Section 21. Inter-Integrated Circuit™ (I^2C™)

HIGHLIGHTS

This section of the manual contains the following major topics:

21.1 Overview

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

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

- Where the dsPIC30F acts as a Slave Device
- Where the dsPIC30F acts as a Master Device in a Single Master System
  (Slave may also be active)
- Where the dsPIC30F acts as a Master/Slave Device in a Multi-Master System
  (Bus collision detection and arbitration available)

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

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

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

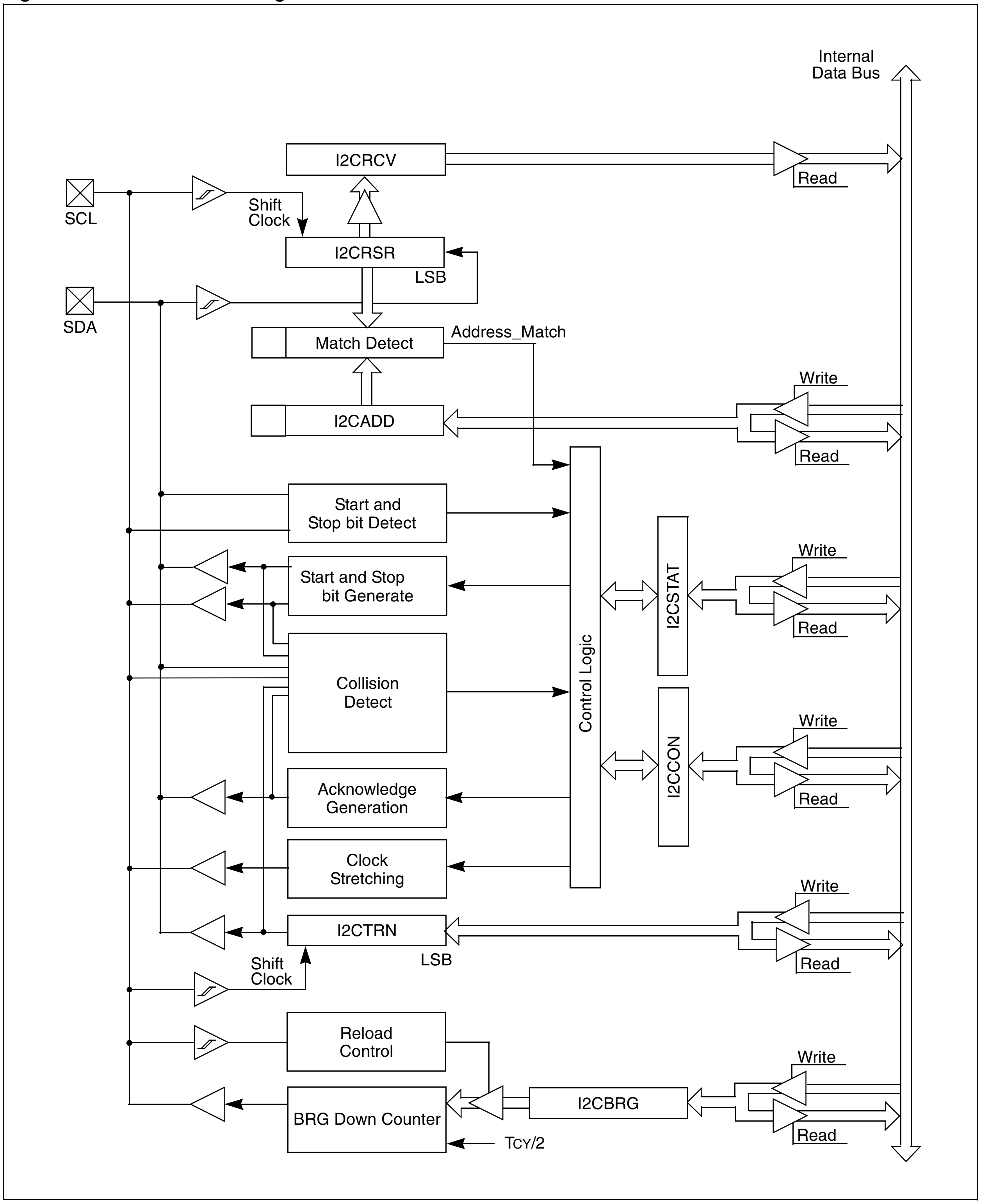
The I^2C module contains a baud rate generator. The I^2C baud rate generator does not consume other timer resources in the device.

21.1.1 Module Features

- Independent Master and Slave logic
- Multi-Master support. No messages lost in arbitration.
- Detects 7-bit and 10-bit device addresses
- Detects general call addresses as defined in the I^2C protocol
- Bus Repeater mode. Accept all messages as a slave regardless of the address.
- Automatic SCL clock stretching provides delays for the processor to respond to a slave data request.
- Supports 100 kHz and 400 kHz bus specifications.

Figure 21-1 shows the I^2C module block diagram.

Figure 21-1: I^2C™ Block Diagram



21.2 I^{2}C Bus Characteristics

The I^{2}C bus is a two-wire serial interface. Figure 21-2 is a schematic of a typical I^{2}C connection between the dsPIC30F device and a 24LC256 I^{2}C serial EEPROM.

The I^{2}C interface employs a comprehensive protocol to ensure reliable transmission and reception of data. When communicating, one device is the “master” which initiates transfer on the bus and generates the clock signals to permit that transfer, while the other device(s) acts as the “slave” responding to the transfer. The clock line, “SCL”, is output from the master and input to the slave, although occasionally the slave drives the SCL line. The data line, “SDA”, may be output and input from both the master and slave.

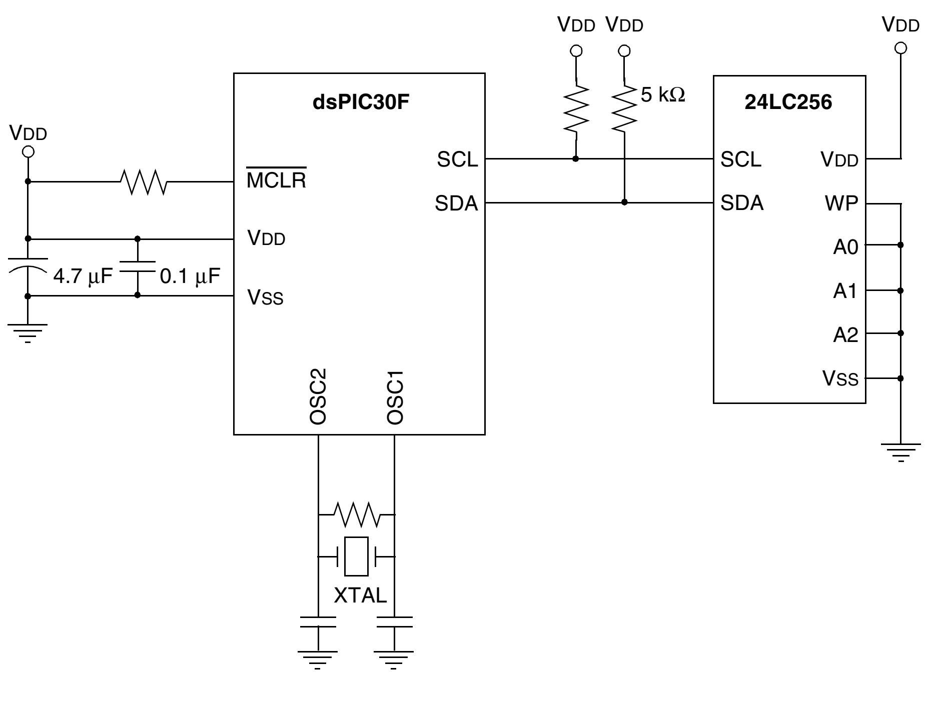
Because the SDA and SCL lines are bidirectional, the output stages of the devices driving the SDA and SCL lines must have an open drain in order to perform the wired-AND function of the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down.

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

- Master-transmitter and Slave-receiver
- Slave-transmitter and Master-receiver

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

Figure 21-2: Typical I^{2}C™ Interconnection Block Diagram



21.2.1 Bus Protocol

The following I2C bus protocol has been defined:

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

Accordingly, the following bus conditions have been defined (Figure 21-3).

21.2.1.1 Start Data Transfer (S)

After a bus Idle state, a HIGH-to-LOW transition of the SDA line while the clock (SCL) is HIGH determines a Start condition. All data transfers must be preceded by a Start condition.

21.2.1.2 Stop Data Transfer (P)

A LOW-to-HIGH transition of the SDA line while the clock (SCL) is HIGH determines a Stop condition. All data transfers must end with a Stop condition.

21.2.1.3 Repeated Start (R)

After a WAIT state, a HIGH-to-LOW transition of the SDA line while the clock (SCL) is HIGH determines a Repeated Start condition. Repeated Starts allow a master to change bus direction without relinquishing control of the bus.

21.2.1.4 Data Valid (D)

The state of the SDA line represents valid data when, after a Start condition, the SDA line is stable for the duration of the HIGH period of the clock signal. There is one bit of data per SCL clock.

21.2.1.5 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 SDA line low for an ACK or release the SDA line for a NACK. The Acknowledge is a one-bit period, using one SCL clock.

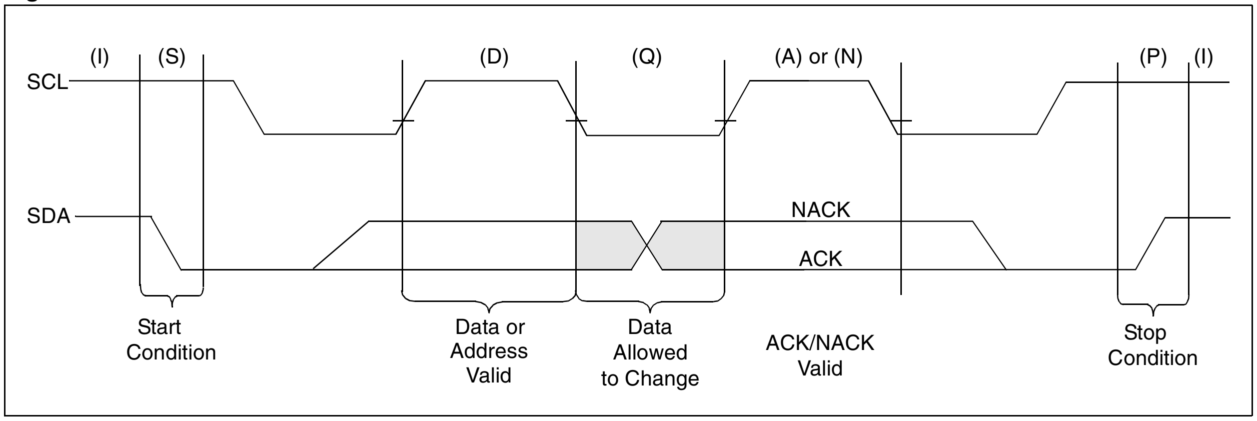
21.2.1.6 WAIT/Data Invalid (Q)

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

21.2.1.7 Bus Idle (I)

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

Figure 21-3: I^2C™ Bus Protocol States

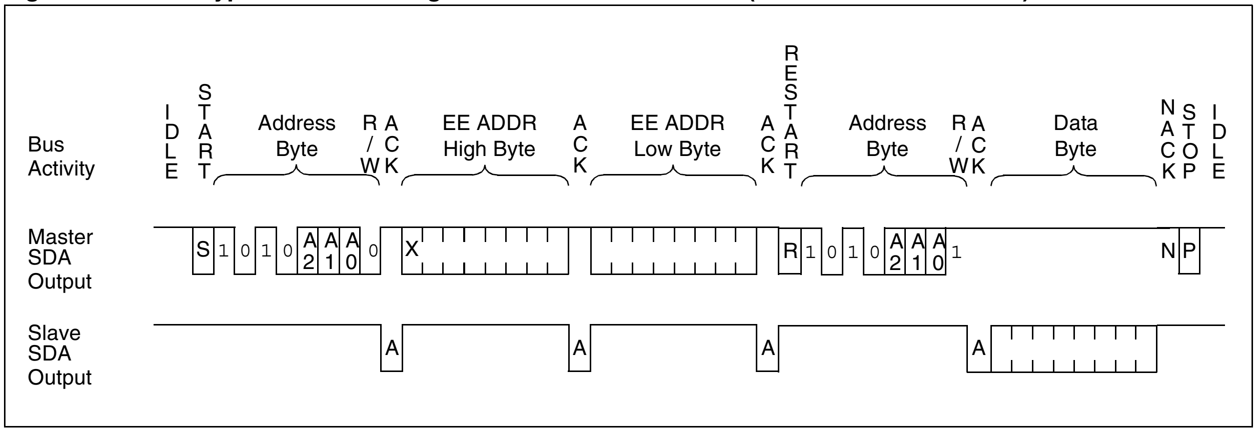


21.2.2 Message Protocol

A typical I^2C message is shown in Figure 21-4. In this example, the message will read a specified byte from a 24LC256 I^2C serial EEPROM. The dsPIC30F device will act as the master and the 24LC256 device will act as the slave.

Figure 21-4 indicates the data as driven by the master device and the data as driven by the slave device, remembering that the combined SDA line is a wired-AND of the master and slave data. The master device controls and sequences the protocol. The slave device will only drive the bus at specifically determined times.

Figure 21-4: A Typical I^2C™ Message: Read of Serial EEPROM (Random Address Mode)



21.2.2.1 Start Message

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

21.2.2.2 Address Slave

In the figure, the first byte is the device address byte that must be the first part of any I^2C message. It contains a device address and a $R/\overline{W}$ bit. Refer to **“Section 26. Appendix”** for additional information on Address Byte formats. Note that $R/\overline{W}$ = `0` for this first address byte, indicating that the master will be a transmitter and the slave will be a receiver.

21.2.2.3 Slave Acknowledge

The receiving device is obliged to generate an Acknowledge signal, “ACK”, after the reception of each byte. The master device must generate an extra SCL clock, which is associated with this Acknowledge bit.

21.2.2.4 Master Transmit

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

21.2.2.5 Repeated Start

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

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

21.2.2.6 Slave Reply

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

21.2.2.7 Master Acknowledge

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

21.2.2.8 Stop Message

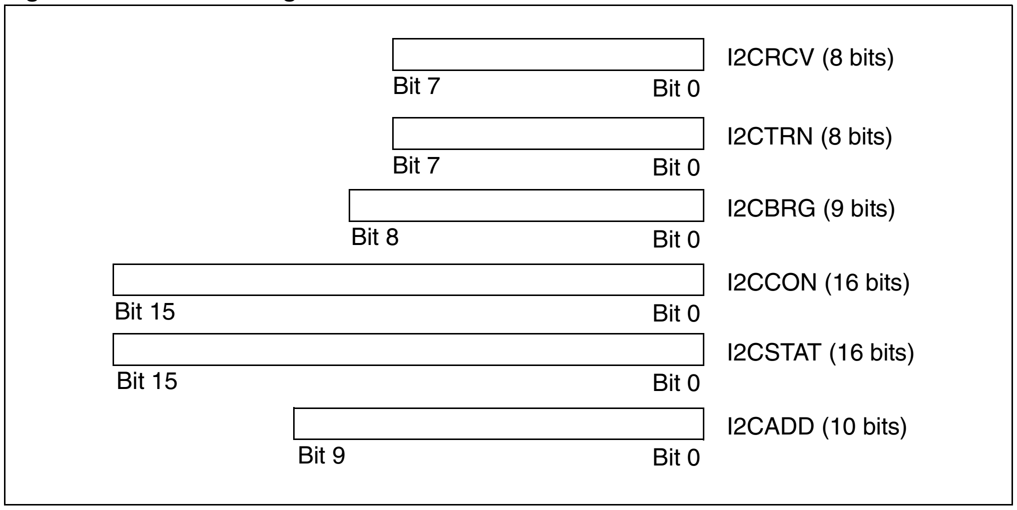
The master sends Stop to terminate the message and return the bus to an Idle state.

21.3 Control and Status Registers

The I^2C module has six user-accessible registers for I^2C operation. The registers are accessible in either Byte or Word mode. The registers are shown in Figure 21-5 and listed below:

- Control Register (I2CCON): This register allows control of the I^2C operation.
- Status Register (I2CSTAT): This register contains status flags indicating the module state during I^2C operation.
- Receive Buffer Register (I2CRCV): This is the buffer register from which data bytes can be read. The I2CRCV register is a read only register.
- Transmit Register (I2CTRN): This is the transmit register; bytes are written to this register during a transmit operation. The I2CTRN register is a read/write register.
- Address Register (I2CADD): The I2CADD register holds the slave device address.
- Baud Rate Generator Reload Register (I2CBRG): Holds the baud rate generator reload value for the I^2C module baud rate generator.

Figure 21-5: I^2C™ Programmer’s Model



Register 21-1 and Register 21-2 define the I^2C module Control and Status registers, I2CCON and I2CSTAT.

The I2CTRN is the register to which transmit data is written. This register is used when the module operates as a master transmitting data to the slave or as a slave sending reply data to the master. As the message progresses, the I2CTRN register shifts out the individual bits. Because of this, the I2CTRN may not be written to unless the bus is Idle. The I2CTRN may be reloaded while the current data is transmitting.

Data being received by either the master or the slave is shifted into a non-accessible Shift register called I2CRSR. When a complete byte is received, the byte transfers to the I2CRCV register. In receive operations, the I2CRSR and I2CRCV create a double-buffered receiver. This allows reception of the next byte to begin before reading the current byte of received data.

If the module receives another complete byte before the software reads the previous byte from the I2CRCV register, a receiver overflow occurs and sets the I2COV (I2CCON<6>). The byte in the I2CRSR is lost.

The I2CADD register holds the slave device address. In 10-bit mode, all bits are relevant. In 7-bit addressing mode, only I2CADD<6:0> are relevant. The A10M (I2CCON<10>) specifies the expected mode of the slave address.

