Section 19. Inter-Integrated Circuit™ (I²C™)

HIGHLIGHTS

This section of the manual contains the following major topics:

19.1 Introduction ............................................................................................................... 19-2
19.2 I²C Bus Characteristics.............................................................................................. 19-4
19.3 Control and Status Registers ....................................................................................... 19-7
19.4 Enabling I²C Operation ............................................................................................. 19-14
19.5 Communicating as a Master in a Single-Master Environment .................................. 19-16
19.6 Communicating as a Master in a Multi-Master Environment .................................... 19-29
19.7 Communicating as a Slave ....................................................................................... 19-32
19.8 Connection Considerations for I²C Bus ................................................................. 19-47
19.9 Operation in Power-Saving Modes .......................................................................... 19-49
19.10 Peripheral Module Disable (PMDx) Registers ......................................................... 19-49
19.11 Effects of a Reset ..................................................................................................... 19-49
19.12 Constant Current Source ....................................................................................... 19-50
19.13 Register Maps ......................................................................................................... 19-52
19.14 Design Tips ............................................................................................................. 19-53
19.15 Related Application Notes ..................................................................................... 19-54
19.16 Revision History ..................................................................................................... 19-55
19.1 INTRODUCTION

The Inter-Integrated Circuit (I²C) module is a serial interface useful for communicating with other peripheral or microcontroller (MCU) devices. The external peripheral devices may be serial EEPROMs, display drivers, analog-to-digital converters, and so on.

The I²C module can operate as any one of the following in the I²C systems:

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

The I²C module contains independent I²C master logic and I²C slave logic; which generates interrupts based on their events. In the multi-master systems, the user software is simply partitioned into the master controller and the slave controller.

When the I²C 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 the multi-master bus arbitration.

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

The I²C module contains a Baud Rate Generator (BRG). The I²C BRG does not consume other timer resources in the device.

Key features of the I²C module include the following:

• Independent master and slave logic
• Multi-master support which prevents message losses in arbitration
• Detects 7-bit and 10-bit device addresses with configurable address masking in Slave mode
• Detects general call addresses as defined in the I²C 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 Platform Management Interface (IPMI) standard

Figure 19-1 illustrates the I²C module block diagram.
19.2  I²C BUS CHARACTERISTICS

The I²C bus is a two-wire serial interface. Figure 19-2 illustrates the schematic of an I²C connection between a dsPIC33F/PIC24H device and a 24LC256 I²C serial EEPROM, which is a typical example for any I²C interface.

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

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

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

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

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

Figure 19-2: Typical I²C™ Interconnection Block Diagram

19.2.1  Bus Protocol

The following I²C bus protocol has been defined:

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

Accordingly, the bus conditions are defined as illustrated in Figure 19-3.

Figure 19-3: I²C™ Bus Protocol States
19.2.1.1 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.

19.2.1.2 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.

19.2.1.3 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.

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

19.2.1.5 ACKNOWLEDGE (A) OR NOT-ACKNOWLEDGE (N)
All data byte transmissions must be Acknowledged (ACK) or Not-Acknowledged (NACK) by the receiver. The receiver will pull the SDAx line low for an ACK or release the SDAx line for a NACK. The Acknowledge is a 1-bit period using one SCLx clock.

19.2.1.6 WAIT/DATA INVALID (Q)
The data on the line must be changed during the low period of the clock signal. The devices may also stretch the clock low time by asserting a low on the SCLx line, causing a wait on the bus.

19.2.1.7 BUS IDLE (I)
Both data and clock lines remain high after a Stop condition and before a Start condition.

19.2.2 Message Protocol
A typical I²C message is illustrated in Figure 19-4. In this example, the message will read a specified byte from a 24LC256 I²C serial EEPROM. The dsPIC33F/PIC24H device will act as the master and the 24LC256 device will act as the slave.

Figure 19-4 illustrates the data as driven by the master device and the slave device, taking into account that the combined SDAx line is a wired-AND of the master and slave data. The master device controls and sequences the protocol. The slave device will only drive the bus at specifically determined times.

![A Typical I²C Message: Read of Serial EEPROM (Random Address Mode)](Figure 19-4)
19.2.2.1 START MESSAGE

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

19.2.2.2 ADDRESS SLAVE

In Figure 19-4, the first byte is the device address byte, which must be the first part of any \( \text{I}^2\text{C} \) message. It contains a device address and a R/W status bit. Note that R/W = 0 for this first address byte, indicating that the master will be a transmitter and the slave will be a receiver.

19.2.2.3 SLAVE ACKNOWLEDGE

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

19.2.2.4 MASTER TRANSMIT

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

19.2.2.5 REPEATED START

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

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

19.2.2.6 SLAVE REPLY

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

19.2.2.7 MASTER ACKNOWLEDGE

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

19.2.2.8 STOP MESSAGE

The master sends a Stop to terminate the message and return the bus to an Idle state.
19.3 CONTROL AND STATUS REGISTERS

The I2C module has seven registers for operation that are accessible by the user application. All registers are accessible in either byte or word mode. The registers are as follows:

- **I2CxCON**: I2Cx Control Register
  This register allows control of the module’s operation.

- **I2CxSTAT**: I2Cx Status Register
  This register contains status flags indicating the module’s state during operation.

- **I2CxMSK**: I2Cx Slave Mode Address Mask Register
  This register designates which bit positions in the I2CADD register can be ignored, which allows for multiple address support.

- **ISRCCON**: Current Source Control Register
  This register allows control of the current source module.

- **I2CxRCV**: I2Cx Receive Buffer Register
  This is the buffer register from which data bytes can be read. The I2CxRCV register is a read-only register.

- **I2CxTRN**: I2Cx Transmit Register
  This is the transmit register. The bytes are written to this register during a transmit operation. The I2CxTRN register is a read/write register.

- **I2CxADD**: I2Cx Address Register
  This register holds the slave device address.

- **I2CxBRG**: I2Cx Baud Rate Generator Reload Register
  This register holds the BRG reload value for the I2C module BRG.

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

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

If the module receives another complete byte before the user software reads the previous byte from the I2CxRCV register, a receiver overflow occurs and sets the I2COV bit (I2CxSTAT<6>). The byte in the I2CxRSR register is lost. Further reception and clock stretching are disabled until the I2C module sees a Start/Repeated, Start/Stop condition on the bus. If the I2COV flag has been cleared, the reception can proceed normally. If the I2COV flag is not cleared, the module will receive the next byte correctly, but will send a NACK. It will then be unable to receive further bytes or stretch the clock until it detects a Start/Repeated and Start/Stop condition.

