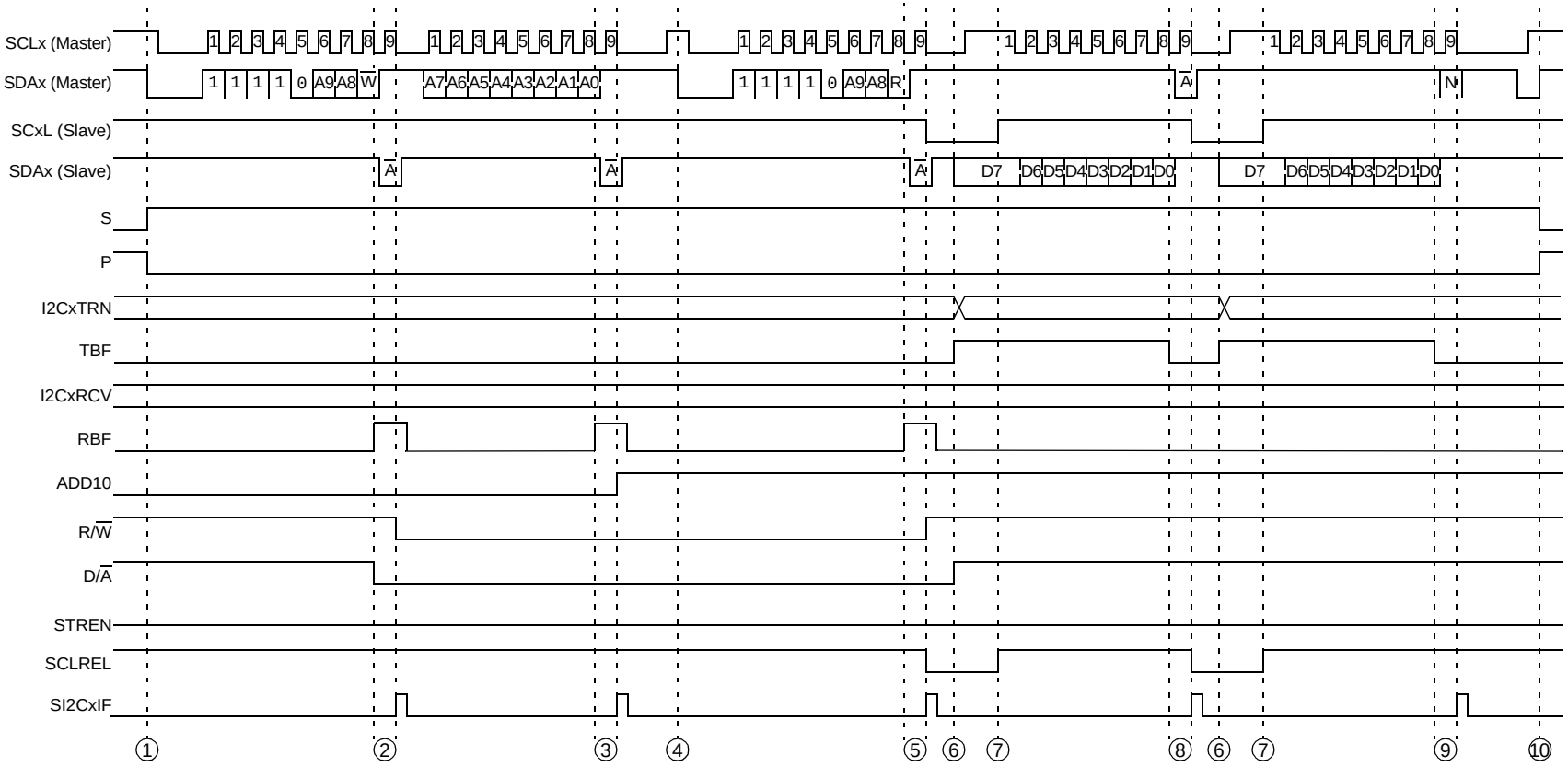
⑥ At the end of ninth clock, if the master has sent an $\overline{ACK}$, module clears SCLREL to suspend clock. Slave generates interrupt.

⑦ At the end of ninth clock, if master sent a NACK, no more data expected. User software should stop writing to I2CxTRN. Module does not suspend clock and will generate an interrupt.

⑧ Slave recognizes Stop event; S and P bits set/clear accordingly.

© 2007-2015 Microchip Technology Inc.

Figure 7-16: Slave Message (Read Data from Slave: 10-Bit Address)



① Slave recognizes Start event; S and P bits set/clear accordingly.