Register 21-1: I2CCON: I^{2}C™ Control Register

Upper Byte:							
R/W-0	U-0	R/W-0	R/W-1 HC	R/W-0	R/W-0	R/W-0	R/W-0
I2CEN	—	I2CSIDL	SCLREL	IPMIEN	A10M	DISSLW	SMEN
bit 15							bit 8

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

bit 15 **I2CEN:** I^{2}C Enable bit
1 = Enables the I^{2}C module and configures the SDA and SCL pins as serial port pins
0 = Disables I^{2}C module. All I^{2}C pins are controlled by port functions.

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

bit 13 **I2CSIDL:** Stop in Idle Mode bit
1 = Discontinue module operation when device enters an Idle mode
0 = Continue module operation in Idle mode

bit 12 **SCLREL:** SCL Release Control bit (when operating as I^{2}C Slave)
1 = Release SCL clock
0 = Hold SCL clock low (clock stretch)
If STREN = 1:
Bit is R/W (i.e., software may write '0' to initiate stretch and write '1' to release clock)
Hardware clear at beginning of slave transmission.
Hardware clear at end of slave reception.
If STREN = 0:
Bit is R/S (i.e., software may only write '1' to release clock)
Hardware clear at beginning of slave transmission.

bit 11 **IPMIEN:** Intelligent Peripheral Management Interface (IPMI) Enable bit
1 = Enable IPMI Support mode. All addresses Acknowledged.
0 = IPMI mode not enabled

bit 10 **A10M:** 10-bit Slave Address bit
1 = I2CADD is a 10-bit slave address
0 = I2CADD is a 7-bit slave address

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

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

bit 7 **GCEN:** General Call Enable bit (when operating as I^{2}C slave)
1 = Enable interrupt when a general call address is received in the I2CRSR
(module is enabled for reception)
0 = General call address disabled

bit 6 **STREN:** SCL Clock Stretch Enable bit (when operating as I^{2}C slave)
Used in conjunction with SCLREL bit.
1 = Enable software or receive clock stretching
0 = Disable software or receive clock stretching

bit 5 **ACKDT:** Acknowledge Data bit (When operating as I^{2}C Master. Applicable during master receive.)
Value that will be transmitted when the software initiates an Acknowledge sequence.
1 = Send NACK during acknowledge
0 = Send ACK during acknowledge

Register 21-1: I2CCON: I^{2}C™ Control Register (Continued)

bit 4 **ACKEN:** Acknowledge Sequence Enable bit
(When operating as I^{2}C master. Applicable during master receive.)
`1` = Initiate Acknowledge sequence on SDA and SCL 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 (when operating as I^{2}C master)
`1` = Enables Receive mode for I^{2}C
Hardware clear at end eighth bit of master receive data byte.
`0` = Receive sequence not in progress

bit 2 **PEN:** Stop Condition Enable bit (when operating as I^{2}C master)
`1` = Initiate Stop condition on SDA and SCL pins
Hardware clear at end of master Stop sequence.
`0` = Stop condition not in progress

bit 1 **RSEN:** Repeated Start Condition Enabled bit (when operating as I^{2}C master)
`1` = Initiate Repeated Start condition on SDA and SCL pins
Hardware clear at end of master Repeated Start sequence.
`0` = Repeated Start condition not in progress

bit 0 **SEN:** Start Condition Enabled bit (when operating as I^{2}C master)
`1` = Initiate Start condition on SDA and SCL pins
Hardware clear at end of master Start sequence.
`0` = Start condition not in progress

Legend:		
R = Readable	C = Clearable bit	U = Unimplemented bit, read as ‘0’
W = Writable	HS = Set by Hardware	S = Settable bit
HC = Cleared by Hardware	‘0’ = Bit cleared at POR	x = Bit is unknown at POR
‘1’ = Bit is set at POR		

Register 21-2: I2CSTAT: I^2C™ Status Register

Upper Byte:

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

Lower Byte:

R/C-0 HS	R/W-0 HS	R-0 HS, HC	R/C-0 HS, HC	R/C-0 HS, HC	R-0 HS, HC	R-0 HS, HC	R-0 HS, HC
IWCOL	I2COV	D_A	P	S	R_W	RBF	TBF
bit 7							bit 0

bit 15 **ACKSTAT:** Acknowledge Status bit
(When operating as I^2C master. Applicable to master transmit operation.)
1 = NACK received from slave
0 = ACK received from slave
Hardware set or clear at end of slave Acknowledge.

bit 14 **TRSTAT:** Transmit Status bit
(When operating as I^2C master. Applicable to master 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'

bit 10 **BCL:** Master Bus Collision Detect bit
1 = A bus collision has been detected during a master operation
0 = No collision
Hardware set at detection of bus collision.

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

bit 8 **ADD10:** 10-bit Address Status bit
1 = 10-bit address was matched
0 = 10-bit address was not matched
Hardware set at match of 2nd byte of matched 10-bit address.
Hardware clear at Stop detection.

bit 7 **IWCOL:** Write Collision Detect bit
1 = An attempt to write the I2CTRN register failed because the I^2C module is busy
0 = No collision
Hardware set at occurrence of write to I2CTRN while busy (cleared by software).

bit 6 **I2COV:** Receive Overflow Flag bit
1 = A byte was received while the I2CRCV register is still holding the previous byte
0 = No overflow
Hardware set at attempt to transfer I2CRSR to I2CRCV (cleared by software).

bit 5 **D_A:** Data/Address bit (when operating as I^2C slave)
1 = Indicates that the last byte received was data
0 = Indicates that the last byte received was device address
Hardware clear at device address match.
Hardware set by write to I2CTRN or by reception of slave byte.

Register 21-2: I2CSTAT: I^{2}C™ Status Register (Continued)

bit 4	**P:** Stop bit `1` = Indicates that a Stop bit has been detected last `0` = Stop bit was not detected last Hardware set or clear when Start, Repeated Start or Stop detected.
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 bit Information (when operating as I^{2}C slave) `1` = Read - indicates data transfer is output from slave `0` = Write - indicates data transfer is input to slave Hardware set or clear after reception of I^{2}C device address byte.
bit 1	**RBF:** Receive Buffer Full Status bit `1` = Receive complete, I2CRCV is full `0` = Receive not complete, I2CRCV is empty Hardware set when I2CRCV written with received byte. Hardware clear when software reads I2CRCV.
bit 0	**TBF:** Transmit Buffer Full Status bit `1` = Transmit in progress, I2CTRN is full `0` = Transmit complete, I2CTRN is empty Hardware set when software writes I2CTRN. Hardware clear at completion of data transmission.

Legend:		
R = Readable	W = Writable	C = Clearable bit
HC = Cleared by Hardware	HS = Set by Hardware	U = Unimplemented bit, read as ‘0’
‘1’ = Bit is set at POR	‘0’ = Bit cleared at POR	x = Bit is unknown at POR

21.4 Enabling I^2C Operation

The module is enabled by setting the I2CEN (I2CCON<15>) bit.

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

When initially enabled, the module will release SDA and SCL pins, putting the bus into the Idle state. The master functions will remain in the Idle state unless software sets a control bit to initiate a master event. The slave functions will begin to monitor the bus. If the slave logic detects a Start event and a valid address on the bus, the slave logic will begin a slave transaction.

21.4.1 Enabling I^2C I/O

Two pins are used for bus operation. These are the SCL pin, which is the clock, and the SDA 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 SDA and SCL pins. The module software need not be concerned with the state of the port I/O of the pins, the module overrides the port state and direction. At initialization, the pins are tri-state (released).

21.4.2 I^2C Interrupts

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

The master interrupt is called MI2CIF and is activated on completion of a master message event.

The following events generate the MI2CIF interrupt.

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

The slave interrupt is called SI2CIF and is activated on detection of a message directed to the slave.

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

21.4.3 Setting Baud Rate when Operating as a Bus Master

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

The reload value for the baud rate generator is the I2CBRG register, as shown in Figure 21-6. When the baud rate generator is loaded with this value, the generator counts down to '0' and stops until another reload has taken place. The generator count is decremented twice per instruction cycle (T_{CY}). The baud rate generator is reloaded automatically on baud rate restart. For example, if clock synchronization is taking place, the baud rate generator will be reloaded when the SCL pin is sampled high.

Note: I2CBRG value of `0x0` is not supported.

To compute the baud rate generator reload value, use the following equation.

Equation 21-1:

$$I2CBRG = \left(\frac{F_{CY}}{F_{SCL}} - \frac{F_{CY}}{1,111,111} \right) - 1$$

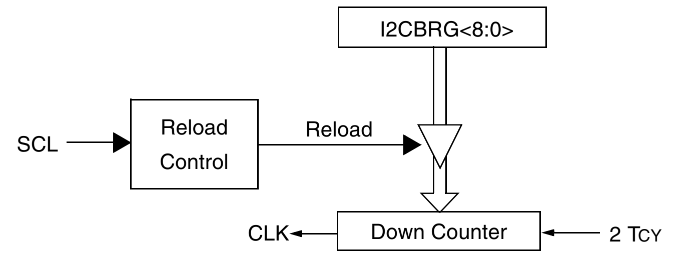
Table 21-1: I^2C™ Clock Rates

Required System F_{SCL}	F_{CY}	I2CBRG Decimal	I2CBRG HEX	Actual F_{SCL}
100 kHz	30 MHz	272	`0x110`	100 kHz
100 kHz	20 MHz	181	`0x0B5`	100 kHz
100 kHz	1 MHz	8	`0x008`	101 kHz
400 kHz	10 MHz	15	`0x00F`	400 kHz
400 kHz	5 MHz	7	`0x007`	400 kHz
400 kHz	1 MHz	1	`0x001`	345 kHz**
1 MHz*	11 MHz	1	`0x001`	1 MHz*
1 MHz	1 MHz	0	`0x000 (invalid)`	1 MHz

*F_{CY} = 11 MHz is the minimum input clock frequency to have F_{SCL} = 1 MHz.

** This is closest value to 400 kHz for this value of F_{CY}.

Figure 21-6: Baud Rate Generator Block Diagram



21.5 Communicating as a Master in a Single Master Environment

Typical operation of the I^2C module in a system is using the I^2C to communicate with an I^2C peripheral, such as an I^2C serial memory. In an I^2C system, the master controls the sequence of all data communication on the bus. In this example, the dsPIC30F and its I^2C module have the role of the single master in the system. As the single master, it is responsible for generating the SCL clock and controlling the message protocol.

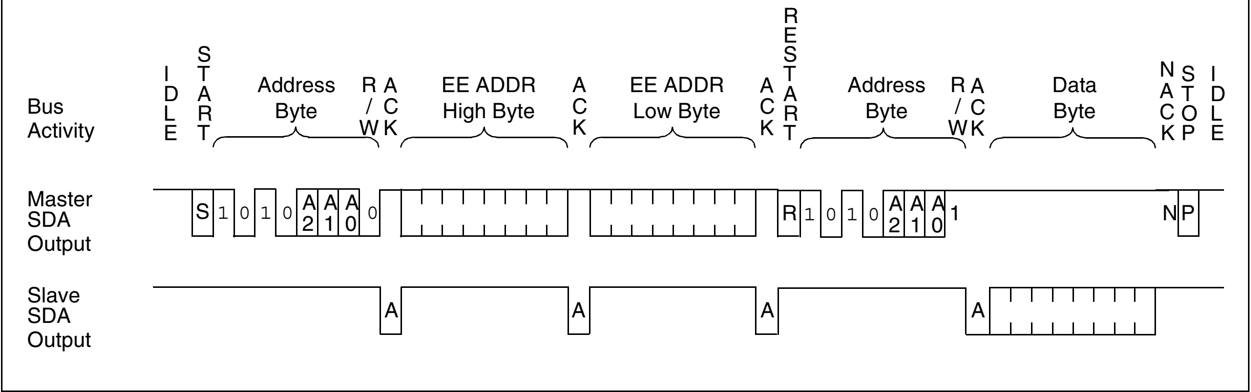
In the I^2C module, the module controls individual portions of the I^2C message protocol, however, sequencing of the components of the protocol to construct a complete message is a software task.

For example, a typical operation in a single master environment may be to read a byte from an I^2C serial EEPROM. This example message is depicted in Figure 21-7.

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

1. Assert a Start condition on SDA and SCL.
2. Send the I^2C device address byte to the slave with a write indication.
3. Wait for and verify an Acknowledge from the slave.
4. Send the serial memory address high byte to the slave.
5. Wait for and verify an Acknowledge from the slave.
6. Send the serial memory address low byte to the slave.
7. Wait for and verify an Acknowledge from the slave.
8. Assert a Repeated Start condition on SDA and SCL.
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 SDA and SCL.

Figure 21-7: A Typical I^2C™ Message: Read Of Serial EEPROM (Random Address Mode)



The I^2C module supports Master mode communication with the inclusion of Start and Stop generators, data byte transmission, data byte reception, Acknowledge generator and a baud rate generator.

Generally, the software will write to a control register to start a particular step, then wait for an interrupt or poll status to wait for completion.

Subsequent sub-sections detail each of these operations

21.5.1 Generating Start Bus Event

To initiate a Start event, the software sets the Start enable bit, SEN (I2CCON<0>). Prior to setting the Start bit, the software can check the P (I2CSTAT<4>) status bit to ensure that the bus is in an Idle state.

Figure 21-8 shows the timing of the Start condition.

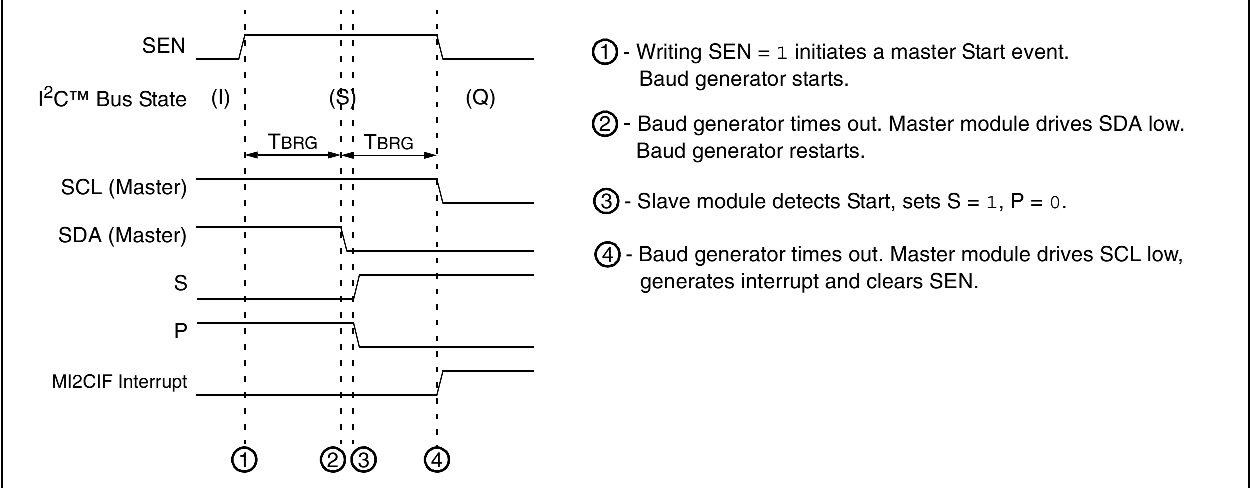
- Slave logic detects the Start condition, sets the S bit (I2CSTAT<3>) and clears the P bit (I2CSTAT<4>).
- SEN bit is automatically cleared at completion of the Start condition.
- MI2CIF interrupt generated at completion of the Start condition.
- After Start condition, SDA line and SCL line are left low (Q state).

21.5.1.1 IWCOL Status Flag

If the software writes the I2CTRN when a Start sequence is in progress, then IWCOL is set and the contents of the transmit buffer are ignored.

Note: Because queueing of events is not allowed, writing to the lower 5 bits of I2CCON is disabled until the Start condition is complete.

Figure 21-8: Master Start Timing Diagram


SEN
I2C™ Bus State
(I)
(S)
(Q)
TBRG
TBRG
SCL (Master)
SDA (Master)
S
P
MI2CIF Interrupt
1
2
3
4
1 - Writing SEN = 1 initiates a master Start event. Baud generator starts.
2 - Baud generator times out. Master module drives SDA low. Baud generator restarts.
3 - Slave module detects Start, sets S = 1, P = 0.
4 - Baud generator times out. Master module drives SCL low, generates interrupt and clears SEN.

21.5.2 Sending Data to a Slave Device

Transmission of a data byte, a 7-bit device address byte or the second byte of a 10-bit address, is accomplished by simply writing the appropriate value to the I2CTRN register. Loading this register will start the following process:

- The software loads the I2CTRN with the data byte to transmit.
- Writing I2CTRN sets the buffer full flag bit, TBF (I2CSTAT<0>).
- The data byte is shifted out the SDA pin until all 8 bits are transmitted. Each bit of address/data will be shifted out onto the SDA pin after the falling edge of SCL.
- On the ninth SCL clock, the module shifts in the ACK bit from the slave device and writes its value into the ACKSTAT bit (I2CCON<15>).
- The module generates the MI2CIF interrupt at the end of the ninth SCL clock cycle.

Note that the module does not generate or validate the data bytes. The contents and usage of the byte is dependant on the state of the message protocol maintained by the software.

21.5.2.1 Sending a 7-bit Address to the Slave

Sending a 7-bit device address involves sending 1 byte to the slave. A 7-bit address byte must contain the 7 bits of I^2C device address and a R/$\overline{W}$ bit that defines if the message will be a write to the slave (master transmission and slave receiver) or a read from the slave (slave transmission and master receiver).

21.5.2.2 Sending a 10-bit Address to the Slave

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

21.5.2.3 Receiving Acknowledge from the Slave

On the falling edge of the eighth SCL clock, the TBF bit is cleared and the master will de-assert the SDA pin allowing the slave to respond with an Acknowledge. The master will then generate a ninth SCL clock.

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

The status of $\overline{ACK}$ is written into the Acknowledge status bit, ACKSTAT (I2CSTAT<15>), on the falling edge of the ninth SCL clock. After the ninth SCL clock, the module generates the MI2CIF interrupt and enters an Idle state until the next data byte is loaded into I2CTRN.

21.5.2.4 ACKSTAT Status Flag

The ACKSTAT bit (I2CCON<15>) is cleared when the slave has sent an Acknowledge ($\overline{ACK}$ = `0`), and is set when the slave does not Acknowledge ($\overline{ACK}$ = `1`).