The I2CxADD register holds the slave device address. In 10-bit Addressing mode, all bits are relevant. In 7-bit Addressing mode, only the I2CxADD<6:0> bits are relevant. Note that the I2CxADD<6:0> bits correspond to the upper 7 bits in the address byte. The Read/Write bit is not included in the value in this register. The A10M bit (I2CxCON<10>) specifies the expected mode of the slave address. By using the I2CxMSK register with the I2CxADD register in either Slave Addressing mode, one or more bit positions can be removed from the exact address matching, allowing the module in Slave mode to respond to multiple addresses.
### Register 19-1: I2CxCON: \( \text{I}^2\text{Cx} \) Control Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>I2CEN: ( \text{I}^2\text{Cx} ) Enable bit</td>
<td>1 = Enables the ( \text{I}^2\text{Cx} ) module and configures the SDAx and SCLx pins as serial port pins</td>
<td>Enables the ( \text{I}^2\text{Cx} ) module and configures the SDAx and SCLx pins as serial port pins; all ( \text{I}^2\text{C} ) pins are controlled by port functions</td>
</tr>
<tr>
<td>14</td>
<td>Unimplemented: Read as ‘0’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>I2CSIDL: Stop in Idle Mode bit</td>
<td>1 = Discontinue module operation when device enters Idle mode</td>
<td>Discontinue module operation when device enters Idle mode</td>
</tr>
<tr>
<td>12</td>
<td>SCLREL: SCLx Release Control bit (when operating as ( \text{I}^2\text{C} ) slave)</td>
<td>1 = Release SCLx clock</td>
<td>Release SCLx clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Hold SCLx clock low (clock stretch)</td>
<td>Hold SCLx clock low (clock stretch)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If STREN = 1:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>User software may write ‘0’ to initiate a clock stretch and write ‘1’ to release the clock. Hardware clear at the beginning of every slave data byte transmission. Hardware clear at the end of every slave address byte reception. Hardware clear at the end of every slave data byte reception.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If STREN = 0:</td>
<td>User software may only write ‘1’ to release the clock. Hardware clear at the beginning of every slave data byte transmission. Hardware clear at the end of every slave address byte reception.</td>
</tr>
<tr>
<td>11</td>
<td>IPMIEN: IPMI Enable bit(1)</td>
<td>1 = IPMI Support mode is enabled; all addresses Acknowledged</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A10M: 10-bit Slave Address bit</td>
<td>1 = I2CxADD register is a 10-bit slave address</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>DISSLW: Disable Slew Rate Control bit</td>
<td>1 = Stew rate control disabled</td>
<td>Disable Slew Rate Control bit</td>
</tr>
<tr>
<td>8</td>
<td>SMEN: SMBus Input Levels bit</td>
<td>1 = Enable I/O pin thresholds compliant with SMBus specification</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>GCEN: General Call Enable bit (when operating as ( \text{I}^2\text{C} ) slave)</td>
<td>1 = Enable interrupt when a general call address is received in the I2CxRSR register (module is enabled for reception)</td>
<td>General Call Enable bit (when operating as ( \text{I}^2\text{C} ) slave)</td>
</tr>
<tr>
<td>6</td>
<td>STREN: SCLx Clock Stretch Enable bit (( \text{I}^2\text{C} ) Slave mode only; used in conjunction with SCLREL bit)</td>
<td>1 = Enable user software or receive clock stretching</td>
<td></td>
</tr>
</tbody>
</table>

Note 1: The IPMIEN bit should not be set when the \( \text{I}^2\text{C} \) module is operating as a master.
**Register 19-1: I2CxCON: I²Cx Control Register (Continued)**

bit 5  **ACKDT**: Acknowledge Data bit (I²C Master mode; receive operation only)
- Value that will be transmitted when the user software initiates an Acknowledge sequence.
  - 1 = Send NACK during Acknowledge
  - 0 = Send ACK during Acknowledge

bit 4  **ACKEN**: Acknowledge Sequence Enable bit (I²C 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 (I²C Master mode)
- 1 = Enables Receive mode for I²C (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 (I²C 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 (I²C 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 (I²C 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

**Note 1:** The IPMIEN bit should not be set when the I²C module is operating as a master.
Register 19-2: I2CxSTAT: I²Cx Status Register

<table>
<thead>
<tr>
<th>R-0, HSC</th>
<th>R-0, HSC</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>R/C-0, HS</th>
<th>R-0, HSC</th>
<th>R-0, HSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKSTAT</td>
<td>TRSTAT</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>BCL</td>
<td>GCSTAT</td>
<td>ADD10</td>
</tr>
</tbody>
</table>

- **ACKSTAT**: Acknowledge Status bit
  - 1 = NACK received from slave
  - 0 = ACK received from slave
  - Hardware set or clear at end of Slave or Master Acknowledge.

- **TRSTAT**: Transmit Status bit (I²C Master mode transmit operation)
  - 1 = Master transmit is in progress (8 bits + 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’

- **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.

- **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

- **IWCOL**: Write Collision Detect bit
  - 1 = An attempt to write the I2CxTRN register failed because the I²C module is busy
  - 0 = No collision
  - Hardware set at occurrence of write to I2CxTRN register while busy (cleared by software).

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

- **D/A**: Data/Address bit (I²C Slave mode)
  - 1 = Indicates that the last byte received was data
  - 0 = Indicates that the last byte received was a device address
  - Hardware clear at device address match; hardware set by reception of slave byte or is set after the transmission is complete and the TBF flag is cleared.

bit 4 **P**: Stop bit

The **Legend**:
- **R** = Readable bit
- **C** = Clearable bit
- **U** = Unimplemented bit, read as ‘0’
- **HS** = Set in Hardware
- **HSC** = Hardware Set/Cleared
- **-n** = Value at Reset
- ‘1’ = Bit is set
- ‘0’ = Bit is clear
- **x** = Bit is unknown
Register 19-2:  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/W: Read/Write Information bit (when operating as I2C slave)
1 = Read: data transfer is output from slave
0 = Write: data transfer is input to slave
Hardware set or clear after reception of I2C device address byte.

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

bit 0  
TBF: Transmit Buffer Full Status bit
1 = Transmit in progress; I2CxTRN register is full
0 = Transmit complete; I2CxTRN register is empty
Hardware set when user software writes to I2CxTRN register; hardware clear at completion of data transmission.
Register 19-3: I2CxMSK: I²Cx Slave Mode Address Mask Register

<table>
<thead>
<tr>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AMSK9</td>
<td>AMSK8</td>
</tr>
<tr>
<td>bit 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 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 |       |       |       |       |       |       |       |

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
### Section 19. Inter-Integrated Circuit™ (I²C™)

**Register 19-4: ISRCCON: Current Source Control Register\(^{(1)}\)**

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>U-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISRCEN</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>OUTSEL&lt;2:0&gt;</td>
<td>bit 15</td>
<td>bit 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>—</td>
<td>—</td>
<td>ISRCCAL&lt;5:0&gt;</td>
<td>(2)</td>
<td>bit 7</td>
<td>bit 0</td>
<td></td>
</tr>
</tbody>
</table>

#### Legend:
- **R** = Readable bit
- **W** = Writable bit
- **U** = Unimplemented bit, read as ‘0’
- \(-n\) = Value at POR
- ‘1’ = Bit is set
- ‘0’ = Bit is cleared
- **x** = Bit is unknown

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

- **bit 14-11**  
  **Unimplemented**: Read as ‘0’

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

- **bit 7-6**  
  **Unimplemented**: Read as ‘0’

- **bit 5-0**  
  **ISRCCAL<5:0>**: Current Source Calibration bits\(^{(2)}\)

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

#### Note 2:  
The calibration value must be retrieved from Flash memory and stored in this location at start-up time.
19.4  ENABLING I\textsuperscript{2}C OPERATION

The I\textsuperscript{2}C module is enabled by setting the I2CEN bit (I2CxCON<15>).

The I\textsuperscript{2}C module fully implements all master and slave functions. When the module is enabled, the master and slave functions are active simultaneously and will respond according to the user software or bus events.

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

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

19.4.1  I\textsuperscript{2}C I/O Pins

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

19.4.2  I\textsuperscript{2}C Interrupts

The I\textsuperscript{2}C module generates two interrupts: MI2CxIF and SI2CxIF. The MI2CxIF interrupt is assigned to the master events and the SI2CxIF interrupt is assigned to the slave events. These interrupts set a corresponding interrupt flag bit and interrupt the user software process if the corresponding interrupt enable bit is set and the corresponding interrupt priority is higher than the CPU interrupt priority.

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

• Start condition
• Stop condition
• Data transfer byte transmitted/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 the following events:

• Detection of a valid device address (including general call)
• Request to transmit data (ACK) or to stop data transmission (NACK)
• Reception of data

19.4.3  Setting Baud Rate When Operating as a Bus Master

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

The reload value for the BRG is the I2CxBRG register as illustrated in Figure 19-5. When the BRG is loaded with this value, the generator counts down to zero and stops until another reload has taken place. The generator count is decremented twice per instruction cycle (TCy). The BRG is reloaded automatically on baud rate restart. For example, if clock synchronization is taking place, the BRG will be reloaded when the SCLx pin is sampled high.

Note: I2CxBRG register values that are less than two are not supported.
Equation 19-1 shows the formula for computing the BRG reload value.