② Slave receives first address byte. Write indicated. Slave Acknowledges and generates interrupt. User software reads I2CxRCV register.

③ Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt. User software reads I2CxRCV register.

④ Master sends a Repeated Start to redirect the message.

⑤ Slave receives resend of first address byte. User software reads I2CxRCV register. Read indicated. Slave suspends clock.

⑥ User software writes I2CxTRN with response data.

⑦ User software sets SCLREL to release clock hold. Master resumes clocking and slave transmits data byte.

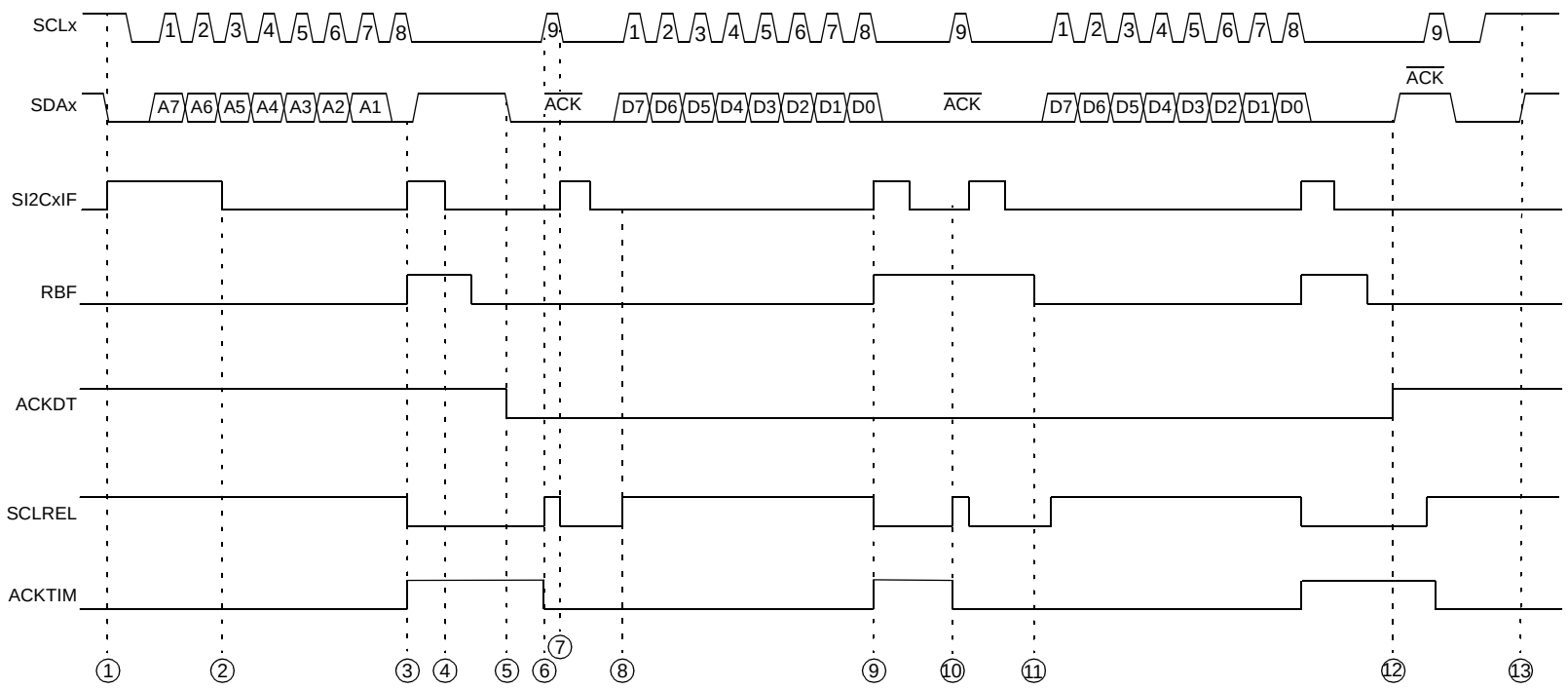
⑧ At the end of ninth clock, if master sent an $\overline{ACK}$, module clears SCLREL to suspend clock. Slave generates interrupt.

⑨ At the end of ninth clock, if master sent a NACK, no more data expected. User software should stop writing to I2CxTRN. Module does not suspend clock and will generate an interrupt.

⑩ Slave recognizes Stop event; S and P bits set/clear accordingly.

© 2007-2015 Microchip Technology Inc.

