Section 24. Inter-Integrated Circuit™ (I^2C™)

HIGHLIGHTS

This section of the manual contains the following major topics:

24.1 Overview 24-2
24.2 I^2C Bus Characteristics 24-4
24.3 Control and Status Registers 24-7
24.4 Enabling I^2C Operation 24-13
24.5 Communicating as a Master in a Single Master Environment 24-15
24.6 Communicating as a Master in a Multi-Master Environment 24-28
24.7 Communicating as a Slave 24-31
24.8 Connection Considerations for I^2C Bus 24-46
24.9 Module Operation During PWRSAV Instruction 24-47
24.11 Effects of a Reset 24-47
24.12 Register Maps 24-48
24.13 Electrical Specifications 24-49
24.14 Design Tips 24-53
24.15 Related Application Notes 24-54
24.16 Revision History 24-55

24.1 OVERVIEW

The Inter-Integrated Circuit (I^2C) module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, display drivers, A/D converters, etc.

The I^2C module can operate in any of the following I^2C systems:

- As a slave device
- As a master device in a single master system (slave may also be active)
- As a master/slave device in a multi-master system (bus collision detection and arbitration available)

The I^2C module contains independent I^2C master logic and I^2C slave logic, each generating interrupts based on their events. In multi-master systems, the software is simply partitioned into master controller and 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 other masters in a multi-master system. No messages are lost during multi-master bus arbitration.

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

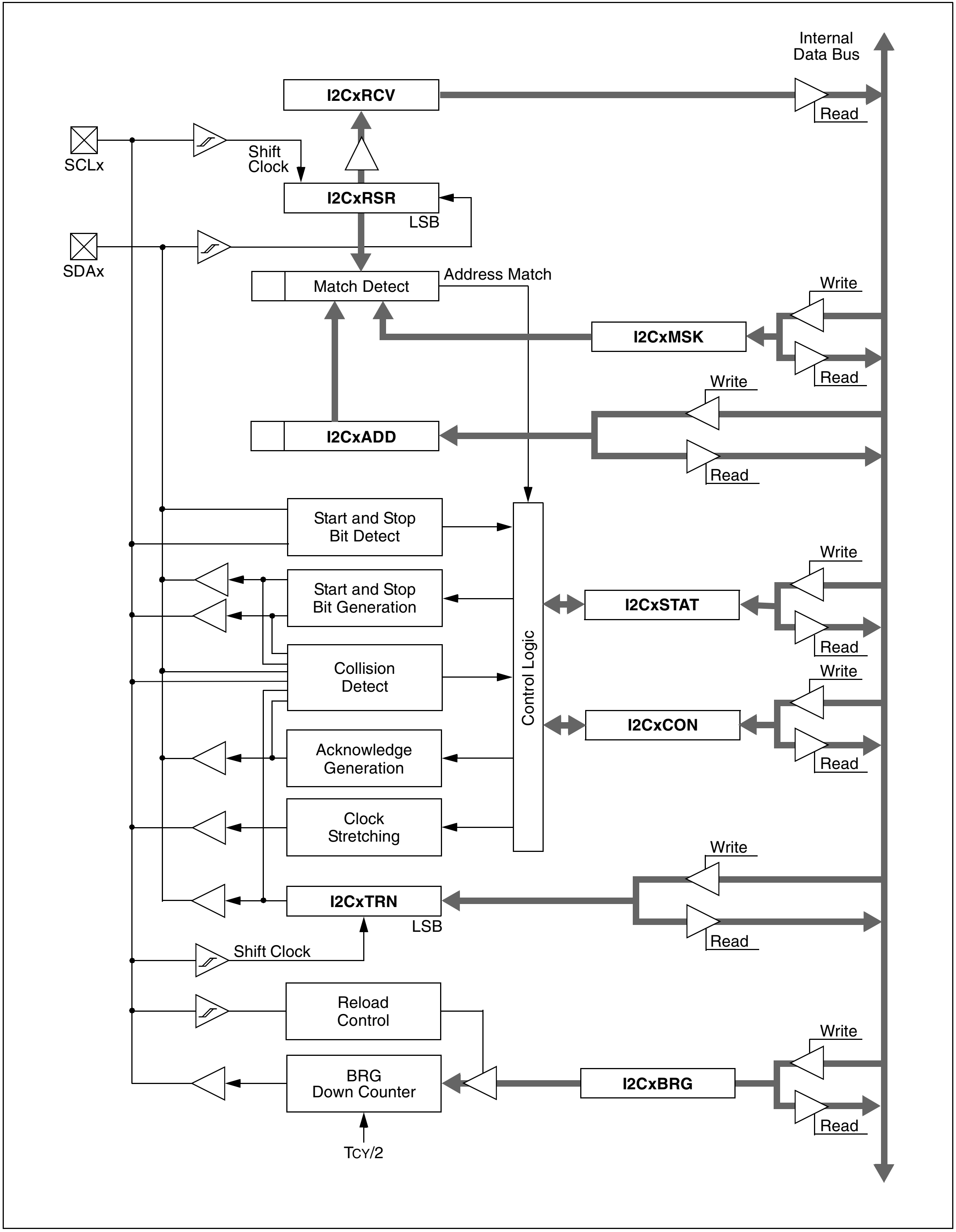
The I^2C module contains a Baud Rate Generator. The I^2C Baud Rate Generator does not consume other timer resources in the device.

Key features of the I^2C module include:

- 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 regardless 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 Peripheral Management Interface (IPMI) standard

Figure 24-1 shows the I^2C module block diagram.

Figure 24-1: I^2C™ Block Diagram



24.2 I^{2}C BUS CHARACTERISTICS

The I^{2}C bus is a two-wire serial interface. Figure 24-2 shows a schematic of an I^{2}C connection between a PIC24F device and a 24LC256 I^{2}C serial EEPROM, which is a typical example for any I^{2}C interface.

The interface employs a comprehensive protocol to ensure reliable transmission and reception of data. When communicating, one device is the “master” which initiates transfer on the bus and generates the clock signals to permit that transfer, while the other device(s) acts 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.

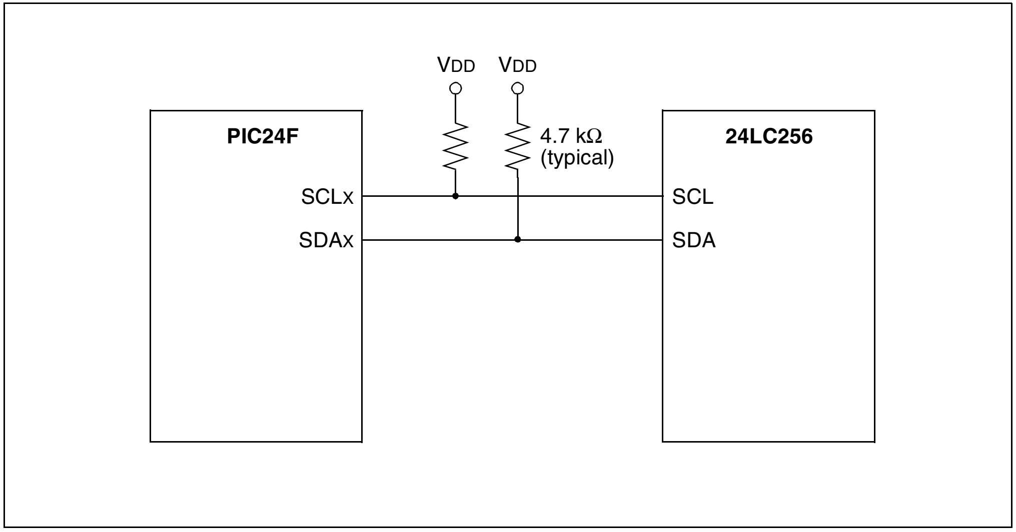
Because the SDAx and SCLx lines are bidirectional, the output stages of the devices driving the SDAx and SCLx lines must have an open-drain in order to perform the wired AND function of the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down.

In the I^{2}C interface protocol, each device has an address. When a master wishes to initiate a data transfer, it first transmits the address of the device that it wishes to “talk” to. All devices “listen” to see if this is their address. Within this address, bit 0 specifies if the master wishes to read from or write to the slave device. The master and slave are always in opposite modes (transmitter/receiver) of operation during a data transfer. That is, they can be thought of as operating in either of these two relations:

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

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

Figure 24-2: Typical I^{2}C™ Interconnection Block Diagram



24.2.1 Bus Protocol

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

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

Accordingly, the following bus conditions have been defined (Figure 24-3).

24.2.1.1 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.

24.2.1.2 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.

24.2.1.3 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 addressed slave device without relinquishing control of the bus.

24.2.1.4 DATA VALID (D)

The state of the SDAx line represents valid data when, after a Start condition, 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.

24.2.1.5 ACKNOWLEDGE (A) OR NOT ACKNOWLEDGE (N)

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

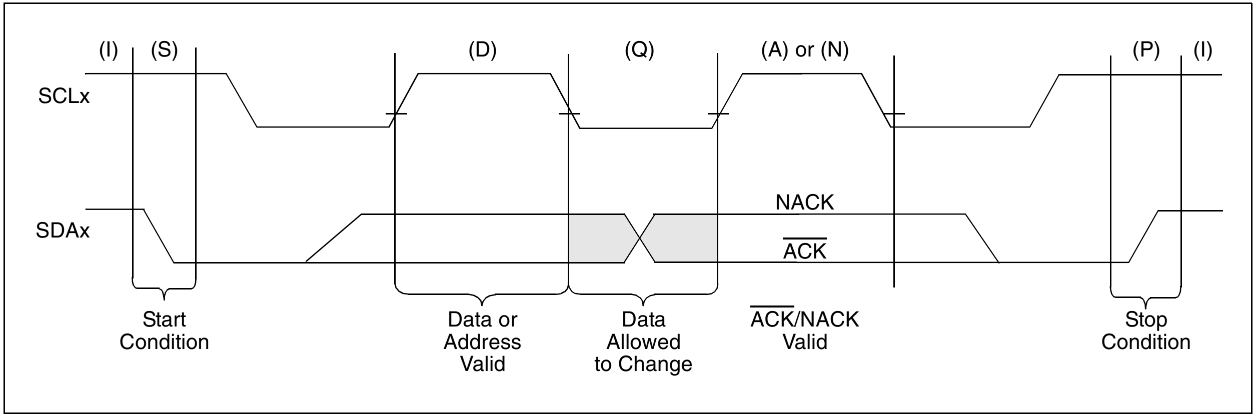
24.2.1.6 WAIT/DATA INVALID (Q)

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

24.2.1.7 BUS IDLE (I)

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

Figure 24-3: I^2C™ Bus Protocol States

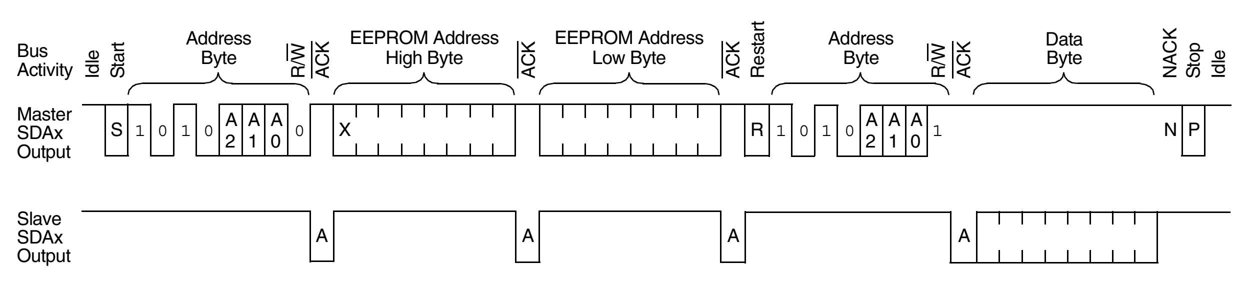


24.2.2 Message Protocol

A typical I^2C message is shown in Figure 24-4. In this example, the message will read a specified byte from a 24LC256 I^2C serial EEPROM. The PIC24F device will act as the master and the 24LC256 device will act as the slave.

Figure 24-4 indicates the data as driven by the master device and the data as driven by the slave device, remembering 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 24-4: A Typical I²C™ Message: Read of Serial EEPROM (Random Address Mode)



24.2.2.1 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 “device address byte” or “data byte”.

24.2.2.2 ADDRESS SLAVE

In Figure 24-4, the first byte is the device address byte that must be the first part of any I^2C message. It contains a device address and a $R/\overline{W}$ bit. Refer to **Appendix A** (check Microchip web site for availability: www.microchip.com) for additional information on address byte formats. 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.

24.2.2.3 SLAVE ACKNOWLEDGE

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

24.2.2.4 MASTER TRANSMIT

The next 2 bytes, sent by the master to the slave, are data bytes containing the location of the requested EEPROM data byte. The slave must Acknowledge each of the data bytes.

24.2.2.5 REPEATED START

At this point, the slave EEPROM has the address information necessary to return the requested data byte to the master. However, the $R/\overline{W}$ bit from the first device address byte specified master transmission and slave reception. The bus must be turned in the other direction for the slave to send data to the master.

To do 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 the $R/\overline{W}$ = `1` to indicate slave transmission and master reception.

24.2.2.6 SLAVE REPLY

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

24.2.2.7 MASTER ACKNOWLEDGE

During reads, a master must terminate data requests to the slave by Not Acknowledging (generate a “NACK”) on the last byte of the message.

24.2.2.8 STOP MESSAGE

The master sends Stop to terminate the message and return the bus to an Idle state.

24.3 CONTROL AND STATUS REGISTERS

The I^2C module has seven user-accessible registers for operation. All registers are accessible in either Byte or Word mode. The registers are:

- Control Register (I2CxCON): This register (Register 24-1) allows control of the module’s operation.
- Status Register (I2CxSTAT): This register (Register 24-2) contains status flags indicating the module’s state during operation.
- Address Mask Register (I2CxMSK): This register (Register 24-3) designates which bit positions in I2CxADD can be ignored, which allows for multiple address support.
- Receive Buffer Register (I2CxRCV): This is the buffer register from which data bytes can be read. The I2CxRCV register is a read-only register.
- Transmit Register (I2CxTRN): This is the transmit register; bytes are written to this register during a transmit operation. The I2CxTRN register is a read/write register.
- Address Register (I2CxADD): This register holds the slave device address.
- Baud Rate Generator Reload Register (I2CxBRG): Holds the Baud Rate Generator reload value for the I^2C module Baud Rate Generator.

I2CxTRN is the register to which transmit data is written. This register is used when the module operates as a master transmitting data to the slave, or as a slave sending reply data to the master. As the message progresses, the I2CxTRN register shifts out the individual bits. Because of this, the I2CxTRN may not be written to unless the bus is Idle. The I2CxTRN may be reloaded while the current data is transmitting.

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 create a double-buffered receiver. This allows reception of the next byte to begin before reading the current byte of received data.

If the module receives another complete byte before the 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 is lost.

The I2CxADD register holds the slave device address. In 10-Bit Addressing mode, all bits are relevant. In 7-Bit Addressing mode, only I2CxADD<6:0> are relevant. The A10M bit (I2CxCON<10>) specifies the expected mode of the slave address. By using the I2CxMSK register with I2CxADD in either Slave Addressing mode, one or more bit positions can be removed from exact address matching, allowing the module in Slave mode to respond to multiple addresses.

Register 24-1: 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	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:		U = Unimplemented bit, read as ‘0’	
R = Readable bit	W = Writable bit	HS = Set in Hardware	HC = Cleared in Hardware
-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 I^2C pins are controlled by port functions.

bit 14 **Unimplemented:** Read as ‘`0`’

bit 13 **I2CSIDL:** Stop in Idle Mode bit

`1` = Discontinue module operation when device enters an Idle mode
`0` = Continue module operation in Idle mode

bit 12 **SCLREL:** SCLx Release Control bit (when operating as I^2C slave)

`1` = Release SCLx clock
`0` = Hold SCLx clock low (clock stretch)

If STREN = `1`:
Bit is R/W (i.e., software may write ‘`0`’ to initiate stretch and write ‘`1`’ to release clock). Hardware clear at beginning of slave transmission and at end of slave reception.