Equation 19-1: BRG Reload Value Calculation

\[
I_{2C}BRG = \left[ \left( \frac{1}{F_{SCL}} - PGD \right) \cdot F_{CY} \right] - 2
\]

Table 19-1: \(I_2C\) Clock Rates

<table>
<thead>
<tr>
<th>Required System F&lt;sub&gt;SCL&lt;/sub&gt;</th>
<th>F&lt;sub&gt;CY&lt;/sub&gt;</th>
<th>PGD(^{(1)})</th>
<th>(I_2CxBRG) Decimal</th>
<th>(I_2CxBRG) Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 kHz</td>
<td>40 MHz</td>
<td>130 ns</td>
<td>392.8</td>
<td>0x188</td>
</tr>
<tr>
<td>100 kHz</td>
<td>20 MHz</td>
<td>130 ns</td>
<td>195.4</td>
<td>0x0C3</td>
</tr>
<tr>
<td>100 kHz</td>
<td>10 MHz</td>
<td>130 ns</td>
<td>96.7</td>
<td>0x060</td>
</tr>
<tr>
<td>400 kHz</td>
<td>20 MHz</td>
<td>130 ns</td>
<td>45.4</td>
<td>0x02D</td>
</tr>
<tr>
<td>400 kHz</td>
<td>10 MHz</td>
<td>130 ns</td>
<td>21.7</td>
<td>0x015</td>
</tr>
<tr>
<td>400 kHz</td>
<td>5 MHz</td>
<td>130 ns</td>
<td>9.85</td>
<td>0x009</td>
</tr>
<tr>
<td>1 MHz</td>
<td>10 MHz</td>
<td>130 ns</td>
<td>6.7</td>
<td>0x006</td>
</tr>
</tbody>
</table>

Note 1: The typical value of the Pulse Gobbler Delay (PGD) is 130 ns. Refer to the specific device data sheet for more information.

Note: Equation 19-1 and Table 19-1 are only design guidelines. Due to system-dependant parameters, the actual baud rate may differ slightly. Testing is required to confirm that the actual baud rate meets the system requirements. Otherwise, the value of \(I_2CxBRG\) may need to be adjusted.
The I²C module’s typical operation in a system is using the I²C to communicate with an I²C peripheral, such as an I²C serial memory. In an I²C system, the master controls the sequence of all data communication on the bus. In this example, the dsPIC33F/PIC24H and its I²C module have the role of the single-master in the system. As the single-master, it is responsible for generating the SCLx clock and controlling the message protocol.

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

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

For example, a typical operation in a single-master environment is to read a byte from an I²C serial EEPROM. Figure 19-6 illustrates the example message.

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

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

Figure 19-6: A Typical I²C™ Message: Read of Serial EEPROM (Random Address Mode)

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

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

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

Figure 19-7 illustrates the timing of the Start condition.

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

19.5.1.1 IWCOL STATUS FLAG

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

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

![Figure 19-7: Master Start Timing Diagram](image)

19.5.2 Sending Data to a Slave Device

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

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

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

The sequence of events that occur during Master Transmission and Master Reception are provided in Figure 19-8.
19.5.2.1 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 I2C device address and a R/W status bit that defines whether the message will be a write to the slave (master transmission and slave reception) or a read from the slave (slave transmission and master reception).

Note 1: In 7-bit Addressing mode, each node using the I2C protocol should be configured with a unique address that is stored in the I2CxADD register.

2: While transmitting the address byte, the master must shift the address bits <7:0> left by 1 bit, and configure bit 0 as the R/W bit.
19.5.2.2 SENDING A 10-BIT ADDRESS TO THE SLAVE

Sending a 10-bit device address involves sending two bytes to the slave. The first byte contains 5 bits of the I2C 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/W status bit in the first byte must be ‘0’, indicating master transmission and slave reception. If the message data is also directed toward the slave, a Repeated Start sequence with the R/W status bit at ‘1’ will change the R/W state of the message to a read of the slave.

Note 1: In 10-bit Addressing mode, each node using the I2C protocol should be configured with a unique address that is stored in the I2CxADD register.

2: While transmitting the first address byte, the master must shift the bits <9:8> left by one bit, and configure bit 0 as the R/W bit.

19.5.2.3 RECEIVING ACKNOWLEDGE FROM THE SLAVE

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

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

The status of ACK is written into the ACKSTAT bit (I2CxSTAT<15>), on the falling edge of the ninth SCLx clock. After the ninth SCLx clock, the module generates the MII2CxIF interrupt and enters into the Idle state until the next data byte is loaded into the I2CxTRN register.

19.5.2.4 ACKSTAT STATUS FLAG

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

19.5.2.5 TBF STATUS FLAG

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

19.5.2.6 IWCOL STATUS FLAG

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

Note: Because queueing of events is not allowed, writing to the lower 5 bits of the I2CxCON register is disabled until the transmit condition is complete.
19.5.3 Receiving Data from a Slave Device

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

Note: The lower 5 bits of the I2CxCON register 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 following events occur:

- The RCEN bit (I2CxCON<3>) is automatically cleared
- The contents of the I2CxRSR register transfer into the I2CxRCV register
- The RBF status bit (I2CxSTAT<1>) is set
- The I2C module generates the MI2CxIF interrupt

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

The sequence of events that occur during Master Transmission and Master Reception are provided in Figure 19-9.

Figure 19-9: 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 BRG starts. SCLx remains low.
3. The BRG times out. The Master attempts to release SCLx.
4. When slave releases SCLx, the BRG restarts.
5. The BRG times out. The MSB of response is shifted to I2CxRSR register. SCLx is driven low for the next baud interval.
6. At the falling edge of the eighth SCLx clock, the I2CxRSR register is transferred to the I2CxRCV register. The module clears the RCEN bit. The RBF status bit is set. Master generates interrupt.
19.5.3.1 RBF STATUS FLAG

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

19.5.3.2 I2COV STATUS FLAG

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

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

19.5.3.3 IWCOL STATUS FLAG

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

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

19.5.4 Acknowledge Generation

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

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

Figure 19-10 illustrates an ACK sequence and Figure 19-11 illustrates a NACK sequence. The ACKDT bit (I2CxCON<5>), specifies ACK or NACK.

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

19.5.4.1 IWCOL STATUS FLAG

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

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

Figure 19-10: Master Acknowledge (ACK) Timing Diagram

1. Writing ACKDT = 0 specifies sending an ACK.
2. Writing ACKEN = 1 initiates a master Acknowledge event.
3. BRG starts. SCLx remains low.
4. When SCLx detected low, module drives SDAx low.
5. The BRG times out. Module releases SCLx. BRG restarts.
6. BRG times out.
7. Module drives SCLx low, then releases SDAx.
8. Module clears ACKEN. Master generates interrupt.
19.5.5 Generating Stop Bus Event

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

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

When the PEN bit is set, the master generates the Stop sequence as illustrated in Figure 19-12.

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

19.5.5.1 IWCOL STATUS FLAG

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

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

---

Figure 19-11: Master Not-Acknowledge (NACK) Timing Diagram

---

Figure 19-12: Master Stop Timing Diagram
### Generating Repeated Start Bus Event

Setting the RSEN bit (I2CxCON<1>), enables generation of a master Repeated Start sequence as illustrated in Figure 19-13.

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

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

The following is the Repeated Start sequence:

1. The slave detects the Start condition, sets the S status bit (I2CxSTAT<3>) and clears the P status bit (I2CxSTAT<4>).
2. The RSEN bit is automatically cleared.
3. The I²C module generates the MI2CxIF interrupt.

#### IWCOL STATUS FLAG

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

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

---

**Figure 19-13: Master Repeated Start Timing Diagram**

1. Writing RSEN = 1 initiates a master Repeated Start event. BRG starts. Module drives SCLx low and releases SDAx.
2. The BRG times out. Module releases SCLx. BRG restarts.
3. The BRG times out. Module drives SDAx low. BRG restarts.
4. Slave logic detects Start. Module sets $S = 1$ and $P = 0$.
5. The BRG times out. Module drives SCLx low. Module clears RSEN. Master generates interrupt.
19.5.7 Building Complete Master Messages

As described at the beginning of 19.5 “Communicating as a Master in a Single-Master Environment”, the user software is responsible for constructing messages with the correct message protocol. The module controls individual portions of the I2C message protocol; however, sequencing of the components of the protocol to construct a complete message is performed by the user software.