Figure 7-17: I²C Slave, 7-Bit Address, Reception (STREN = 1, AHEN = 1, DHEN = 1)



① Detecting Start bit, enables address detection, interrupt is set if SCEN is set.

② User software clears the interrupt flag.

③ Slave receives first address byte. Write Indicated. If AHEN = 1, SCLREL is cleared by hardware. ACKTIM and interrupt are asserted.

④ User software clears the interrupt flag.

⑤ ACKDT is written with an $\overline{ACK}$ by user software.

⑥ User software sets SCLREL to release the clock hold; ACKTIM is cleared by hardware.

⑦ Hardware stretches the clock after $\overline{ACK}$ (if STREN = 1).

⑧ User software sets SCLREL to release clock hold.

⑨ I2CxRCV is loaded with I2CxRSR. RBF is set. If DHEN = 1, SCLREL is cleared by hardware and ACKTIM is asserted.

⑩ User software sets SCLREL to release the clock hold. ACKTIM is cleared by hardware. After Acknowledgment, hardware stretches the clock.

⑪ User software reads I2CxRCV, that clears the RBF flag.

⑫ NACK sent by master.

⑬ Module recognizes the Stop event.

© 2007-2015 Microchip Technology Inc.

8.0 CONNECTION CONSIDERATIONS FOR I^2C BUS

Because the I^2C bus is a wired-AND bus connection, Pull-up Resistors (R_P) on the bus are required, as illustrated in Figure 8-1. The Series Resistors (R_S) are optional and are used to improve the Electrostatic Discharge (ESD) susceptibility.

The values of the resistors, R_P and R_S, depend on the following parameters:

- Supply voltage
- Bus capacitance
- Number of connected devices (input current + leakage current)
- Input level selection (I^2C or System Management Bus (SMBus))

Because the device must be able to pull the bus low against R_P, current drawn by R_P must be greater than the I/O pin minimum sink current, I_{OL} at V_{OLMAX}**,** for the device output stage. Equation 8-1 shows the formula for computing the minimum pull-up resistance.

Equation 8-1: Minimum Pull-up Resistance

$$R_{PMIN} = \frac{(V_{DDMAX} - V_{OLMAX})}{I_{OL}}$$

In a 400 kHz system, a minimum rise time specification of 300 ns exists; in a 100 kHz system, the specification is 1000 ns. Because R_P must pull the bus up against the total Capacitance, C_B, with a maximum rise time of 300 ns to (V_{DD} – 0.7V), the Maximum Resistance for the Pull-up (R_{PMAX}) is computed using the formula as shown in Equation 8-2.

Equation 8-2: Maximum Pull-up Resistance

$$\frac{-tR}{C_B * [ln(1 - (V_{DDMAX} - V_{ILMAX}))]}$$

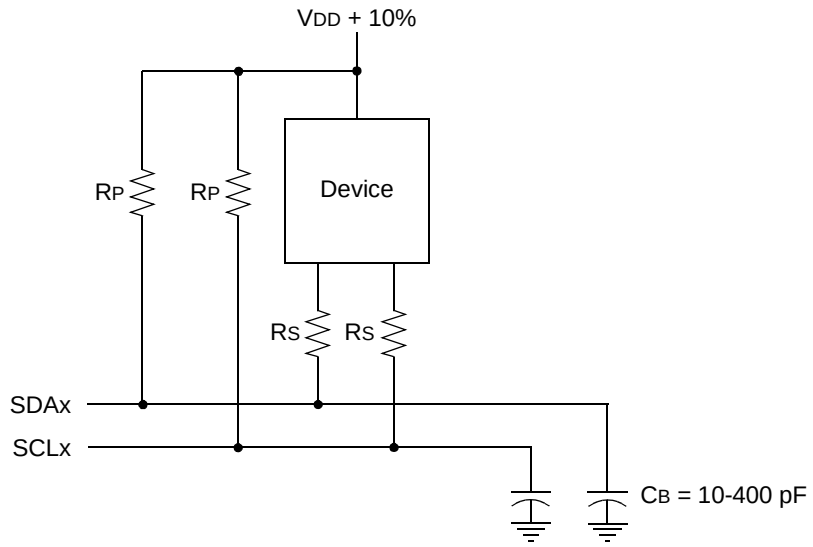
The maximum value for R_S is determined by the desired noise margin for the low level. R_S cannot drop enough voltage to make the device V_{OL} and the voltage across R_S more than the maximum V_{IL}. Equation 8-3 shows the formula for computing the maximum value for R_S.