If STREN = `0`:
Bit is R/S (i.e., software may only write ‘`1`’ to release clock). Hardware clear at beginning of slave transmission

bit 11 **IPMIEN:** Intelligent Peripheral Management Interface (IPMI) Enable bit

`1` = IPMI Support mode is enabled; all addresses Acknowledged
`0` = IPMI Support mode disabled

bit 10 **A10M:** 10-Bit Slave Address bit

`1` = I2CxADD is a 10-bit slave address
`0` = I2CxADD is a 7-bit slave address

bit 9 **DISSLW:** Disable Slew Rate Control bit

`1` = Slew rate control disabled
`0` = Slew rate control enabled

bit 8 **SMEN:** SMBus Input Levels bit

`1` = Enable I/O pin thresholds compliant with SMBus specification
`0` = Disable SMBus input thresholds

bit 7 **GCEN:** General Call Enable bit (when operating as I^2C slave)

`1` = Enable interrupt when a general call address is received in the I2CxRSR (module is enabled for reception)
`0` = General call address disabled

bit 6 **STREN:** SCLx Clock Stretch Enable bit (I^2C Slave mode only, used in conjunction with SCLREL bit)

`1` = Enable software or receive clock stretching
`0` = Disable software or receive clock stretching

Register 24-1: I2CxCON: I2Cx Control Register (Continued)

bit 5	**ACKDT:** Acknowledge Data bit (I^{2}C Master mode, receive operation only) Value that will be transmitted when the software initiates an Acknowledge sequence. `1` = Send NACK during Acknowledge `0` = Send $\overline{\text{ACK}}$ during Acknowledge
bit 4	**ACKEN:** Acknowledge Sequence Enable bit (I^{2}C Master mode receive operation) `1` = Initiate Acknowledge sequence on SDAx and SCLx pins and transmit ACKDT data bit (hardware clear at end of master Acknowledge sequence) `0` = Acknowledge sequence not in progress
bit 3	**RCEN:** Receive Enable bit (I^{2}C Master mode) `1` = Enables Receive mode for I^{2}C (hardware clear at end of eighth bit of master receive data byte) `0` = Receive sequence not in progress
bit 2	**PEN:** Stop Condition Enable bit (I^{2}C Master mode) `1` = Initiate Stop condition on SDAx and SCLx pins (hardware clear at end of master Stop sequence) `0` = Stop condition not in progress
bit 1	**RSEN:** Repeated Start Condition Enable bit (I^{2}C Master mode) `1` = Initiate Repeated Start condition on SDAx and SCLx pins (hardware clear at end of master Repeated Start sequence) `0` = Repeated Start condition not in progress
bit 0	**SEN:** Start Condition Enable bit (I^{2}C Master mode) `1` = Initiate Start condition on SDAx and SCLx pins (hardware clear at end of master Start sequence) `0` = Start condition not in progress

Register 24-2: I2CxSTAT: I2Cx Status Register

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

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

Legend:		U = Unimplemented bit, read as ‘0’	
R = Readable bit	C = Clearable bit	HS = Set in Hardware	HSC = Hardware Set/Cleared
-n = Value at Reset	‘1’ = Bit is set	‘0’ = Bit is clear	x = Bit is unknown

bit 15 **ACKSTAT:** Acknowledge Status bit (I^2C Master mode transmit operation)

`1` = NACK received from slave
`0` = $\overline{\text{ACK}}$ received from slave
Hardware set or clear at end of slave Acknowledge.

bit 14 **TRSTAT:** Transmit Status bit (I^2C Master mode transmit operation)

`1` = Master transmit is in progress (8 bits + $\overline{\text{ACK}}$)
`0` = Master transmit is not in progress
Hardware set at beginning of master transmission; hardware clear at end of slave Acknowledge.

bit 13-11 **Unimplemented:** Read as ‘`0`’

bit 10 **BCL:** Master Bus Collision Detect bit

`1` = A bus collision has been detected during a master operation
`0` = No collision
Hardware set at detection of bus collision.

bit 9 **GCSTAT:** General Call Status bit

`1` = General call address was received
`0` = General call address was not received
Hardware set when address matches general call address; hardware clear 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 set at match of 2nd byte of matched 10-bit address; hardware clear at Stop detection.

bit 7 **IWCOL:** Write Collision Detect bit

`1` = An attempt to write the I2CxTRN register failed because the I^2C module is busy
`0` = No collision
Hardware set at occurrence of write to I2CxTRN while busy (cleared by software).

bit 6 **I2COV:** Receive Overflow Flag bit

`1` = A byte was received while the I2CxRCV register is still holding the previous byte
`0` = No overflow
Hardware set at attempt to transfer I2CxRSR to I2CxRCV (cleared by software).

bit 5 **D/$\overline{\text{A}}$:** Data/$\overline{\text{Address}}$ bit (I^2C Slave mode)

`1` = Indicates that the last byte received was data
`0` = Indicates that the last byte received was device address
Hardware clear at device address match; hardware set by write to I2CxTRN or by reception of slave byte.

bit 4 **P:** Stop bit

`1` = Indicates that a Stop bit has been detected last
`0` = Stop bit was not detected last
Hardware set or clear when Start, Repeated Start or Stop detected.

Register 24-2: I2CxSTAT: I2Cx Status Register (Continued)

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 set or clear when Start, Repeated Start or Stop detected.
bit 2	**R/$\overline{\text{W}}$:** Read/$\overline{\text{Write}}$ Information bit (when operating as I^{2}C slave) `1` = Read: data transfer is output from slave `0` = Write: data transfer is input to slave Hardware set or clear after reception of I^{2}C device address byte.
bit 1	**RBF:** Receive Buffer Full Status bit `1` = Receive complete, I2CxRCV is full `0` = Receive not complete, I2CxRCV is empty Hardware set when I2CxRCV is written with received byte; hardware clear when software reads I2CxRCV.
bit 0	**TBF:** Transmit Buffer Full Status bit `1` = Transmit in progress, I2CxTRN is full `0` = Transmit complete, I2CxTRN is empty Hardware set when software writes to I2CxTRN; hardware clear at completion of data transmission.

Register 24-3: 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
—	—	—	—	—	—	AMSK9	AMSK8
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
AMSK7	AMSK6	AMSK5	AMSK4	AMSK3	AMSK2	AMSK1	AMSK0
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 **AMSKx:** Mask for Address Bit x Select bit

For 10-Bit Address:

`1` = Enable masking for bit Ax of incoming message address; bit match not required in this position

`0` = Disable masking for bit Ax; bit match required in this position

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

`1` = Enable masking for bit Ax + 1 of incoming message address; bit match not required in this position

`0` = Disable masking for bit Ax + 1; bit match required in this position

24.4 ENABLING I^2C OPERATION

The module is enabled by setting the I2CEN (I2CxCON<15>) bit.

The I^2C 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 software or bus events.

When initially enabled, the module will release SDAx and SCLx pins, putting the bus into the Idle state. The master functions will remain in the Idle state unless software sets a control bit to initiate a master event. 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.

24.4.1 Enabling I^2C I/O

Two pins are used for 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 module software need not be concerned with the state of the port I/O of the pins, the module overrides, the port state and direction. At initialization, the pins are tri-state (released).

24.4.2 I^2C Interrupts

The I^2C module generates two interrupts. One interrupt, MI2CxIF, is assigned to master events; the other interrupt, SI2CxIF, is assigned to slave events. These interrupts will set a corresponding interrupt flag bit and will interrupt the software process if the corresponding interrupt enable bit is set and the corresponding interrupt priority is high enough.

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

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

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

- Detection of a valid device address (including general call)
- Request to transmit data
- Reception of data

24.4.3 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, 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 2 TBRG intervals.

The reload value for the Baud Rate Generator is the I2CxBRG register, as shown in Figure 24-5. When the Baud Rate Generator is loaded with this value, the generator counts down to 0 and stops until another reload has taken place. The generator count is decremented twice per instruction cycle (TCY). The Baud Rate Generator is reloaded automatically on baud rate restart. For example, if clock synchronization is taking place, the Baud Rate Generator will be reloaded when the SCLx pin is sampled high.

Note:	I2CxBRG values of less than 2 are not supported.

To compute the Baud Rate Generator reload value, use the following equation:

Equation 24-1:

$$F_{SCL} = \frac{F_{CY}}{2 \cdot (\text{I2CxBRG} + 1)} \quad \text{or} \quad \text{I2CxBRG} = \left(\frac{F_{CY}}{2 \cdot F_{SCL}}\right) - 1$$

Note 1: Based on $T_{CY} = F_{OSC}/2$; Doze mode and PLL are disabled.

Table 24-1: I²C™ Clock Rates

Required System FSCL	FCY	I2CxBRG Value (Decimal)	I2CxBRG Value (Hexadecimal)	Actual FSCL
100 kHz	16 MHz	79	4F	100 kHz
100 kHz	8 MHz	39	27	100 kHz
100 kHz	4 MHz	19	13	100 kHz
400 kHz	16 MHz	19	13	400 kHz
400 kHz	8 MHz	9	9	400 kHz
400 kHz	4 MHz	4	4	400 kHz
400 kHz	2 MHz	2	2	333 kHz(1)
1 MHz	16 MHz	7	7	1 MHz
1 MHz	8 MHz	3	3	1 MHz(2)
1 MHz	4 MHz	1	1	1 MHz(3)

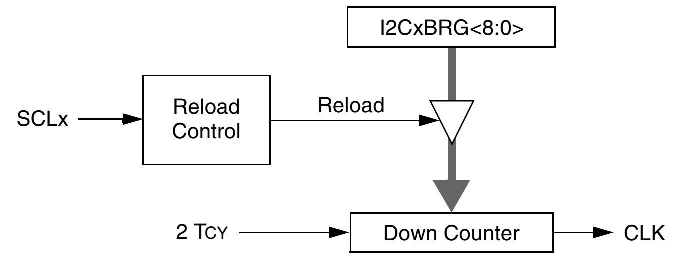
Legend: Shaded rows represent invalid reload values for a given FSCL and FCY.

Note 1: This is the closest value to 400 kHz for this value of FCY.

2: FCY = 2 MHz is the minimum input clock frequency to have FSCL = 1 MHz.

3: I2CxBRG cannot have a value of less than 2.

Figure 24-5: Baud Rate Generator Block Diagram



24.5 COMMUNICATING AS A MASTER IN A SINGLE MASTER ENVIRONMENT

The I^2C module's typical operation in a system is using the I^2C to communicate with an I^2C peripheral, such as an I^2C serial memory. In an I^2C system, the master controls the sequence of all data communication on the bus. In this example, the PIC24F and its I^2C 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.

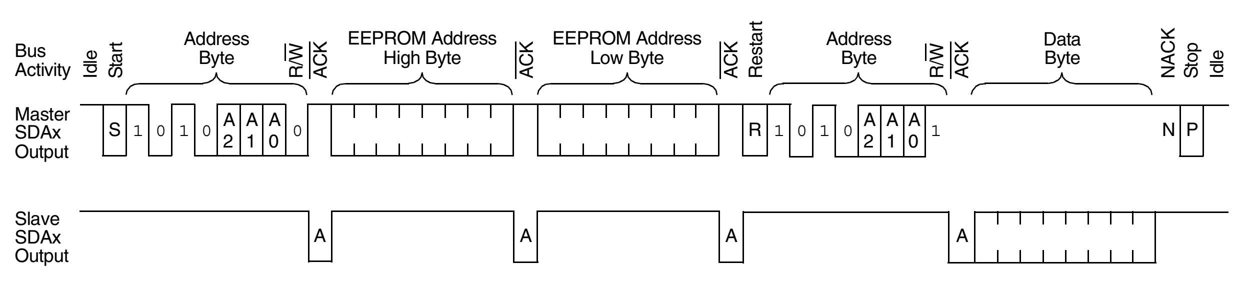
In the I^2C module, the module controls individual portions of the I^2C message protocol; however, sequencing of the components of the protocol to construct a complete message is a software task.

For example, a typical operation in a single master environment may be to read a byte from an I^2C serial EEPROM. This example message is depicted in Figure 24-6.

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

1. Assert a Start condition on SDAx and SCLx.
2. Send the I^2C 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 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 24-6: Typical I^2C™ Message: Read of Serial EEPROM (Random Address Mode)



The I^2C module supports Master mode communication with the inclusion of Start and Stop generators, data byte transmission, data byte reception, Acknowledge generator and a Baud Rate Generator. Generally, the software will write to a control register to start a particular step, then wait for an interrupt or poll status to wait for completion.

Subsequent sections detail each of these operations.

> **Note:** The I^2C module does not allow queueing of events. For instance, the 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 will not be written to and the IWCOL bit will be set, indicating that this write to the I2CxTRN did not occur.

24.5.1 Generating Start Bus Event

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

Figure 24-7 shows the timing of the Start condition.

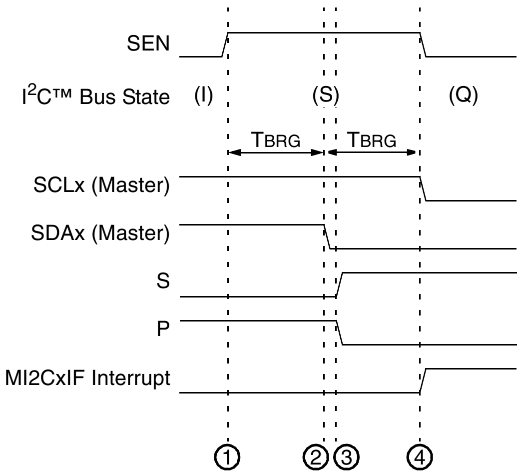
- Slave logic detects the Start condition, sets the S bit (I2CxSTAT<3>) and clears the P bit (I2CxSTAT<4>).
- SEN bit is automatically cleared at completion of the Start condition.
- MI2CxIF interrupt generated at completion of the Start condition.
- After Start condition, SDAx line and SCLx line are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

24.5.1.1 IWCOL STATUS FLAG

If the software writes the I2CxTRN when a Start sequence is in progress, then IWCOL is set and the contents of the transmit buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of I2CxCON is disabled until the Start condition is complete.

Figure 24-7: Master Start Timing Diagram



① - Writing SEN = `1` initiates a master Start event.
Baud Rate Generator starts.

② - Baud Rate Generator times out. Master module drives SDAx low.
Baud Rate Generator restarts.

③ - Slave module detects Start, sets S = `1`, P = `0`.

④ - Baud Rate Generator times out. Master module drives SCLx low, generates interrupt and clears SEN.

24.5.2 Sending Data to a Slave Device

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

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

Note that 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 software.

24.5.2.1 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^{2}C device address and a R/$\overline{\text{W}}$ bit that defines if 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).

24.5.2.2 SENDING A 10-BIT ADDRESS TO THE SLAVE

Sending a 10-bit device address involves sending 2 bytes to the slave. The first byte contains 5 bits of the I^{2}C device address reserved for 10-Bit Addressing modes and 2 bits of the 10-bit address. Because the next byte, which contains the remaining 8 bits of the 10-bit address must be received by the slave, the R/$\overline{\text{W}}$ 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 the data. However, if the master expects a reply from the slave, a Repeated Start sequence with the R/$\overline{\text{W}}$ bit at ‘`1`’ will change the R/$\overline{\text{W}}$ state of the message to a read of the slave.

24.5.2.3 RECEIVING ACKNOWLEDGE FROM THE SLAVE

On the falling edge of the eighth SCLx clock, the TBF 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 $\overline{\text{ACK}}$ bit during the ninth bit time, if an address match occurs, or if 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 $\overline{\text{ACK}}$ is written into the Acknowledge Status bit, ACKSTAT (I2CxSTAT<15>), on the falling edge of the ninth SCLx clock. After the ninth SCLx clock, the module generates the MI2CxIF interrupt and enters an Idle state until the next data byte is loaded into I2CxTRN.

24.5.2.4 ACKSTAT STATUS FLAG

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

24.5.2.5 TBF STATUS FLAG

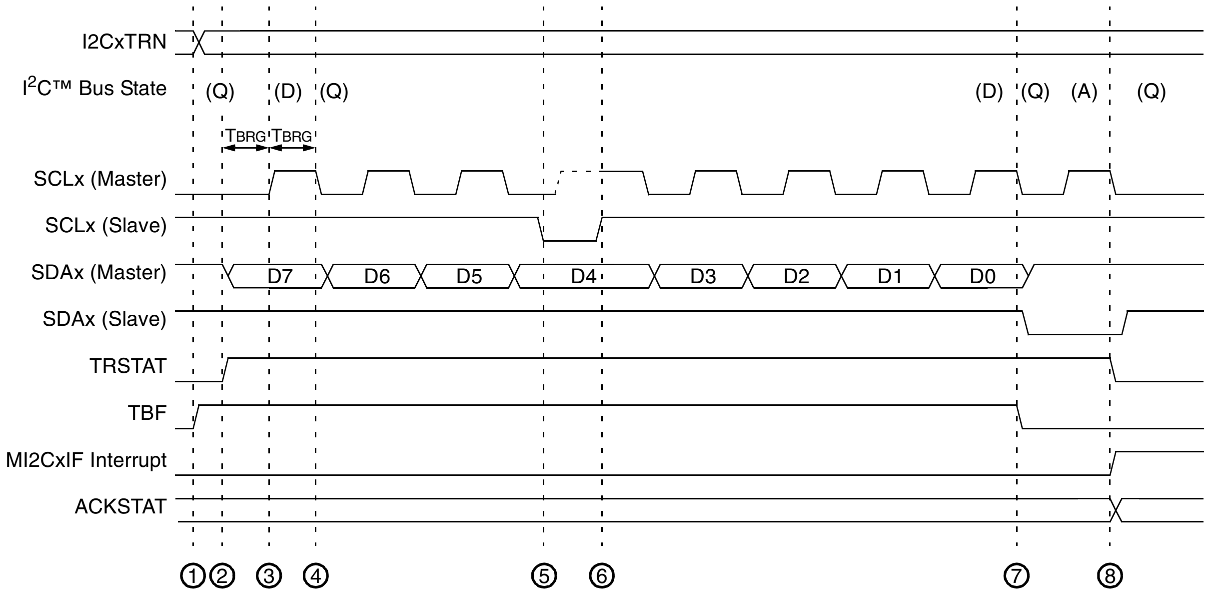
When transmitting, the TBF bit (I2CxSTAT<0>) is set when the CPU writes to I2CXTRN and is cleared when all 8 bits are shifted out.

24.5.2.6 IWCOL STATUS FLAG

If the software writes the I2CXTRN when a transmit is already in progress (i.e., the module is still shifting out a data byte), then IWCOL is set and the contents of the buffer are unchanged (the write doesn’t occur). IWCOL must be cleared in software.

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

Figure 24-8: Master Transmission Timing Diagram



① - Writing the I2CxTRN register will start a master transmission event. TBF bit is set.

② - Baud Rate Generator starts. The MSB of the I2CxTRN drives SDAx. SCLx remains low. TRSTAT bit is set.

③ - Baud Rate Generator times out. SCLx released. Baud Rate Generator restarts.

④ - Baud Rate Generator times out. SCLx driven low. After SCLx detected low, next bit of I2CxTRN 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 wait. Baud Rate Generator restarts.

⑦ - At falling edge of 8th SCLx clock, master releases SDAx. TBF bit is cleared. Slave drives $\overline{\text{ACK}}$/NACK.

⑧ - At falling edge of 9th SCLx clock, master generates interrupt. SCLx remains low until next event.
Slave releases SDAx. TRSTAT bit is clear.

24.5.3 Receiving Data from a Slave Device

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

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

After the falling edge of the eighth SCLx clock:

- The RCEN bit is automatically cleared.
- The contents of the I2CxRSR transfer into the I2CxRCV.
- The RBF flag bit is set.
- The module generates the MI2CxIF interrupt.