21.5.2.5 TBF Status Flag

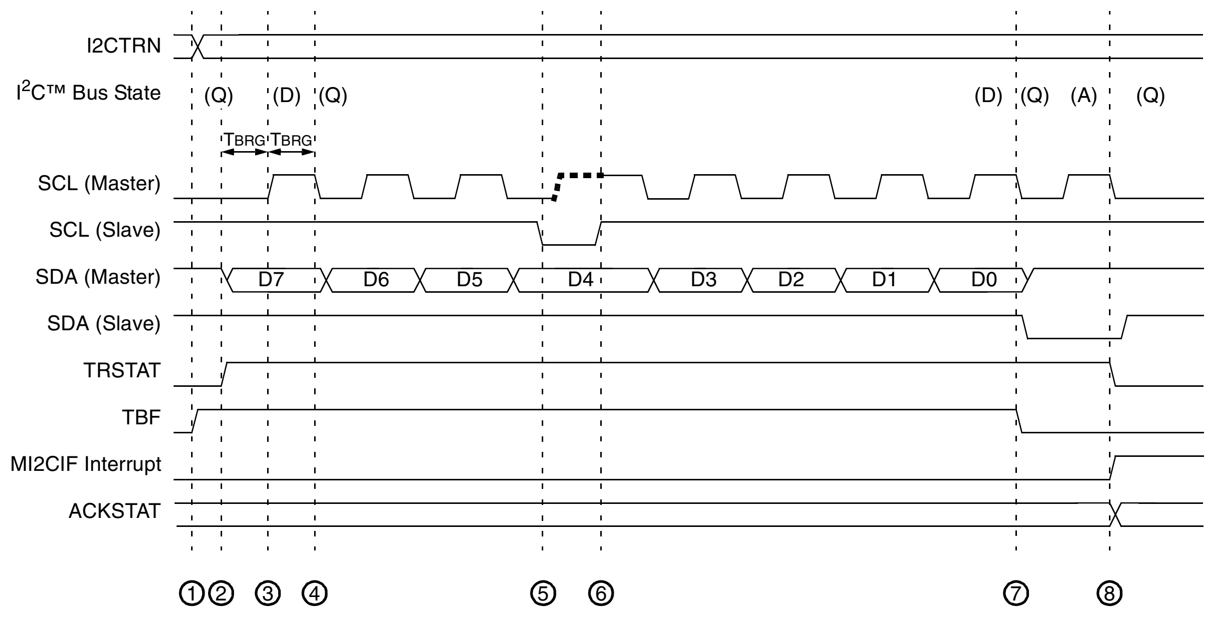
When transmitting, the TBF bit (I2CSTAT<0>) is set when the CPU writes to I2CTRN and is cleared when all 8 bits are shifted out.

21.5.2.6 IWCOL Status Flag

If the software writes the I2CTRN when a transmit is already in progress (i.e., the module is still shifting out a data byte), then IWCOL is set and the contents of the buffer are ignored. IWCOL must be cleared in software.

Note: Because queueing of events is not allowed, writing to the lower 5 bits of I2CCON is disabled until the transmit condition is complete.

Figure 21-9: Master Transmission Timing Diagram



① - Writing the I2CTRN register will start a master transmission event. TBF bit is set.

② - Baud generator starts. The MSB of the I2CTRN drives SDA. SCL remains low. TRSTAT bit is set.

③ - Baud generator times out. SCL released. Baud generator restarts.

④ - Baud generator times out. SCL driven low. After SCL detected low, next bit of I2CTRN drives SDA.

⑤ - While SCL is low, the slave can also pull SCL low to initiate a WAIT (clock stretch).

⑥ - Master has already released SCL, and slave can release to end WAIT. Baud generator restarts.

⑦ - At falling edge of 8th SCL clock, master releases SDA. TBF bit is cleared. Slave drives ACK/NACK.

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

21.5.3 Receiving Data from a Slave Device

Setting the receive enable bit, RCEN (I2CCON<3>), enables the master to receive data from a slave device.

Note:	The lower 5 bits of I2CCON must be ‘0’ before attempting to set the RCEN bit. This ensures the master logic is inactive.

The master logic begins to generate clocks and before each falling edge of the SCL, SDA line is sampled and data is shifted into the I2CRSR.

After the falling edge of the eighth SCL clock:

- The RCEN bit is automatically cleared.
- The contents of the I2CRSR transfer into the I2CRCV.
- The RBF flag bit is set.
- The module generates the MI2CIF interrupt.

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

21.5.3.1 RBF Status Flag

When receiving data, the RBF bit is set when an device address or data byte is loaded into I2CRCV from I2CRSR. It is cleared when software reads the I2CRCV register.

21.5.3.2 I2COV Status Flag

If another byte is received in the I2CRSR while the RBF bit remains set and the previous byte remains in the I2CRCV register, the I2COV bit is set and the data in the I2CRSR is lost.

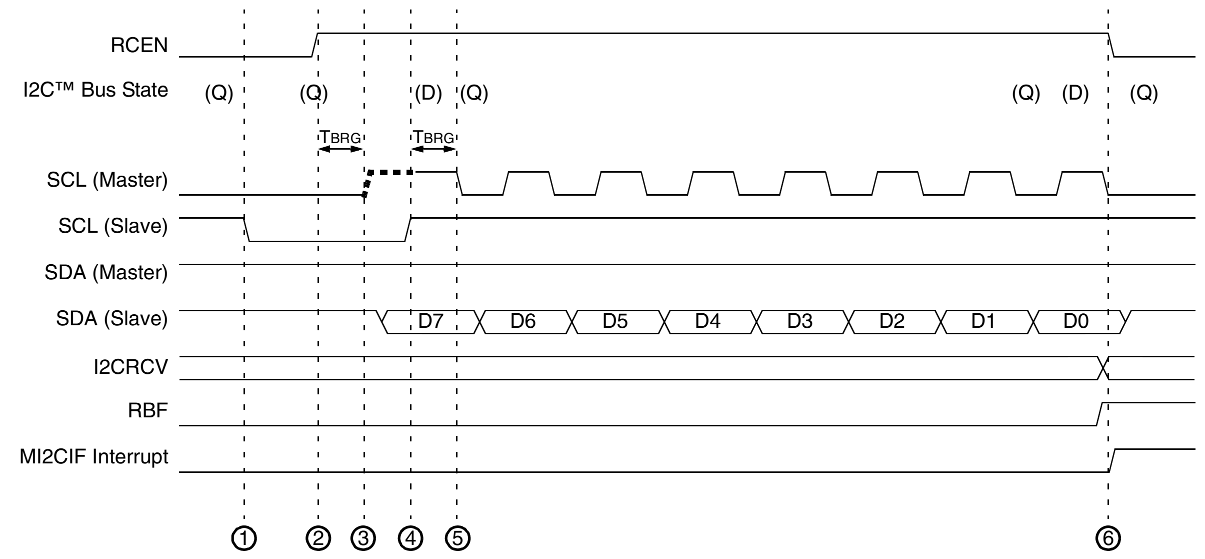
Leaving I2COV set does not inhibit further reception. If RBF is cleared by reading the I2CRCV, and the I2CRSR receives another byte, that byte will be transferred to the I2CRCV.

21.5.3.3 IWCOL Status Flag

If the software writes the I2CTRN when a receive is already in progress (i.e., I2CRSR is still shifting in a data byte), then the IWCOL bit is set and the contents of the buffer are ignored.

Note:	Since queueing of events is not allowed, writing to the lower 5 bits of I2CCON is disabled until the data reception condition is complete.

Figure 21-10: Master Reception Timing Diagram



1 - Typically, the slave can pull SCL low (clock stretch) to request a wait to prepare data response. The slave will drive MSB of data response on SDA when ready.

2 - Writing the RCEN bit will start a master reception event. The baud generator starts. SCL remains low.

3 - Baud generator times out. Master attempts to release SCL.

4 - When slave releases SCL, baud generator restarts.

5 - Baud generator times out. MSB of response shifted to I2CRSR. SCL driven low for next baud interval.

6 - At falling edge of 8th SCL clock, I2CRSR transferred to I2CRCV. Module clears RCEN bit. RBF bit is set. Master generates interrupt.

21.5.4 Acknowledge Generation

Setting the Acknowledge sequence enable bit, ACKEN (I2CCON<4>), enables generation of a master Acknowledge sequence.

Note:	The lower 5 bits of I2CCON must be '0' (master logic inactive) before attempting to set the ACKEN bit.

Figure 21-11 shows an $\overline{\text{ACK}}$ sequence and Figure 21-12 shows a NACK sequence. The Acknowledge data bit, ACKDT (I2CCON<5>), specifies $\overline{\text{ACK}}$ or NACK.

After two baud periods:

- The ACKEN bit is automatically cleared.
- The module generates the MI2CIF interrupt.

21.5.4.1 IWCOL Status Flag

If the software writes the I2CTRN when an Acknowledge sequence is in progress, then IWCOL is set and the contents of the buffer are ignored.

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of I2CCON is disabled until the Acknowledge condition is complete.

Figure 21-11: Master Acknowledge (ACK) Timing Diagram

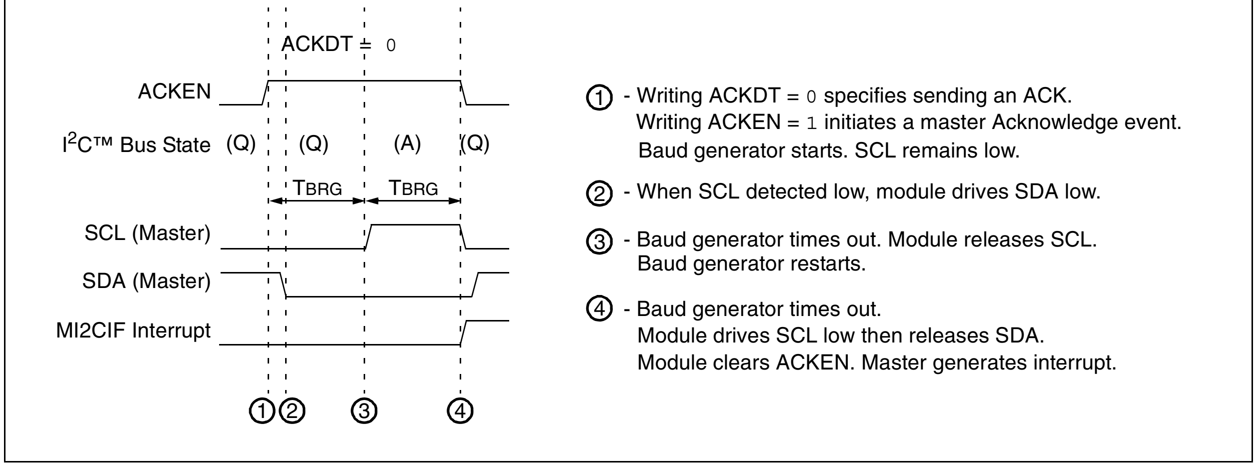
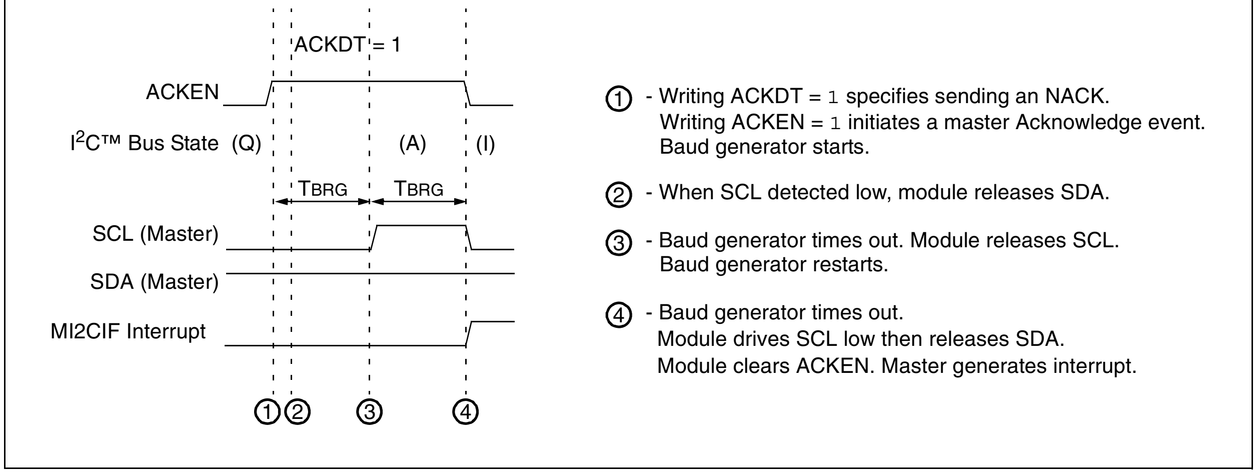

ACKDT = 0
ACKEN
I²C™ Bus State (Q) (Q) (A) (Q)
TBRG TBRG
SCL (Master)
SDA (Master)
MI2CIF Interrupt
① ② ③ ④
① - Writing ACKDT = 0 specifies sending an ACK.
Writing ACKEN = 1 initiates a master Acknowledge event.
Baud generator starts. SCL remains low.
② - When SCL detected low, module drives SDA low.
③ - Baud generator times out. Module releases SCL.
Baud generator restarts.
④ - Baud generator times out.
Module drives SCL low then releases SDA.
Module clears ACKEN. Master generates interrupt.

Figure 21-12: Master Not Acknowledge (NACK) Timing Diagram


ACKDT = 1
ACKEN
I²C™ Bus State (Q) (A) (I)
TBRG TBRG
SCL (Master)
SDA (Master)
MI2CIF Interrupt
① ② ③ ④
① - Writing ACKDT = 1 specifies sending an NACK.
Writing ACKEN = 1 initiates a master Acknowledge event.
Baud generator starts.
② - When SCL detected low, module releases SDA.
③ - Baud generator times out. Module releases SCL.
Baud generator restarts.
④ - Baud generator times out.
Module drives SCL low then releases SDA.
Module clears ACKEN. Master generates interrupt.

21.5.5 Generating Stop Bus Event

Setting the Stop sequence enable bit, PEN (I2CCON<2>), enables generation of a master Stop sequence.

Note:	The lower 5 bits of I2CCON must be ‘0’ (master logic inactive) before attempting to set the PEN bit.

When the PEN bit is set, the master generates the Stop sequence as shown in Figure 21-13.

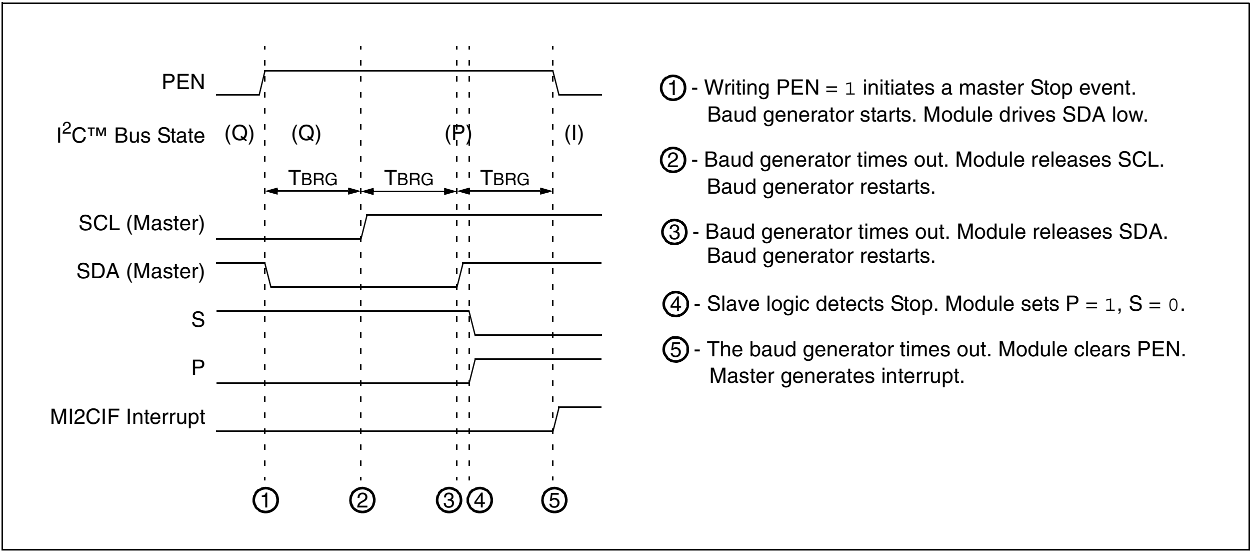
- The slave detects the Stop condition, sets the P bit (I2CSTAT<4>) and clears the S bit (I2CSTAT<3>).
- The PEN bit is automatically cleared.
- The module generates the MI2CIF interrupt.

21.5.5.1 IWCOL Status Flag

If the software writes the I2CTRN when a Stop sequence is in progress, then the IWCOL bit is set and the contents of the buffer are ignored.

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of I2CCON is disabled until the Stop condition is complete.

Figure 21-13: Master Stop Timing Diagram


PEN
I2C™ Bus State
(Q)
(Q)
(P)
(I)
TBRG
TBRG
TBRG
SCL (Master)
SDA (Master)
S
P
MI2CIF Interrupt
1
2
3
4
5
1 - Writing PEN = 1 initiates a master Stop event. Baud generator starts. Module drives SDA low.
2 - Baud generator times out. Module releases SCL. Baud generator restarts.
3 - Baud generator times out. Module releases SDA. Baud generator restarts.
4 - Slave logic detects Stop. Module sets P = 1, S = 0.
5 - The baud generator times out. Module clears PEN. Master generates interrupt.

21.5.6 Generating Repeated Start Bus Event

Setting the Repeated Start sequence enable bit, RSEN (I2CCON<1>), enables generation of a master Repeated Start sequence (see Figure 21-14).

Note:	The lower 5 bits of I2CCON must be ‘0’ (master logic inactive) before attempting to set the RSEN bit.