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

The user software can use the SEN, RSEN, PEN, RCEN and ACKEN bits (Least Significant 5 bits of the I2CxCON register) and the TRSTAT status bit as a ‘state’ flag when progressing through a message. For example, Table 19-2 shows some example state numbers associated with bus states.

Table 19-2: Master Message Protocol States

<table>
<thead>
<tr>
<th>Example State Number</th>
<th>I2CxCON&lt;4:0&gt;</th>
<th>TRSTAT (I2CxSTAT&lt;14&gt;)</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000</td>
<td>0</td>
<td>Bus Idle or Wait</td>
</tr>
<tr>
<td>1</td>
<td>00001</td>
<td>N/A</td>
<td>Sending Start Event</td>
</tr>
<tr>
<td>2</td>
<td>00000</td>
<td>1</td>
<td>Master Transmitting</td>
</tr>
<tr>
<td>3</td>
<td>00010</td>
<td>N/A</td>
<td>Sending Repeated Start Event</td>
</tr>
<tr>
<td>4</td>
<td>00100</td>
<td>N/A</td>
<td>Sending Stop Event</td>
</tr>
<tr>
<td>5</td>
<td>01000</td>
<td>N/A</td>
<td>Master Reception</td>
</tr>
<tr>
<td>6</td>
<td>10000</td>
<td>N/A</td>
<td>Master Acknowledgement</td>
</tr>
</tbody>
</table>

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

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

As each event completes and generates an interrupt, the interrupt handler may check the state number. Therefore, for a start state, the interrupt handler will confirm execution of the Start sequence and then start a master transmission event to send the I2C 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 19-14 illustrates a more detailed examination of the same message sequence as shown in Figure 19-6. Figure 19-15 illustrates a few simple examples of the messages using 7-bit addressing format. Figure 19-16 illustrates an example of a 10-bit addressing format message sending data to a slave. Figure 19-17 illustrates an example of a 10-bit addressing format message receiving data from a slave.
<table>
<thead>
<tr>
<th></th>
<th>SEN</th>
<th>RSEN</th>
<th>RCEN</th>
<th>ACKEN</th>
<th>AKDT</th>
<th>PEN</th>
<th>I2CxTRN</th>
<th>TBF</th>
<th>I2CxRCV</th>
<th>RBF</th>
<th>ACKSTAT</th>
<th>MI2CxIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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/W status bit clear, indicating a write.
3. Writing the I2CxTRN register starts a master transmission. The data is the first byte of the EEPROM data address.
4. Writing the I2CxTRN register starts a master transmission. The data is the second byte of the EEPROM data address.
5. Setting the RSEN bit starts a Repeated Start event.
6. Writing the I2CxTRN register starts a master transmission. The data is the second byte of the EEPROM data address, but with R/W status bit set, indicating a read.
7. Setting the RCEN bit starts a master reception. On interrupt, the user software reads the I2CxRCV register, which clears the RBF status bit.
8. Setting the ACKEN bit starts an Acknowledge event. ACKDT = 1 to send NACK.
9. Setting the PEN bit starts a master Stop event.

Figure 19-14: Master Message (Typical I²C™ Message: Read of Serial EEPROM)
Figure 19-15: 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 status bit clear.
3. Writing the I2CxTRN register starts a master transmission. The data is the message byte.
4. Setting the PEN bit starts a master Stop event.
5. Setting the SEN bit starts a Start event. An interrupt is generated on completion of the Start event.

6. Writing the I2CxTRN register starts a master transmission. The data is the address byte with R/W status bit set.
7. Setting the RCEN bit starts a master reception.
8. Setting the ACKEN bit starts an Acknowledge event. ACKDT = 1 to send NACK.
9. Setting the PEN bit starts a master Stop event.
Setting the SEN bit starts a Start event.

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

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

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

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

Setting the PEN bit starts a master Stop event.
Figure 19-17: 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/W status bit cleared.
3. Writing the I2CxTRN register starts a master transmission. The data is the second byte of the address.
4. Setting the RSEN bit starts a master Restart event.
5. Writing the I2CxTRN register starts a master transmission. The data is a re-send of the first byte with the R/W status bit set.
6. Setting the RCEN bit starts a master reception. On interrupt, the user software reads the I2CxRCV register, which clears the RBF status bit.
7. Setting the ACKEN bit starts an Acknowledge event. ACKDT = 0 to send ACK.
8. Setting the RCEN bit starts a master reception.
9. Setting the ACKEN bit starts an Acknowledge event. ACKDT = 1 to send NACK.
10. Setting the PEN bit starts a master Stop event.
19.6 COMMUNICATING AS A MASTER IN A MULTI-MASTER ENVIRONMENT

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

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

### 19.6.1 Multi-Master Operation

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

### 19.6.2 Master Clock Synchronization

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

The clock synchronization occurs when the master deasserts the SCLx pin (SCLx intended to float high). When the SCLx pin is released, the BRG is suspended from counting until the SCLx pin is actually sampled high. When the SCLx pin is sampled high, the BRG is reloaded with the contents of I2CxBRG<8:0> and begins counting. This ensures that the SCLx high time will always be at least one BRG rollover count in the event that the clock is held low by an external device, as illustrated in Figure 19-18.

**Figure 19-18: 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.
19.6.3 Bus Arbitration and Bus Collision

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

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

The first master will set the BCL bit (I2CxSTAT<10>), and generate a master interrupt. The Master module will reset the I2C port to its Idle state.

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

The states where arbitration can be lost are:
- Start condition
- Repeated Start condition
- Address, Data or Acknowledge bit
- Stop condition

19.6.4 Detecting Bus Collisions and Re-sending Messages

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

The user software is expecting an interrupt at the completion of the master event. The user software can check the BCL status bit to determine if the master event completed successfully or a bus collision occurred. If a bus collision occurs, the user software must abort sending the rest of the pending message and prepare to re-send the entire message sequence, beginning with the Start condition, after the bus returns to Idle state. The user software can monitor the S and P status bits to wait for an Idle bus. When the user software executes the master Interrupt Service Routine (ISR) and the I2C bus is free, the user software can resume communication by asserting a Start condition.

19.6.5 Bus Collision During a Start Condition

Before issuing a Start condition, the user software should verify an Idle state of the bus using the S and P status bits. Two masters may attempt to initiate a message at a similar point in time. Typically, the masters will synchronize clocks and continue arbitration into the message until one loses arbitration. However, the following conditions can cause a bus collision to occur during a Start:
- If the SDA and SCL pins are at a low logic state at the beginning of the Start condition, or
- If the SCL line is at a low logic state before the SDA line is driven low

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

19.6.6 Bus Collision During a Repeated Start Condition

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

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

If the user software is properly checking the bus state, it is unlikely that a bus collision will occur on a Start condition. However, because another master can, at the same time, check the bus and initiate its own Start condition, it is likely that SDAx arbitration will occur and synchronize the Start of two masters. In this condition, both masters begin and continue to transmit their messages until one master loses arbitration on a message bit. The SCLx clock synchronization keeps the two masters synchronized until one loses arbitration. Figure 19-19 illustrates an example of the message bit arbitration.

Figure 19-19: Bus Collision During Message Bit Transmission

![Diagram of bus collision during message bit transmission]

**19.6.8 Bus Collision During a Stop Condition**

If the master software loses track of the state of the $^2$C bus, many conditions exist that can cause a bus collision during a Stop condition. In this case, the master generating the Stop condition will lose arbitration and generate a bus collision interrupt.
19.7 COMMUNICATING AS A SLAVE

In some systems, particularly where multiple processors communicate with each other, the dsPIC33F/PIC24H device can communicate as a slave, as illustrated in Figure 19-20. When the I²C module is enabled, the slave is active. The slave cannot initiate a message, it can only respond to a message sequence initiated by a master. The master requests a response from a particular slave as defined by the device address byte in the I²C protocol. The slave replies to the master at the appropriate times as defined by the protocol.