When the CPU reads the buffer, the RBF flag bit is automatically cleared. The software can process the data and then do an Acknowledge sequence.

24.5.3.1 RBF STATUS FLAG

When receiving data, the RBF bit is set when a device address or data byte is loaded into I2CxRCV from I2CxRSR. It is cleared when software reads the I2CxRCV register.

24.5.3.2 I2COV STATUS FLAG

If another byte is received in the I2CxRSR while the RBF bit remains set, and the previous byte remains in the I2CxRCV register, the I2COV bit is set and the data in the I2CxRSR is lost.

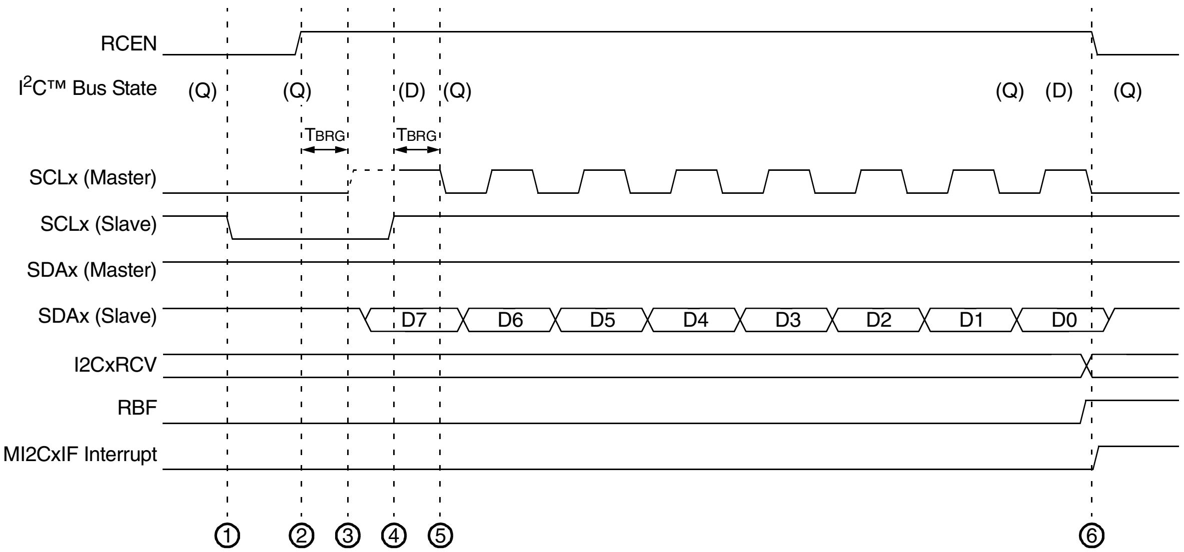
Leaving I2COV set does not inhibit further reception. If RBF is cleared by reading the I2CxRCV, and the I2CxRSR receives another byte, that byte will be transferred to the I2CxRCV.

24.5.3.3 IWCOL STATUS FLAG

If the software writes the I2CxTRN when a receive is already in progress (i.e., I2CxRSR is still shifting in a data byte), then the IWCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

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

Figure 24-9: Master Reception Timing Diagram



① - Typically, the slave can pull SCLx low (clock stretch) to request a wait to prepare 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 Baud Rate Generator starts. SCLx remains low.

③ - Baud Rate Generator times out. Master attempts to release SCLx.

④ - When slave releases SCLx, Baud Rate Generator restarts.

⑤ - Baud Rate Generator times out. MSB of response shifted to I2CxRSR. SCLx driven low for next baud interval.

⑥ - At falling edge of 8th SCLx clock, I2CxRSR transferred to I2CxRCV. Module clears RCEN bit. RBF bit is set. Master generates interrupt.

24.5.4 Acknowledge Generation

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

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

Figure 24-10 shows an $\overline{\text{ACK}}$ sequence and Figure 24-11 shows a NACK sequence. The Acknowledge Data bit, ACKDT (I2CxCON<5>), specifies $\overline{\text{ACK}}$ or NACK.

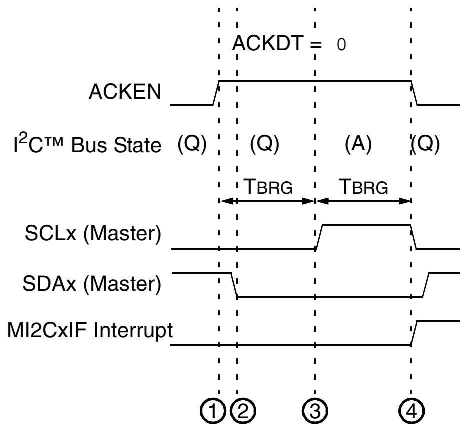
After two baud periods, the ACKEN bit is automatically cleared and the module generates the MI2CxIF interrupt.

24.5.4.1 IWCOL STATUS FLAG

If the software writes the I2CxTRN when an Acknowledge sequence is in progress, then IWCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

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

Figure 24-10: Master Acknowledge ($\overline{\text{ACK}}$) Timing Diagram



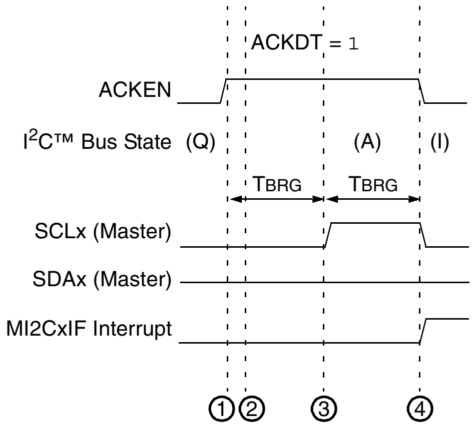
① - Writing ACKDT = `0` specifies sending an $\overline{\text{ACK}}$.
Writing ACKEN = `1` initiates a master Acknowledge event.
Baud Rate Generator starts. SCLx remains low.

② - When SCLx detected low, module drives SDAx low.

③ - Baud Rate Generator times out. Module releases SCLx.
Baud Rate Generator restarts.

④ - Baud Rate Generator times out.
Module drives SCLx low, then releases SDAx.
Module clears ACKEN. Master generates interrupt.

Figure 24-11: Master Not Acknowledge (NACK) Timing Diagram



① - Writing ACKDT = `1` specifies sending a NACK.
Writing ACKEN = `1` initiates a master Acknowledge event.
Baud Rate Generator starts.

② - When SCLx detected low, module releases SDAx.

③ - Baud Rate Generator times out. Module releases SCLx.
Baud Rate Generator restarts.

④ - Baud Rate Generator times out.
Module drives SCLx low, then releases SDAx.
Module clears ACKEN. Master generates interrupt.

24.5.5 Generating Stop Bus Event

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

> **Note:** The lower 5 bits of I2CxCON 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 shown in Figure 24-12.

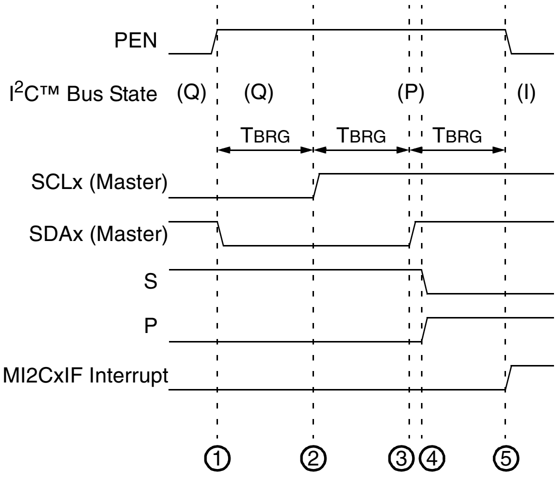
- The slave detects the Stop condition, sets the P bit (I2CxSTAT<4>) and clears the S bit (I2CxSTAT<3>).
- The PEN bit is automatically cleared.
- The module generates the MI2CxIF interrupt.

24.5.5.1 IWCOL STATUS FLAG

If the software writes the I2CxTRN when a Stop sequence is in progress, then the IWCOL bit is set and the contents of the buffer are unchanged (the write doesn’t occur).

> **Note:** Because queueing of events is not allowed, writing to the lower 5 bits of I2CxCON is disabled until the Stop condition is complete.

Figure 24-12: Master Stop Timing Diagram



① - Writing PEN = `1` initiates a master Stop event.
Baud Rate Generator starts. Module drives SDAx low.

② - Baud Rate Generator times out. Module releases SCLx.
Baud Rate Generator restarts.

③ - Baud Rate Generator times out. Module releases SDAx.
Baud Rate Generator restarts.

④ - Slave logic detects Stop. Module sets P = `1`, S = `0`.

⑤ - The Baud Rate Generator times out. Module clears PEN.
Master generates interrupt.

24.5.6 Generating Repeated Start Bus Event

Setting the Repeated Start Enable bit, RSEN (I2CxCON<1>), enables generation of a master Repeated Start sequence (see Figure 24-13).

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

To generate a Repeated Start condition, software sets the RSEN bit (I2CxCON<1>). The module asserts the SCLx pin low. When the module samples the SCLx pin low, the module releases the SDAx pin for one Baud Rate Generator count (TBRG). When the Baud Rate Generator times out and if the module samples SDAx high, the module deasserts the SCLx pin. When the module samples SCLx pin high, the Baud Rate Generator reloads and begins counting. SDAx and SCLx must be sampled high for one TBRG. This action is then followed by assertion of the SDAx pin low for one TBRG while SCLx is high.

The following is the Repeated Start sequence:

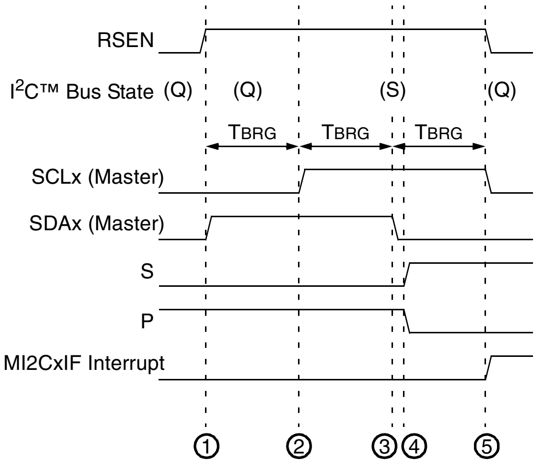
- The slave detects the Start condition, sets the S bit (I2CxSTAT<3>) and clears the P bit (I2CxSTAT<4>).
- The RSEN bit is automatically cleared.
- The module generates the MI2CxIF interrupt.

24.5.6.1 IWCOL STATUS FLAG

If the software writes the I2CxTRN when a Repeated Start sequence is in progress, then IWCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

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

Figure 24-13: Master Repeated Start Timing Diagram



① - Writing RSEN = `1` initiates a master Repeated Start event. Baud Rate Generator starts. Module drives SCLx low and releases SDAx.

② - Baud Rate Generator times out. Module releases SCLx. Baud Rate Generator restarts.

③ - Baud Rate Generator times out. Module drives SDAx low. Baud Rate Generator restarts.

④ - Slave logic detects Start. Module sets S = `1`, P = `0`.

⑤ - The Baud Rate Generator times out. Module drives SCLx low. Module clears RSEN. Master generates interrupt.

24.5.7 Building Complete Master Messages

As described at the beginning of **Section 24.5 "Communicating as a Master in a Single Master Environment"**, the 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 a software task.

The software can use polling or interrupt methods while using the module. The examples shown use interrupts.

The software can use the SEN, RSEN, PEN, RCEN and ACKEN bits (Least Significant 5 bits of the I2CxCON register) and the TRSTAT bit as a "state" flag when progressing through a message. For example, Table 24-2 shows some example state numbers associated with bus states.

Table 24-2: Master Message Protocol States

Example State Number	I2CxCON<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 Acknowledgement

Note: Example state numbers for reference only. User software may assign as desired.

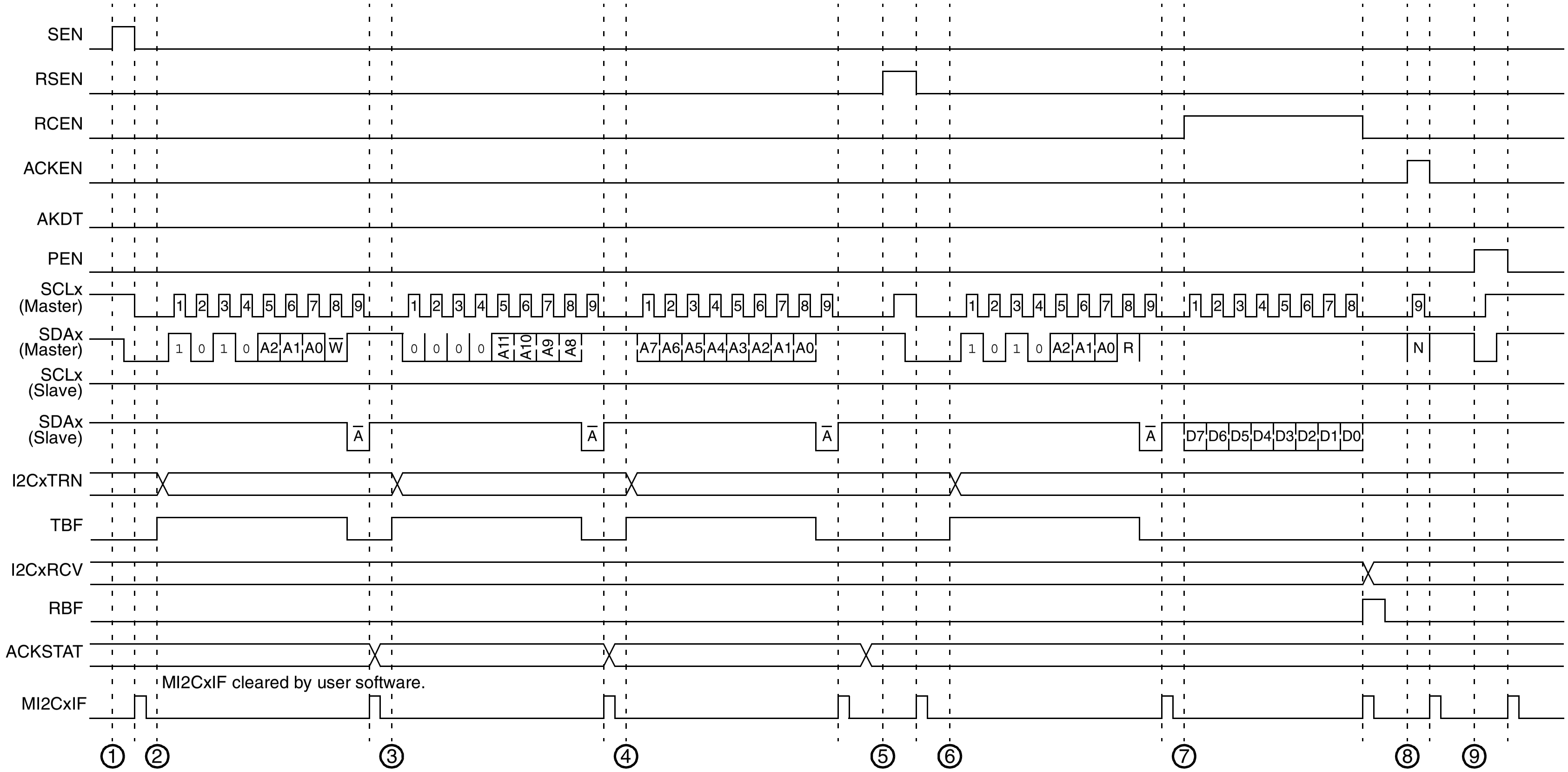
The software will begin a message by issuing a Start command. The software will record the state number corresponding to the Start.

As each event completes and generates an interrupt, the interrupt handler may check the state number. So, 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 24-14 provides a more detailed examination of the same message sequence of Figure 24-6. Figure 24-15 shows some simple examples of messages using 7-bit addressing format. Figure 24-16 shows an example of a 10-bit addressing format message sending data to a slave. Figure 24-17 shows an example of a 10-bit addressing format message receiving data from a slave.

Figure 24-14: Master Message (Typical I^{2}C™ Message: Read of Serial EEPROM)



① - Setting the SEN bit starts a Start event.

② - Writing the I2CxTRN register starts a master transmission. The data is the serial EEPROM device address byte, with R/$\overline{W}$ 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.