To generate a Repeated Start condition, software sets the RSEN bit (I2CCON<1>). The module asserts the SCL pin low. When the module samples the SCL pin low, the module releases the SDA pin for one baud rate generator count (TBRG). When the baud rate generator times out, if the module samples SDA high, the module de-asserts the SCL pin. When the module samples SCL pin high, the baud rate generator reloads and begins counting. SDA and SCL must be sampled high for one TBRG. This action is then followed by assertion of the SDA pin low for one TBRG while SCL is high.

The following is the Repeated Start sequence:

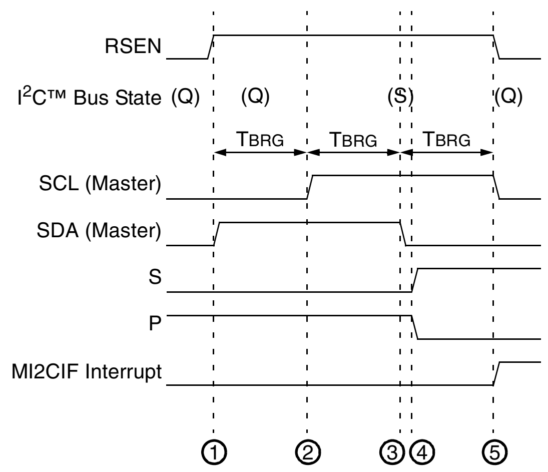
- The slave detects the Start condition, sets the S bit (I2CSTAT<3>) and clears the P bit (I2CSTAT<4>).
- The RSEN bit is automatically cleared.
- The module generates the MI2CIF interrupt.

21.5.6.1 IWCOL Status Flag

If the software writes the I2CTRN when a Repeated Start sequence is in progress, then IWCOL is set and the contents of the buffer are ignored.

Note:	Because queueing of events is not allowed, writing of the lower 5 bits of I2CCON is disabled until the Repeated Start condition is complete.

Figure 21-14: Master Repeated Start Timing Diagram



① - Writing RSEN = `1` initiates a master Repeated Start event. Baud generator starts. Module drives SCL low and releases SDA.

② - Baud generator times out. Module releases SCL. Baud generator restarts.

③ - Baud generator times out. Module drives SDA low. Baud generator restarts.

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

⑤ - The baud generator times out. Module drives SCL low. Module clears RSEN. Master generates interrupt.

21.5.7 Building Complete Master Messages

As described at the beginning of Section 21.5, the software is responsible for constructing messages with the correct message protocol. The module controls individual portions of the I^2C message protocol, however, sequencing of the components of the protocol to construct a complete message is a software task.

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

The software can use the SEN, RSEN, PEN, RCEN and ACKEN bits (Least Significant 5 bits of the I2CCON register) and the TRSTAT bit as a "state" flag when progressing through a message. For example, Table 21-2 shows some example state numbers associated with bus states.

Table 21-2: Master Message Protocol States

Example State Number	I2CCON<4:0>	TRSTAT (I2CSTAT<14>)	State
0	`00000`	`0`	Bus Idle or WAIT
1	`00001`	n/a	Sending Start Event
2	`00000`	`1`	Master Transmitting
3	`00010`	n/a	Sending Repeated Start Event
4	`00100`	n/a	Sending Stop Event
5	`01000`	n/a	Master Reception
6	`10000`	n/a	Master Acknowledgement

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

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

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

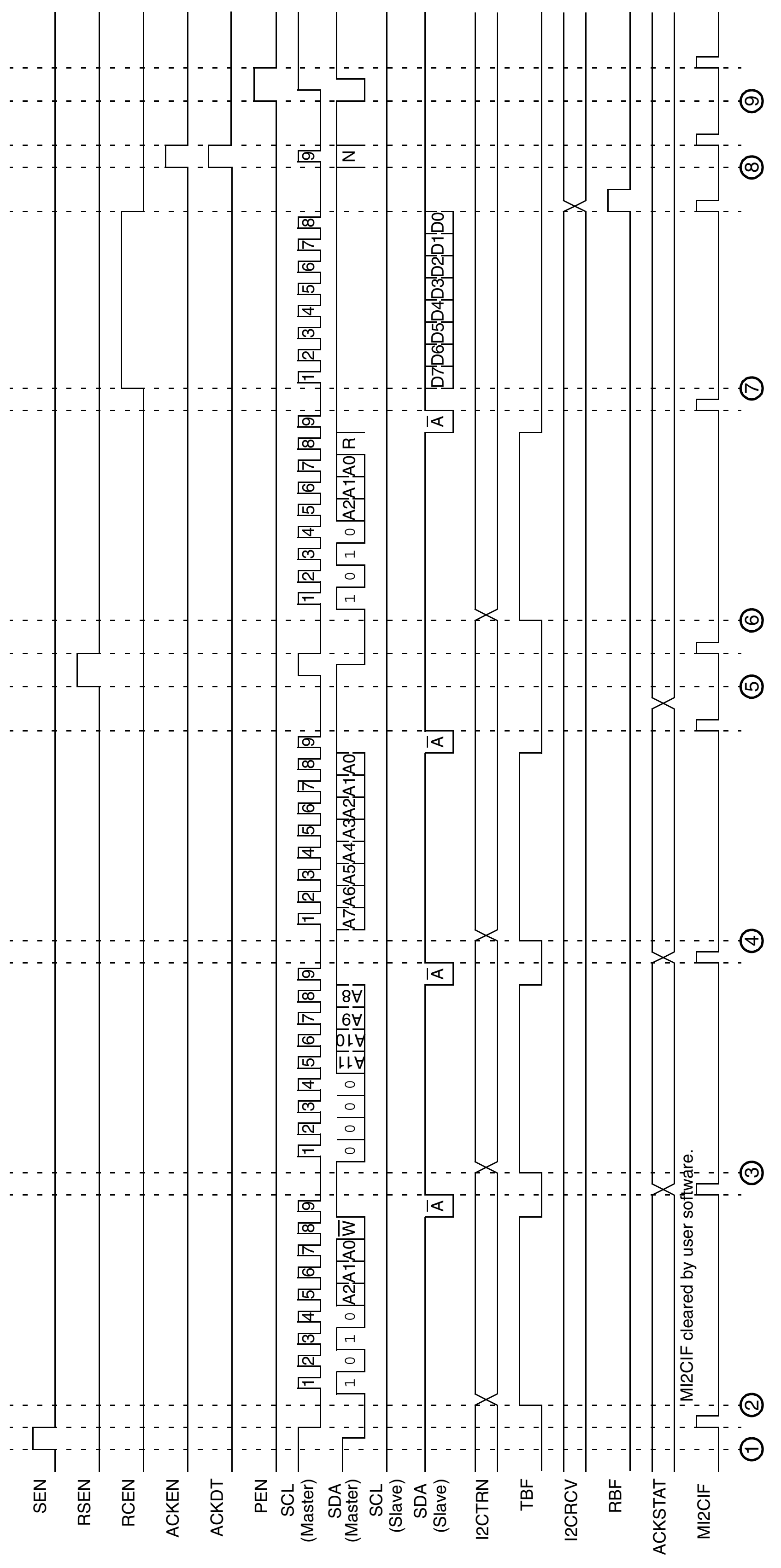
Figure 21-15 provides a more detailed examination of the same message sequence of Figure 21-7.

Figure 21-16 shows some simple examples of messages using 7-bit addressing format.

Figure 21-17 shows an example of a 10-bit address format message sending data to a slave.

Figure 21-18 shows an example of a 10-bit address format message receiving data from a slave.

Figure 21-15: Master Message (Typical I²C™ Message: Read of Serial EEPROM)



① - Setting the SEN bit starts a Start event.

② - Writing the I2CTRN register starts a master transmission. The data is the serial EE device address byte, with R/$\overline{W}$ clear indicating a write.

③ - Writing the I2CTRN register starts a master transmission. The data is the first byte of the EE data address.

④ - Writing the I2CTRN register starts a master transmission. The data is the second byte of the EE data address.

⑤ - Setting the RSEN bit starts a Repeated Start event.

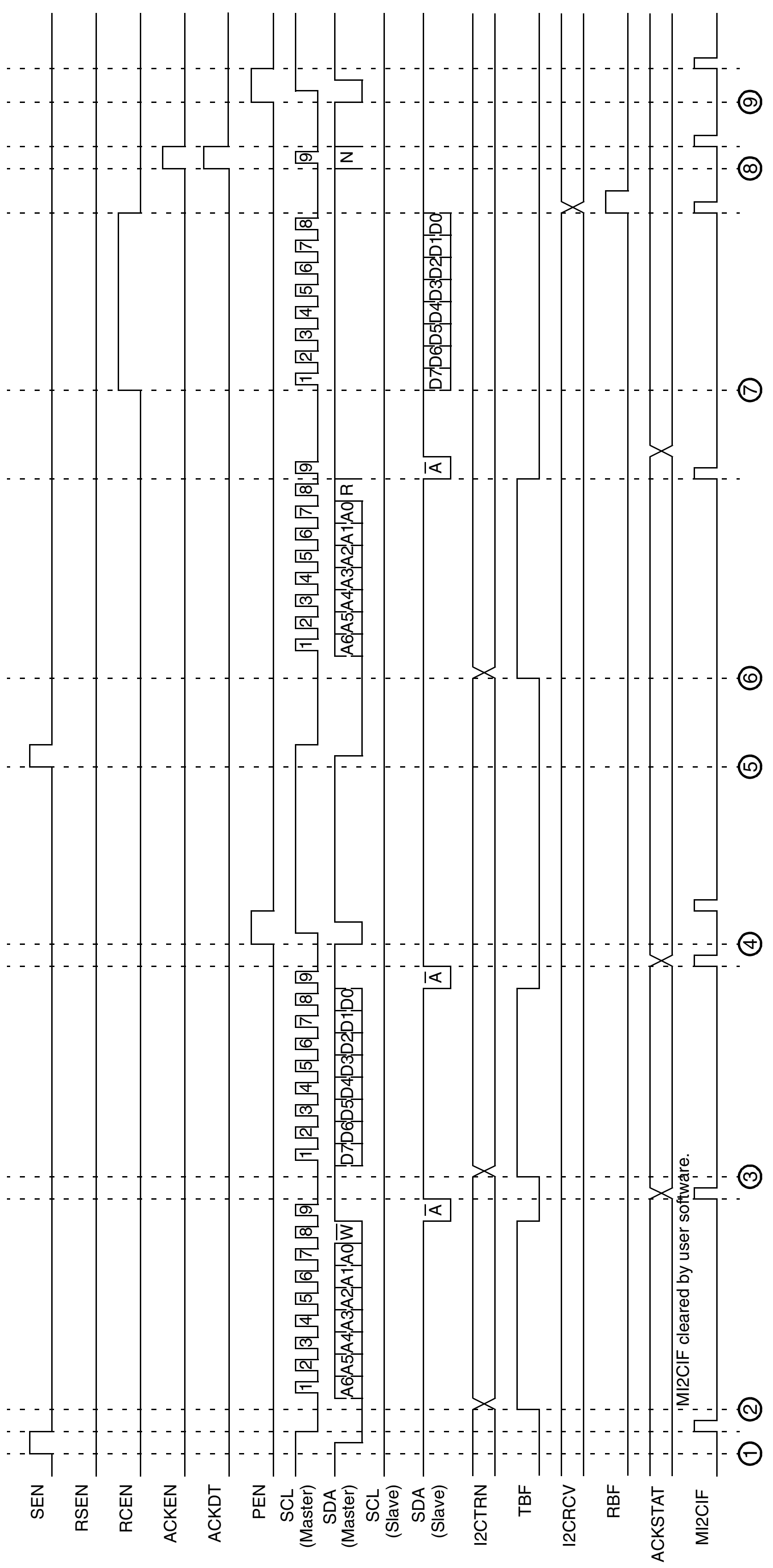
⑥ - Writing the I2CTRN register starts a master transmission. The data is a resend of the serial EE device address byte, but with R/$\overline{W}$ bit set indicating a read.

⑦ - Setting the RCEN bit starts a master reception. On interrupt, the software reads the I2CRCV register, which clears the RBF flag.

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 21-16: Master Message (7-bit Address: Transmission And Reception)



① - Setting the SEN bit starts a Start event.

② - Writing the I2CTRN register starts a master transmission. The data is the address byte with $R/\overline{W}$ bit clear.

③ - Writing the I2CTRN register starts a master transmission. The data is the message byte.

④ - Setting the PEN bit starts a master Stop event.

⑤ - Setting the SEN bit starts a Start event.

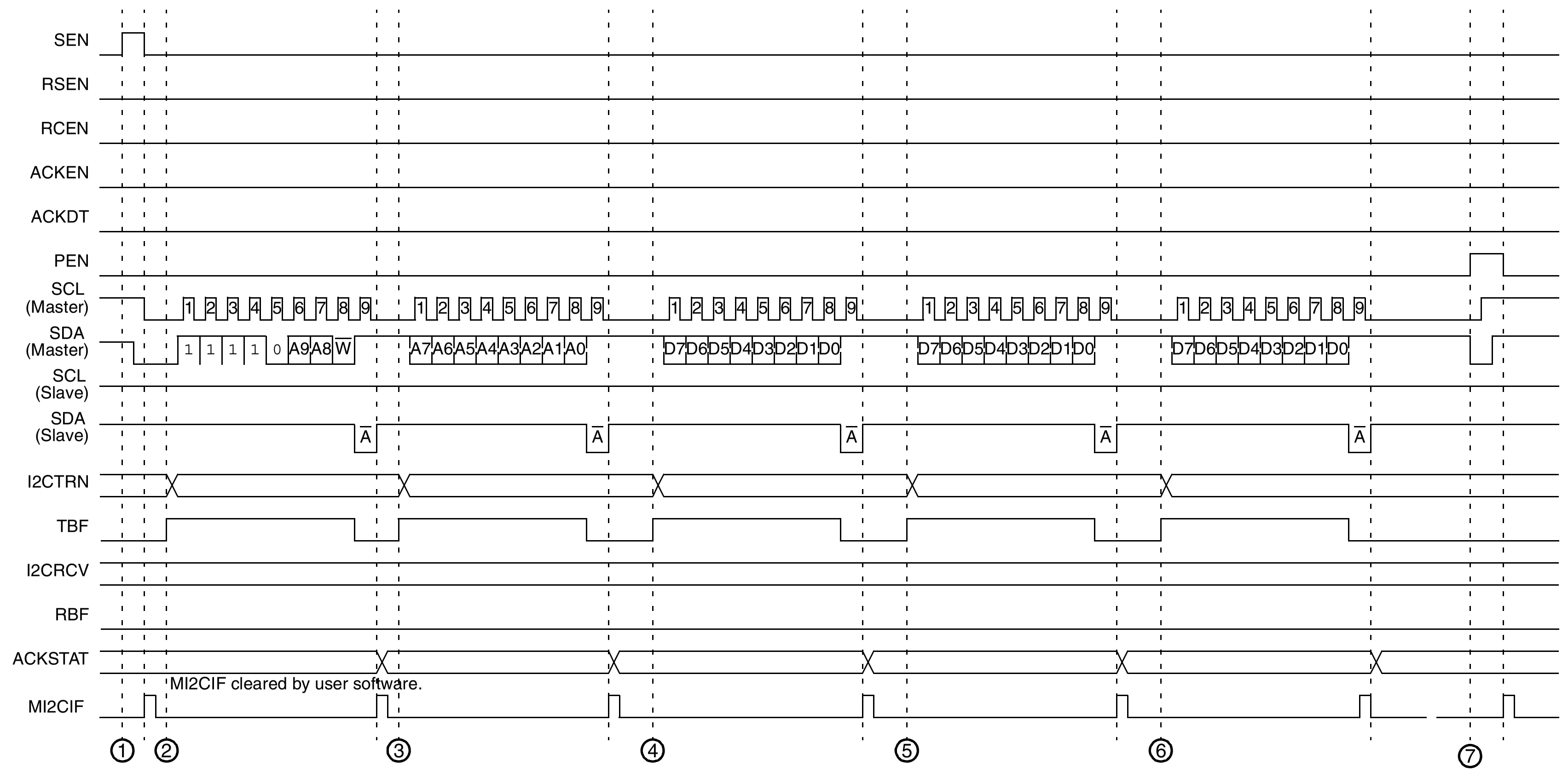
⑥ - Writing the I2CTRN register starts a master transmission. The data is the address byte with $R/\overline{W}$ bit set.

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.

Figure 21-17: Master Message (10-bit Transmission)



① - Setting the SEN bit starts a Start event.

② - Writing the I2CTRN register starts a master transmission. The data is the first byte of the address.

③ - Writing the I2CTRN register starts a master transmission. The data is the second byte of the address.