Equation 8-3: Maximum Series Resistance

$$R_{SMAX} = \frac{(V_{ILMAX} - V_{OLMAX})}{I_{OLMAX}}$$

Note:	The SCLx clock input must have a minimum high and low time for proper operation. Refer to the **"Electrical Characteristics"** chapter in the specific device data sheet for more information on the high and low times of the I^2C bus specification, and requirements of the I^2C module and I/O pins.

Figure 8-1: Sample Device Configuration for I²C Bus



Note: I²C devices with input levels related to VDD must have one common supply line to which the pull-up resistor is also connected.

8.1 Integrated Signal Conditioning

The SCLx and SDAx pins have an input glitch filter. The I²C bus requires this filter in both the 100 kHz and 400 kHz systems.

When operating on a 400 kHz bus, the I²C bus specification requires a slew rate control of the device pin output. This slew rate control is integrated into the device. If the DISSLW bit (I2CxCON<9> or I2CxCONL<9>) is cleared, the slew rate control is active. For other bus speeds, the I²C bus specification does not require slew rate control and the DISSLW bit should be set.

Some system implementations of I²C buses require different input levels for VILMAX and VIHMIN. In a normal I²C system, VILMAX is 0.3 VDD; VIHMIN is 0.7 VDD. By contrast, in a SMBus system, VILMAX is set at 0.8V, while VIHMIN is set at 2.1V.

The SMEN bit (I2CxCON<8> or I2CxCONL<8>) controls the input levels. Setting SMEN (= 1) changes the input levels to SMBus specifications.

© 2007-2015 Microchip Technology Inc.

9.0 OPERATION IN POWER-SAVING MODES

9.1 Sleep Mode in Slave Mode

The I^2C module can wake-up from Sleep mode on detecting a valid slave address match. Because all bit shifting is done with reference to the external SCLx signal, generated by an I^2C master, transmissions and receptions can continue even while in Sleep mode.

Note:	As per the slave I^2C behavior, a slave interrupt is generated only on an address match. Therefore, when an I^2C slave is in Sleep mode and it receives a message from the master, the clock required to match the received address is derived from the master. Only on an address match, will the interrupt be generated and the device can wake-up from Sleep, provided the interrupt has been enabled and an ISR has been defined.

9.2 Sleep Mode in Master Mode

If Sleep occurs in the middle of a master transmission, and the state machine is partially into a transmission as the clocks stop, the behavior of the module will be undefined. Similarly, if Sleep occurs in the middle of a master reception, the module behavior will also be undefined. The transmitter and receiver will stop at Sleep when in Master mode. Register contents are not affected by going into Sleep mode or coming out of Sleep mode; there is no automatic way to prevent Sleep entry if a transmission or reception is pending. The user software must synchronize the Sleep entry with I^2C operation to avoid undefined module behavior.

9.3 When the Device Enters Idle Mode

When the device executes a `PWRSAV 1` instruction, the device enters Idle mode. The module enters a power-saving state in Idle mode, depending on the I2CSIDL bit (I2CxCON<13> or I2CxCONL<13>). If I2CSIDL = 1, the module enters a power-saving mode, similar to actions while entering Sleep mode. If I2CSIDL = 0, the module does not enter a power-saving mode and continues to operate normally.

10.0 PERIPHERAL MODULE DISABLE (PMDx) REGISTERS

The Peripheral Module Disable (PMDx) registers provide a method to disable the I^2C modules by stopping all clock sources supplied to that module. When a peripheral is disabled through the appropriate PMDx control bit, the peripheral is in a minimum power consumption state. The control and status registers associated with the peripheral are also disabled, so writes to those registers will have no effect and read values will be invalid. A peripheral module is only enabled if the I2CxMD bit in the PMDx register is cleared.

11.0 EFFECTS OF A RESET

A Reset disables the I^2C module and terminates any active or pending message activity. Refer to the I2Cx Control (I2CxCON or I2CxCONH and I2CxCONL) and I2Cx Status (I2CxSTAT) registers' definitions (Register 3-1, Register 3-2, Register 3-3 and Register 3-4) for the Reset conditions of those registers.

Note 1:	The I2CxCON or I2CxCONL and I2CxCONH are not available on all devices. Refer to the specific device data sheet for availability.
2:	In this discussion, 'Idle' refers to the CPU power-saving state. The lower case 'idle' refers to the time when the I^2C module is not transferring data on the bus.