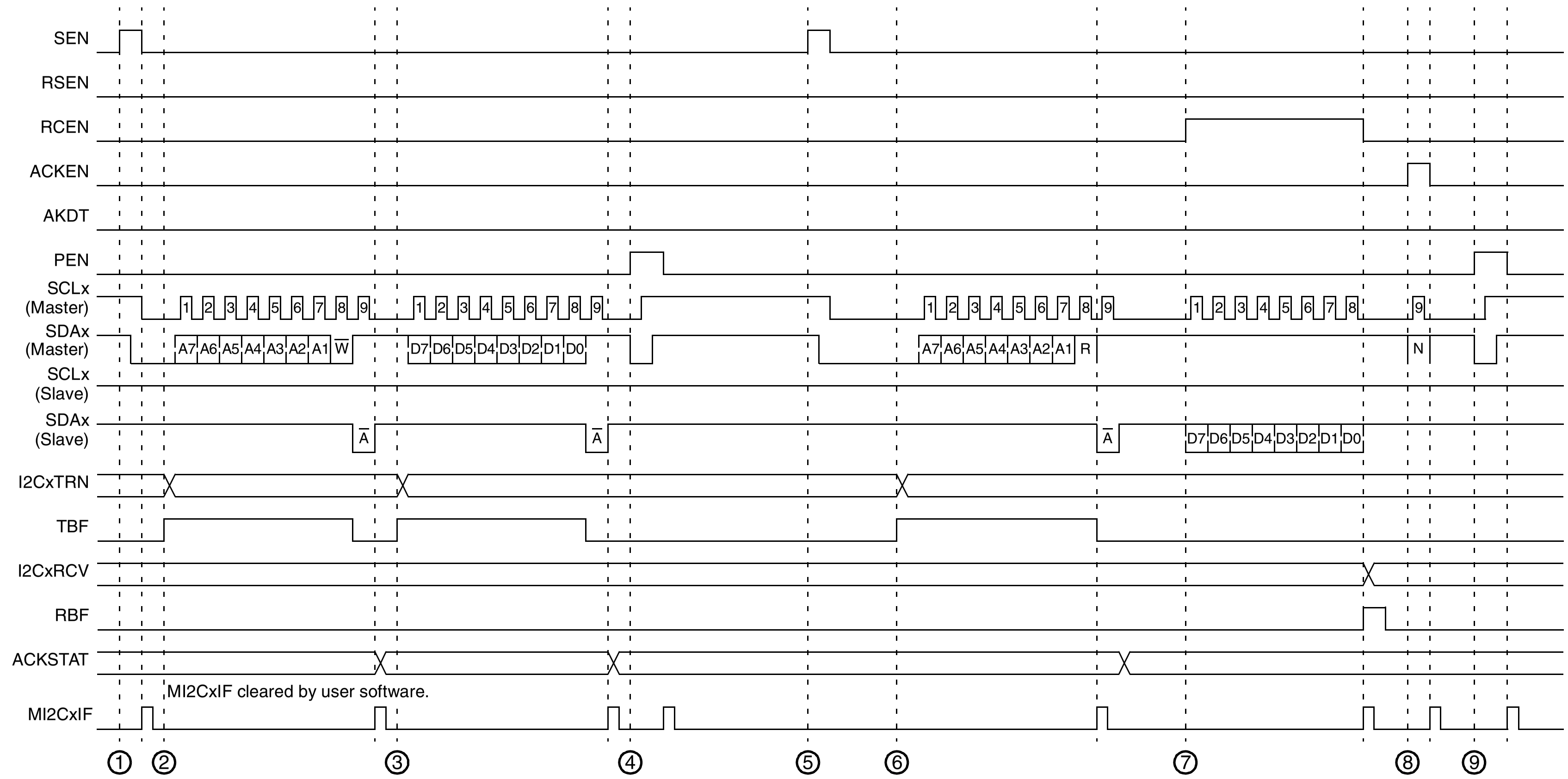
⑥ - Writing the I2CxTRN register starts a master transmission. The data is a resend of the serial EEPROM device address byte, but with R/$\overline{W}$ bit set indicating a read.

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

⑧ - Setting the ACKEN bit starts an Acknowledge event. ACKDT = 0 to send NACK.

⑨ - Setting the PEN bit starts a master Stop event.

Figure 24-15: Master Message (7-Bit Address: Transmission And Reception)



① - Setting the SEN bit starts a Start event.

② - Writing the I2CxTRN register starts a master transmission. The data is the address byte with R/W 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 starts a Start event.

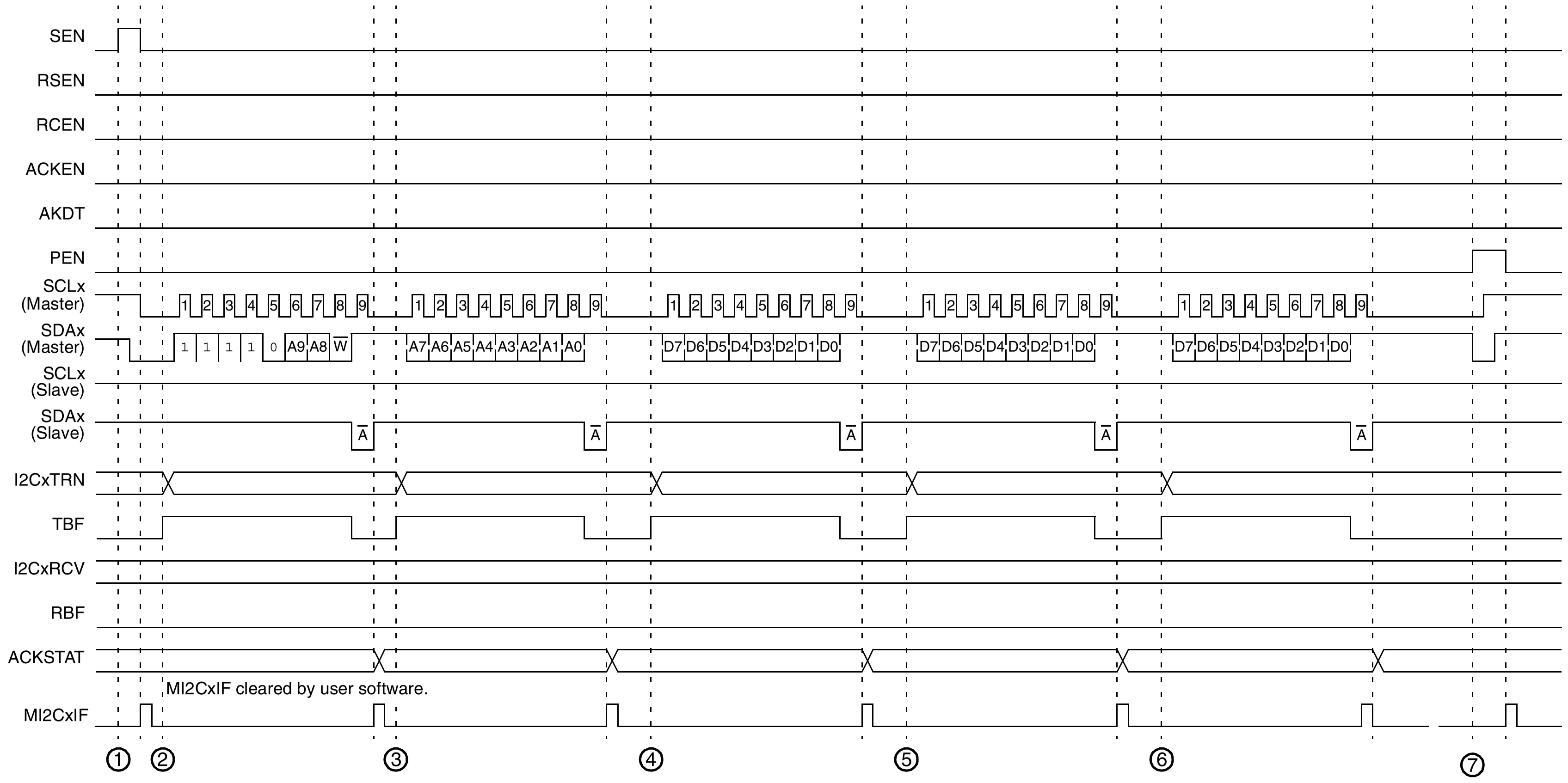
⑥ - Writing the I2CxTRN register starts a master transmission. The data is the address byte with R/W bit set.

⑦ - Setting the RCEN bit starts a master reception.

⑧ - Setting the ACKEN bit starts an Acknowledge event. ACKDT = 0 to send NACK.

⑨ - Setting the PEN bit starts a master Stop event.

Figure 24-16: Master Message (10-Bit Transmission)



① - Setting the SEN bit starts 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.

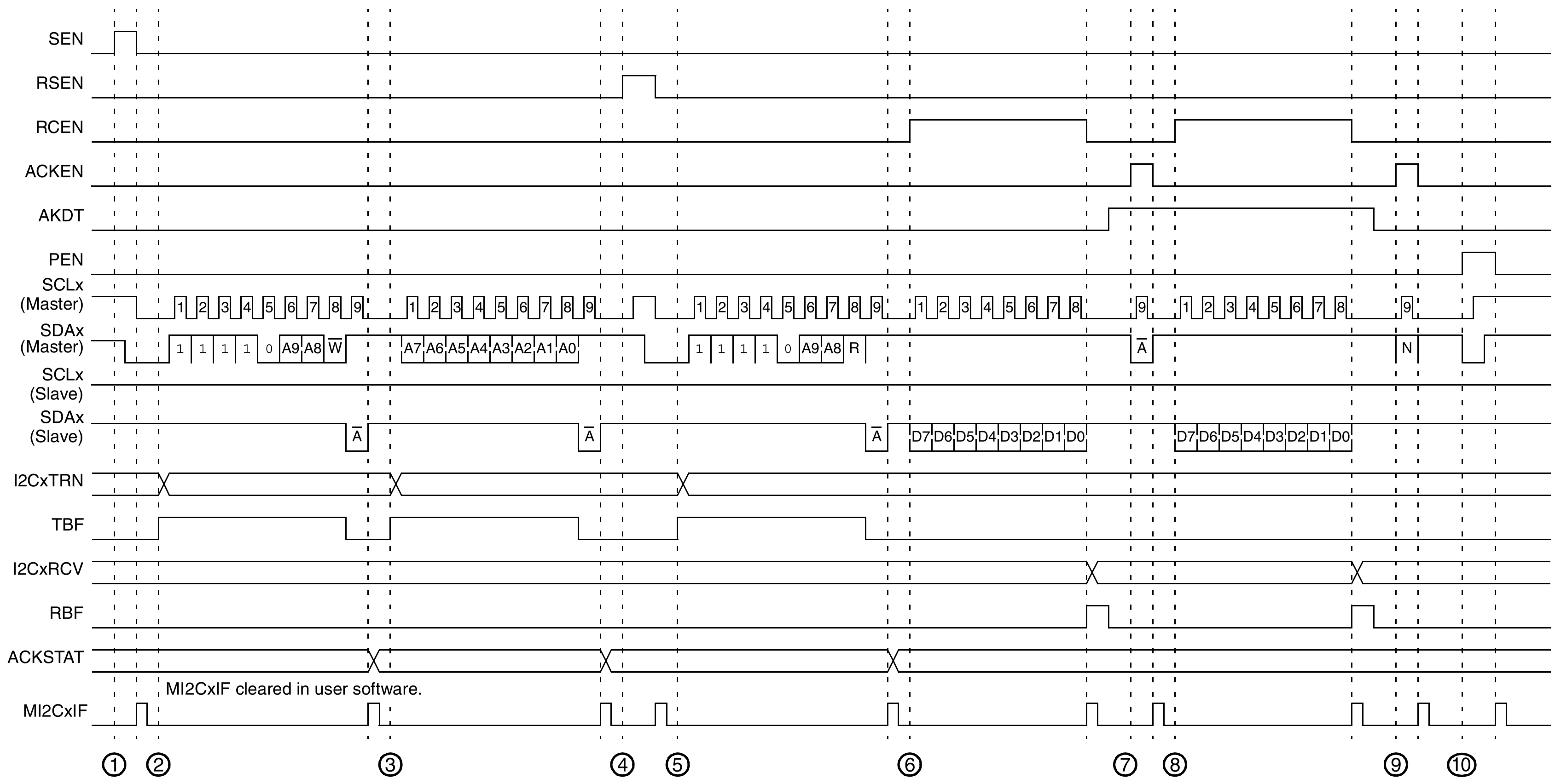
④ - Writing the I2CxTRN register starts a master transmission. The data is the first byte of the message data.

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.

Figure 24-17: Master Message (10-Bit Reception)



① - Setting the SEN bit starts a Start event.

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

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

⑦ - Setting the ACKEN bit starts an Acknowledge event, ACKDT = `1` to send $\overline{\text{ACK}}$.

⑧ - Setting the RCEN bit starts a master reception.

⑨ - Setting the ACKEN bit starts an Acknowledge event, ACKDT = `0` to send NACK.

⑩ - Setting the PEN bit starts a master Stop event.

24.6 COMMUNICATING AS A MASTER IN A MULTI-MASTER ENVIRONMENT

The I^2C protocol allows for more than one master to be attached to a system bus. Remembering 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. Clock synchronization ensures that multiple nodes can synchronize their SCLx clocks to result in one common clock on the SCLx line. Bus arbitration ensures that if more than one node attempts a message transaction, one, and only one node, will be successful in completing the message. The other nodes will lose bus arbitration and be left with a bus collision.

24.6.1 Multi-Master Operation

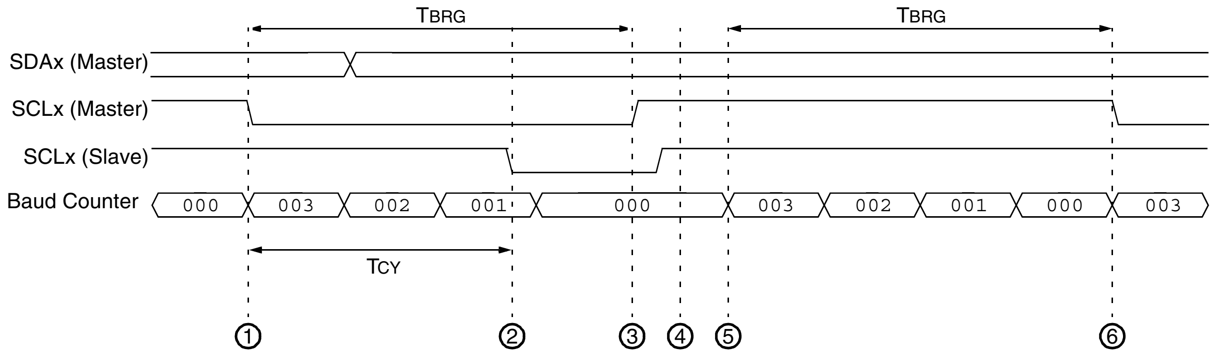
The master module has no special settings to enable multi-master operation. The module performs clock synchronization and bus arbitration at all times. If the module is used in a single master environment, clock synchronization will only occur between the master and slaves and bus arbitration will not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

24.6.2 Master Clock Synchronization

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

Clock synchronization occurs when the master deasserts the SCLx pin (SCLx intended to float high). When the SCLx pin is released, the Baud Rate Generator (BRG) is suspended from counting until the SCLx pin is actually sampled high. When the SCLx pin is sampled high, the Baud Rate Generator 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 shown in Figure 24-18.

Figure 24-18: 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 counter.

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

⑤ - The baud counter rollover occurs on next cycle.

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

24.6.3 Bus Arbitration and Bus Collision

Bus arbitration supports multi-master system operation.

The wired AND nature of the SDAx line permits arbitration. Arbitration takes place when the first master outputs a '`1`' on SDAx by letting SDAx float high and simultaneously, the second master outputs a '`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 a '`1`', yet the data sampled on SDAx is a '`0`'. This is the definition of a bus collision.

The first master will set the Bus Collision bit, BCL (I2CxSTAT<10>), and generate a master 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 module, with the result placed in the BCL bit.

The states where arbitration can be lost are:

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

24.6.4 Detecting Bus Collisions and Resending Messages

When a bus collision occurs, the module sets the BCL bit and generates a master interrupt. If bus collision occurs during a byte transmission, the transmission is halted, the TBF flag is cleared and the SDAx and SCLx pins are deasserted. If 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.

The software is expecting an interrupt at the completion of the master event. The software can check the BCL bit to determine if the master event completed successfully or if a collision occurred. If a collision occurs, the 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 an Idle state. The software can monitor the S and P bits to wait for an Idle bus. When the software services the master Interrupt Service Routine and the I^2C bus is free, the software can resume communication by asserting a Start condition.

24.6.5 Bus Collision During a Start Condition

Before issuing a Start command, the 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. However, certain conditions can cause a bus collision to occur during a Start. In this case, the master that loses arbitration during the Start bit generates a bus collision interrupt.

24.6.6 Bus Collision During a Repeated Start Condition

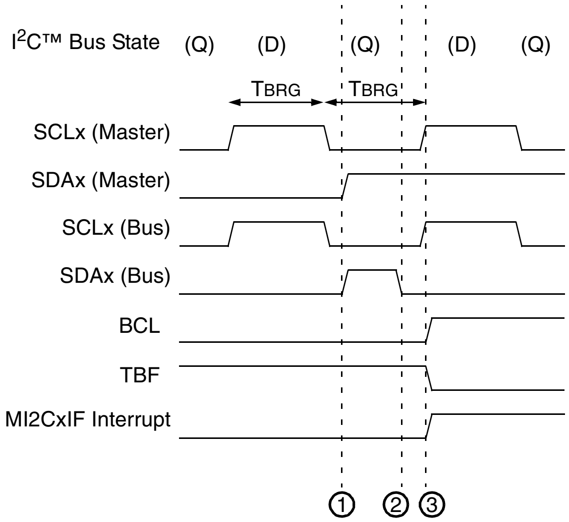
Should two masters not collide throughout an address byte, a bus collision may occur when one master attempts to assert a Repeated Start while another transmits data. In this case, the master generating the Repeated Start will lose arbitration and generate a bus collision interrupt.

24.6.7 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 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 a very similar 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 will begin and continue to transmit their messages until one master loses arbitration on a message bit. Remember that the SCLx clock synchronization will keep the two masters synchronized until one loses arbitration. Figure 24-19 shows an example of message bit arbitration.

Figure 24-19: Bus Collision During Message Bit Transmission



① - Master transmits bit value of '1' in next SCLx clock. Module releases SDAx.

② - Another master on bus transmits bit value of '0' in next SCLx clock. Another master pulls SDAx low.

③ - Baud Rate Generator times out. Module attempts to verify SDAx high. Bus collision detected.
Module releases SDAx, SCLx. Module sets BCL bit and clears TBF bit. Master generates interrupt.

24.6.8 Bus Collision During a Stop Condition

If the master software loses track of the state of the I^2C bus, there are conditions which 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.

24.7 COMMUNICATING AS A SLAVE

In some systems, particularly where multiple processors communicate with each other, the PIC24F device may communicate as a slave (see Figure 24-20). When the module is enabled, the slave module is active. The slave may not 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 module 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 software task. However, the slave module detects when the device address matches the address specified by the software for that slave.