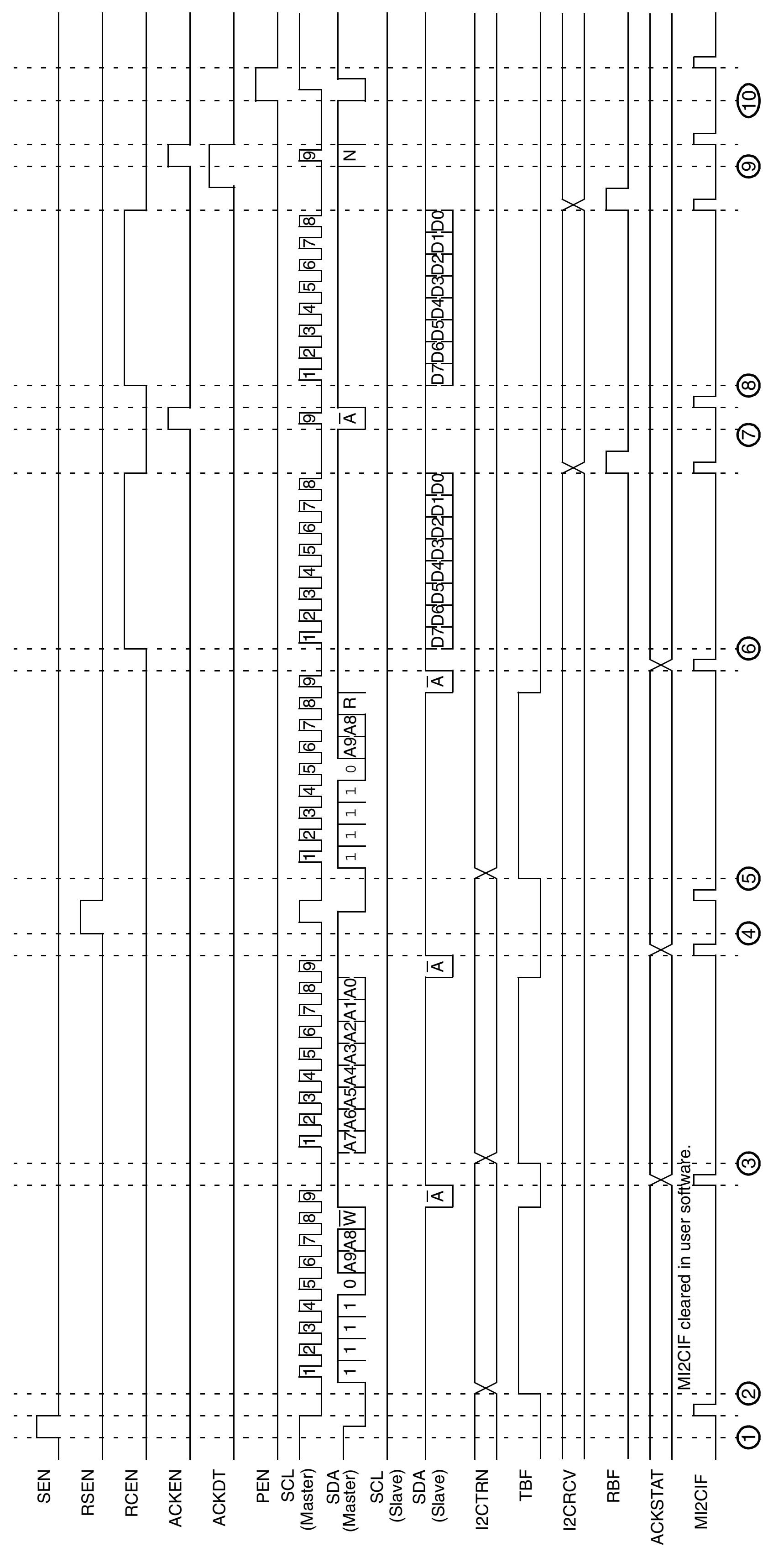
④ - Writing the I2CTRN register starts a master transmission. The data is the first byte of the message data.

⑤ - Writing the I2CTRN register starts a master transmission. The data is the second byte of the message data.

⑥ - Writing the I2CTRN register starts a master transmission. The data is the third byte of the message data.

⑦ - Setting the PEN bit starts a master Stop event.

Figure 21-18: Master Message (10-bit Reception)



1 - Setting the SEN bit starts a Start event.

2 - Writing the I2CTRN register starts a master transmission. The data is the first byte of the address with the R/$\overline{W}$ bit cleared.

3 - Writing the I2CTRN 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 I2CTRN register starts a master transmission. The data is a resend of the first byte with the R/$\overline{W}$ bit set.

6 - Setting the RCEN bit starts a master reception. On interrupt, the software reads the I2CRCV register, which clears the RBF flag.

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

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.

21.6 Communicating as a Master in a Multi-Master Environment

The I2C protocol allows for more than one master to be attached to a system bus. Remembering that a master can initiate message transactions and generate clocks for the bus, the protocol has methods to account for situations where more than one master is attempting to control the bus. Clock synchronization ensures that multiple nodes can synchronize their SCL clocks to result in one common clock on the SCL line. Bus arbitration ensures that if more than one node attempts a message transaction, one and only one node will be successful in completing the message. The other nodes will lose bus arbitration and be left with a bus collision.

21.6.1 Multi-Master Operation

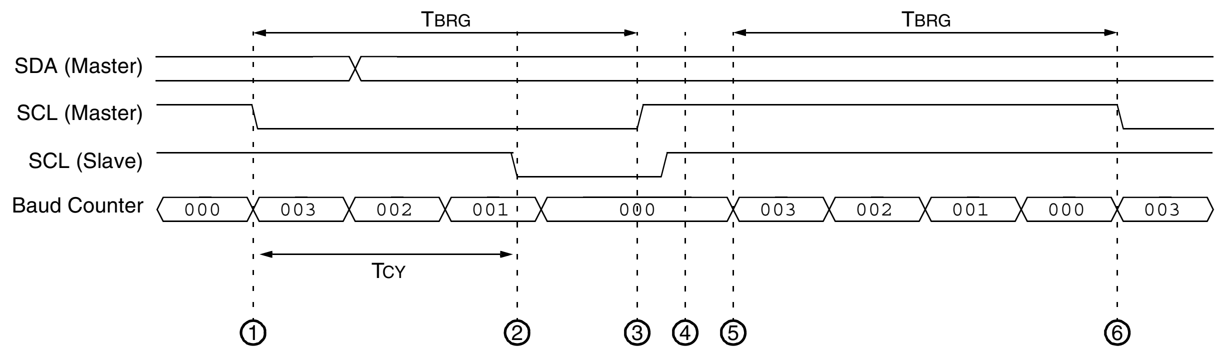
The master module has no special settings to enable multi-master operation. The module performs clock synchronization and bus arbitration at all times. If the module is used in a single master environment, clock synchronization will only occur between the master and slaves and bus arbitration will not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

21.6.2 Master Clock Synchronization

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

Clock synchronization occurs when the master de-asserts the SCL pin (SCL intended to float high). When the SCL pin is released, the baud rate generator (BRG) is suspended from counting until the SCL pin is actually sampled high. When the SCL pin is sampled high, the baud rate generator is reloaded with the contents of I2CBRG<8:0> and begins counting. This ensures that the SCL high time will always be at least one BRG rollover count in the event that the clock is held low by an external device, as shown in Figure 21-19.

Figure 21-19: Baud Rate Generator Timing with Clock Synchronization



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

② - The slave has pulled SCL low to initiate a wait.

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

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

⑤ - The baud counter rollover occurs on next cycle.

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

21.6.3 Bus Arbitration and Bus Collision

Bus arbitration supports multi-master system operation.

The wired-and nature of the SDA line permits arbitration. Arbitration takes place when the first master outputs a ‘1’ on SDA by letting SDA float high and, simultaneously, the second master outputs a ‘0’ on SDA by pulling SDA low. The SDA 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 SDA is a ‘1’ yet the data sampled on SDA is a ‘0’. This is the definition of a bus collision.

The first master will set the bus collision bit, BCL (I2CSTAT<10>), and generate a master interrupt. The master module will reset the I^{2}C port to its Idle state.

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

The states where arbitration can be lost are:

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

21.6.4 Detecting Bus Collisions and Resending Messages

When a bus collision occurs, the module sets the BCL bit and generates a master interrupt. If bus collision occurs during a byte transmission, the transmission is halted, the TBF flag is cleared and the SDA and SCL pins are de-asserted. If bus collision occurs during a Start, Repeated Start, Stop or Acknowledge condition, the condition is aborted, the respective control bits in the I2CCON register are cleared and the SDA and SCL lines are de-asserted.

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

21.6.5 Bus Collision During a Start Condition

Before issuing a Start command, the software should verify an Idle state of the bus using the S and P status bits. Two masters may attempt to initiate a message at a similar point in time. Typically, the masters will synchronize clocks and continue arbitration into the message until one loses arbitration. However, certain conditions can cause a bus collision to occur during a Start. In this case, the master that loses arbitration during the Start bit generates a bus collision interrupt.

21.6.6 Bus Collision During a Repeated Start Condition

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

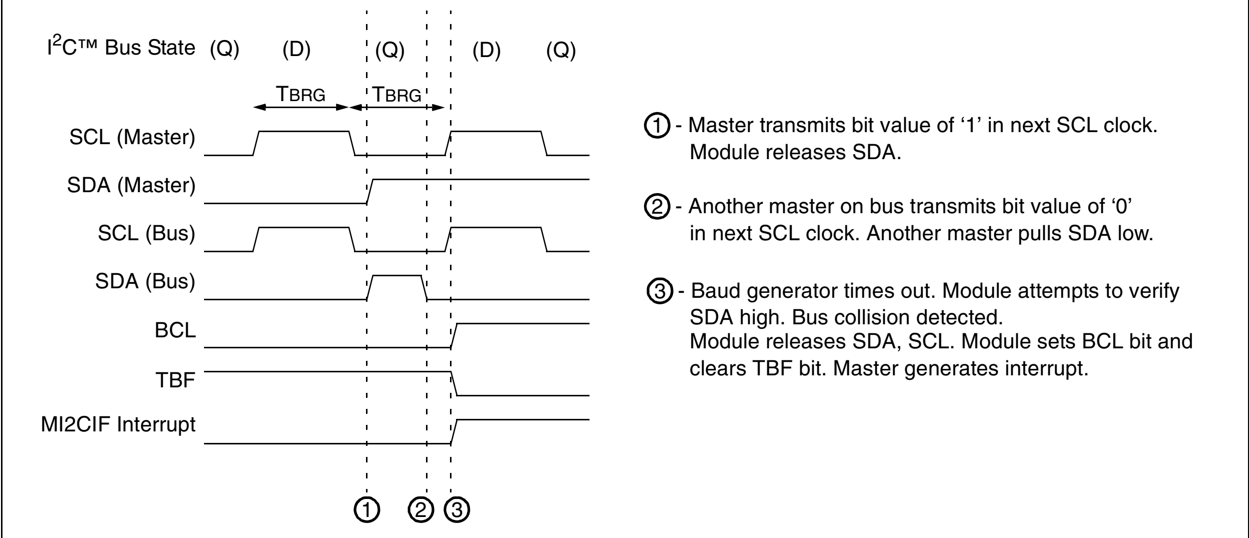
21.6.7 Bus Collision During Message Bit Transmission

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

If the software is properly checking the bus state, it is unlikely that a bus collision will occur on a Start condition. However, because another master can at a very similar time, check the bus and initiate its own Start condition, it is likely that SDA arbitration will occur and synchronize the starts of two masters. In this condition, both masters will begin and continue to transmit their messages until one master loses arbitration on a message bit. Remember that SCL clock synchronization will keep the two masters synchronized until one loses arbitration.

Figure 21-20 shows an example of message bit arbitration.

Figure 21-20: Bus Collision During Message Bit Transmission


I2C™ Bus State
(Q)
(D)
(Q)
(D)
(Q)
TBRG
TBRG
SCL (Master)
SDA (Master)
SCL (Bus)
SDA (Bus)
BCL
TBF
MI2CIF Interrupt
1 - Master transmits bit value of ‘1’ in next SCL clock. Module releases SDA.
2 - Another master on bus transmits bit value of ‘0’ in next SCL clock. Another master pulls SDA low.
3 - Baud generator times out. Module attempts to verify SDA high. Bus collision detected. Module releases SDA, SCL. Module sets BCL bit and clears TBF bit. Master generates interrupt.
1
2
3

21.6.8 Bus Collision During a Stop Condition

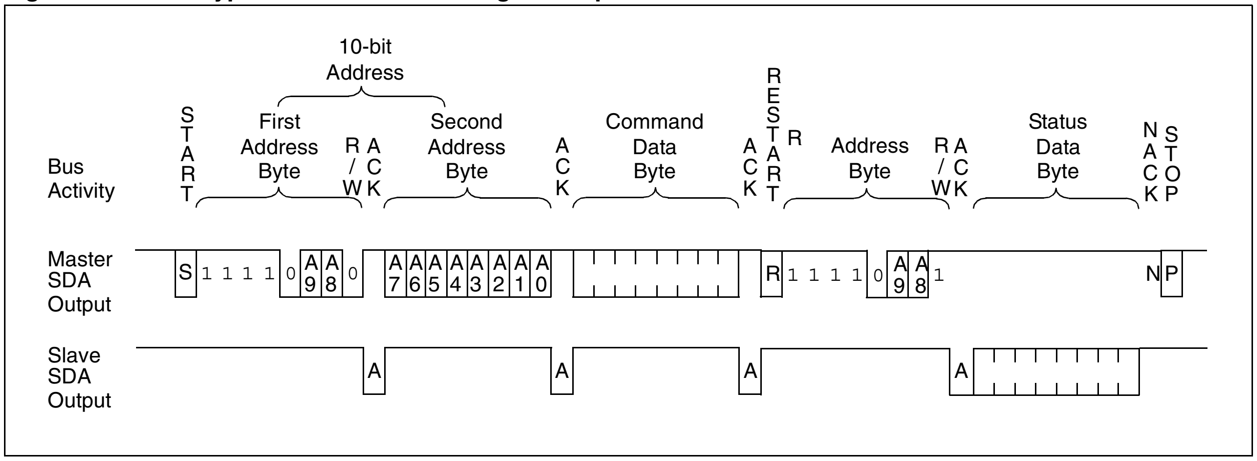
If the master software loses track of the state of the I^2C bus, there are conditions which cause a bus collision during a Stop condition. In this case, the master generating the Stop condition will lose arbitration and generate a bus collision interrupt.

21.7 Communicating as a Slave

In some systems, particularly where multiple processors communicate with each other, the dsPIC30F device may communicate as a slave (see Figure 21-21). When the module is enabled, the slave module is active. The slave may not initiate a message, it can only respond to a message sequence initiated by a master. The master requests a response from a particular slave as defined by the device address byte in the I^{2}C protocol. The slave module replies to the master at the appropriate times as defined by the protocol.

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

Figure 21-21: A Typical Slave I^{2}C™ Message: Multiprocessor Command/Status



After a Start condition, the slave module will receive and check the device address. The slave may specify either a 7-bit address or a 10-bit address. When a device address is matched, the module will generate an interrupt to notify the software that its device is selected. Based on the R/$\overline{W}$ bit sent by the master, the slave will either receive or transmit data. If the slave is to receive data, the slave module automatically generates the Acknowledge ($\overline{ACK}$), loads the I2CRCV register with the received value currently in the I2CRSR register and notifies the software through an interrupt. If the slave is to transmit data, the software must load the I2CTRN register.

21.7.1 Sampling Receive Data

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

21.7.2 Detecting Start and Stop Conditions

The slave module will detect Start and Stop conditions on the bus and indicate that status on the S bit (I2CSTAT<3>) and P bit (I2CSTAT<4>). The Start (S) and Stop (P) bits are cleared when a Reset occurs or when the module is disabled. After detection of a Start or Repeated Start event, the S bit is set and the P bit is cleared. After detection of a Stop event, the P bit is set and the S bit is clear.

21.7.3 Detecting the Address

Once the module has been enabled, the slave module waits for a Start condition to occur. After a Start, depending on the A10M bit (I2CCON<10>), the slave will attempt to detect a 7-bit or 10-bit address. The slave module will compare 1 received byte for a 7-bit address or 2 received bytes for a 10-bit address. A 7-bit address also contains a R/$\overline{W}$ bit that specifies the direction of data transfer after the address. If R/$\overline{W}$ = `0`, a write is specified and the slave will receive data from the master. If R/$\overline{W}$ = `1`, a read is specified and the slave will send data to the master. The 10-bit address contains a R/$\overline{W}$ bit, however by definition, it is always R/$\overline{W}$ = `0` because the slave must receive the second byte of the 10-bit address.

Table 21-3: **Slave Addresses Suppported by the I^2C™ Module**:

0x00	General call address or start byte
0x01-0x03	Reserved
0x04-0x77	Valid 7-bit addresses
0x78-0x7b	Valid 10-bit addresses (lower 7 bits)
0x7c-0x7f	Reserved

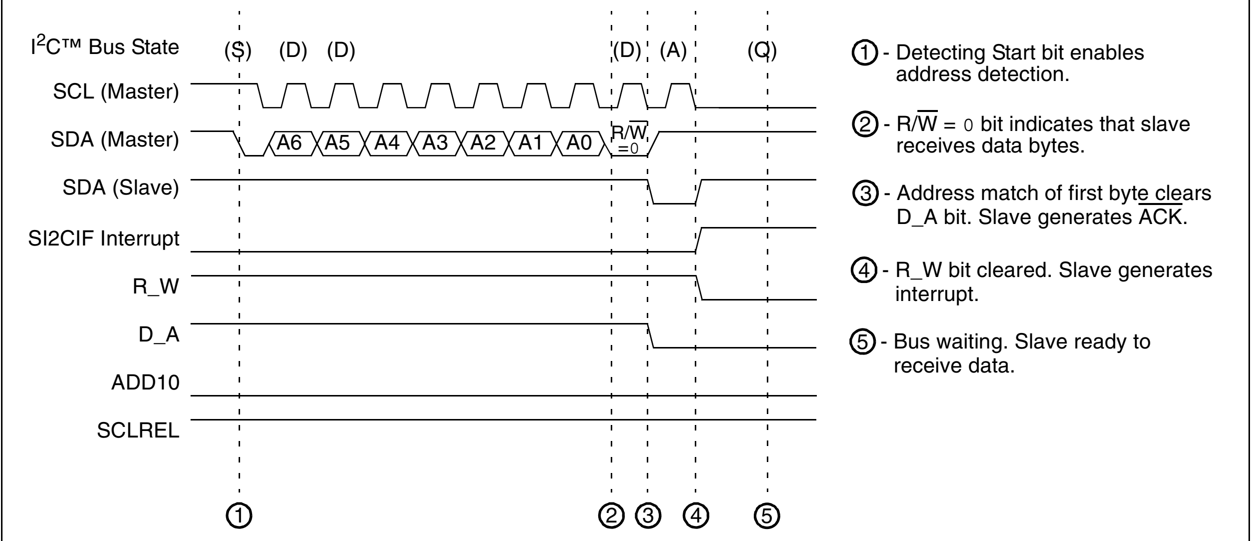
Refer to *dsPIC30F Family Reference Manua*l (DS70046) for descriptions of register bit fields.

21.7.3.1 7-bit Address and Slave Write

Following the Start condition, the module shifts 8 bits into the I2CRSR register (see Figure 21-22). The value of register I2CRSR<7:1> is compared to the value of the I2CADD<6:0> register. The device address is compared on the falling edge of the eighth clock (SCL). If the addresses match, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D_A and R_W bits are cleared.
3. The module generates the SI2CIF interrupt on the falling edge of the ninth SCL clock.
4. The module will wait for the master to send data.