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

Figure 19-20: A Typical Slave I²C™ Message: Multiprocessor Command/Status

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

19.7.1 Sampling Receive Data

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

19.7.2 Detecting Start and Stop Conditions

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

19.7.3 Detecting the Address

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

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

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

19.7.3.2 7-BIT ADDRESS AND SLAVE WRITE

Following the Start condition, the module shifts 8 bits into the I2CxRSR register as illustrated in Figure 19-21. The value of the I2CxRSR register is evaluated against that of the I2CxADD and I2CxMSK registers on the falling edge of the eighth clock (SCLx). If the address is valid (that is, an exact match between unmasked bit positions), the following events occur:

- An ACK is generated
- The D/A and R/W status bits are cleared
- The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock
- The module waits for the master to send data

![Figure 19-21: Slave Write 7-bit Address Detection Timing Diagram](image)

19.7.3.3 7-BIT ADDRESS AND SLAVE READ

When a slave read is specified by having \( R/W = 1 \) in a 7-bit address byte, the process of detecting the device address is similar to that for a slave write, as illustrated in Figure 19-22. If the addresses match, the following events occur:

- An ACK is generated
- The D/A status bit is cleared and the R/W status bit is set
- The module generates the SI2CxIF interrupt on the falling edge of the ninth SCLx clock

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

**Note:** The SCLREL bit will automatically clear after detection of a slave read address, regardless of the state of the STREN bit.
19.7.3.4 10-BIT ADDRESSING MODE

In 10-bit Addressing mode, the slave must receive two device address bytes as illustrated in Figure 19-23. The 5 Most Significant bits (MSbs) of the first address byte specify a 10-bit address. The R/W status bit of the address must specify a write, causing the slave device to receive the second address byte. For a 10-bit address, the first byte would equal '11110 A9 A8 0', where A9 and A8 are the 2 MSbs of the address.

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

Following the Start condition, the module shifts eight bits into the I2CxRSR register. The value of the I2CxRSR<2:1> bits are evaluated against the value of the I2CADD<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 I2CADD<9:8>. (If both bits are masked, a match is not needed.) If the address is valid, the following events occur:

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

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

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

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

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

The addressing procedure for the \( \text{I}^2\text{C} \) bus is such that the first byte after a Start condition usually determines which slave device the master is addressing. The exception is the general call address, which can address all devices. When this address is used, all the enabled devices respond with an Acknowledge. The general call address is one of the eight addresses reserved for specific purposes by the \( \text{I}^2\text{C} \) protocol. It consists of all ‘0’s with \( \text{R/W} = 0 \). 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 as illustrated in Figure 19-24. Following a Start bit detect, 8 bits are shifted into the I2CxRSR register and the address is compared against the I2CxADD register and the general call address.

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

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

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

Note: General call addresses are 7-bit addresses. If configuring the slave for 10-bit addresses and the A10M and GCEN bits are set, the slave will continue to detect the 7-bit general call address.
19.7.3.6 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 bus uses the I^2C nodes as message repeaters in a distributed network. To allow a node to repeat all messages, the slave must accept all messages, regardless of the device address.

To enable the IPMI mode, set the IPMIEN bit (I2CxCON<11>) as illustrated in Figure 19-25. Regardless of the state of the A10M and GCEN bits or the value loaded in the I2CxADD register, all addresses are accepted. This includes all valid 7-bit addresses, General Call, Start Byte, CBUS, Reserved and HS modes, as well as 10-bit address preambles.

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

If a 7-bit address does not match the contents of I2CxADD<6:0>, the slave will return to an Idle state and ignore any activity on the I2C bus 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 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> but the second byte of the 10-bit address does not match I2CxADD<7:0>, the slave will return to an Idle state and ignore all bus activity until after the Stop condition.

19.7.3.8 ADDRESSES RESERVED FROM MASKING

Even when enabled, there are several addresses that are ignored by the I2C module. For these addresses, an Acknowledge will not be issued independent of the mask setting. These addresses are listed in Table 19-3.

Table 19-3: Reserved I2C™ Bus Addresses

<table>
<thead>
<tr>
<th>Slave Address</th>
<th>R/W Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 000</td>
<td>0</td>
<td>General Call Address[1]</td>
</tr>
<tr>
<td>0000 000</td>
<td>1</td>
<td>Start Byte</td>
</tr>
<tr>
<td>0000 001</td>
<td>x</td>
<td>CBUS Address</td>
</tr>
<tr>
<td>0000 010</td>
<td>x</td>
<td>Reserved</td>
</tr>
<tr>
<td>0000 011</td>
<td>x</td>
<td>Reserved</td>
</tr>
<tr>
<td>0000 1xx</td>
<td>x</td>
<td>HS Mode Master Code</td>
</tr>
<tr>
<td>1111 1xx</td>
<td>x</td>
<td>Reserved</td>
</tr>
<tr>
<td>1111 0xx</td>
<td>x</td>
<td>10-bit Slave Upper Byte[2]</td>
</tr>
</tbody>
</table>

Note 1: Address will be Acknowledged only if GCEN = 1.
Note 2: Match on this address can only occur as the upper byte in the 10-bit Addressing mode.

19.7.4 RECEIVING DATA FROM A MASTER DEVICE

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

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

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

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

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

Table 19-4 shows what happens when a data transfer byte is received, given the status of the RBF and I2COV status bits. If the RBF status bit is already set when the slave attempts to transfer to the I2CxRCV register, the transfer does not occur but the interrupt is generated and the I2COV status bit is set. If both the RBF and I2COV status bits are set, the slave acts similarly. The shaded cells show the condition where user software did not properly clear the overflow condition.

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

Table 19-4: Data Transfer Received Byte Actions

<table>
<thead>
<tr>
<th>Status Bits as Data Byte Received</th>
<th>Transfer I2CxRSR to I2CxRCV</th>
<th>Generate ACK</th>
<th>Generate SI2CxIF Interrupt (interrupt occurs if enabled)</th>
<th>Set RBF</th>
<th>Set I2COV</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBF 0 I2COV 0</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No change</td>
</tr>
<tr>
<td>1 0</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No change</td>
<td>Yes</td>
</tr>
<tr>
<td>1 1</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No change</td>
<td>Yes</td>
</tr>
<tr>
<td>0 1</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No change</td>
</tr>
</tbody>
</table>

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

19.7.4.2 WAIT STATES DURING SLAVE RECEIPTIONS

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

The STREN bit (I2CxCON<6>) enables a bus wait to occur on slave receptions. When STREN = 1 at the falling edge of the ninth SCLx clock of a received byte, the slave clears the SCLREL bit. Clearing the SCLREL bit causes the slave to pull the SCLx line low, initiating a wait. The SCLx clock of the master and slave will synchronize, as shown in 19.6.2 “Master Clock Synchronization”.

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

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

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

Figure 19-26 illustrates a simple receive message. Because it is a 7-bit address message, only one interrupt occurs for the address bytes. Then, interrupts occur for each of four data bytes. At an interrupt, the user software may monitor the status bits, RBF (I2CxSTAT<1>), D/Ä (I2CxSTAT<5>), and R/Ŵ (I2CxSTAT<2>) to determine the condition of the byte received.

Figure 19-27 illustrates a similar message using a 10-bit address. In this case, two bytes are required for the address.

Figure 19-28 illustrates a case where the user software does not respond to the received byte and the buffer overruns. On reception of the second byte, the module will automatically NACK the master transmission. Generally, this causes the master to re-send the previous byte. The I2COV status bit (I2CxSTAT<6>) indicates that the buffer has overrun. The I2CxRCV register buffer retains the contents of the first byte. On reception of the third byte, the buffer is still full, and again, the module will NACK the master. After this, the user software finally reads the buffer. Reading the buffer will clear the RBF status bit; however, the I2COV status bit remains set. The user software must clear the I2COV status bit (I2CxSTAT<6>). The next received byte is moved to the I2CxRCV register buffer and the module responds with an ACK.