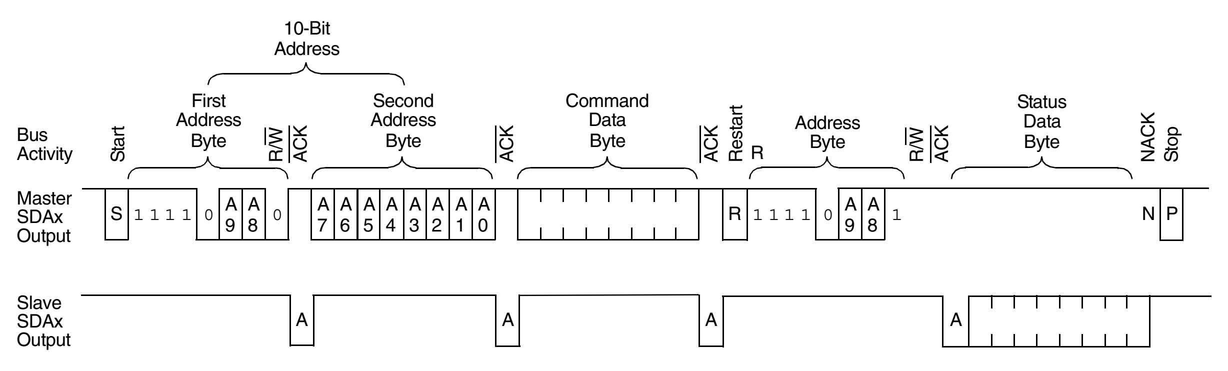


Figure 24-20: A Typical Slave I^2C™ Message: Multiprocessor Command/Status

After a Start condition, the slave module will receive and check the device address. The slave may 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 software that its device is selected. Based on the R/$\overline{W}$ bit sent by the master, the slave will either receive or transmit data. If the slave is to receive data, the slave module automatically generates the Acknowledge ($\overline{ACK}$), loads the I2CxRCV register with the received value currently in the I2CxRSR register and notifies the software through an interrupt. If the slave is to transmit data, the software must load the I2CxTRN register.

24.7.1 Sampling Receive Data

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

24.7.2 Detecting Start and Stop Conditions

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

24.7.3 Detecting the Address

Once the module has been enabled, the slave module waits for a Start condition to occur. After a Start, depending on the A10M bit (I2CxCON<10>), the slave will attempt to detect a 7-bit or 10-bit address. The slave module will compare 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/$\overline{W}$ bit that specifies the direction of data transfer after the address. If R/$\overline{W}$ = `0`, a write is specified and the slave will receive data from the master. If R/$\overline{W}$ = `1`, a read is specified and the slave will send data to the master. The 10-bit address contains an R/$\overline{W}$ bit, however, by definition, it is always R/$\overline{W}$ = `0` because the slave must receive the second byte of the 10-bit address.

24.7.3.1 SLAVE ADDRESS MASKING

The I2CxMSK register masks 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 module will respond 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 module will Acknowledge addresses ‘`0000000`’ and ’`0100000`’ as valid.

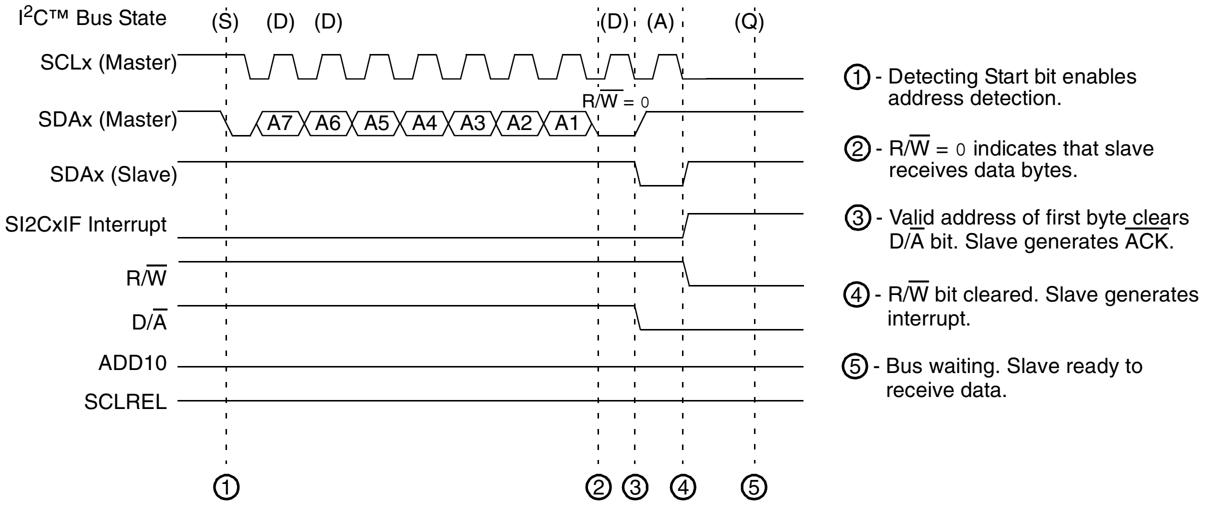
To enable address masking, the IPMI (Intelligent Peripheral Management Interface) must be disabled by clearing the IPMIEN bit (I2CxCON<11>).

24.7.3.2 7-BIT ADDRESS AND SLAVE WRITE

Following the Start condition, the module shifts 8 bits into the I2CxRSR register (see Figure 24-21). The value of register I2CxRSR<7:1> is evaluated against that of the I2CxADD<6:0> and I2CxMSK<6:0> registers, on the falling edge of the eighth clock (SCLx). If the address is valid (i.e., an exact match between unmasked bit positions), the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D/$\overline{\text{A}}$ and R/$\overline{\text{W}}$ bits are cleared.
3. The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock.
4. The module will wait for the master to send data.

Figure 24-21: Slave Write 7-Bit Address Detection Timing Diagram



① - Detecting Start bit enables address detection.

② - R/$\overline{\text{W}}$ = `0` indicates that slave receives data bytes.

③ - Valid address of first byte clears D/$\overline{\text{A}}$ bit. Slave generates $\overline{\text{ACK}}$.

④ - R/$\overline{\text{W}}$ bit cleared. Slave generates interrupt.

⑤ - Bus waiting. Slave ready to receive data.

24.7.3.3 7-BIT ADDRESS AND SLAVE READ

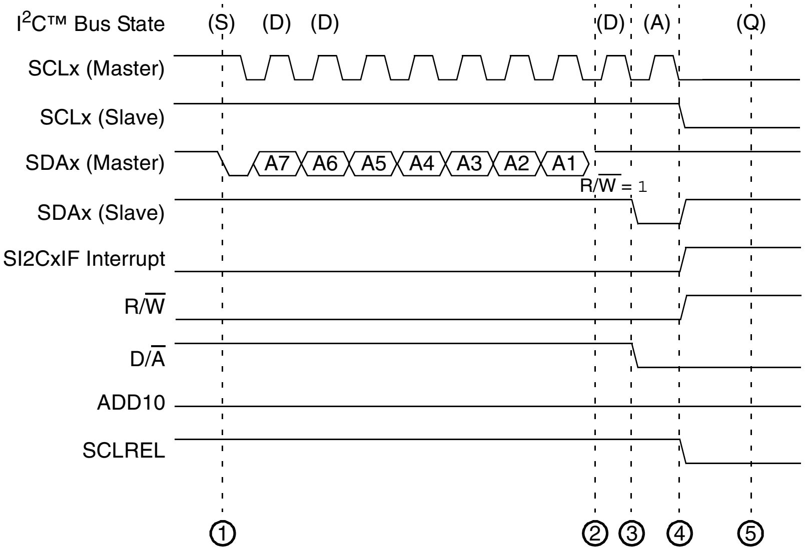
When a slave read is specified by having R/$\overline{\text{W}}$ = `1` in a 7-bit address byte, the process of detecting the device address is similar to that for a slave write (see Figure 24-22). If the addresses match, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D/$\overline{\text{A}}$ bit is cleared and the R/$\overline{\text{W}}$ bit is set.
3. The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock.

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

Note: SCLREL will automatically clear after detection of a slave read address regardless of the state of the STREN bit.

Figure 24-22: Slave Read 7-Bit Address Detection Timing Diagram



① - Detecting Start bit enables address detection.

② - $R/\overline{W}$ = `1` indicates that slave sends data bytes.

③ - Valid address of first byte clears $D/\overline{A}$ bit. Slave generates $\overline{\text{ACK}}$.

④ - $R/\overline{W}$ bit set. Slave generates interrupt. SCLREL cleared. Slave pulls SCLx low while SCLREL = `0`.

⑤ - Bus waiting. Slave prepares to send data.

24.7.3.4 10-BIT ADDRESSING MODE

In 10-Bit Addressing mode, the slave must receive two device address bytes (see Figure 24-23). The five Most Significant bits (MSbs) of the first address byte specify a 10-bit address. The $R/\overline{W}$ 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 two MSbs of the address.

The I2CxMSK register can mask any bit position in a 10-bit address. The two MSbs of I2CxMSK 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 are evaluated against the value of the I2CxADD<9:8> and I2CxMSK<9:8> bits, while the value of the I2CxRSR<7:3> bits are compared to '`11110`'. Address evaluation occurs on the falling edge of the eighth clock (SCLx). For the address to be valid, I2CxRSR<7:3> must equal '`11110`', while I2CxRSR<2:1> must exactly match any unmasked bits in I2CxADD<9:8>. (If both bits are masked, a match is not needed.) If the address is valid, the following events occur:

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

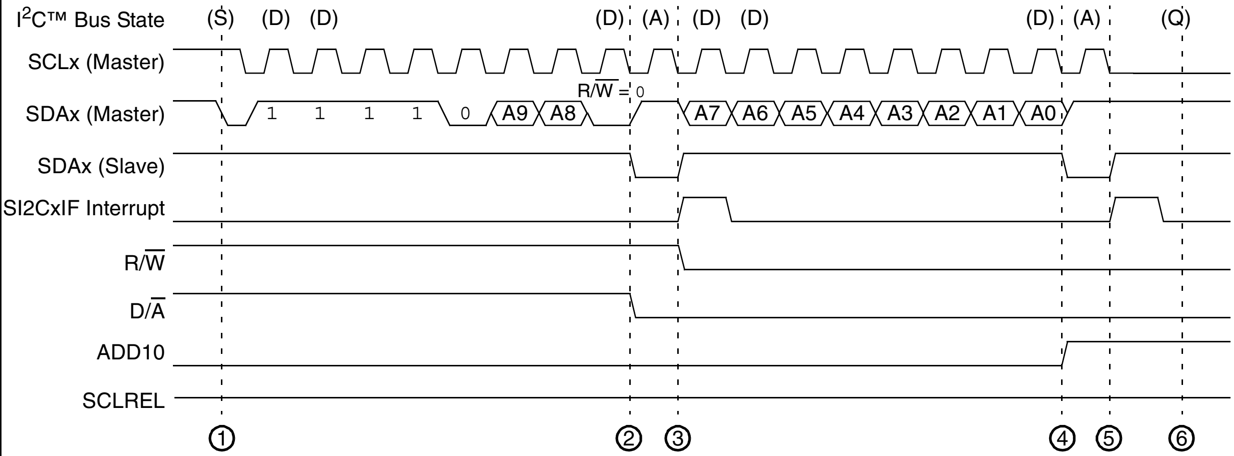
The module does generate an interrupt after the reception of the first byte of a 10-bit address, however, this interrupt is of little use.

The module will continue to receive the second byte into I2CxRSR. This time, the I2CxRSR<7:0> bits are evaluated against the the I2CADD<7:0> and I2CxMSK<7:0> bits. If the lower byte of the addresses is valid as previously described, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The ADD10 bit is set.
3. The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock.
4. 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 module only matches the first 7-bit address, '`11110 A9 A8 0`'.

Figure 24-23: 10-Bit Address Detection Timing Diagram



① - Detecting Start bit enables address detection.

② - Address match of first byte clears D/$\overline{\text{A}}$ bit and causes slave logic to generate $\overline{\text{ACK}}$.

③ - Reception of first byte clears R/$\overline{\text{W}}$ bit. Slave logic generates interrupt.

④ - Address match of first and second byte sets ADD10 and causes slave logic to generate $\overline{\text{ACK}}$.

⑤ - Reception of second byte completes 10-bit address. Slave logic generates interrupt.

⑤ - Bus waiting. Slave ready to receive data.

24.7.3.5 GENERAL CALL OPERATION

The addressing procedure for the I²C 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 enabled devices should respond with an Acknowledge. The general call address is one of eight addresses reserved for specific purposes by the I²C protocol. It consists of all '`0`'s with R/$\overline{\text{W}}$ = `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>), is set (see Figure 24-24). Following a Start bit detect, eight bits are shifted into the I2CxRSR and the address is compared against the I2CxADD, and is also compared to the general call address.

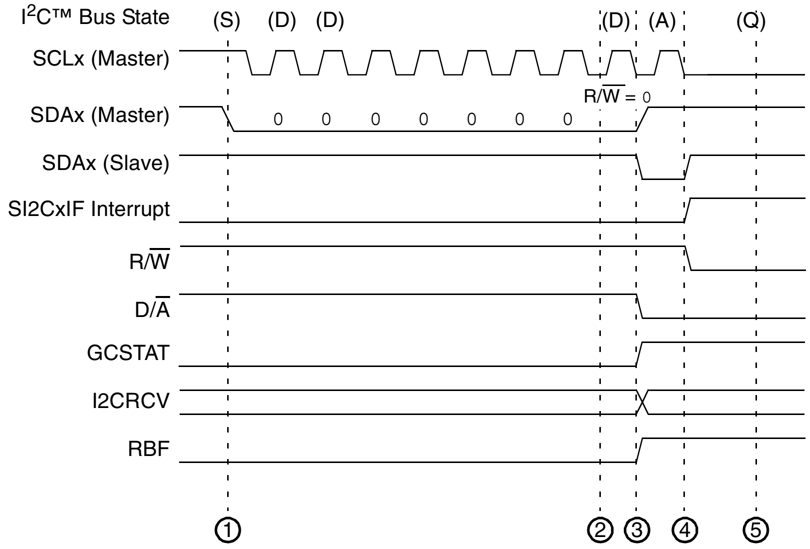
If the general call address matches, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. Slave module will set the GCSTAT bit (I2CxSTAT<9>).
3. The D/$\overline{\text{A}}$ and R/$\overline{\text{W}}$ bits are cleared.
4. The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock.
5. The I2CxRSR is transferred to the I2CxRCV and the RBF flag bit is set (during the eighth bit).
6. The module will wait 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 bit to determine if the device address was device specific or a general call address.

Note that general call addresses are 7-bit addresses. If configuring the slave module for 10-bit addresses and the A10M and GCEN bits are set, the slave module will continue to detect the 7-bit general call address.

Figure 24-24: General Call Address Detection Timing Diagram (GCEN = `1`)



① - Detecting Start bit enables address detection.

② - All '`0`'s and $R/\overline{W}$ = `0` indicates general call.

③ - Valid address clears $D/\overline{A}$ bit and sets GCSTAT.
Slave generates $\overline{ACK}$.
Address loaded into I2CxRCV.

④ - $R/\overline{W}$ bit cleared. Slave generates interrupt.

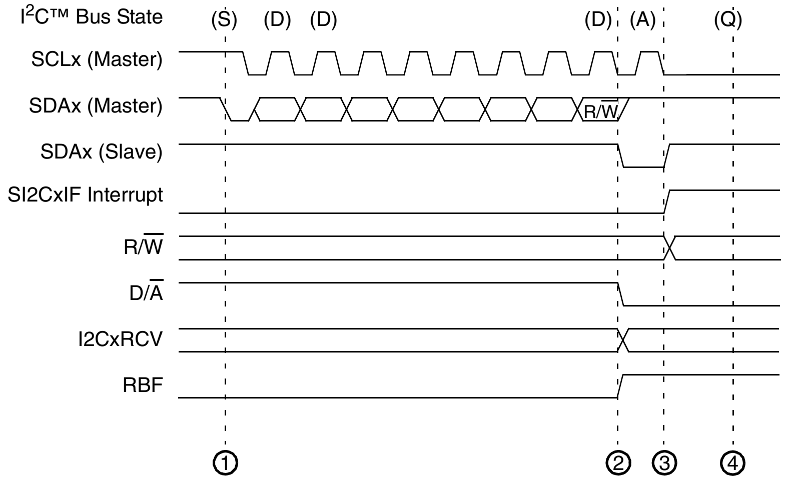
⑤ - Bus waiting. Slave ready to receive data.

24.7.3.6 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 (Intelligent Peripheral Management Interface) bus uses I^2C nodes as message repeaters in a distributed network. To allow a node to repeat all messages, the slave module must accept all messages, regardless of the device address.

To enable IPMI mode, set the IPMIEN bit (I2CxCON<11>); see Figure 24-25. Regardless of the state of the A10M and GCEN bits, or the value loaded in the I2CxADD register, all addresses will be accepted.

Figure 24-25: 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 $D/\overline{A}$ bit.
Slave generates $\overline{ACK}$.
Address loaded into I2CxRCV.

③ - $R/\overline{W}$ bit set/clear. Slave generates interrupt.

④ - Bus waiting.

24.7.3.7 WHEN AN ADDRESS IS INVALID

If a 7-bit address does not match the contents of I2CxADD<6:0>, the slave module 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 does not match the contents of I2CxADD<9:8>, the slave module 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 I2CxADD<9:8>, however, the second byte of the 10-bit address does not match I2CxADD<7:0>, the slave module will return to an Idle state and ignore all bus activity until after the Stop condition.

24.7.3.8 ADDRESSES RESERVED FROM MASKING

Even when enabled, there are several addresses that are excluded in hardware from masking. For these addresses, an Acknowledge will not be issued independent of the mask setting. These addresses are listed in Table 24-3.