Figure 21-22: Slave Write 7-bit Address Detection Timing Diagram


I^2C™ Bus State
(S)
(D)
(D)
(D)
(A)
(Q)
SCL (Master)
SDA (Master)
A6
A5
A4
A3
A2
A1
A0
R/$\overline{\text{W}}$ =0
SDA (Slave)
SI2CIF Interrupt
R_W
D_A
ADD10
SCLREL
1
2
3
4
5
1 - Detecting Start bit enables address detection.
2 - R/$\overline{\text{W}}$ = 0 bit indicates that slave receives data bytes.
3 - Address match of first byte clears D_A bit. Slave generates $\overline{\text{ACK}}$.
4 - R_W bit cleared. Slave generates interrupt.
5 - Bus waiting. Slave ready to receive data.

21.7.3.2 7-bit Address and Slave Read

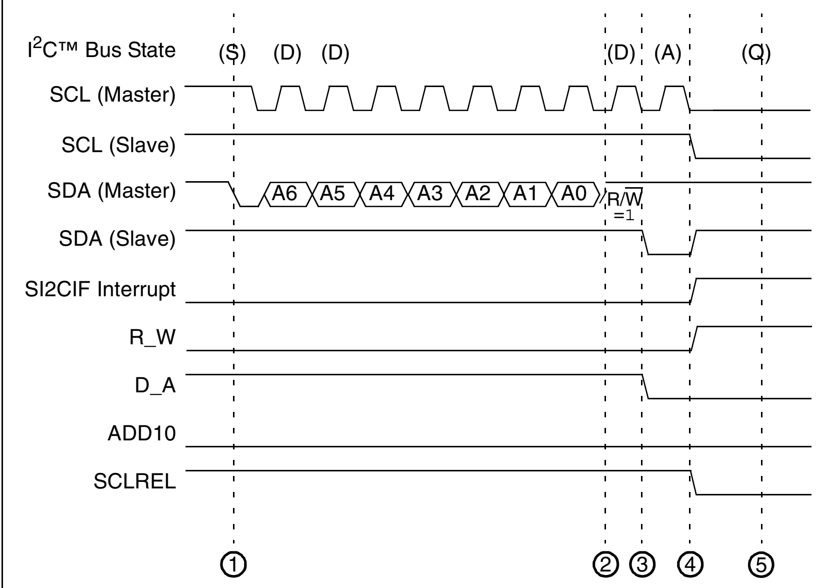
When a slave read is specified by having R/$\overline{\text{W}}$ = `1` in a 7-bit address byte, the process of detecting the device address is similar to that for a slave write (see Figure 21-23). If the addresses match, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D_A bit is cleared and the R_W bit is set.
3. The module generates the SI2CIF interrupt on the falling edge of the ninth SCL clock.

Since the slave module is expected to reply with data at this point, it is necessary to suspend the operation of the I^{2}C bus to allow the software to prepare a response. This is done automatically when the module clears the SCLREL bit. With SCLREL low, the slave module will pull down the SCL clock line, causing a wait on the I^{2}C bus. The slave module and the I^{2}C bus will remain in this state until the software writes the I2CTRN register with the response data and sets the SCLREL bit.

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

Figure 21-23: Slave Read 7-bit Address Detection Timing Diagram



1 - Detecting Start bit enables address detection.

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

3 - Address match of first byte clears D_A bit. Slave generates $\overline{\text{ACK}}$.

4 - R_W bit set. Slave generates interrupt. SCLREL cleared. Slave pulls SCL low while SCLREL = `0`.

5 - Bus waiting. Slave prepares to send data.

21.7.3.3 10-bit Address

In 10-bit Address mode, the slave must receive two device address bytes (see Figure 21-24). The five Most Significant bits (MSbs) of the first address byte specify a 10-bit address. The R/$\overline{\text{W}}$ bit of the address must specify a write, causing the slave device to receive the second address byte. For a 10-bit address the first byte would equal `‘11110 A9 A8 0’`, where `A9` and `A8` are the two MSbs of the address.

Following the Start condition, the module shifts 8 bits into the I2CRSR register. The value of register I2CRSR<2:1> is compared to the value of the I2CADD<9:8> register. The value of I2CRSR<7:3> is compared to `‘11110’`. The device address is compared on the falling edge of the eighth clock (SCL). If the addresses match, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D_A and R_W bits are cleared.
3. The module generates the SI2CIF interrupt on the falling edge of the ninth SCL clock.

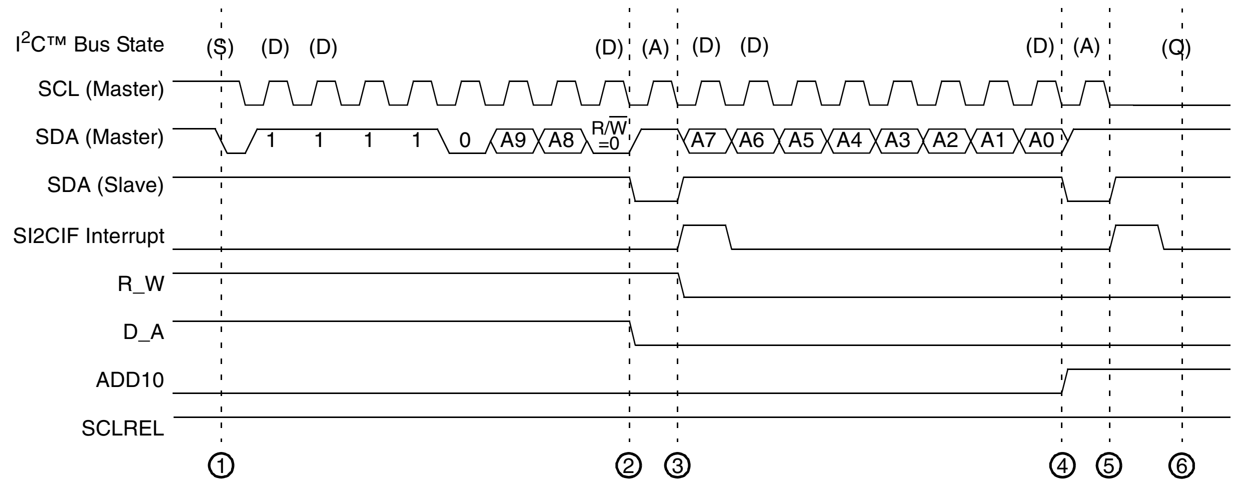
The module does generate an interrupt after the reception of the first byte of a 10-bit address, however this interrupt is of little use.

The module will continue to receive the second byte into I2CRSR. This time, I2CRSR<7:0> is compared to I2CADD<7:0>. If the addresses match, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The ADD10 bit is set.
3. The module generates the SI2CIF interrupt on the falling edge of the ninth SCL clock.
4. The module will wait for the master to send data or initiate a Repeated Start condition.

Note:	Following a Repeated Start condition in 10-bit mode, the slave module only matches the first 7-bit address, `‘11110 A9 A8 0’`.

Figure 21-24: 10-bit Address Detection Timing Diagram



① - Detecting Start bit enables address detection.

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

③ - Reception of first byte clears R_W bit. Slave logic generates interrupt.

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

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

⑤ - Bus waiting. Slave ready to receive data.

21.7.3.4 General Call Operation

The addressing procedure for the I^{2}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 enabled devices should respond with an Acknowledge. The general call address is one of eight addresses reserved for specific purposes by the I^{2}C protocol. It consists of all '0's with R/$\overline{W}$ = 0. The general call is always a slave write operation.

The general call address is recognized when the general call enable bit, GCEN (I2CCON<7>), is set (see Figure 21-25). Following a Start bit detect, 8 bits are shifted into the I2CRSR and the address is compared against the I2CADD, and is also compared to the general call address.

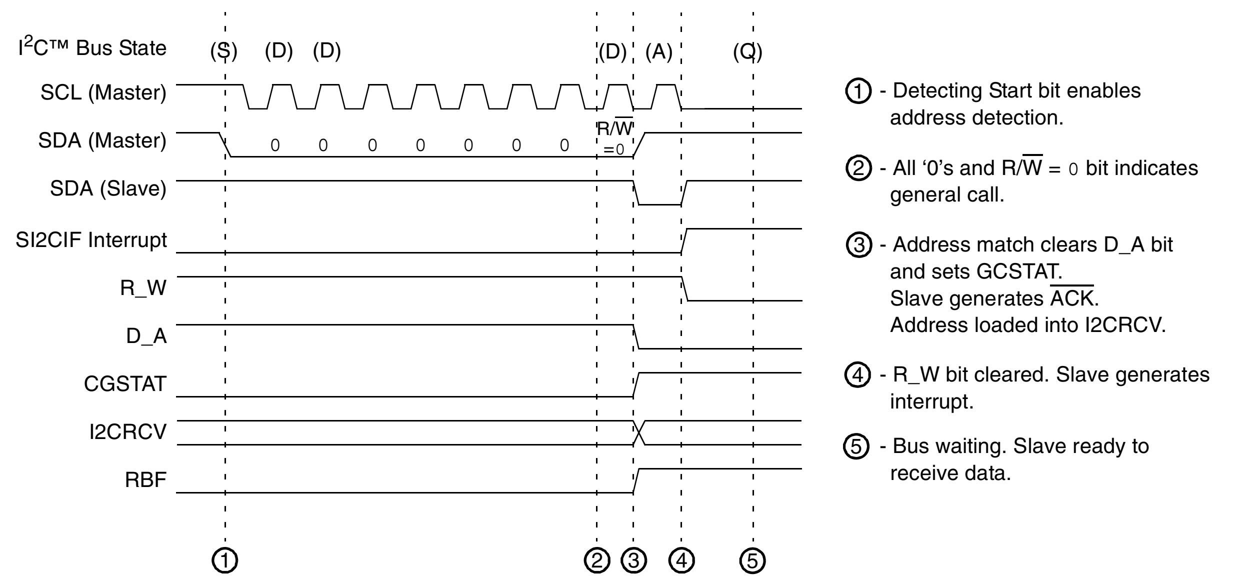
If the general call address matches, the following events occur:

1. An $\overline{ACK}$ is generated.
2. Slave module will set the GCSTAT bit (I2CSTAT<9>).
3. The D_A and R_W bits are cleared.
4. The module generates the SI2CIF interrupt on the falling edge of the ninth SCL clock.
5. The I2CRSR is transferred to the I2CRCV and the RBF flag bit is set (during the eighth bit).
6. The module will wait for the master to send data.

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

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

Figure 21-25: General Call Address Detection Timing Diagram (GCEN = 1)

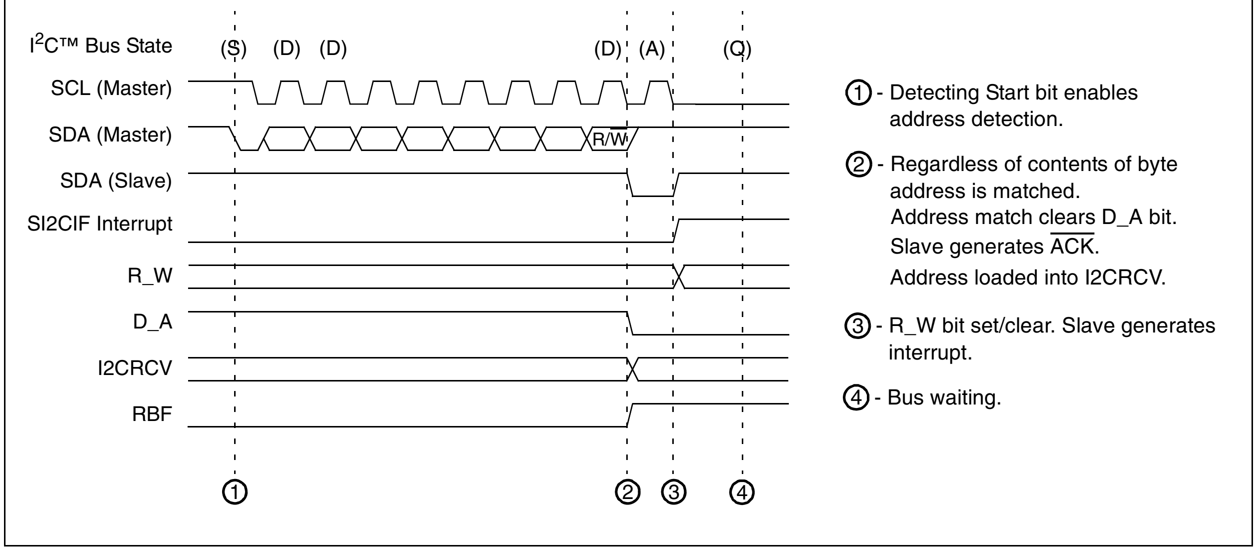

I²C™ Bus State
(S) (D) (D) (D) (A) (Q)
SCL (Master)
SDA (Master)
0 0 0 0 0 0 0
R/W̄ =0
SDA (Slave)
SI2CIF Interrupt
R_W
D_A
CGSTAT
I2CRCV
RBF
1 2 3 4 5
1 - Detecting Start bit enables address detection.
2 - All '0's and R/W̄ = 0 bit indicates general call.
3 - Address match clears D_A bit and sets GCSTAT. Slave generates ACK̄. Address loaded into I2CRCV.
4 - R_W bit cleared. Slave generates interrupt.
5 - Bus waiting. Slave ready to receive data.

21.7.3.5 Receiving All Addresses (IPMI Operation)

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

Setting the IPMIEN bit (I2CCON<11>) enables this mode (see Figure 21-26). Regardless of the state of the I2CADD register and the A10M and GCEN bits, all addresses will be accepted.

Figure 21-26: IPMI Address Detection Timing Diagram (IPMIEN = 1)


I2C™ Bus State
(S) (D) (D) (D) (A) (Q)
SCL (Master)
SDA (Master)
R/W
SDA (Slave)
SI2CIF Interrupt
R_W
D_A
I2CRCV
RBF
1 2 3 4
1 - Detecting Start bit enables address detection.
2 - Regardless of contents of byte address is matched. Address match clears D_A bit. Slave generates ACK. Address loaded into I2CRCV.
3 - R_W bit set/clear. Slave generates interrupt.
4 - Bus waiting.

21.7.3.6 When an Address is Invalid

If a 7-bit address does not match the contents of I2CADD<6:0>, the slave module will return to an Idle state and ignore all bus activity until after the Stop condition.

If the first byte of a 10-bit address does not match the contents of I2CADD<9:8>, the slave module will return to an Idle state and ignore all bus activity until after the Stop condition.

If the first byte of a 10-bit address matches the contents of I2CADD<9:8>, however, the second byte of the 10-bit address does not match I2CADD<7:0>, the slave module will return to an Idle state and ignore all bus activity until after the Stop condition.

21.7.4 Receiving Data from a Master Device

When the R/$\overline{W}$ bit of the device address byte is zero and an address match occurs, the R_W bit (I2CSTAT<2>) is cleared. The slave module enters a state waiting for data sent by the master. After the device address byte, the contents of the data byte are defined by the system protocol and are only received by the slave module.

The slave module shifts 8 bits into the I2CRSR register. On the falling edge of the eighth clock (SCL), the following events occur:

1. The module begins to generate an $\overline{\text{ACK}}$ or NACK.
2. The RBF bit is set to indicate received data.
3. The I2CRSR byte is transferred to the I2CRCV register for access by the software.
4. The D_A bit is set.
5. A slave interrupt is generated. Software may check the status of the I2CSTAT register to determine the cause of the event and then clear the SI2CIF flag.
6. The module will wait for the next data byte.

21.7.4.1 Acknowledge Generation

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

1. The buffer full bit, RBF (I2CSTAT<1>), was set before the transfer was received.
2. The overflow bit, I2COV (I2CSTAT<6>), was set before the transfer was received.

Table 21-4 shows what happens when a data transfer byte is received, given the status of the RBF and I2COV bits. If the RBF bit is already set when the slave module attempts to transfer to the I2CRCV, the transfer does not occur but the interrupt is generated and the I2COV bit is set. If both the RBF and I2COV bits are set, the slave module acts similarly. The shaded cells show the condition where software did not properly clear the overflow condition.

Reading the I2CRCV clears the RBF bit. The I2COV is cleared by writing to a '0' through software.

Table 21-4: Data Transfer Received Byte Actions

Status Bits as Data Byte Received		Transfer I2CRSR $\rightarrow$ I2CRCV	Generate $\overline{\text{ACK}}$	Generate SI2CIF Interrupt (Interrupt occurs if enabled)	Set RBF	Set I2COV
RBF	**I2COV**					
`0`	`0`	Yes	Yes	Yes	Yes	No change
`1`	`0`	No	No	Yes	No change	Yes
`1`	`1`	No	No	Yes	No change	Yes
`0`	`1`	Yes	No	Yes	Yes	No change

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

21.7.4.2 WAIT States During Slave Receptions

When the slave module receives a data byte, the master can potentially begin sending the next byte immediately. This allows the software controlling the slave module 9 SCL 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 (I2CCON<6>) enables a bus WAIT to occur on slave receptions. When STREN = `1` at the falling edge of the 9th SCL clock of a received byte, the slave module clears the SCLREL bit. Clearing the SCLREL bit causes the slave module to pull the SCL line low, initiating a WAIT. The SCL clock of the master and slave will synchronize, as shown in **Section 21.6.2 "Master Clock Synchronization"**.

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

21.7.4.3 Example Messages of Slave Reception

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

When the slave detects the valid address, the associated interrupt will notify the software to expect a message. On receive data, as each data byte transfers to the I2CRCV register, an interrupt notifies the software to unload the buffer.

Figure 21-27 shows a simple receive message. Being a 7-bit address message, only one interrupt occurs for the address bytes. Then, interrupts occur for each of four data bytes.