12.0 CONSTANT-CURRENT SOURCE

The constant-current source module is a precision current generator and is used in conjunction with an Analog-to-Digital Converter (ADC) to measure the resistance of external resistors connected to device pins. A typical use of this peripheral is to set the I^2C board address using resistors, as explained in the following sections.

12.1 Features Overview

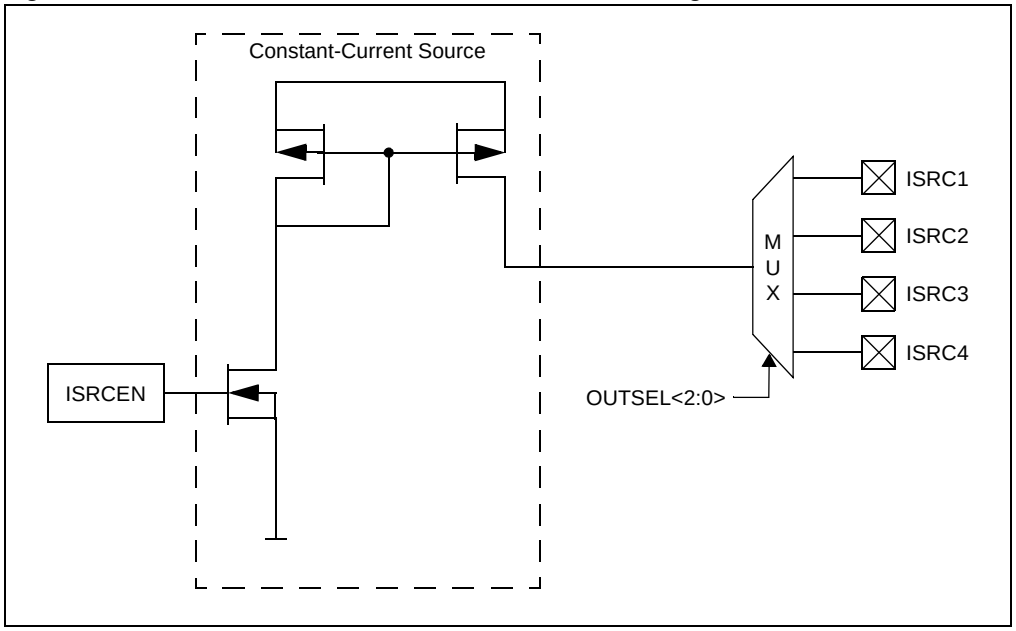
The constant-current source module offers the following major features:

- Constant-current generator (10 µA nominal)
- Internal selectable connection to one out of four pins
- Enable/disable bit

12.2 Module Description

Figure 12-1 shows a functional block diagram of the constant-current source module. It consists of a precision current generator with a nominal value of 10 µA. The module can be enabled and disabled using the ISRCEN bit (ISRCCON<15>). The output of the current generator is internally connected to one out of four pins. The OUTSEL<2:0> bits (ISRCCON<10:8>) allow selection of the target pin. The current source is calibrated during testing. The calibration value is stored into the Flash die serialization area, and should be read and transferred to the ISRCCON register by the application firmware before using the peripheral.

Figure 12-1: Constant-Current Source Module Block Diagram



Note: The constant-current source module is not available on all devices. Refer to the specific device data sheet for availability.

© 2007-2015 Microchip Technology Inc.

12.3 Module Applications

An end application would enable the user to determine the resistance of a couple of external resistors that encode in their resistance an address value that the application can use when communicating over a communications network. A typical application example is PMBus™.

The user application enables the current source and selects a pin where the current is to be directed. A resistor is externally connected from this pin to ground. The current flow through the external resistor generates a voltage. ADC channels, AN4 through AN7, can be used to read the generated voltage. The calculated resistance value of the external resistor can be translated into a number of bits. Allowing for the current source accuracy, the accuracy of the external resistors, and the accuracy of the ADC, 10 to 12 unique resistance values can be read for each resistor. Therefore, two external resistors in the user's application circuit can encode 100 to 144 unique values for use as module identification addresses, which allows two device pins to replace device pins that are binary encoded. This results in a large pin count savings, especially in a low pin count package.

The current source output is nominally 10 µA, and the maximum output voltage that can be reached while still maintaining current regulation is AV_{DD} – 0.5V. Table 12-1 shows some example resistor values that can be used to encode an address into a resistor. The table uses common 5% tolerance resistor values, and the chosen values are separated by two standard resistor values, which is approximately a 30% value separation.