Figure 19-29 highlights clock stretching while receiving data. In the previous examples, the STREN bit (I2CxCON<6>) is equal to '0', which disables clock stretching on receive messages. In this example, the user software sets STREN to enable clock stretching. When STREN = 1, the module will automatically clock stretch after each received data byte, allowing the user software more time to move the data from the buffer. If RBF = 1 at the falling edge of the ninth clock, the module automatically clears the SCLREL bit (I2CxCON<12>) and pulls the SCLx bus line low. As shown with the second received data byte, if the user software can read the buffer and clear the RBF status bit before the falling edge of the ninth clock, the clock stretching will not occur. The user software can also suspend the bus at any time. By clearing the SCLREL bit, the module pulls the SCLx line low after it detects the bus SCLx low. The SCLx line remains low, suspending transactions on the bus until the SCLREL bit is set.
Figure 19-26: 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 status 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. The byte moved to I2CxRCV register sets the RBF status bit. Slave generates interrupt. Slave Acknowledges reception.
4 User software reads I2CxRCV register. RBF status bit clears.
5 Slave recognizes Stop event; S and P status bits set/clear accordingly.
Figure 19-27: Slave Message (Write Data to Slave: 10-bit Address; Address Matches; A10M = 1; GCEN = 0; IPMIEN = 0)

2. Slave receives address byte. High order address matches.
   Slave Acknowledges and generates interrupt. Address byte is moved to I2CRCV register and is read by user software to prevent buffer overflow.
3. Slave receives address byte. Low order address matches.
   Slave Acknowledges and generates interrupt. Address byte is moved to I2CRCV register and is read by user software to prevent buffer overflow.
4. Next received byte is message data. Byte moved to I2CRCV register, sets RBF.
   Slave Acknowledges and generates interrupt.
5. User software reads I2CRCV register. RBF bit clears.
Figure 19-28: 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 is moved to I2CxRCV register and must be read by user software to prevent buffer overflow.

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


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

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

6. User software clears I2COV bit. Reception will still not be able to proceed normally until the module sees a stop/repeated start bit. If neither of these conditions is met, an additional transmission will be received correctly, but send a NACK and set the I2COV bit again.
Figure 19-29: Slave Message (Write Data to Slave: 7-bit Address; Clock Stretching Enabled; \text{A}10\text{M} = 0; \text{GCEN} = 0; \text{IPMIEN} = 0)

1. User software sets the STREN bit to enable clock stretching.
2. Slave receives address byte. I2CRCV register is read by user software to prevent buffer overflow.
3. Next received byte is message data. The byte moved to I2CRCV register, sets RBF.
4. Because RBF = 1 at ninth clock, automatic clock stretch begins. Slave clears SCLREL bit. Slave pulls SCL line low to stretch clock.
5. User software reads I2CRCV register. RBF bit clears.
6. User software sets SCLREL bit to release clock.
7. Slave does not clear SCLREL because RBF = 0 at this time.
8. User software may clear SCLREL to cause a clock hold. Module must detect SCL low before asserting SCL low.
9. User software may set SCLREL to release a clock hold.
19.7.5 Sending Data to a Master Device

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

| Note: | When the IPMIEN bit (I2CxCON<11>) is equal to ‘1’ (IPMI mode), the I2C module assumes that the R/W bit is ‘0’. Therefore, the slave transmission function is disabled. If the R/W bit is ‘1’, the I2C module will trigger an interrupt. This interrupt should be ignored (that is, the I2C interrupt flags should be cleared) and the I2C slave transmission event should be aborted. |

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

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

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

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

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

If the SDAx line is high, indicating a Not-Acknowledge (NACK), the data transfer is complete. The slave resets and generates an interrupt, and it waits for detection of the next Start bit.

19.7.5.1 WAIT STATES DURING SLAVE TRANSMISSIONS

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

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

The slave clears the SCLREL bit (I2CxCON<12>). Clearing the SCLREL bit causes the slave to pull the SCLx line low, initiating a wait. The SCLx clock of the master and slave will synchronize, as shown in 19.6.2 “Master Clock Synchronization”.

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

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

19.7.5.2 EXAMPLE MESSAGES OF SLAVE TRANSMISSION

The slave transmissions for 7-bit address messages are illustrated in Figure 19-30. When the address matches and the R/W status bit of the address indicates a slave transmission, the module automatically initiates clock stretching by clearing the SCLREL bit and generates an interrupt to indicate a response byte is required. The user software writes the response byte into the I2CxTRN register. As the transmission completes, the master responds with an ACK. If the master replies with an ACK, the master expects more data and the module again clears the SCLREL bit and generates another interrupt. If the master responds with a NACK, no more data is required and the module will not stretch the clock but will generate an interrupt.

The slave transmissions for 10-bit address messages require the slave to first recognize a 10-bit address. Because the master must send two bytes for the address, the R/W status bit in the first byte of the address specifies a write. To change the message to a read, the master sends a Repeated Start and repeats the first byte of the address with the R/W status bit specifying a read. At this point, the slave transmission begins as illustrated in Figure 19-31.
Figure 19-30: Slave Message (Read Data from Slave: 7-bit Address)

2. Slave receives address byte. Address matches. Slave generates interrupt. Address byte is moved to I2CRCV register and is read by user software to prevent buffer overflow. R/W = 1 to indicate read from slave. SCLREL = 0 to suspend master clock.
3. User software writes I2CTRN with response data. TBF = 1 indicates that buffer is full. Writing I2CTRN sets D/A, indicating data byte.
4. User software sets SCLREL to release clock hold. Master resumes clocking and slave transmits data byte.
5. After last bit, module clears TBF bit indicating buffer is available for next byte.
6. At the end of ninth clock, if the master has sent ACK, module clears SCLREL to suspend clock. Slave generates interrupt.
7. At the end of ninth 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 19-31: Slave Message (Read Data from Slave: 10-bit Address)

4. Master sends a Repeated Start to redirect the message.
6. User software writes I2CTR with response data.
7. User software sets SCLREL to release clock hold. Master resumes clocking and slave transmits data byte.
8. At the end of ninth clock, if master sent ACK, module clears SCLREL to suspend clock. Slave generates interrupt.
9. At the end of ninth 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.
19.8 CONNECTION CONSIDERATIONS FOR I\textsuperscript{2}C BUS

Because the I\textsuperscript{2}C bus is a wired-AND bus connection, pull-up resistors (R\textsubscript{P}) on the bus are required as illustrated in Figure 19-32. The series resistors (R\textsubscript{S}) are optional and are used to improve the Electrostatic Discharge (ESD) susceptibility. The values of the resistors, R\textsubscript{P} and R\textsubscript{S}, depend on the following parameters:

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

Because the device must be able to pull the bus low against R\textsubscript{P}, current drawn by R\textsubscript{P} must be greater than the I/O pin minimum sink current, I\textsubscript{OL} at V\textsubscript{OLMAX}, for the device output stage. Equation 19-2 shows the formula for computing the minimum pull-up resistance.

**Equation 19-2: Minimum Pull-up Resistance**

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

In a 400 kHz system, a minimum rise time specification of 300 ns exists; in a 100 kHz system, the specification is 1000 ns. Because R\textsubscript{P} must pull the bus up against the total capacitance, C\textsubscript{B}, with a maximum rise time of 300 ns to (V\textsubscript{DD} – 0.7V), the maximum resistance for the pull-up (R\textsubscript{PMAX}) is computed using the formula as shown in Equation 19-3.

**Equation 19-3: Maximum Pull-up Resistance**

\[
C_{B} \times \frac{-tR}{I_{OL} \times (V_{DDMAX} - V_{ILMAX})}
\]

The maximum value for R\textsubscript{S} is determined by the desired noise margin for the low level. R\textsubscript{S} cannot drop enough voltage to make the device V\textsubscript{OL} and the voltage across R\textsubscript{S} more than the maximum V\textsubscript{IL}. Equation 19-4 shows the formula for computing the maximum value for R\textsubscript{S}.