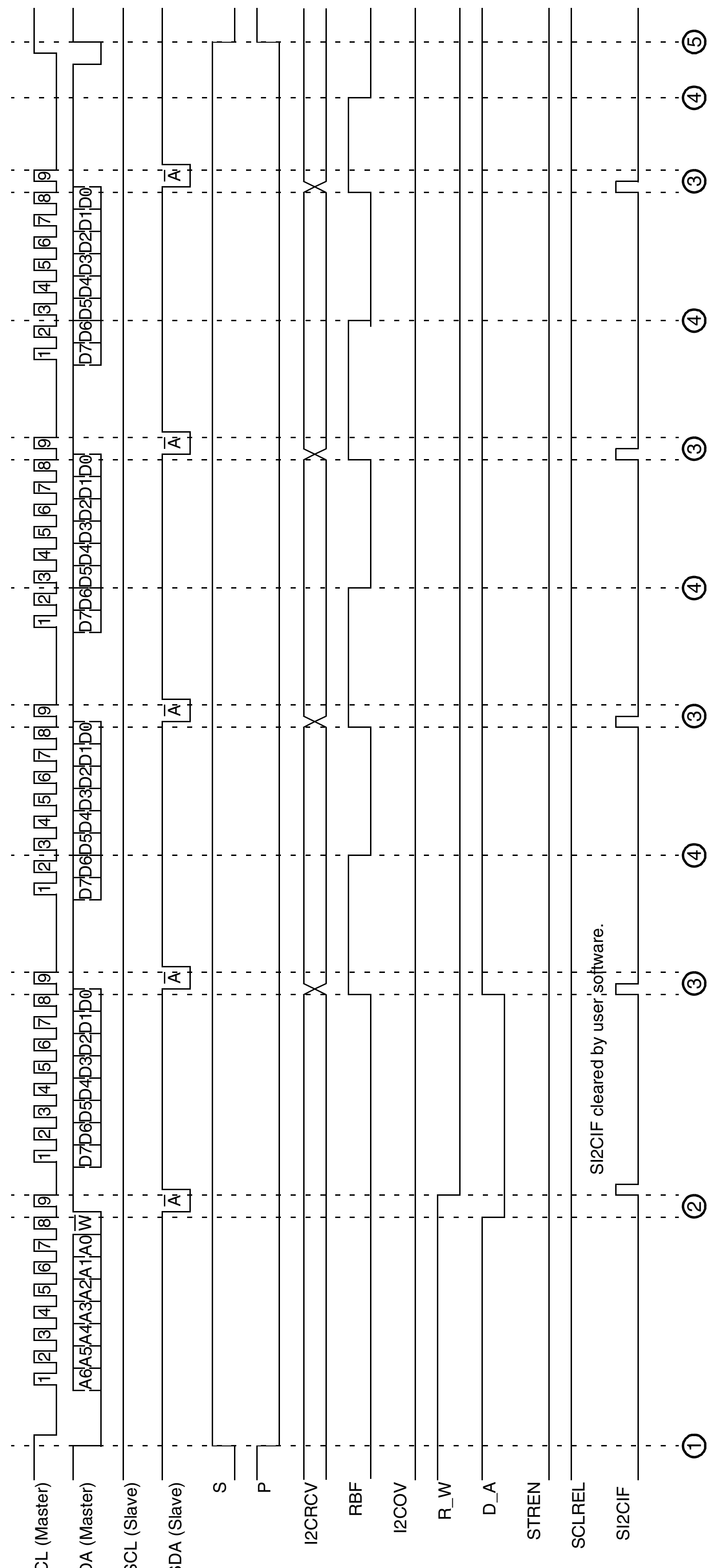
At an interrupt, the software may monitor the RBF, D_A and R_W bits to determine the condition of the byte received.

Figure 21-28 shows a similar message using a 10-bit address. In this case, two bytes are required for the address.

Figure 21-29 shows a case where the software does not respond to the received byte and the buffer overruns. On reception of the second byte, the module will automatically NACK the master transmission. Generally, this causes the master to resend the previous byte. The I2COV bit indicates that the buffer has overrun. The I2CRCV buffer retains the contents of the first byte. On reception of the third byte, the buffer is still full and again the module will NACK the master. After this, the software finally reads the buffer. Reading the buffer will clear the RBF bit, however the I2COV bit remains set. The software must clear the I2COV bit. The next received byte will be moved to the I2CRCV buffer and the module will respond with a $\overline{\text{ACK}}$.

Figure 21-30 highlights clock stretching while receiving data. Note in the previous examples, STREN = `0` which disables clock stretching on receive messages. In this example, the software sets STREN to enable clock stretching. When STREN = `1`, the module will automatically clock stretch after each received data byte, allowing the software more time to move the data from the buffer. Note that if RBF = `1` at the falling edge of the 9th clock, the module will automatically clear the SCLREL bit and pull the SCL bus line low. As shown with the second received data byte, if the software can read the buffer and clear the RBF before the falling edge of the 9th clock, the clock stretching will not occur. The software can also suspend the bus at any time. By clearing the SCLREL bit, the module will pull the SCL line low after it detects the bus SCL low. The SCL line will remain low, suspending transactions on the bus until the SCLREL bit is set.

Figure 21-27: Slave Message (Write Data to Slave: 7-bit Address; Address Matches; A10M = 0; GCEN = 0; IPMIEN = 0)



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

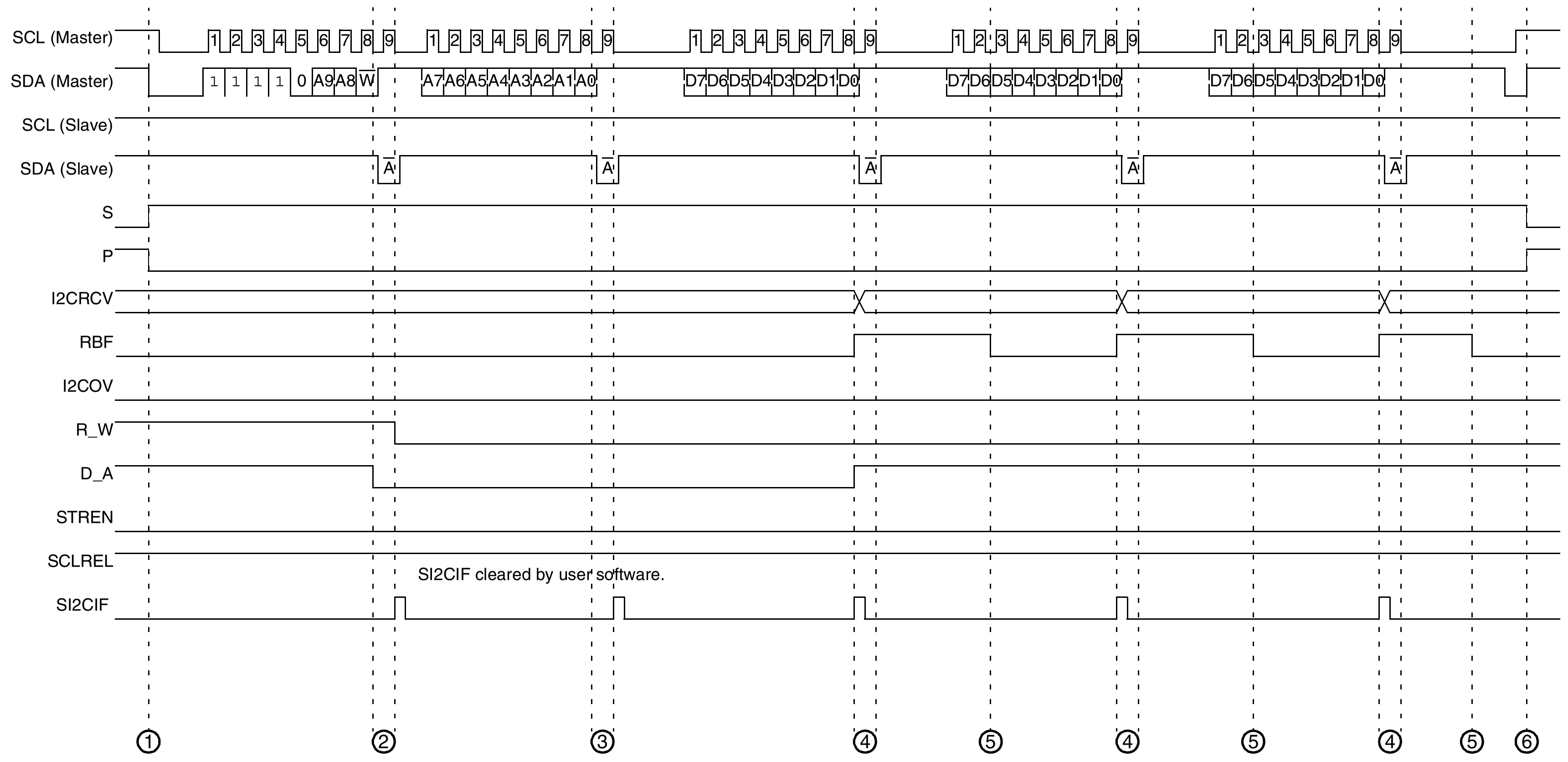
2 - Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt. Address byte is moved to I2CRCV register and must be read by user software to prevent buffer overflow.

3 - Next received byte is message data. Byte moved to I2CRCV register, sets RBF. Slave generates interrupt. Slave Acknowledges reception.

4 - Software reads I2CRCV register. RBF bit clears.

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

Figure 21-28: Slave Message (Write Data to Slave: 10-bit Address; Address Matches; A10M=1; GCEN=0; IPMIEN=0)



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

② - Slave receives address byte. High order address matches. Slave Acknowledges and generates interrupt. Address byte not moved to I2CRCV register.

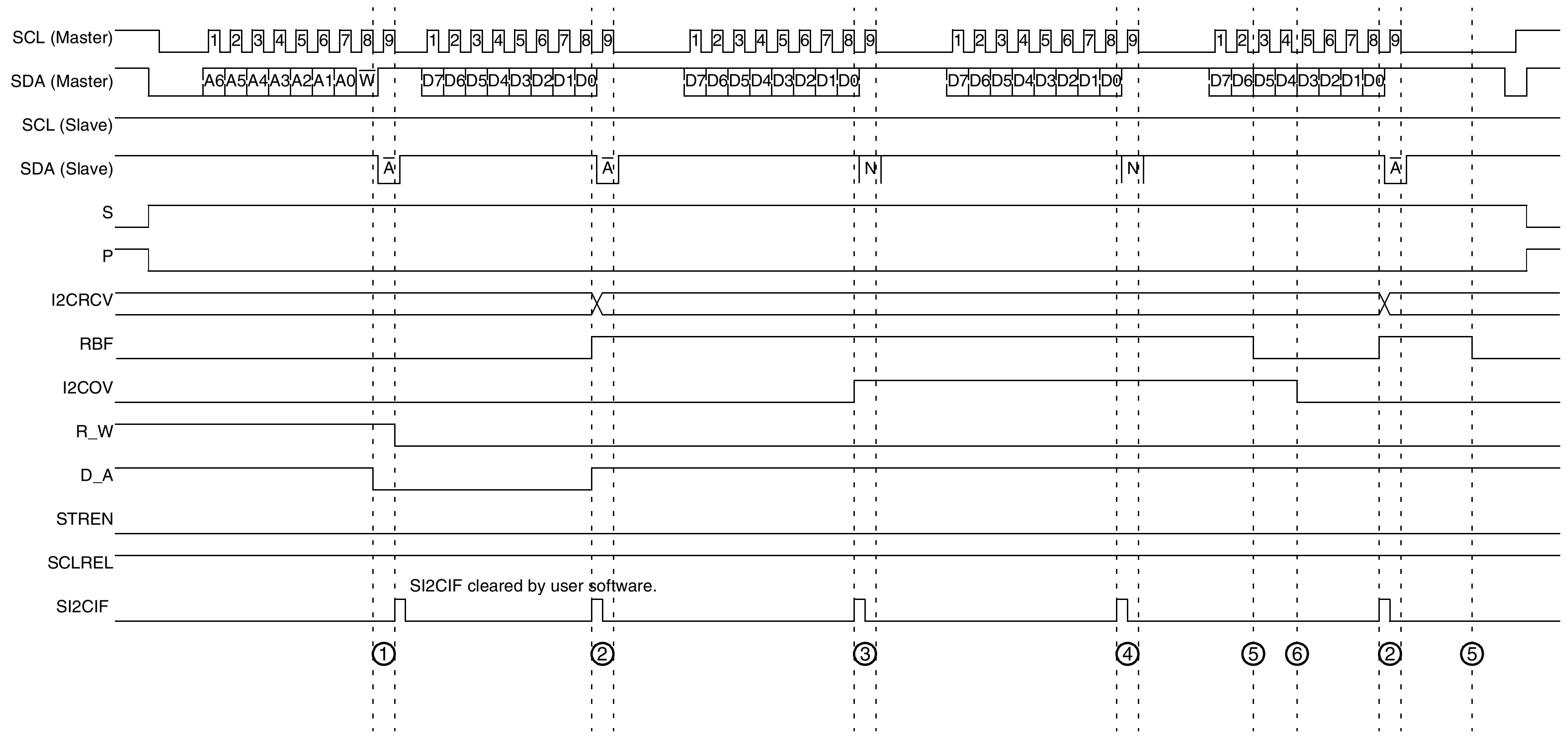
③ - Slave receives address byte. Low order address matches. Slave Acknowledges and generates interrupt. Address byte not moved to I2CRCV register.

④ - Next received byte is message data. Byte moved to I2CRCV register, sets RBF. Slave Acknowledges and generates interrupt.

⑤ - Software reads I2CRCV register. RBF bit clears.

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

Figure 21-29: Slave Message (Write Data to Slave: 7-bit Address; Buffer Overrun; A10M = 0; GCEN = 0; IPMIEN = 0)



① - Slave receives address byte. Address matches. Slave generates interrupt. Address byte not moved to I2CRCV register.

② - Next received byte is message data. Byte moved to I2CRCV register, sets RBF. Slave generates interrupt. Slave Acknowledges reception.

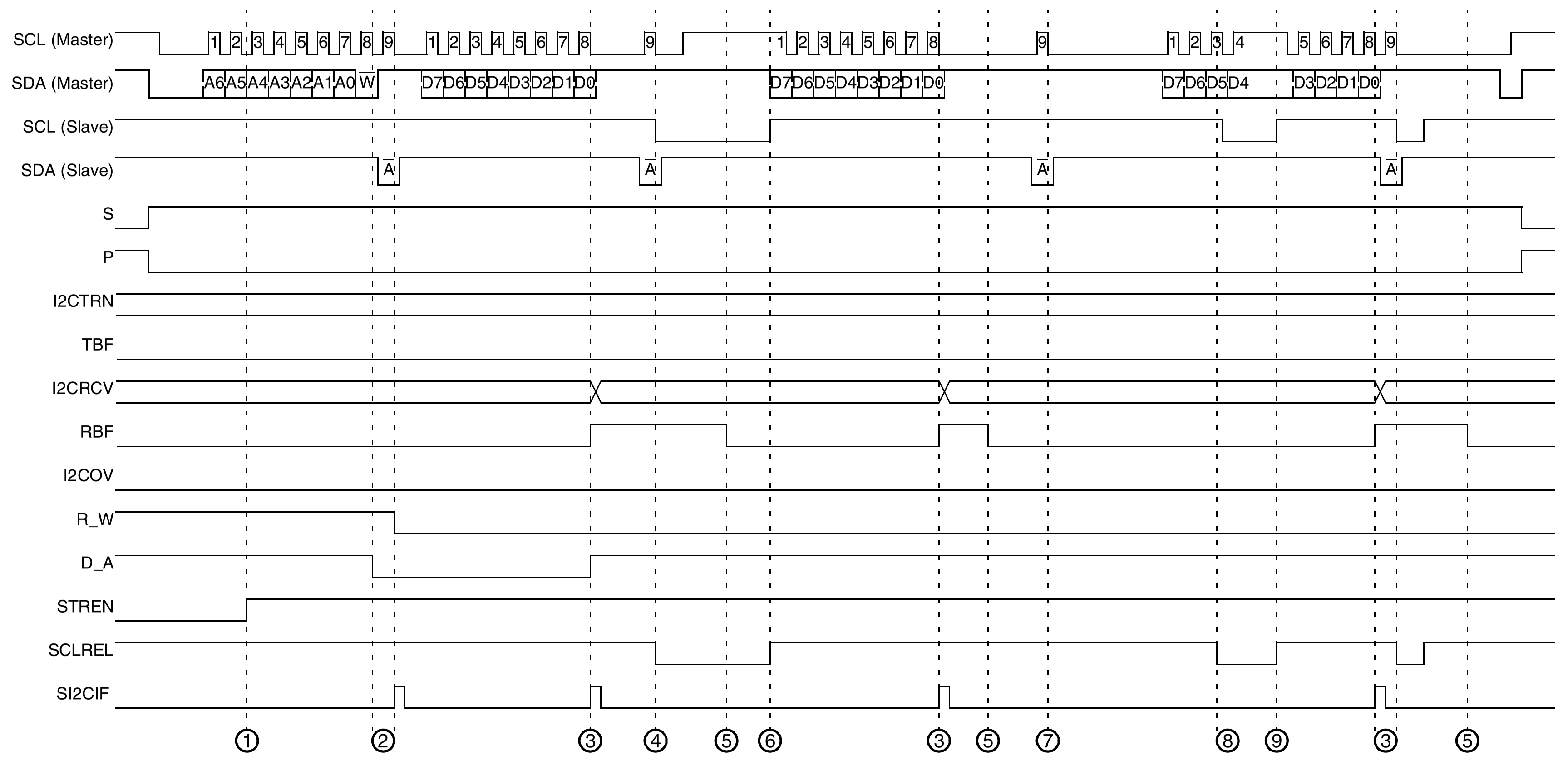
③ - Next byte received before I2CRCV read by software. I2CRCV register unchanged. I2COV overflow bit set. Slave generates interrupt. Slave sends NACK for reception.

④ - Next byte also received before I2CRCV read by software. I2CRCV register unchanged. Slave generates interrupt. Slave sends NACK for reception.

⑥ - Software reads I2CRCV register. RBF bit clears.

⑦ - Software clears I2COV bit.

Figure 21-30: Slave Message (Write Data to Slave: 7-bit Address; Clock Stretching Enabled; A10M = 0; GCEN = 0; IPMIEN = 0)



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

② - Slave receives address byte.