Table 12-1: Example PMBus™ Address Encoding

Resistor	Voltage	Address
13k	130 mV	1
18k	180 mV	2
24k	240 mV	3
33k	330 mV	4
43k	430 mV	5
56k	560 mV	6
75k	750 mV	7
100k	1V	8
130k	1.3V	9
180k	1.8V	10
240k	2.4V	11

Note: Other external resistor values may be used as required by the application. Using tighter tolerance resistors and reducing the spacing between the values of the resistors will allow more addresses to be encoded into each resistor in the application circuit.

13.0 REGISTER MAPS

A summary of the registers associated with the I²C module is provided in Table 13-1 and Table 13-2.

Table 13-1: I2Cx Register Map

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
I2CxRCV	—	—	—	—	—	—	—	—	I2Cx Receive Register								0000
I2CxTRN	—	—	—	—	—	—	—	—	I2CxTransmit Register								00FF
I2CxBRG	—	—	—	—	—	—	—	I2Cx Baud Rate Generator Register									0000
I2CxCON(1)	I2CEN	—	I2CSIDL	SCLREL	IPMIEN	A10M	DISSLW	SMEN	GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	1000
I2CxSTAT	ACKSTAT	TRSTAT	ACKTIM(1)	—	—	BCL	GCSTAT	ADD10	IWCOL	I2COV	D/$\overline{A}$	P	S	R/$\overline{W}$	RBF	TBF	0000
I2CxADD	—	—	—	—	—	—	I2Cx Address Register										0000
I2CxMSK	—	—	—	—	—	—	I2Cx Address Mask Register										0000
ISRCCON	ISRCEN	—	—	—	—	OUTSEL2	OUTSEL1	OUTSEL0	—	—	ISRCCAL5	ISRCCAL4	ISRCCAL3	ISRCCAL2	ISRCCAL1	ISRCCAL0	0000

Legend: — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Note 1: These registers and/or bits are not available on all devices. Refer to the specific device data sheet for availability.

Table 13-2: I2Cx Register Map

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
I2CxRCV	—	—	—	—	—	—	—	—	I2Cx Receive Register								0000
I2CxTRN	—	—	—	—	—	—	—	—	I2Cx Transmit Register								00FF
I2CxBRG	—	—	—	—	—	—	—	I2Cx Baud Rate Generator Register									0000
I2CxCONL(1)	I2CEN		I2CSIDL	SCLREL	STRICT	A10M	DISSLW	SMEN	GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	1000
I2CxCONH(1)	—	—	—	—	—	—	—	—	—	PCIE	SCIE	BOEN	SDAHT	SBCDE	AHEN	DHEN	0000
I2CxSTAT	ACKSTAT	TRSTAT	ACKTIM(1)	—	—	BCL	GCSTAT	ADD10	IWCOL	I2COV	D/$\overline{A}$	P	S	R/$\overline{W}$	RBF	TBF	0000
I2CxADD	—	—	—	—	—	—	I2Cx Address Register										0000
I2CxMSK	—	—	—	—	—	—	I2Cx Address Mask Register										0000
ISRCCON	ISRCEN	—	—	—	—	OUTSEL2	OUTSEL1	OUTSEL0	—	—	ISRCCAL5	ISRCCAL4	ISRCCAL3	ISRCCAL2	ISRCCAL1	ISRCCAL0	0000

Legend: — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Note 1: These registers and/or bits are not available on all devices. Refer to the specific device data sheet for availability.

© 2007-2015 Microchip Technology Inc.

14.0 DESIGN TIPS

Question 1: ***I am operating as a bus master and transmitting data. Why do slave and receive interrupts keep occurring at the same time?***

Answer: The master and slave circuits are independent. The slave will receive events from the bus sent by the master.

Question 2: ***I am operating as a slave and I write data to the I2CxTRN register. Why is the data not being transmitted?***

Answer: The slave enters an automatic Wait when preparing to transmit. Ensure that you set the SCLREL bit to release the I^2C clock.

Question 3: ***How do I tell what state the master module is in?***

Answer: Looking at the condition of the SEN, RSEN, PEN, RCEN, ACKEN and TRSTAT bits will indicate the state of the master module. If all bits are '0', the module is Idle.

Question 4: ***Operating as a slave, I receive a byte while STREN = 0. What should the user software do if it cannot process the byte before the next one is received?***

Answer: Because STREN was '0', the module did not generate an automatic Wait on the received byte. However, the user software may, at any time during the message, set STREN and then clear SCLREL. This will cause a Wait on the next opportunity to synchronize the SCLx clock.