**Equation 19-4: Maximum Series Resistance**

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

**Note:** The SCLx clock input must have a minimum high and low time for proper operation. Refer to the “Electrical Characteristics” chapter in the specific device data sheet for more details on the high and low times of the I\textsuperscript{2}C bus specification, and requirements of the I\textsuperscript{2}C module and I/O pins.
19.8.1 Integrated Signal Conditioning

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

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

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

The SMEN bit (I²CxCON<8>) controls the input levels. Setting SMEN (= 1) changes the input levels to SMBus specifications.
19.9 OPERATION IN POWER-SAVING MODES

19.9.1 Sleep Mode in Slave Mode

The I²C module can wake-up from Sleep mode on detecting a valid slave address match. Because all bit shifting is done with reference to the external SCL signal generated by an I²C master, transmissions and receptions can continue even while in Sleep mode.

| Note: | As per the slave I²C behavior, a slave interrupt is generated only on an address match. Therefore, when an I²C slave is in Sleep mode and it receives a message from the master, the clock required to match the received address is derived from the master. Only on an address match, will the interrupt be generated and the device can wake-up from Sleep, provided the interrupt has been enabled and an ISR has been defined. |

19.9.2 Sleep Mode in Master Mode

If Sleep occurs in the middle of a master transmission, and the state machine is partially into a transmission as the clocks stop, the behavior of the module will be undefined. Similarly, if Sleep occurs in the middle of a master reception, the module behavior will also be undefined. The transmitter and receiver will stop at Sleep when in Master mode. Register contents are not affected by going into Sleep mode or coming out of Sleep mode; there is no automatic way to prevent Sleep entry if a transmission or reception is pending. The user software must synchronize Sleep entry with I²C operation to avoid undefined module behavior.

19.9.3 When the Device Enters Idle Mode

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

19.10 PERIPHERAL MODULE DISABLE (PMDx) REGISTERS

The Peripheral Module Disable (PMDx) registers provide a method to disable the I²C modules by stopping all clock sources supplied to that module. When a peripheral is disabled through the appropriate PMDx control bit, the peripheral is in a minimum power consumption state. The control and status registers associated with the peripheral are also disabled, so writes to those registers will have no effect and read values will be invalid. A peripheral module is only enabled if the I2CxMD bit in the PMDx register is cleared.

19.11 EFFECTS OF A RESET

A Reset disables the I²C module and terminates any active or pending message activity. Refer to the I2CxCON and I2CxSTAT register definitions (Register 19-1 and Register 19-2) 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 I²C module is not transferring data on the bus. |
19.12 CONSTANT CURRENT SOURCE

The Constant Current Source module is a precision current generator and is used in conjunction with an Analog-to-Digital Converter (ADC) to measure the resistance of external resistors connected to device pins. A typical use of this peripheral is to set the I²C board address using resistors, as explained in the following sections.

19.12.1 Features Overview

The Constant Current Source module offers the following major features:

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

19.12.2 Module Description

Figure 19-33 shows a functional block diagram of the Constant Current Source module. It consists of a precision current generator, with a nominal value of 10 µA. The module can be enabled and disabled using the ISRCEN bit (ISRCCON<15>). The output of the current generator is internally connected to one out of four pins. The OUTSEL<2:0> bits (ISRCCON<10:8>) allow selection of the target pin. The current source is calibrated during testing. The calibration value is stored into the Flash die serialization area and should be read and transferred to the ISRCCON register by the application firmware, before using the peripheral.

Figure 19-33: Constant Current Source Module Block Diagram
### 19.12.3 Module Applications

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

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

The current source output is nominally 10 µA, and the maximum output voltage that can be reached while still maintaining current regulation is \( V_{DD} - 0.5V \). Table 19-5 shows some example resistor values that can be used to encode an address into a resistor. The table uses common 5% tolerance resistor values, and the chosen values are separated by two standard resistor values, which is approximately a 30% value separation.

#### Table 19-5: Example PMBus™ Address Encoding

<table>
<thead>
<tr>
<th>Resistor</th>
<th>Voltage</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>13k</td>
<td>130 mV</td>
<td>1</td>
</tr>
<tr>
<td>18k</td>
<td>180 mV</td>
<td>2</td>
</tr>
<tr>
<td>24k</td>
<td>240 mV</td>
<td>3</td>
</tr>
<tr>
<td>33k</td>
<td>330 mV</td>
<td>4</td>
</tr>
<tr>
<td>43k</td>
<td>430 mV</td>
<td>5</td>
</tr>
<tr>
<td>56k</td>
<td>560 mV</td>
<td>6</td>
</tr>
<tr>
<td>75k</td>
<td>750 mV</td>
<td>7</td>
</tr>
<tr>
<td>100k</td>
<td>1V</td>
<td>8</td>
</tr>
<tr>
<td>130k</td>
<td>1.3V</td>
<td>9</td>
</tr>
<tr>
<td>180k</td>
<td>1.8V</td>
<td>10</td>
</tr>
<tr>
<td>240k</td>
<td>2.4V</td>
<td>11</td>
</tr>
</tbody>
</table>

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

A summary of the registers associated with the I^2^C module is provided in Table 19-6.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>All Resets</th>
</tr>
</thead>
<tbody>
<tr>
<td>I^2^CxRCV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>I^2^CxTRN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00FF</td>
<td></td>
</tr>
<tr>
<td>I^2^CxBRG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>I^2^CxCON</td>
<td>I2CEN</td>
<td>—</td>
<td>I2CSIDL</td>
<td>SCLREL</td>
<td>IPMIE</td>
<td>A10M</td>
<td>DISSLW</td>
<td>SMEN</td>
<td>GCEN</td>
<td>STREN</td>
<td>ACKDT</td>
<td>ACKEN</td>
<td>RCEN</td>
<td>PEN</td>
<td>RSEN</td>
<td>SEN</td>
<td>1000</td>
</tr>
<tr>
<td>I^2^CxSTAT</td>
<td>ACKSTAT</td>
<td>TRSTAT</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>BCL</td>
<td>GCSTAT</td>
<td>ADD10</td>
<td>IWCOL</td>
<td>I2COV</td>
<td>D/A</td>
<td>P</td>
<td>S</td>
<td>R/W</td>
<td>RBF</td>
<td>TBF</td>
<td>0000</td>
</tr>
<tr>
<td>I^2^CxADD</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0000</td>
</tr>
<tr>
<td>I^2^CxMSK</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0000</td>
</tr>
<tr>
<td>ISRCON</td>
<td>ISRCEN</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>OUTSEL&lt;2:0&gt;</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0000</td>
<td></td>
</tr>
</tbody>
</table>

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

**Note 1:** Not all bits are available for all devices. Please refer to the specific device data sheet for details.
19.14 DESIGN TIPS

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

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

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

**Answer:** The slave enters an automatic wait when preparing to transmit. Ensure that you set the SCLREL bit to release the I²C clock.

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

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

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

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

**Question 5:** My I²C system is a multi-master system. Why are my messages being corrupted when I attempt to send them?

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

**Question 6:** My I²C system is a multi-master system. How can I tell when it is okay to begin a message?

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

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

**Answer:** You must wait for each event on the I²C 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 I²C interrupt before data is written to the I2CxTRN register.
19.15 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 dsPIC33F/PIC24H 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\textsuperscript{2}C™) module include the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Application Note #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of the SSP Module in the I\textsuperscript{2}C™ Multi-Master Environment</td>
<td>AN578</td>
</tr>
<tr>
<td>Using the PIC\textsuperscript{®} Devices’ SSP and MSSP Modules for Slave I\textsuperscript{2}C™ Communication</td>
<td>AN734</td>
</tr>
<tr>
<td>Using the PICmicro\textsuperscript{®} MSSP Module for Master I\textsuperscript{2}C™ Communications</td>
<td>AN735</td>
</tr>
<tr>
<td>An I\textsuperscript{2}C™ Network Protocol for Environmental Monitoring</td>
<td>AN736</td>
</tr>
</tbody>
</table>

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the dsPIC33F/PIC24H family of devices.
19.16 REVISION HISTORY

Revision A (February 2007)
This is the initial released version of this document.