Table 24-3: Reserved I^2C Bus Addresses(1)

7-Bit Address Mode:		
Slave Address	**R/$\overline{\text{W}}$ Bit**	**Description**
`0000 000`	`0`	General Call Address(1)
`0000 000`	`1`	Start Byte
`0000 001`	`x`	CBUS 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: Match on this address can only occur as the upper byte in the 10-Bit Addressing mode.

24.7.4 Receiving Data From a Master Device

When the R/$\overline{\text{W}}$ bit of the device address byte is zero and an address match occurs, the R/$\overline{\text{W}}$ bit (I2CxSTAT<2>) is cleared. The slave module enters a state waiting for data 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 module.

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

1. The module begins to generate an $\overline{\text{ACK}}$ or NACK.
2. The RBF bit is set to indicate received data.
3. The I2CxRSR byte is transferred to the I2CxRCV register for access by the software.
4. The D/$\overline{\text{A}}$ bit is set.
5. A slave interrupt is generated. Software may check the status of the I2CxSTAT register to determine the cause of the event and then clear the SI2CxIF flag.
6. The module will wait for the next data byte.

24.7.4.1 ACKNOWLEDGE GENERATION

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

1. The buffer full bit, RBF (I2CxSTAT<1>), was set before the transfer was received.
2. The overflow bit, I2COV (I2CxSTAT<6>), was set before the transfer was received.

Table 24-4 shows what happens when a data transfer byte is received, given the status of the RBF and I2COV bits. If the RBF bit is already set when the slave module attempts to transfer to the I2CxRCV, the transfer does not occur, but the interrupt is generated and the I2COV bit is set. If both the RBF and I2COV bits are set, the slave module acts similarly. The shaded cells show the condition where software did not properly clear the overflow condition.

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

Table 24-4: 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
RBF	I2COV					
0	0	Yes	Yes	Yes	Yes	No change
1	0	No	No	Yes	No change	Yes
1	1	No	No	Yes	No change	Yes
0	1	Yes	No	Yes	Yes	No change

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

24.7.4.2 WAIT STATES DURING SLAVE RECEPTIONS

When the slave module receives a data byte, the master can potentially begin sending the next byte immediately. This allows the software, controlling the slave module, nine 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>) enables a bus wait to occur on slave receptions. When STREN = `1` at the falling edge of the 9th SCLx clock of a received byte, the slave module clears the SCLREL bit. Clearing the SCLREL bit causes the slave module to pull the SCLx line low, initiating a wait. The SCLx clock of the master and slave will synchronize, as shown in **Section 24.6.2 "Master Clock Synchronization"**.

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

24.7.4.3 EXAMPLE MESSAGES OF SLAVE RECEPTION

Receiving a slave message is a rather automatic process. The 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 software to expect a message. On receive data, as each data byte transfers to the I2CxRCV register, an interrupt notifies the software to unload the buffer.

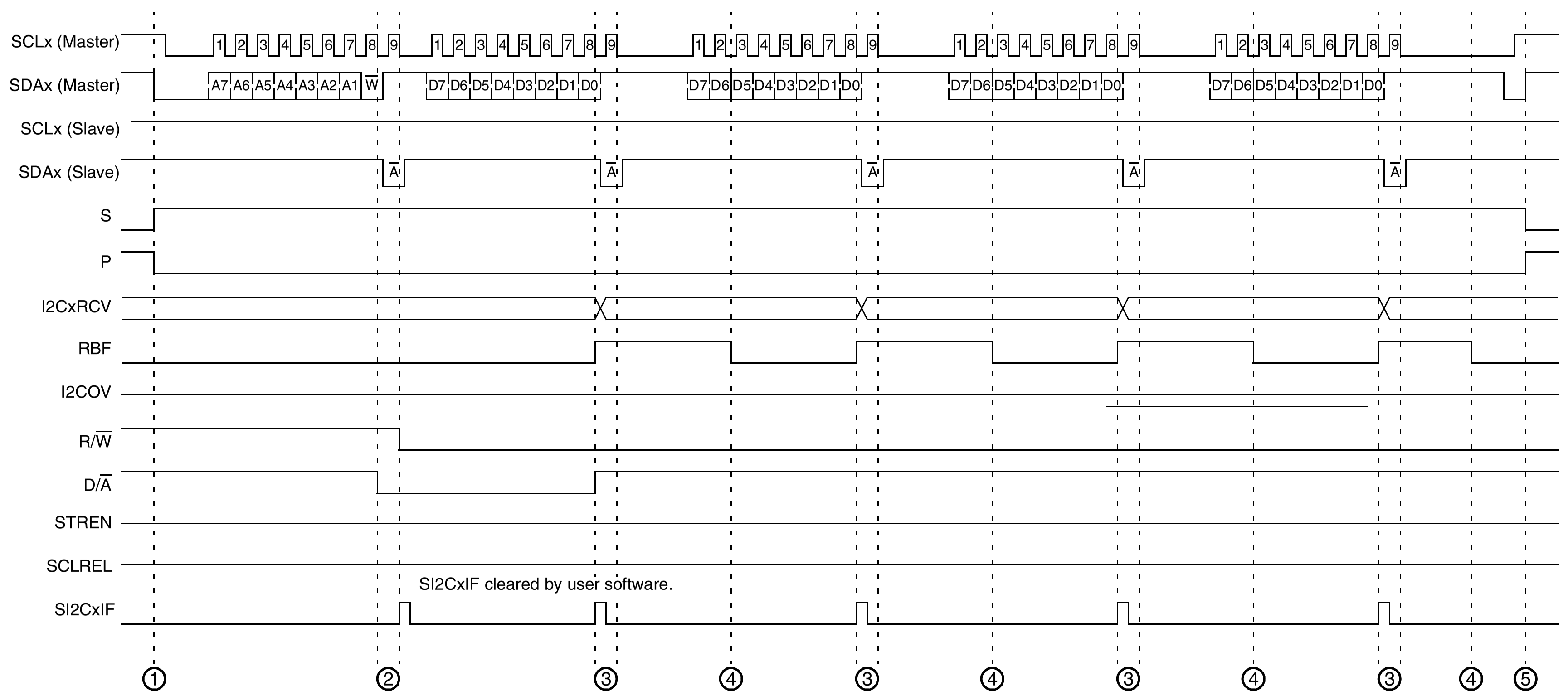
Figure 24-26 shows a simple receive message. Being a 7-bit address message, only one interrupt occurs for the address bytes. Then, interrupts occur for each of four data bytes. At an interrupt, the software may monitor the RBF, D/$\overline{\text{A}}$ and R/$\overline{\text{W}}$ bits to determine the condition of the byte received.

Figure 24-27 shows a similar message using a 10-bit address. In this case, two bytes are required for the address.

Figure 24-28 shows a case where the 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 bit indicates that the buffer has overrun. The I2CxRCV 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 software finally reads the buffer. Reading the buffer will clear the RBF bit, however, the I2COV bit remains set. The software must clear the I2COV bit. The next received byte will be moved to the I2CxRCV buffer and the module will respond with an $\overline{\text{ACK}}$.

Figure 24-29 highlights clock stretching while receiving data. Note in the previous examples, STREN = `0`, which disables clock stretching on receive messages. In this example, the software sets STREN to enable clock stretching. When STREN = `1`, the module will automatically clock stretch after each received data byte, allowing the software more time to move the data from the buffer. Note that if RBF = `1` at the falling edge of the 9th clock, the module will automatically clear the SCLREL bit and pull the SCLx bus line low. As shown with the second received data byte, if the software can read the buffer and clear the RBF before the falling edge of the 9th clock, the clock stretching will not occur. The software can also suspend the bus at any time. By clearing the SCLREL bit, the module will pull the SCLx line low after it detects the bus SCLx low. The SCLx line will remain low, suspending transactions on the bus until the SCLREL bit is set.

Figure 24-26: Slave Message (Write Data to Slave: 7-Bit Address; Address Matches; A10M = 0; GCEN = 0; IPMIEN = 0)



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

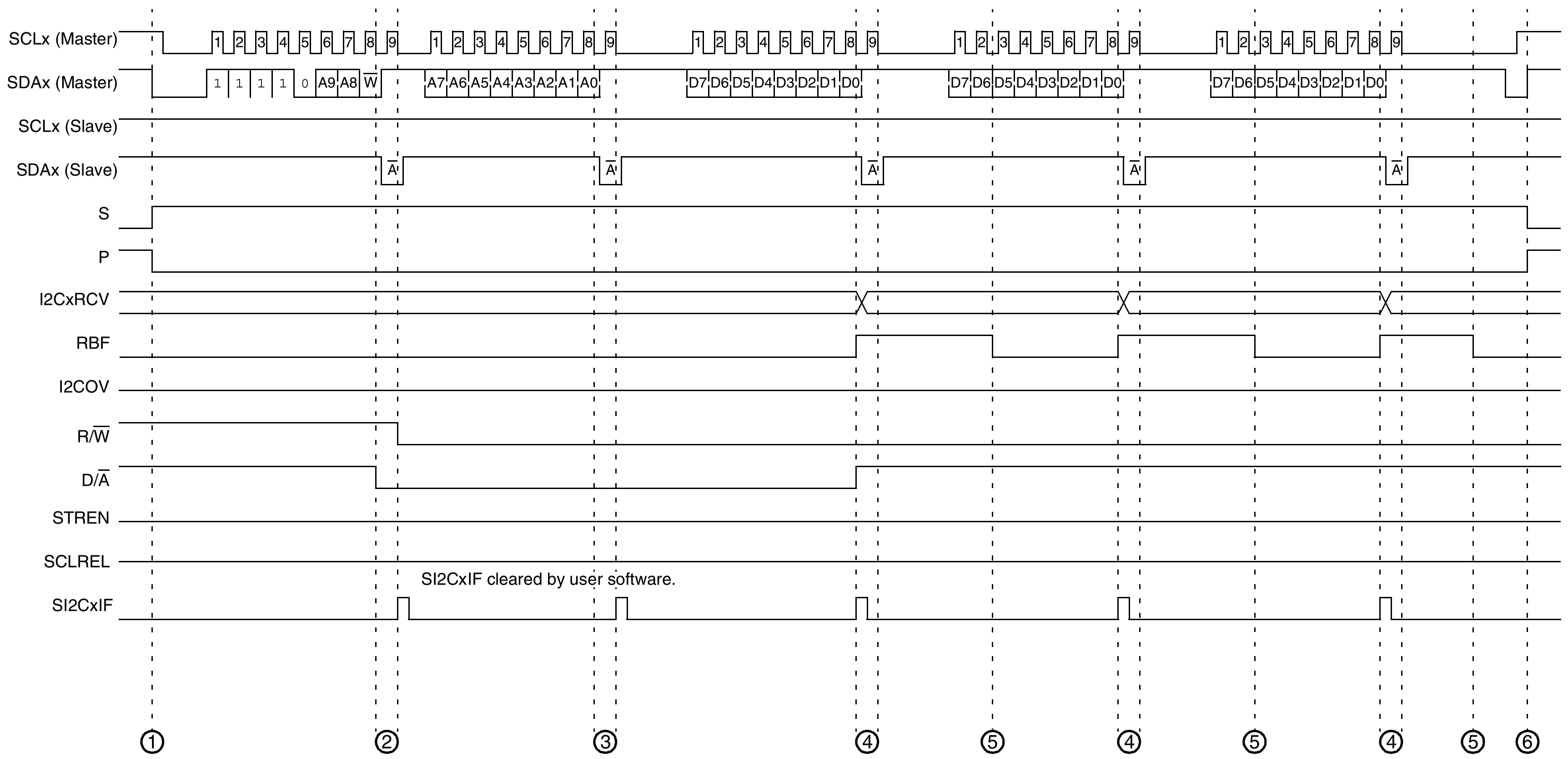
② - Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt. Address byte is moved to I2CxRCV register and must be read by user software to prevent buffer overflow.

③ - Next received byte is message data. Byte moved to I2CxRCV register, sets RBF. Slave generates interrupt. Slave Acknowledges reception.

④ - Software reads I2CxRCV register. RBF bit clears.

⑤ - Slave recognizes Stop event, S and P bits set/clear accordingly.

Figure 24-27: Slave Message (Write Data to Slave: 10-Bit Address; Address Matches; A10M = 1; GCEN = 0; IPMIEN = 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 not moved to I2CxRCV register.

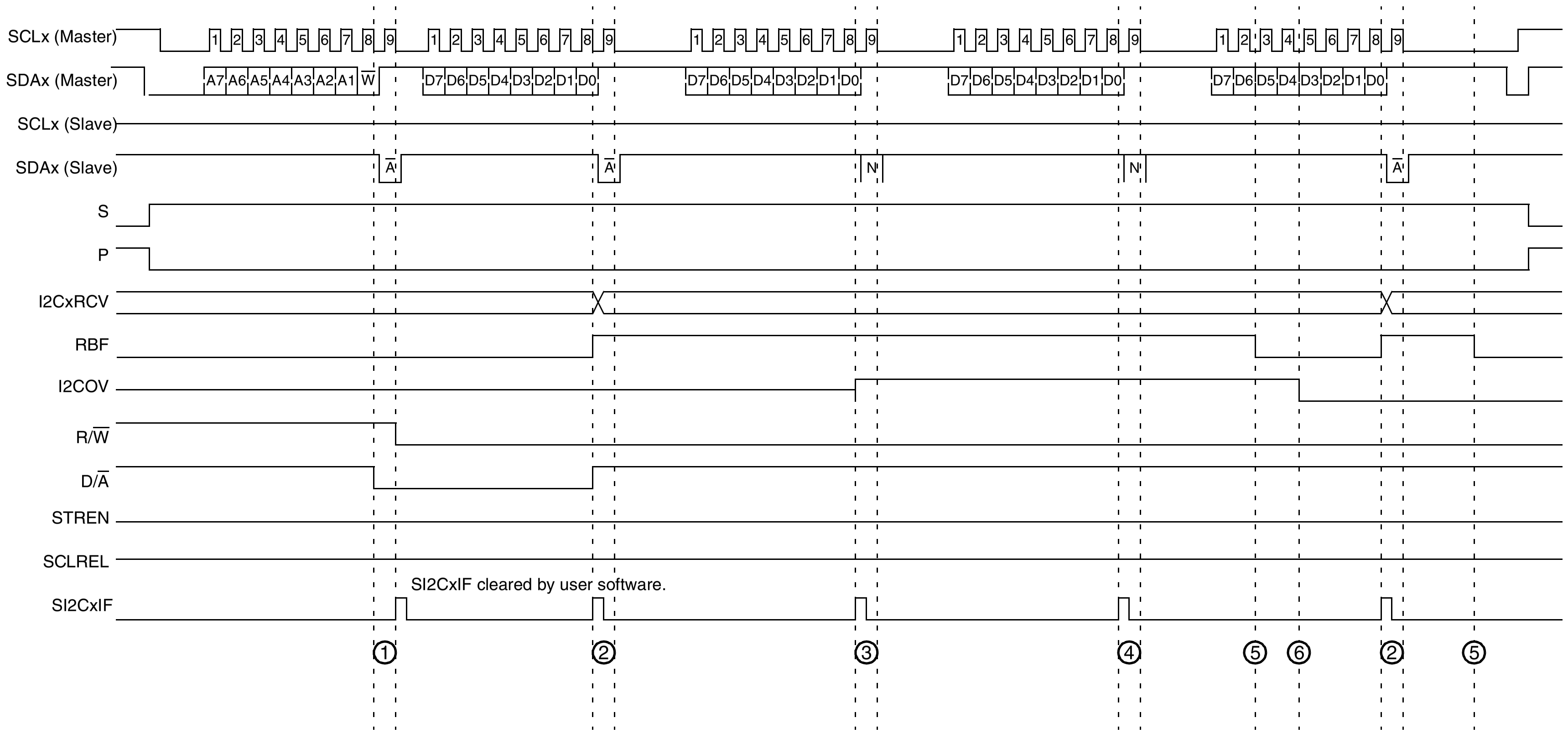
③ - Slave receives address byte. Low-order address matches. Slave Acknowledges and generates interrupt. Address byte not moved to I2CxRCV register.

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

⑤ - Software reads I2CxRCV register. RBF bit clears.

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

Figure 24-28: Slave Message (Write Data to Slave: 7-Bit Address; Buffer Overrun; A10M = 0; GCEN = 0; IPMIEN = 0)



① - Slave receives address byte. Address matches. Slave generates interrupt. Address byte not moved to I2CxRCV register.