Question 5: ***My I^2C system is a multi-master system. Why are my messages being corrupted when I attempt to send them?***

Answer: In a multi-master system, other masters may cause bus collisions. In the ISR for the master, check the BCL status bit to ensure that the operation completed without a collision. If a collision is detected, the message must be resent from the beginning.

Note:	In some devices, a separate bus collision interrupt (I2CxBCIF) is generated instead of a master event interrupt (MI2CxIF).

Question 6: ***My I^2C system is a multi-master system. How can I tell when it is okay to begin a message?***

Answer: Check the S status bit. If S = 0, the bus is Idle.

Question 7: ***I tried to send a Start condition on the bus, then transmit a byte by writing to the I2CxTRN register. The byte did not get transmitted. Why?***

Answer: You must wait for each event on the I^2C bus to complete before starting the next one. In this case, you should poll the SEN bit to determine when the Start event completed or wait for the master I^2C interrupt before data is written to the I2CxTRN register.

15.0 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the dsPIC33/PIC24 device families, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Inter-Integrated Circuit (I^2C) module include the following:

Title	Application Note #
Use of the SSP Module in the I^2C™ Multi-Master Environment	AN578
Using the Mid-Range Enhanced Core PIC16 Devices' MSSP Module for Slave I^2C™ Communication	AN734
Using the PICmicro® MSSP Module for Master I^2C™ Communications	AN735
An I^2C™ Network Protocol for Environmental Monitoring	AN736

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the dsPIC33/PIC24 device families.

© 2007-2015 Microchip Technology Inc.

16.0 REVISION HISTORY

Revision A (February 2007)

This is the initial released version of this document.

Revision B (August 2008)

This revision includes the following corrections and updates:

- Updated bit definitions for the ACKSTAT bit (I2CxSTAT<15>) and the D/$\overline{A}$ bit (I2CxSTAT<5>) in Register 3-4.
- Updated the I2CBRG denominator from 1,111,111 to 10,000,000 in Equation 4-1.
- Updated the I^2C clock rate values in Table , removed the table notes and added a general note just after the table.
- Updated the last two paragraphs in **Section 3.0 "Control and Status Registers"** to clarify the shift of matching address bytes in the I2CxRSR register to the I2CxRCV register.
- Updated **Section 4.0 "Enabling I^2C Operation"** to clarify that the master function is enabled when the SEN bit is set, and when data is loaded into the I2CxTRN register.
- Several sections were updated to clarify NACK status in Slave mode. The affected sections are:
  - **Section 4.2 "I^2C Interrupts"**
  - **Section 7.5 "Sending Data to a Master Device"**
  - Figure 7-13 through Figure 7-16
- The IPMIEN bit was incorrectly described as the Intelligent *Peripheral* Management Interface Enable bit. All occurrences have been updated to Intelligent *Platform* Management Interface bit.
- Updated **Section 9.2 "Sleep Mode in Master Mode"** to clarify what occurs when entering Sleep mode while transmitting.
- Updated the slave message RBF status bit information in Figure 7-11 through Figure 7-16.
- Additional minor corrections such as language and formatting updates are incorporated throughout the document.

Revision C (November 2009)

This revision includes the following corrections and updates:

- The document was updated to include the PIC24H families of devices
- Added Note 1 to the IPMIEN bit in the I2CxCON register (Register 3-1)
- Updated the bit status to HSC (Hardware Set/Cleared) for the P and S bits in the I2CxSTAT register (Register 3-4)
- Updated the BRG Reload Value Calculation (Equation 4-1)
- Added a shaded note on the SDAx and SCLx pins to **Section 2.0 "I^2C Bus Characteristics"** and **Section 8.0 "Connection Considerations for I^2C Bus"**
- Added a shaded note after the first paragraph in **Section 5.0 "Communicating as a Master in a Single Master Environment"** and **Section 6.0 "Communicating as a Master in a Multi-Master Environment"**
- Removed the Actual FSCL column, added the PGD column, and updated the Decimal and Hexadecimal values in the I^2C Clock Rates table (Table)
- Added a sentence to the end of the second paragraph and added a shaded note in **Section 7.3.8 "Receiving All Addresses (IPMI Operation)"**
- Added a shaded note to the first paragraph of **Section 7.5 "Sending Data to a Master Device"**
- Updated the last sentence of the first paragraph in **Section 7.5.2 "Example Messages of Slave Transmission"** to clarify that an interrupt will be generated.
- Added a shaded note to **Section 9.1 "Sleep Mode in Slave Mode"**

Revision D (April 2011)

This revision includes the following updates:

- Equations:
  - Updated Equation 8-1, Equation 8-2 and Equation 8-3
- Figures:
  - Updated Figure 5-10
- Notes:
  - Removed the note in **Section 2.0 "I²C Bus Characteristics"**
  - Added Note 2 in **Section 5.2.1 "Sending a 7-Bit Address to the Slave"**
  - Added Note 2 in **Section 5.2.3 "Sending a 10-Bit Address to the Slave"**
  - Added a note in **Section 7.5.1 "Wait States During Slave Transmissions"**
  - Replaced the existing note in **Section 8.0 "Connection Considerations for I²C Bus"**
  - Added Note 1 in Table 13-1
- Registers:
  - Updated the bit 12 description in Register 3-1
- Sections:
  - Updated **Section 8.0 "Connection Considerations for I²C Bus"**
- Updated the Family Reference Manual name to dsPIC33F/PIC24H Family Reference Manual
- Additional minor corrections such as language and formatting updates were incorporated throughout the document

Revision E (September 2011)

This revision includes the following updates:

- Figures:
  - Added Figure 12-1
- Sections:
  - Removed reference to dsPIC30F documentation in **Section 2.2.2 "Address Slave"**
  - Added **Section 12.0 "Constant-Current Source"**
- Additional minor corrections such as language and formatting updates were incorporated throughout the document

© 2007-2015 Microchip Technology Inc.

Revision F (February 2014)

This revision includes the following updates:

- Sections:
  - Updated **Section 1.0 “Introduction”**
  - Updated **Section 3.0 “Control and Status Registers”**
  - Updated **Section 4.0 “Enabling I^2C Operation”**, **Section 5.0 “Communicating as a Master in a Single Master Environment”**
  - Included **Section 5.2.2 “STRICT Support in Master Mode”**
  - Updated **Section 6.3 “Bus Arbitration and Bus Collision”**, **Section 6.4 “Detecting Bus Collisions and Resending Messages”**
  - Included **Section 7.2.1 “Interrupt on Start/Repeated Start and Stop Conditions (Slave Mode)”**
  - Updated **Section 7.3.2 “7-Bit Address and Slave Write”**
  - Included **Section 7.3.3 “7-Bit Address and Slave Write with the AHEN and DHEN Bits”**
  - Updated **Section 7.3.4 “7-Bit Address and Slave Read”**
  - Included **Section 7.3.6 “Slave Mode Bus Collision”**
  - Updated note in **Section 7.3.7 “General Call Operation”**
  - Included **Section 7.3.9 “STRICT Support”**
  - Note included in **Section 5.4 “Acknowledge Generation”**
  - Included **Section 7.4.2 “Receive Buffer Overwrite (I^2C Slave Mode Only)”**
  - Updated **Section 14.0 “Design Tips”**
- Figures
  - Updated Figure 1-1, Figure 2-1, Figure 7-2, Figure 7-5, Figure 7-7, Figure 7-9, Figure 7-11, Figure 7-12, Figure 7-13 and Figure 7-14
  - Included new Figure 7-17
- Tables
  - Update Table 5-1
  - Updated note in Table 7-2
  - Updated Table 7-3
- Registers
  - Added Register 3-2 and Register 3-3
  - Updated Register 3-4
  - Updated register map Table 13-1
- Equation
  - Updated Equation 4-1
- Additional minor corrections, such as language and formatting updates, were incorporated throughout the document.

Revision G (November 2015)

This revision includes the following updates:

- Sections
  - Added a note to **Section 2.0 “I^2C Bus Characteristics”**
  - Added a note to **Section 5.1 “Generating Start Bus Event”**
- Figures
  - Updated Figure 7-6, Figure 7-14, Figure 7-15 and Figure 7-16
- Additional minor corrections, such as language and formatting updates, were incorporated throughout the document.
- Removed the trademark from Inter-Integrated Circuit (I^2C) as it is no longer active.

NOTES:

© 2007-2015 Microchip Technology Inc.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.** Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, flexPWR, JukeBlox, KEELOQ, KEELOQ logo, Kleer, LANCheck, MediaLB, MOST, MOST logo, MPLAB, OptoLyzer, PIC, PICSTART, PIC^{32} logo, RightTouch, SpyNIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

The Embedded Control Solutions Company and mTouch are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, ECAN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, KleerNet, KleerNet logo, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, RightTouch logo, REAL ICE, SQI, Serial Quad I/O, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2007-2015, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-0055-4

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

07/14/15

© 2007-2015 Microchip Technology Inc.