Revision B (August 2008)
This revision includes the following corrections and updates:
- Updated bit definitions for the ACKSTAT bit (I2CxSTAT<15>) and the D/A bit (I2CxSTAT<5>) in Register 19-2.
- Updated the I2CBRG denominator from 1,111,111 to 10,000,000 in Equation 19-1.
- Updated the I2C clock rate values in Table 19-1, removed the table notes and added a general note just after the table.
- Updated the last two paragraphs in 19.3 “Control and Status Registers” to clarify the shift of matching address bytes in the I2CxRSR register to the I2CxRCV register.
- Updated “” to clarify that the master function is enabled when the SEN bit is set, and when data is loaded into the I2CxTRN register.
- Several sections were updated to clarify NACK status in Slave mode. The affected sections are:
  - 19.4.2 “I2C Interrupts”
  - 19.7.5 “Sending Data to a Master Device”
  - Figure 19-28 through Figure 19-31
- The IPMIEN bit was incorrectly described as the Intelligent Peripheral Management Interface Enable bit. All occurrences have been updated to Intelligent Platform Management Interface bit.
- Updated 19.9.2 “Sleep Mode in Master Mode” to clarify what occurs when entering Sleep mode while transmitting.
- Updated the Slave Message RBF status bit information in Figure 19-26 through Figure 19-31.
- Additional minor corrections such as language and formatting updates are incorporated throughout the document.

Revision C (November 2009)
This revision includes the following corrections and updates:
- The document was updated to include the PIC24H families of devices
- Added Note 1 to the IPMIEN bit in the I2CxCON register (Register 19-1)
- Updated the bit status to HSC (Hardware Set/Cleared) for the P and S bits in the I2CxSTAT register (Register 19-2)
- Updated the BRG Reload Value Calculation (Equation 19-1)
- Added a shaded note on the SDA and SCL pins to 19.2 “I2C Bus Characteristics” and 19.8 “Connection Considerations for I2C Bus”
- Added a shaded note after the first paragraph in 19.5 “Communicating as a Master in a Single-Master Environment” and 19.6 “Communicating as a Master in a Multi-Master Environment”
- Removed the Actual F SCL column, added the PGD column, and updated the Decimal and Hexadecimal values in the I2C Clock Rates table (Table 19-1)
- Added a sentence to the end of the second paragraph and added a shaded note in 19.7.3.6 “Receiving All Addresses (IPMI Operation)”
- Added a shaded note to the first paragraph of 19.7.5 “Sending Data to a Master Device”
- Updated the last sentence of the first paragraph in 19.7.5.2 “Example Messages of Slave Transmission” to clarify that an interrupt will be generated.
- Added a shaded note to 19.9.1 “Sleep Mode in Slave Mode”
Revision D (April 2011)

This revision includes the following updates:

- Equations:
  - Updated Equation 19-2, Equation 19-3 and Equation 19-4
- Figures:
  - Updated Figure 19-15
- Notes:
  - Removed the note in 19.2 “I2C Bus Characteristics”
  - Added Note 2 in 19.5.2.1 “Sending a 7-bit Address to the Slave”
  - Added Note 2 in 19.5.2.2 “Sending a 10-bit Address to the Slave”
  - Added a note in 19.7.5.1 “Wait States During Slave Transmissions”
  - Replaced the existing note in 19.8 “Connection Considerations for I2C Bus”
  - Added Note 1 in Table 19-6
- Registers:
  - Updated the bit 12 description in Register 19-1
- Sections:
  - Updated 19.8 “Connection Considerations for I2C Bus”
  - Additional minor corrections such as language and formatting updates were incorporated throughout the document

Revision E (September 2011)

This revision includes the following updates:

- Figures:
  - Added Figure 19-33
- Sections:
  - Removed reference to dsPIC30F documentation in 19.2.2.2 “Address Slave”
  - Added 19.12 “Constant Current Source”
- Additional minor corrections such as language and formatting updates were incorporated throughout the document
Note the following details of the code protection feature on Microchip devices:

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

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

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

Trademarks
The Microchip name and logo, the Microchip logo, dsPIC, KEELOG, KEELOG logo, MPLAB, PIC, PICmicro, PICSTART, PIC² logo, rPIC and Uni/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.
Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, P ICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rFLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.
All other trademarks mentioned herein are property of their respective companies.
Printed on recycled paper.
ISBN: 978-1-61341-633-4
# Worldwide Sales and Service

<table>
<thead>
<tr>
<th>AMERICAS</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>EUROPE</th>
</tr>
</thead>
</table>
| Corporate Office  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
http://www.microchip.com/support  
Web Address:  
www.microchip.com | Asia Pacific Office  
Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2401-1200  
Fax: 852-2401-3431 | India - Bangalore  
Tel: 91-80-3090-4444  
Fax: 91-80-3090-4123 | Austria - Wels  
Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393 |
| Atlanta  
Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455 | China - Beijing  
Tel: 86-10-8569-7000  
Fax: 86-10-8528-2104 | India - New Delhi  
Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632 | Denmark - Copenhagen  
Tel: 45-4450-2828  
Fax: 45-4485-2829 |
| Boston  
Westborough, MA  
Tel: 774-760-0087  
Fax: 774-760-0088 | China - Chengdu  
Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889 | India - Pune  
Tel: 91-20-2566-1512  
Fax: 91-20-2566-1513 | France - Paris  
Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79 |
| Chicago  
Itasca, IL  
Tel: 630-285-0071  
Fax: 630-285-0075 | China - Chongqing  
Tel: 86-23-9890-9588  
Fax: 86-23-9890-9500 | Japan - Yokohama  
Tel: 81-45-471-6166  
Fax: 81-45-471-6122 | Germany - Munich  
Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44 |
| Cleveland  
Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643 | China - Hangzhou  
Tel: 86-571-2819-3187  
Fax: 86-571-2819-3189 | Korea - Daegu  
Tel: 82-53-744-4301  
Fax: 82-53-744-4302 | Italy - Milan  
Tel: 39-0331-742611  
Fax: 39-0331-466781 |
| Dallas  
Addison, TX  
Tel: 972-818-7423  
Fax: 972-818-2924 | China - Hong Kong SAR  
Tel: 852-2401-1200  
Fax: 852-2401-3431 | Korea - Seoul  
Tel: 82-2-554-7200  
Fax: 82-2-558-5932 or 82-2-558-5934 | Netherlands - Drunen  
Tel: 31-416-690399  
Fax: 31-416-690340 |
| Detroit  
Farmington Hills, MI  
Tel: 248-538-2250  
Fax: 248-538-2280 | China - Nanjing  
Tel: 86-25-8473-2460  
Fax: 86-25-8473-2470 | Malaysia - Kuala Lumpur  
Tel: 60-3-6201-9857  
Fax: 60-3-6201-9859 | Spain - Madrid  
Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91 |
| Indianapolis  
Noblesville, IN  
Tel: 317-773-8323  
Fax: 317-773-5453 | China - Qingdao  
Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205 | Malaysia - Penang  
Tel: 60-4-227-8870  
Fax: 60-4-227-4068 | UK - Wokingham  
Tel: 44-118-921-5869  
Fax: 44-118-921-5820 |
| Los Angeles  
Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608 | China - Shanghai  
Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066 | Philippines - Manila  
Tel: 63-2-634-9065  
Fax: 63-2-634-9069 | |
| Santa Clara  
Santa Clara, CA  
Tel: 408-961-6444  
Fax: 408-961-6445 | China - Shenyang  
Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393 | Singapore  
Tel: 65-6334-8870  
Fax: 65-6334-8850 | |
| Toronto  
Mississauga, Ontario, Canada  
Tel: 905-673-0699  
Fax: 905-673-6509 | China - Shenzhen  
Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760 | Taiwan - Hsin Chu  
Tel: 886-3-5778-366  
Fax: 886-3-5770-955 | |
|  | China - Wuhan  
Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118 | Taiwan - Kaohsiung  
Tel: 886-7-536-4818  
Fax: 886-7-330-9305 | Taiwan - Taipei  
Tel: 886-2-2500-6610  
Fax: 886-2-2508-0102 | |
|  | China - Xian  
Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256 | Taiwan - Bangkok  
Tel: 66-2-694-1351  
Fax: 66-2-694-1350 | Thailand - Bangkok  
Tel: 66-2-694-1351  
Fax: 66-2-694-1350 |