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

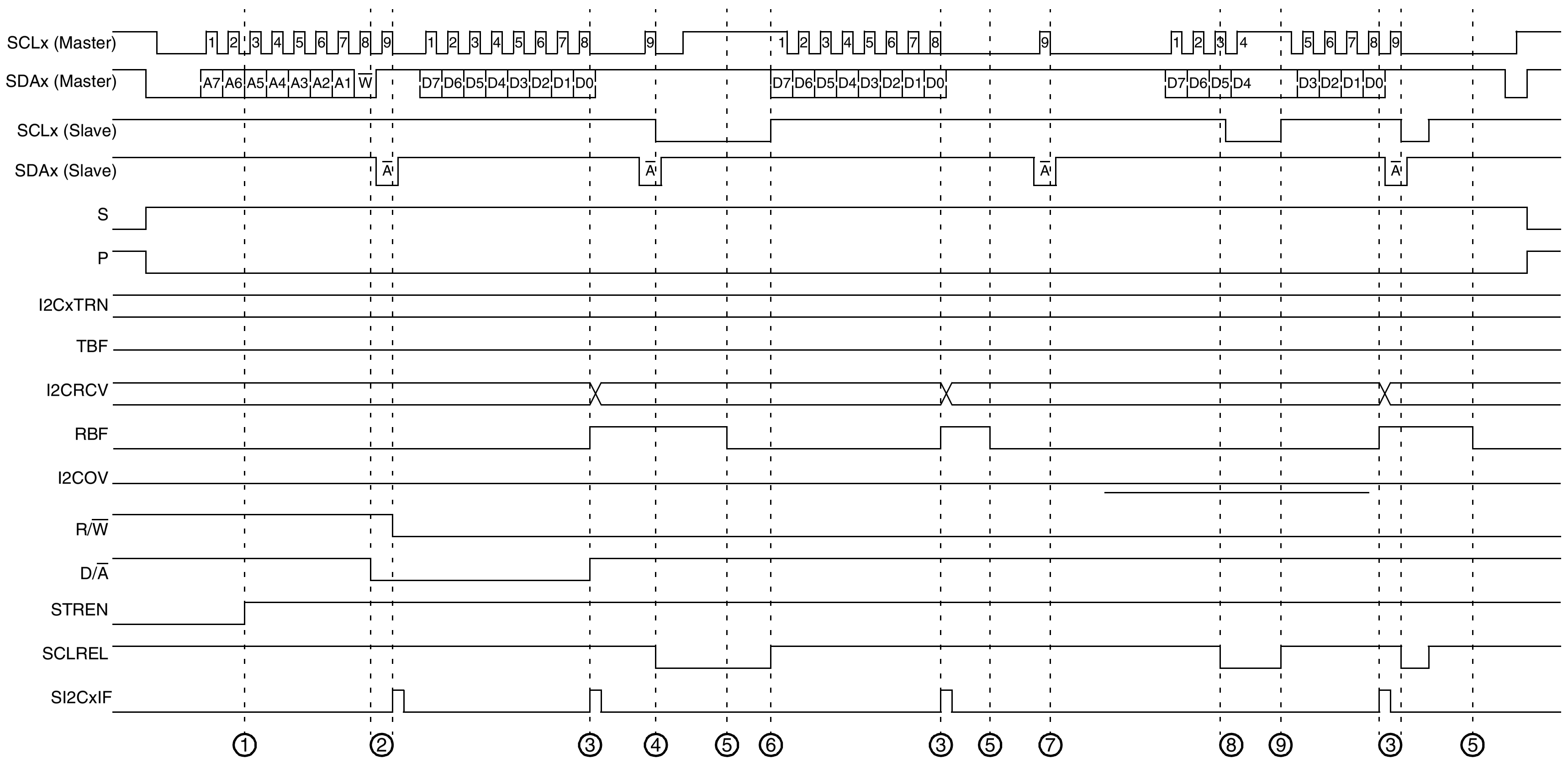
③ - Next byte received before I2CxRCV read by software. I2CxRCV register unchanged. I2COV overflow bit set. Slave generates interrupt. Slave sends NACK for reception.

④ - Next byte also received before I2CxRCV read by software. I2CxRCV register unchanged. Slave generates interrupt. Slave sends NACK for reception.

⑥ - Software reads I2CxRCV register. RBF bit clears.

⑦ - Software clears I2COV bit.

Figure 24-29: Slave Message (Write Data to Slave: 7-Bit Address; Clock Stretching Enabled; A10M = 0; GCEN = 0; IPMIEN = 0)



① - Software sets the STREN bit to enable clock stretching.

② - Slave receives address byte.

③ - Next received byte is message data. Byte moved to I2CxRCV register, sets RBF.

④ - Because RBF = 1 at 9th clock, automatic clock stretch begins.
Slave clears SCLREL bit. Slave pulls SCLx line low to stretch clock.

⑤ - Software reads I2CxRCV register. RBF bit clears.

⑥ - Software sets SCLREL bit to release clock.

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

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

⑨ - Software may set SCLREL to release a clock hold.

24.7.5 Sending Data to a Master Device

When the R/$\overline{\mathrm{W}}$ bit of the incoming device address byte is '`1`' and an address match occurs, the R/$\overline{\mathrm{W}}$ 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 module.

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

The slave module sets the TBF bit. 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 eight bits have been shifted out, the TBF bit will be cleared.

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

If the SDAx line is low, indicating an Acknowledge ($\overline{\mathrm{ACK}}$), the master is expecting more data and the message is not complete. The module generates a slave interrupt to signal more data is requested.

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

If the SDAx line is high, indicating a Not Acknowledge (NACK), then the data transfer is complete. The slave module resets and does not generate an interrupt. The slave module will wait for detection of the next Start bit.

24.7.5.1 WAIT STATES DURING SLAVE TRANSMISSIONS

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

The automatic wait occurs at the falling edge of the 9th SCLx clock of a valid device address byte, or transmitted byte, Acknowledged by the master, indicating expectation of more transmit data.

The slave module clears the SCLREL bit. Clearing the SCLREL bit causes the slave module to pull the SCLx line low, initiating a wait. The SCLx clock of the master and slave will synchronize as shown in **Section 24.6.2 "Master Clock Synchronization"**.

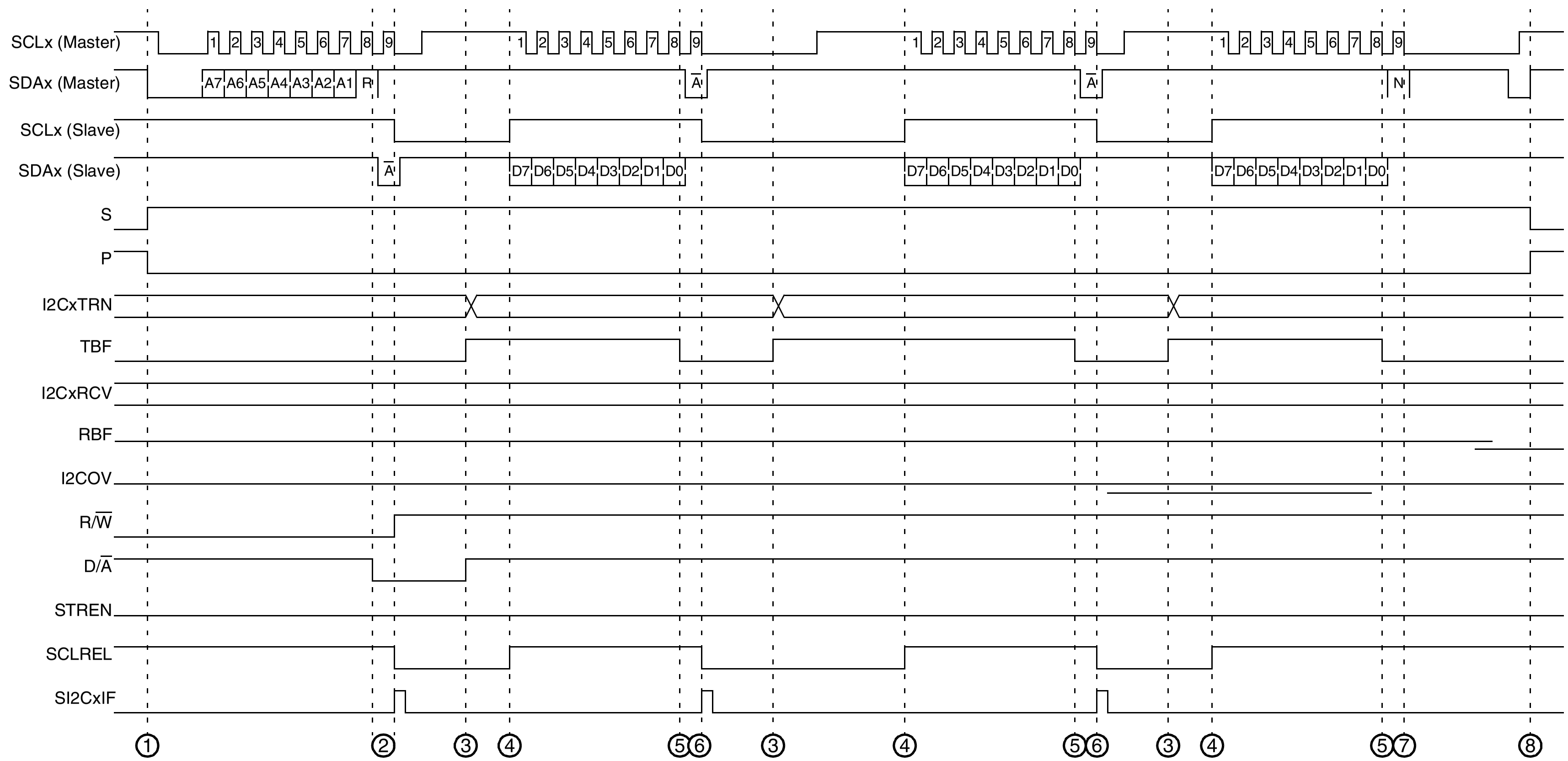
When the software loads the I2CxTRN and is ready to resume transmission, the software sets SCLREL. This causes the slave module to release the SCLx line and the master resumes clocking.

24.7.5.2 EXAMPLE MESSAGES OF SLAVE TRANSMISSION

Slave transmissions for 7-bit address messages are shown in Figure 24-30. When the address matches and the R/$\overline{\mathrm{W}}$ bit of the address indicates a slave transmission, the module will automatically initiate clock stretching by clearing the SCLREL bit and generates an interrupt to indicate a response byte is required. The software will write the response byte into the I2CxTRN register. As the transmission completes, the master will respond with an Acknowledge. If the master replies with an $\overline{\mathrm{ACK}}$, the master expects more data and the module will again clear the SCLREL bit and generate another interrupt. If the master responds with a NACK, no more data is required and the module will not stretch the clock nor generate an interrupt.

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/$\overline{\mathrm{W}}$ bit in the first byte of the address specifies a write. To change the message to a read, the master will send a Repeated Start and repeat the first byte of the address with the R/$\overline{\mathrm{W}}$ bit specifying a read. At this point, the slave transmission begins as shown in Figure 24-31.

Figure 24-30: 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 not moved to I2CxRCV register. R/$\overline{\text{W}}$ = `1` to indicate read from slave. SCLREL = `0` to suspend master clock.

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

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

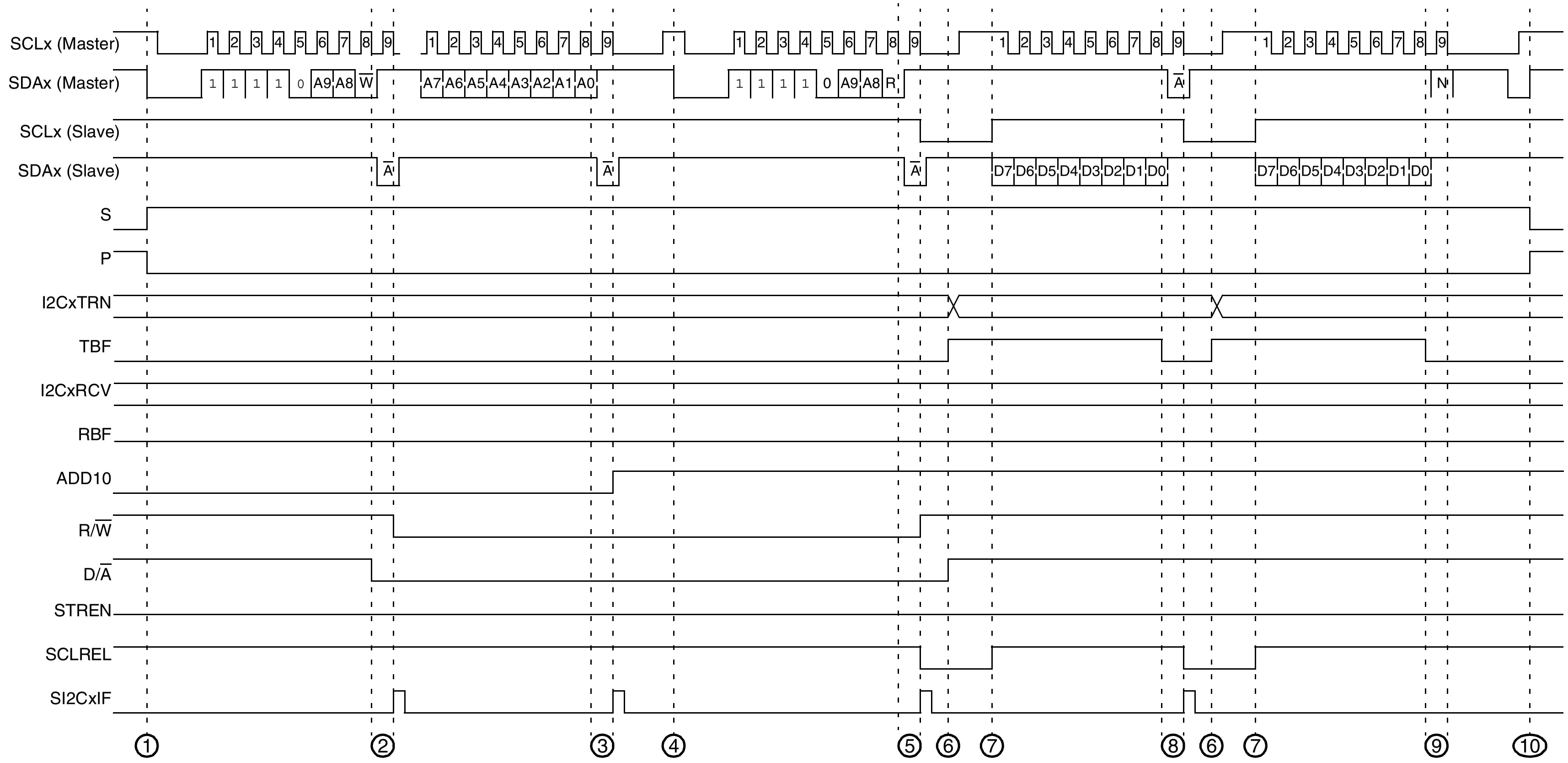
⑤ - After last bit, module clears TBF bit indicating buffer is available for next byte.

⑥ - At end of 9th clock, if master sent $\overline{\text{ACK}}$, module clears SCLREL to suspend clock. Slave generates interrupt.

⑦ - At end of 9th clock, if master sent NACK, no more data expected. Module does not suspend clock and will generate an interrupt.

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

Figure 24-31: 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.

③ - Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt.

④ - Master sends a Repeated Start to redirect the message.

⑤ - Slave receives resend of first address byte. Read indicated. Slave suspends clock.

⑥ - Software writes I2CxTRN with response data.

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

⑧ - At end of 9th clock, if master sent $\overline{\text{ACK}}$, module clears SCLREL to suspend clock. Slave generates interrupt.

⑨ - At end of 9th clock, if master sent NACK, no more data expected. Module does not suspend clock or generate interrupt.

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

24.8 CONNECTION CONSIDERATIONS FOR I^{2}C BUS

By definition of the I^{2}C bus being a wired AND bus connection, pull-up resistors on the bus are required, shown as R_P in Figure 24-32. Series resistors, shown as R_S, are optional and used to improve ESD susceptibility. The values of 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^{2}C or 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} of 6.6 mA at V_{OLMAX} = 0.4V, for the device output stage. For example, with a supply voltage of V_{DD} = 3V + 10%:

Equation 24-2:

$$R_{PMIN} = (V_{DDMAX} - V_{OLMAX})/I_{OL} = (3.3V - 0.6V)/8.5\ mA = 439\Omega$$

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}) must be less than:

Equation 24-3:

$$-t_R/(C_B * (\ln(1 - (V_{DDMAX} - V_{ILMAX})) = -300\ ns/(100\ pF * \ln(1 - (0.99 - 3.3))),\ or\ 2.5\ k\Omega$$

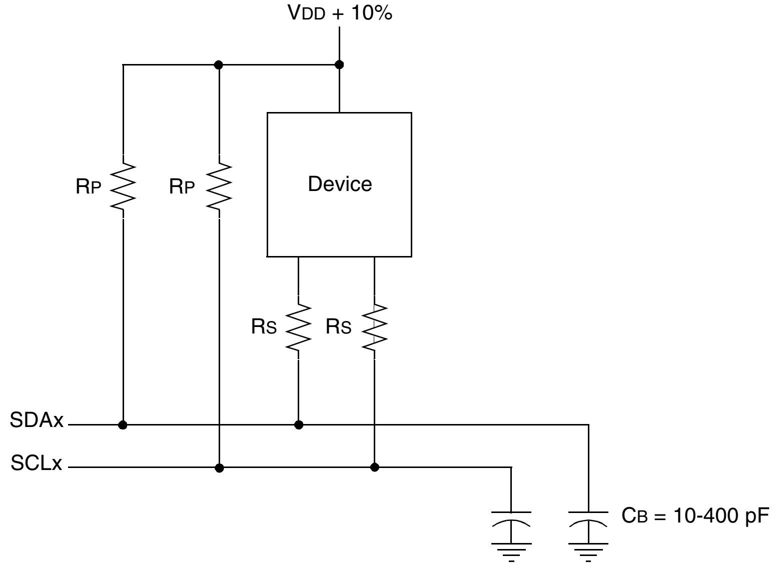
The maximum value for R_S is determined by the desired noise margin for the low level. R_S cannot drop enough voltage to make the device V_{OL} plus voltage across R_S more than the maximum V_{IL}. Mathematically:

Equation 24-4:

$$R_{SMAX} = (V_{ILMAX} - V_{OLMAX})/I_{OLMAX} = (0.3\ V_{DD} - 0.4)/6.6\ mA = 89\Omega$$

The SCLx clock input must have a minimum high and low time for proper operation. The high and low times of the I^{2}C specification, as well as the requirements of the I^{2}C module, are shown in **Section 24.13 "Electrical Specifications"**.

Figure 24-32: Sample Device Configuration for I^{2}C™ Bus



Note: I^{2}C™ devices with input levels related to V_{DD} must have one common supply line to which the pull-up resistor is also connected.

24.8.1 Integrated Signal Conditioning

The SCLx and SDAx pins have an input glitch filter. The I^{2}C bus requires this filter in both the 100 kHz and 400 kHz systems.

When operating on a 400 kHz bus, the I^{2}C 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>) is cleared, the slew rate control is active. For other bus speeds, the I^{2}C specification does not require slew rate control and DISSLW should be set.