③ - Next received byte is message data. Byte moved to I2CRCV register, sets RBF.

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

⑤ - Software reads I2CRCV register. RBF bit clears.

⑥ - Software sets SCLREL bit to release clock.

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

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

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

21.7.5 Sending Data to a Master Device

When the R/$\overline{W}$ bit of the incoming device address byte is one and an address match occurs, the R_W bit (I2CSTAT<2>) is set. At this point, the master device is expecting the slave to respond by sending a byte of data. The contents of the byte are defined by the system protocol and are only transmitted by the slave module.

When the interrupt from the address detection occurs, the software can write a byte to the I2CTRN register to start the data transmission.

The slave module sets the TBF bit. The eight data bits are shifted out on the falling edge of the SCL input. This ensures that the SDA signal is valid during the SCL high time. When all eight bits have been shifted out, the TBF bit will be cleared.

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

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

A slave interrupt is generated on the falling edge of the ninth SCL clock. Software must check the status of the I2CSTAT register and clear the SI2CIF flag.

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

21.7.5.1 WAIT States During Slave Transmissions

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

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

The slave module clears the SCLREL bit. Clearing the SCLREL bit causes the slave module to pull the SCL line low, initiating a WAIT. The SCL clock of the master and slave will synchronize as shown in **Section 21.6.2 “Master Clock Synchronization”**.

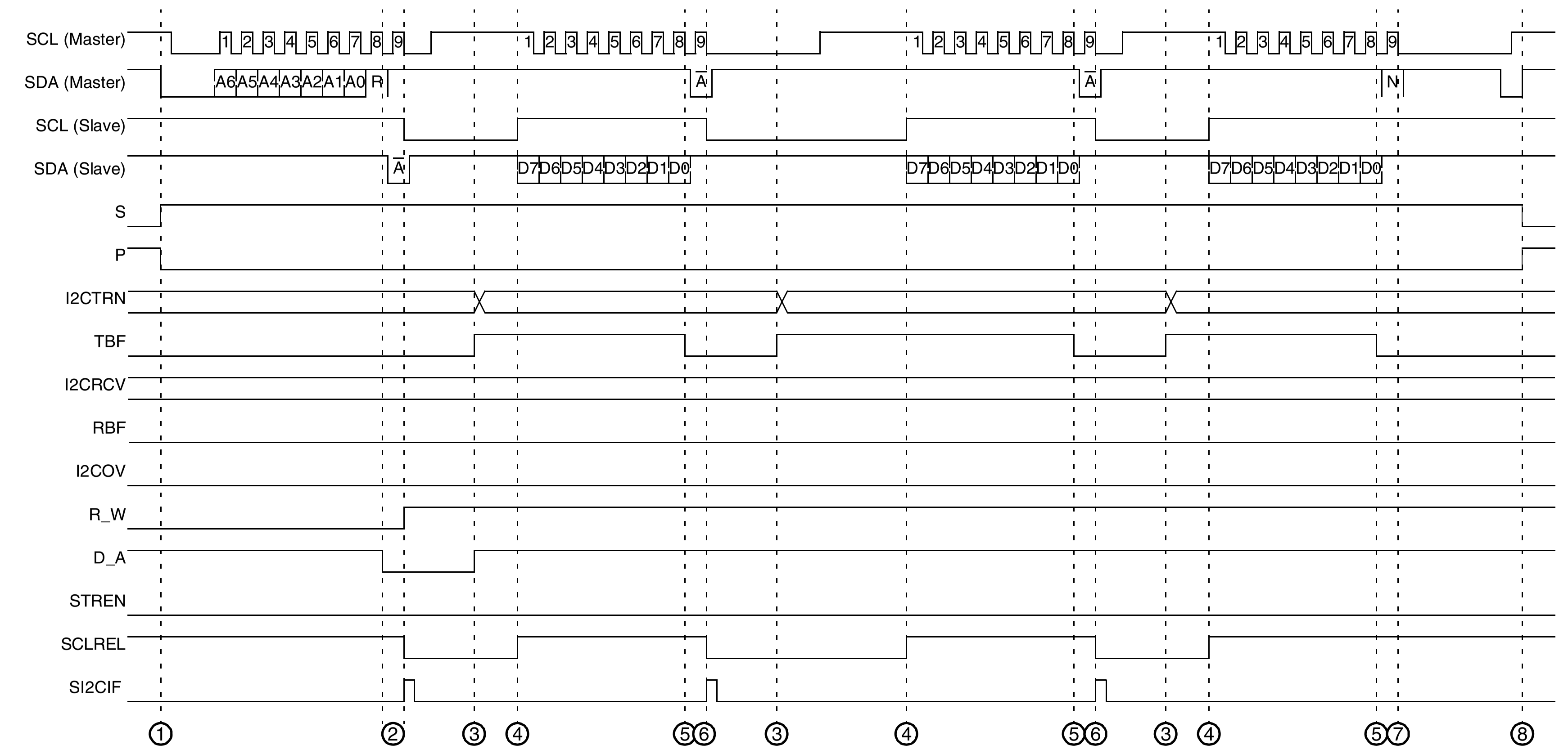
When the software loads the I2CTRN and is ready to resume transmission, the software sets SCLREL. This causes the slave module to release the SCL line and the master resumes clocking.

21.7.5.2 Example Messages of Slave Transmission

Slave transmissions for 7-bit address messages are shown in Figure 21-31. When the address matches and the R/W bit of the address indicates a slave transmission, the module will automatically initiate clock stretching by clearing the SCLREL bit and generate an interrupt to indicate a response byte is required. The software will write the response byte into the I2CTRN register. As the transmission completes, the master will respond with an Acknowledge. If the master replies with an ACK, the master expects more data and the module will again clear the SCLREL bit and generate another interrupt. If the master responds with a NACK, no more data is required and the module will not stretch the clock nor generate an interrupt.

Slave transmissions for 10-bit address messages require the slave to first recognize a 10-bit address. Because the master must send two bytes for the address, the R/W bit in the first byte of the address specifies a write. To change the message to a read, the master will send a Repeated Start and repeat the first byte of the address with the R/W bit specifying a read. At this point, the slave transmission begins as shown in Figure 21-32.

Figure 21-31: Slave Message (Read Data from Slave: 7-bit Address)



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

② - Slave receives address byte. Address matches. Slave generates interrupt. Address byte not moved to I2CRCV register. R_W = 1 to indicate read from slave. SCLREL = 0 to suspend master clock.

③ - Software writes I2CTRN with response data. TBF = 1 indicates that buffer is full. Writing I2CTRN sets D_A, indicating data byte.

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

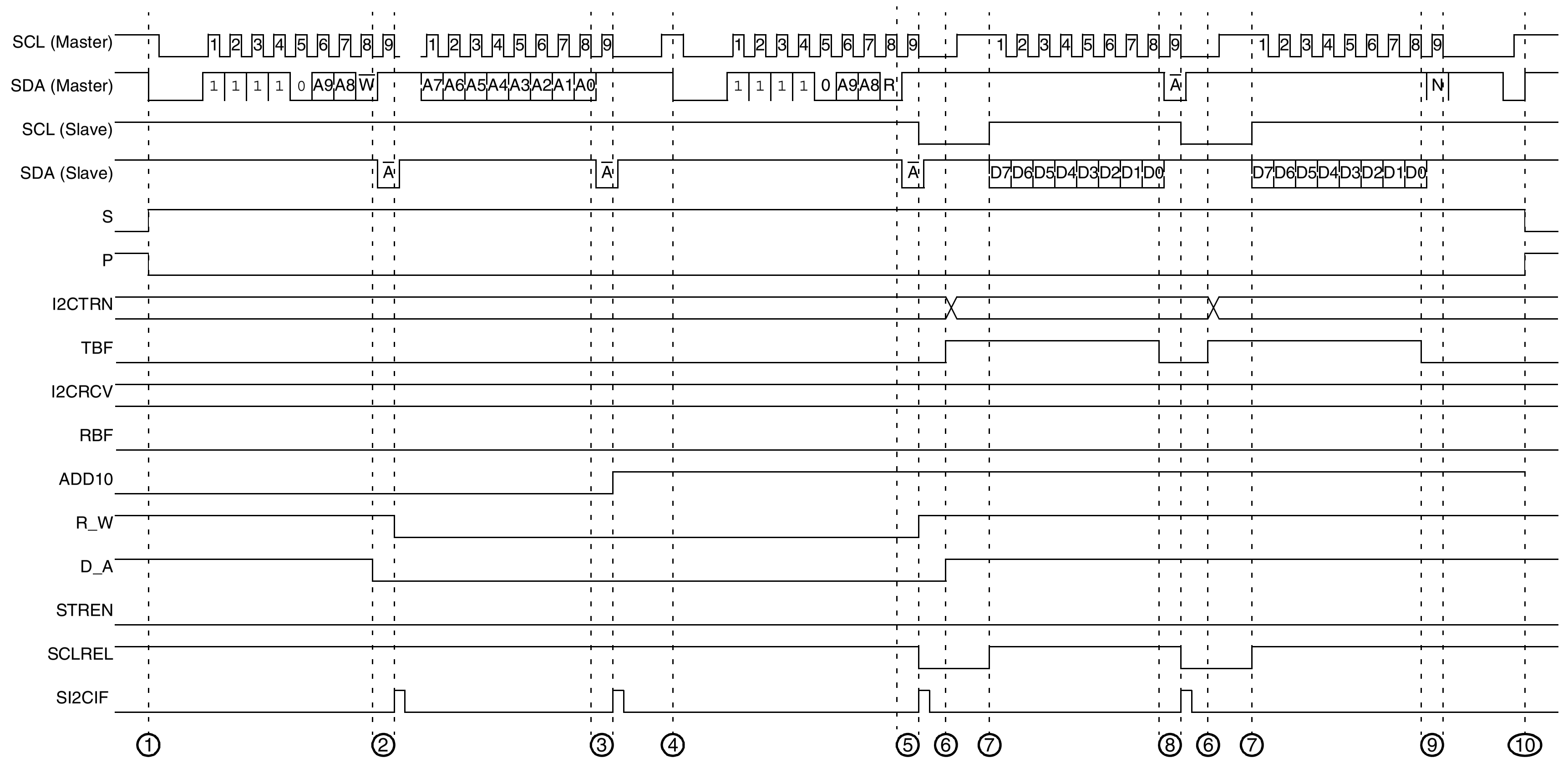
⑤ - After last bit, module clears TBF bit indicating buffer is available for next byte.

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

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

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

Figure 21-32: Slave Message (Read Data from Slave: 10-bit Address)



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

② - Slave receives first address byte. Write indicated. Slave Acknowledges and generates interrupt.

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

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

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

⑥ - Software writes I2CTRN with response data.

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

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

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

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

21.8 Connection Considerations for I²C Bus

By definition of the I²C bus being a wired AND bus connection, pull-up resistors on the bus are required, shown as R_P in Figure 21-33. Series resistors, shown as R_S are optional and used to improve ESD susceptibility. The values of resistors R_P and R_S depend on the following parameters:

- Supply voltage
- Bus capacitance
- Number of connected devices (input current + leakage current)

Because the device must be able to pull the bus low against R_P, current drawn by R_P must be greater than the I/O pin minimum sink current I_{OL} of 3 mA at $V_{OL(MAX)} = 0.4V$ for the device output stage. For example, with a supply voltage of $V_{DD} = 5V +10\%$:

$$R_P(\text{MIN}) = (V_{DD}(MAX) - V_{OL}(MAX)) / I_{OL} = (5.5-0.4) / 3\text{ mA} = 1.7\text{ k}\Omega$$

In a 400 kHz system, a minimum rise time specification of 300 nsec exists and in a 100 kHz system, the specification is 1000 nsec.

Because R_P must pull the bus up against the total capacitance C_B with a maximum rise time of 300 nsec to 0.7 V_{DD}, the maximum resistance for R_P must be less than:

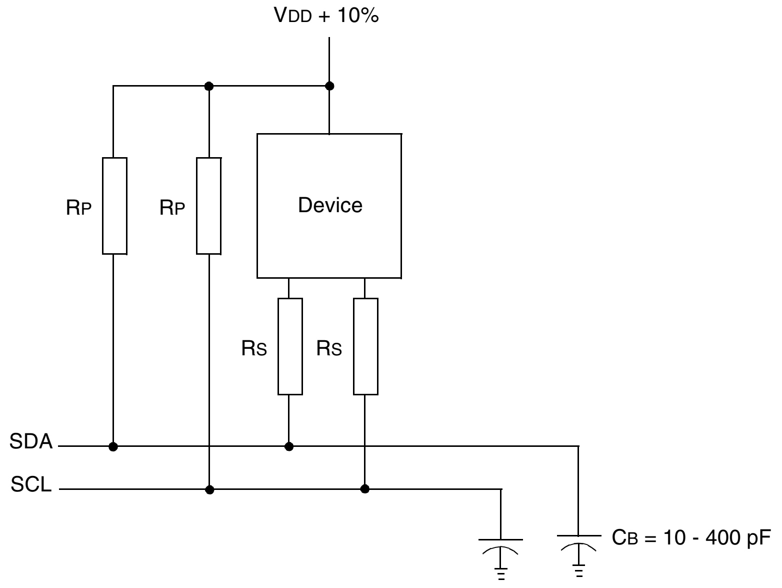
$$R_P(\text{MAX}) = -t_R / C_B * \ln(1 - (V_{IL}(MAX) - V_{DD}(MAX)) = -300\text{ nsec} / (100\text{pf} * \ln(1-0.7)) = 2.5\text{ k}\Omega$$

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

$$R_S(\text{MAX}) = (V_{IL}(MAX) - V_{OL}(MIN)) / I_{OL}(MAX) = (0.3\text{ V}_{DD}-0.4) / 3\text{ mA} = 366\Omega$$

The SCL clock input must have a minimum high and low time for proper operation. The high and low times of the I²C specification as well as the requirements of the I²C module, are shown in the “Electrical Specifications” section in the specific device data sheet.

Figure 21-33: Sample Device Configuration for I²C™ Bus



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

21.8.1 Integrated Signal Conditioning

The SCL and SDA pins have an input glitch filter. The I^2C bus requires this filter in both the 100 kHz and 400 kHz systems.

When operating on a 400 kHz bus, the I^2C specification requires a slew rate control of the device pin output. This slew rate control is integrated into the device. If the DISSLW bit (I2CCON<9>) is cleared, the slew rate control is active. For other bus speeds, the I^2C specification does not require slew rate control and DISSLW should be set.

Some system implementations of I^2C busses require different input levels for $V_{IL(MAX)}$ and $V_{IH(MIN)}$.

In a normal I^2C system:

$V_{IL(MAX)}$ = lesser of 1.5V and 0.3 V_{DD}

$V_{IH(MIN)}$ = greater of 3.0V and 0.7 V_{DD}

In an SMBus (System Management Bus) system:

$V_{IL(MAX)}$ = 0.2 V_{DD}

$V_{IH(MIN)}$ = 0.8 V_{DD}

The SMEN bit (I2CCON<8>) controls the input levels. SMEN is set to change the input levels to SMBus specifications.

21.9 Module Operation During `PWRSAV` Instruction

21.9.1 When the Device Enters Sleep Mode

When the device executes a `PWRSAV 0` instruction, the device enters Sleep mode. When the device enters Sleep mode, the master and slave module abort any pending message activity and reset the state of the modules. Any transmission/reception that is in progress will not continue when the device wakes from Sleep. After the device returns to Operational mode, the master module will be in an Idle state waiting for a message command and the slave module will be waiting for a Start condition. During Sleep, the IWCOL, I2COV and BCL bits are cleared. Additionally, because the master functions are aborted, the SEN, RSEN, PEN, RCEN, ACKEN and TRSTAT bits are cleared. TBF and RBF are cleared and the buffers are available at wake-up.

There is no automatic method to prevent Sleep entry if a transmission or reception is active or pending. The software must synchronize Sleep entry with I^{2}C operation to avoid aborted messages.

During Sleep, the slave module will not monitor the I^{2}C bus. Thus, it is not possible to generate a wake-up event based on the I^{2}C bus using the I^{2}C module. Other interrupt inputs, such as the interrupt-on-change inputs can be used to detect message traffic on a I^{2}C bus and cause a device wake-up.

21.9.2 When the Device Enters Idle Mode

When the device executes a `PWRSAV 1` instruction, the device enters Idle mode. The module will enter a power saving state in Idle mode depending on the I2CSIDL bit (I2CCON<13>).

If I2CSIDL = `1`, the module will enter the Power Saving mode similarly to actions while entering Sleep mode.

If I2CSIDL = `0`, the module will not enter a Power Saving mode. The module will continue to operate normally.

21.10 Effects of a Reset

A Reset disables the I^{2}C module and terminates any active or pending message activity. See the register definitions of I2CCON and I2CSTAT for the Reset conditions of those registers.

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

21.11 Design Tips

Question 1: ***I'm operating as a bus master and transmitting data, however, slave and receive interrupts are also occurring.***

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

Question 2: ***I'm operating as a slave and I write data to the I2CTRN register, but the data did not transmit.***

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

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

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

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

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

Question 5: ***My I^{2}C system is a multi-master system. When I attempt to send a message, it is being corrupted.***

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

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

Answer: Look at the S and P bits. If S = `0` and P = `0` the bus is Idle. If S = `0` and P = `1`, the bus is Idle.

Question 7: ***I tried to send a Start condition on the bus, then transmit a byte by writing to the I2CTRN register. The byte did not get transmitted. Why?***

Answer: You must wait for each event on the I^{2}C bus to complete before starting the next one. In this case, you should poll the SEN bit to determine when the Start event completed, or wait for the master I^{2}C interrupt before data is written to I2CTRN.

21.12 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 dsPIC30F Product Family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Inter-Integrated Circuit (I^2C) module are:

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

Note: Please visit the Microchip web site (www.microchip.com) for additional Application Notes and code examples for the dsPIC30F Family of devices.

21.13 Revision History

Revision A

This is the initial released revision of this document.

Revision B

This revision has been expanded to contain a full description of the dsPIC30F Inter-Integrated Circuit (I^2C) module.

Revision C

This revision incorporates all known errata at the time of this document update.