Some system implementations of I^{2}C busses require different input levels for VILMAX and VIHMIN. In a normal I^{2}C system, VILMAX is 0.3 VDD; VIHMIN is 0.7 VDD. In contrast, in an SMBus (System Management Bus) system, VILMAX is set at 0.8V, while VIHMIN is set at 2.1V.

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

24.9 MODULE OPERATION DURING `PWRSAV` INSTRUCTION

24.9.1 Sleep Mode in Slave Mode

The module will continue full operation if Sleep mode is entered while the module is configured in Slave mode. Since all bit shifting is done with reference to the external SCL signal, all data transmit and receive operations will continue. The module is then required to assert an asynchronous interrupt as needed.

24.9.2 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, then the transmission is aborted. Similarly, if Sleep occurs in the middle of a master reception, then the reception is aborted. The transmitter and receiver will stop at Sleep when in Master mode; the transmitter and receiver abort functions to ensure that the transmitter or receiver does not continue with a partially completed function at wake-up. Fundamentally, if the I^{2}C module enters Sleep configured in Master mode, the module will reset itself and its state machines during Sleep. This results in the IWCOL, I2COV and BCL bits getting reset. 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 Sleep entry with I^{2}C operation to avoid aborted transmissions.

24.9.3 When the Device Enters Idle Mode

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

24.10 PERIPHERAL MODULE DISABLE (PMD) REGISTER

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

24.11 EFFECTS OF A RESET

A Reset disables the I^{2}C module and terminates any active or pending message activity. See the register definitions of I2CxCON and I2CxSTAT for the Reset conditions of those registers.

Note: In this discussion, ‘Idle’ refers to the CPU power-saving state. The lower case ‘idle’ refers to the time when the I^{2}C module is not transferring data on the bus.

24.12 REGISTER MAPS

A summary of the registers associated with the PIC24F I^{2}C module is provided in Table 24-5.

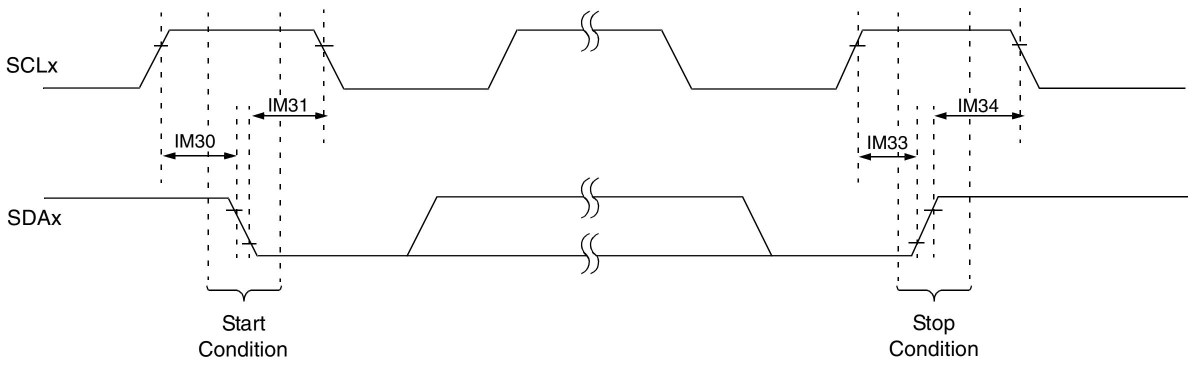
Table 24-5: I2Cx Register Map

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	—	—	—	—	—	—	—	—	Receive Register								`0000`
I2CxTRN	—	—	—	—	—	—	—	—	Transmit Register								`00FF`
I2CxBRG	—	—	—	—	—	—	—	Baud Rate Generator									`0000`
I2CxCON	I2CEN	—	I2CSIDL	SCLREL	IPMIEN	A10M	DISSLW	SMEN	GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	`1000`
I2CxSTAT	ACKSTAT	TRSTAT	—	—	—	BCL	GCSTAT	ADD10	IWCOL	I2COV	$D/\overline{A}$	P	S	$R/\overline{W}$	RBF	TBF	`0000`
I2CxADD	—	—	—	—	—	—	Address Register										`0000`
I2CxMSK	—	—	—	—	—	—	Address Mask										`0000`

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

24.13 ELECTRICAL SPECIFICATIONS

Figure 24-33: I^{2}C™ Bus Start/Stop Bits Timing Characteristics (Master Mode)



Note: Note: Refer to Figure 24-32 for load conditions.

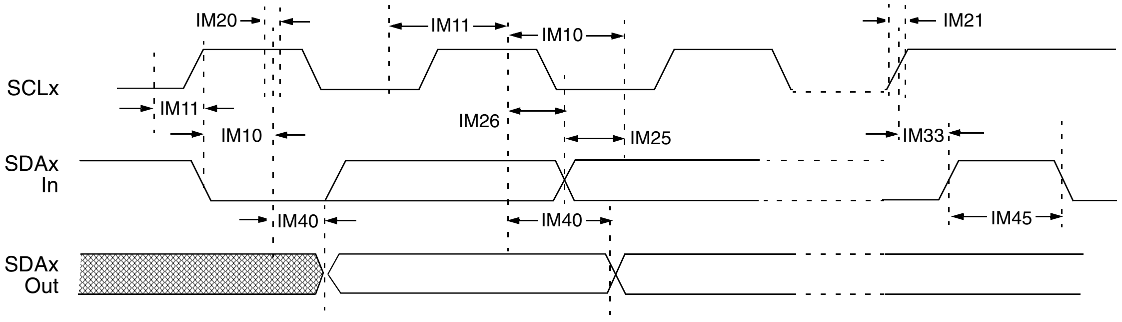
Table 24-6: I^{2}C™ Bus Start/Stop Bit Timing Requirements (Master Mode)

AC CHARACTERISTICS				**Standard Operating Conditions: 2.0V to 3.6V (unless otherwise stated)** Operating temperature -40°C ≤ T_A ≤ +85°C (Industrial)			
Param No.	**Symbol**	**Characteristic**		**Min[(1)]**	**Max**	**Units**	**Conditions**
IM30	$T_{SU:STA}$	Start Condition Setup Time	100 kHz mode	T_{CY}/2 (BRG + 1)	—	µs	Only relevant for Repeated Start condition
			400 kHz mode	T_{CY}/2 (BRG + 1)	—	µs	
			1 MHz mode[(2)]	T_{CY}/2 (BRG + 1)	—	µs	
IM31	$T_{HD:STA}$	Start Condition Hold Time	100 kHz mode	T_{CY}/2 (BRG + 1)	—	µs	After this period, the first clock pulse is generated
			400 kHz mode	T_{CY}/2 (BRG + 1)	—	µs	
			1 MHz mode[(2)]	T_{CY}/2 (BRG + 1)	—	µs	
IM33	$T_{SU:STO}$	Stop Condition Setup Time	100 kHz mode	T_{CY}/2 (BRG + 1)	—	µs	
			400 kHz mode	T_{CY}/2 (BRG + 1)	—	µs	
			1 MHz mode[(2)]	T_{CY}/2 (BRG + 1)	—	µs	
IM34	$T_{HD:STO}$	Stop Condition Hold Time	100 kHz mode	T_{CY}/2 (BRG + 1)	—	ns	
			400 kHz mode	T_{CY}/2 (BRG + 1)	—	ns	
			1 MHz mode[(2)]	T_{CY}/2 (BRG + 1)	—	ns	

Note 1: BRG is the value of the I^{2}C™ Baud Rate Generator. Refer to **Section 24.4.3 "Setting Baud Rate When Operating as a Bus Master"** for details

2: Maximum pin capacitance = 10 pF for all I^{2}C pins (for 1 MHz mode only).

Figure 24-34: I²C™ Bus Data Timing Characteristics (Master Mode)



Note: Refer to Figure 24-32 for load conditions.

Table 24-7: I²C™ Bus Data Timing Requirements (Master Mode)

AC CHARACTERISTICS				**Standard Operating Conditions: 2.0V to 3.6V (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C (Industrial)			
Param No.	**Symbol**	**Characteristic**		**Min(1)**	**Max**	**Units**	**Conditions**
IM10	TLO:SCL	Clock Low Time	100 kHz mode	TCY/2 (BRG + 1)	—	µs	
			400 kHz mode	TCY/2 (BRG + 1)	—	µs	
			1 MHz mode(2)	TCY/2 (BRG + 1)	—	µs	
IM11	THI:SCL	Clock High Time	100 kHz mode	TCY/2 (BRG + 1)	—	µs	
			400 kHz mode	TCY/2 (BRG + 1)	—	µs	
			1 MHz mode(2)	TCY/2 (BRG + 1)	—	µs	
IM20	TF:SCL	SDAx and SCLx Fall Time	100 kHz mode	—	300	ns	CB is specified to be from 10 to 400 pF
			400 kHz mode	20 + 0.1 CB	300	ns	
			1 MHz mode(2)	—	100	ns	
IM21	TR:SCL	SDAx and SCLx Rise Time	100 kHz mode	—	1000	ns	CB is specified to be from 10 to 400 pF
			400 kHz mode	20 + 0.1 CB	300	ns	
			1 MHz mode(2)	—	300	ns	
IM25	TSU:DAT	Data Input Setup Time	100 kHz mode	250	—	ns	
			400 kHz mode	100	—	ns	
			1 MHz mode(2)	TBD	—	ns	
IM26	THD:DAT	Data Input Hold Time	100 kHz mode	0	—	ns	
			400 kHz mode	0	0.9	µs	
			1 MHz mode(2)	TBD	—	ns	
IM40	TAA:SCL	Output Valid From Clock	100 kHz mode	—	3500	ns	
			400 kHz mode	—	1000	ns	
			1 MHz mode(2)	—	—	ns	
IM45	TBF:SDA	Bus Free Time	100 kHz mode	4.7	—	µs	Time the bus must be free before a new transmission can start
			400 kHz mode	1.3	—	µs	
			1 MHz mode(2)	TBD	—	µs	
IM50	CB	Bus Capacitive Loading		—	400	pF	

Legend: TBD = To Be Determined

Note 1: BRG is the value of the I²C Baud Rate Generator. Refer to **Section 24.4.3 "Setting Baud Rate When Operating as a Bus Master"** for details.

2: Maximum pin capacitance = 10 pF for all I²C pins (for 1 MHz mode only).

Figure 24-35: I^2C™ Bus Start/Stop Bits Timing Characteristics (Slave Mode)

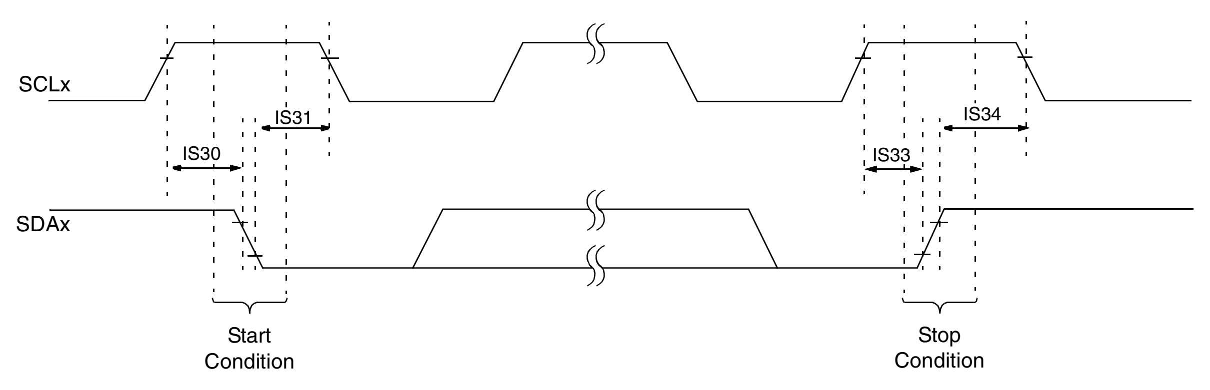


Table 24-8: I^2C™ Bus Start/Stop Bit Timing Requirements (Slave Mode)

AC CHARACTERISTICS				**Standard Operating Conditions: 2.0V to 3.6V (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C (Industrial)			
Param No.	**Symbol**	**Characteristic**		**Min**	**Max**	**Units**	**Conditions**
IS30	TSU:STA	Start Condition Setup Time	100 kHz mode	4.7	—	µs	Only relevant for Repeated Start condition
			400 kHz mode	0.6	—	µs	
			1 MHz mode(1)	0.25	—	µs	
IS31	THD:STA	Start Condition Hold Time	100 kHz mode	4.0	—	µs	After this period, the first clock pulse is generated
			400 kHz mode	0.6	—	µs	
			1 MHz mode(1)	0.25	—	µs	
IS33	TSU:STO	Stop Condition Setup Time	100 kHz mode	4.7	—	µs	
			400 kHz mode	0.6	—	µs	
			1 MHz mode(1)	0.6	—	µs	
IS34	THD:STO	Stop Condition Hold Time	100 kHz mode	4000	—	ns	
			400 kHz mode	600	—	ns	
			1 MHz mode(1)	250	—	ns	

Note 1: Maximum pin capacitance = 10 pF for all I^2C™ pins (for 1 MHz mode only).

Figure 24-36: I^2C™ Bus Data Timing Characteristics (Slave Mode)

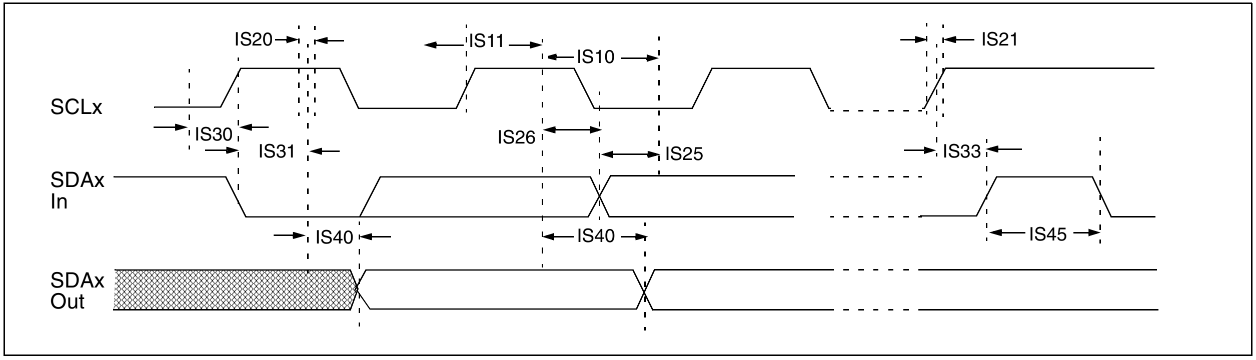


Table 24-9: I^2C™ Bus Data Timing Requirements (Slave Mode)

AC CHARACTERISTICS				Standard Operating Conditions: 2.0V to 3.6V (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C (Industrial)			
Param No.	**Symbol**	**Characteristic**		**Min**	**Max**	**Units**	**Conditions**
IS10	TLO:SCL	Clock Low Time	100 kHz mode	4.7	—	µs	Device must operate at a minimum of 1.5 MHz
			400 kHz mode	1.3	—	µs	Device must operate at a minimum of 10 MHz
			1 MHz mode(1)	0.5	—	µs	
IS11	THI:SCL	Clock High Time	100 kHz mode	4.0	—	µs	Device must operate at a minimum of 1.5 MHz
			400 kHz mode	0.6	—	µs	Device must operate at a minimum of 10 MHz
			1 MHz mode(1)	0.5	—	µs	
IS20	TF:SCL	SDAx and SCLx Fall Time	100 kHz mode	—	300	ns	CB is specified to be from 10 to 400 pF
			400 kHz mode	20 + 0.1 CB	300	ns	
			1 MHz mode(1)	—	100	ns	
IS21	TR:SCL	SDAx and SCLx Rise Time	100 kHz mode	—	1000	ns	CB is specified to be from 10 to 400 pF
			400 kHz mode	20 + 0.1 CB	300	ns	
			1 MHz mode(1)	—	300	ns	
IS25	TSU:DAT	Data Input Setup Time	100 kHz mode	250	—	ns	
			400 kHz mode	100	—	ns	
			1 MHz mode(1)	100	—	ns	
IS26	THD:DAT	Data Input Hold Time	100 kHz mode	0	—	ns	
			400 kHz mode	0	0.9	µs	
			1 MHz mode(1)	0	0.3	µs	
IS40	TAA:SCL	Output Valid From Clock	100 kHz mode	0	3500	ns	
			400 kHz mode	0	1000	ns	
			1 MHz mode(1)	0	350	ns	
IS45	TBF:SDA	Bus Free Time	100 kHz mode	4.7	—	µs	Time the bus must be free before a new transmission can start
			400 kHz mode	1.3	—	µs	
			1 MHz mode(1)	0.5	—	µs	
IS50	CB	Bus Capacitive Loading		—	400	pF	

Note 1: Maximum pin capacitance = 10 pF for all I^2C™ pins (for 1 MHz mode only).

24.14 DESIGN TIPS

Question 1: ***I'm 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 module will receive events from the bus sent by the master.

Question 2: ***I'm operating as a slave and I write data to the I2CxTRN register. Why isn't the data being transmitted?***

Answer: The slave enters an automatic wait when preparing to transmit. Ensure that you set the SCLREL bit to release the I^{2}C 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 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 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^{2}C 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 Interrupt Service Routine for the master, check the BCL bit to ensure that the operation completed without a collision. If a collision is detected, the message must be resent from the beginning.

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

Answer: Look at the S 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^{2}C 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^{2}C interrupt before data is written to I2CxTRN.

24.15 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 PIC24F device family, 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 are:

Title	**Application Note #**
Use of the SSP Module in the I^2C™ Multi-Master Environment	AN578
Using the PICmicro® SSP 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 PIC24F family of devices.

24.16 REVISION HISTORY

Revision A (April 2006)

This is the initial released revision of this document.

NOTES: