Section 24. Inter-Integrated Circuit

HIGHLIGHTS

This section of the manual contains the following topics:

24.1 OVERVIEW

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

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

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

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

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

In a multi-master system, bus collision conflicts with other masters in the system are detected and reported to the application (BCOL Interrupt). The software can terminate, and then restart the message transmission.

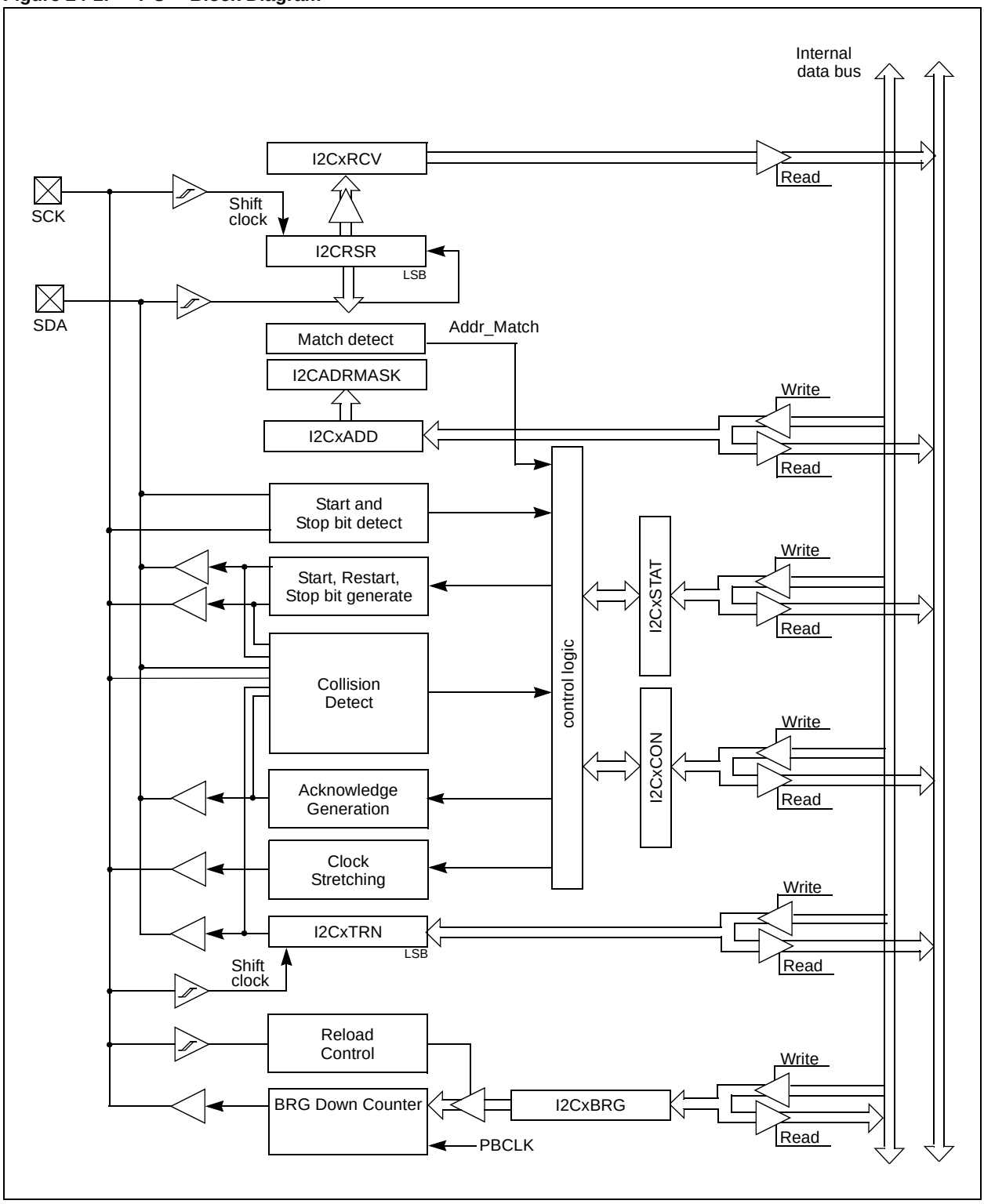
The I^2C module contains a Baud Rate Generator (BRG). The I^2C Baud Rate Generator does not consume other timer resources in the device.

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

- Independent master and slave logic
- Multi-master support which prevents message losses in arbitration
- Detects 7-bit and 10-bit device addresses with configurable address masking in Slave mode
- Detects general call addresses as defined in the I^2C protocol
- Automatic SCLx clock stretching provides delays for the processor to respond to a slave data request
- Supports 100 kHz and 400 kHz bus specifications
- Supports Strict I^2C Reserved Address Rule

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

Figure 24-1: I²C™ Block Diagram



24.2 CONTROL AND STATUS REGISTERS

Note:	Each PIC32MX device variant may have one or more I^2C modules. An 'x' used in the names of pins, control/Status bits, and registers denotes the particular module. Refer to the specific device data sheets for more details.

The PIC32MX I^2C module consists of the following Special Function Registers (SFRs):

- I2CxCON: Control Register for the I^2C Module

  I2CxCONCLR, I2CxCONSET, I2CxCONINV: Atomic Bit Manipulation Write-only Registers for I2CxCON

- I2CxSTAT: Status Register for the I^2C Module

  I2CxSTATCLR, I2CxSTATSET, I2CxSTATINV: Atomic Bit Manipulation Write-only Registers for I2CxSTAT

- I2CxMSK: Address Mask Register (designates which bit positions in I2CxADD can be ignored, which allows for multiple address support)

  I2CxMSKCLR, I2CxMSKSET, I2CxMSKINV: Atomic Bit Manipulation Write-only Registers for I2CxMSK

- I2CxRCV: Receive Buffer Register (read-only)
- I2CxTRN: Transmit Register (read/write)
- I2CxTRNCLR, I2CxTRNSET, I2CxTRNINV: Atomic Bit Manipulation Write-only Registers for I2CxTRN
- I2CxADD: Address Register (holds the slave device address)
- I2CxADDCLR, I2CxADDSET, I2CxADDINV: Atomic Bit Manipulation Write-only Registers for I2CxADD
- I2CxBRG: Baud Rate Generator Reload Register (holds the Baud Rate Generator reload value for the I^2C module Baud Rate Generator)
- I2CxBRGCLR, I2CxBRGSET, I2CxBRGINV: Atomic Bit Manipulation Write-only Registers for I2CxBRG

Each I^2C module also has the following associated bits for interrupt control:

- I2CxMIF: Master Interrupt Flag Status Bits – in IFC0, IFC1 INT Registers
- I2CxSIF: Slave Interrupt Flag Status Bits – in IFS0, IFS1 INT Registers
- I2CxBIF: Bus Collision Interrupt Flag Status Bits – in IFS0, IFS1 INT Registers
- I2CxMIE: Master Interrupt Enable Control Bits – in IEC0, IEC1 INT Registers
- I2CxSIE: Slave Interrupt Enable Control Bits – in IEC0, IEC1 INT Registers
- I2CxBIE: Bus Collision Interrupt Enable Control Bits – in IEC0, IEC1 INT Registers
- I2CxIP<2:0>: Interrupt Priority Control Bits – in IPC6, IPC8 INT Registers
- I2CxIS<1:0>: Interrupt Sub-Priority Control Bits – in IPC6, IPC8 INT Registers

The following table summarizes all I²C-module-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

I²C™ SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
I2CxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	FRZ	SIDL	SCLREL	STRICT	A10M	DISSLW	SMEN
	7:0	GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
I2CxCONCLR	31:0	Writes clear selected bits of I2CxCON, reads yield undefined value							
I2CxCONSET	31:0	Writes set selected bits of I2CxCON, reads yield undefined value							
I2CxCONINV	31:0	Writes invert selected bits of I2CxCON, reads yield undefined value							
I2CxSTAT	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ACKSTAT	TRSTAT	—	—	—	BCL	GCSTAT	ADD10
	7:0	IWCOL	I2COV	D/$\overline{A}$	P	S	R/$\overline{W}$	RBF	TBF
I2CxSTATCLR	31:0	Writes clear selected bits of I2CxSTAT, reads yield undefined value							
I2CxSTATSET	31:0	Writes set selected bits of I2CxSTAT, reads yield undefined value							
I2CxSTATINV	31:0	Writes invert selected bits of I2CxSTAT, reads yield undefined value							
I2CxADD	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	ADD<9:8>	
	7:0	ADD<7:0>							
I2CxADDCLR	31:0	Writes clear selected bits of I2CxADD, reads yield undefined value							
I2CxADDSET	31:0	Writes set selected bits of I2CxADD, reads yield undefined value							
I2CxADDINV	31:0	Writes invert selected bits of I2CxADD, reads yield undefined value							
I2CxMSK	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	MSK<9:8>	
	7:0	MSK<7:0>							
I2CxMSKCLR	31:0	Writes clear selected bits of I2CxMSK, reads yield undefined value							
I2CxMSKSET	31:0	Writes set selected bits of I2CxMSK, reads yield undefined value							
I2CxMSKINV	31:0	Writes invert selected bits of I2CxMSK, reads yield undefined value							
I2CxBRG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	I2CxBRG<11:8>			
	7:0	I2CxBRG<7:0>							
I2CxBRGCLR	31:0	Writes clear selected bits of I2CxBRG, reads yield undefined value							
I2CxBRGSET	31:0	Writes set selected bits of I2CxBRG, reads yield undefined value							
I2CxBRGINV	31:0	Writes invert selected bits of I2CxBRG, reads yield undefined value							
I2CxTRN	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	I2CTxDATA<7:0>							
I2CxTRNCLR	31:0	Writes clear selected bits of I2CxTRN, reads yield undefined value							
I2CxTRNSET	31:0	Writes set selected bits of I2CxTRN, reads yield undefined value							
I2CxTRNINV	31:0	Writes invert selected bits of I2CxTRN, reads yield undefined value							

I²C™ SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
I2CxRCV	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	—
	7:0	I2CRXDATA<7:0>							
IFS0	31:24	I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
	23:16	SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
	15:8	INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
	7:0	INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
IFS1	31:24	—	—	—	—	—	—	USBIF	FCEIF
	23:16	—	—	—	—	DMA3IF	DMA2IF	DMA1IF	DMA0IF
	15:8	RTCCIF	FSCMIF	I2C2MIF	I2C2SIF	I2C2BIF	U2TXIF	U2RXIF	U2EIF
	7:0	SPI2RXIF	SPI2TXIF	SPI2EIF	CMP2IF	CMP1IF	PMPIF	AD1IF	CNIF
IEC0	31:24	I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
	23:16	SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
	15:8	INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
	7:0	INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
IEC1	31:24	—	—	—	—	—	—	USBIE	FCEIE
	23:16	—	—	—	—	DMA3IE	DMA2IE	DMA1IE	DMA0IE
	15:8	RTCCIE	FSCMIE	I2C2MIE	I2C2SIE	I2C2BIE	U2TXIE	U2RXIE	U2EIE
	7:0	SPI2RXIE	SPI2TXIE	SPI2EIE	CMP2IE	CMP1IE	PMPIE	AD1IE	CNIE
IPC6	31:24	—	—	—	AD1IP<2:0>			AD1IS<1:0>	
	23:16	—	—	—	CNIP<2:0>			CNIS<1:0>	
	15:8	—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
	7:0	—	—	—	U1IP<2:0>			U1IS<1:0>	
IPC8	31:24	—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
	23:16	—	—	—	FSCMIP<2:0>			FSCMIS<1:0>	
	15:8	—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
	7:0	—	—	—	U2IP<2:0>			U2IS<1:0>	

Register 24-1: I2CxCON: I²C Control Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
ON	FRZ	SIDL	SCLREL	STRICT	A10M	DISSLW	SMEN
bit 15							bit 8

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

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ON:** I²C Enable bit
1 = Enables the I²C module and configures the SDA and SCL pins as serial port pins
0 = Disables I²C module; all I²C pins are controlled by PORT functions

Note: When using 1:1 PBCLK divisor, the user’s software should not read/write the peripheral’s SFRs in the SYSCLK cycle immediately following the instruction that clears the module’s ON bit.

bit 14 **FRZ:** Freeze in DEBUG Mode Control bit (Read/Write only in DEBUG mode; otherwise read as ‘0’)
1 = Freeze module operation when in DEBUG mode
0 = Do not freeze module operation when in DEBUG mode

Note: FRZ is writable in Debug Exception mode only, it is forced to ‘0’ in normal mode.

bit 13 **SIDL:** Stop in IDLE Mode bit
1 = Discontinue module operation when device enters IDLE mode
0 = Continue module operation in IDLE mode

bit 12 **SCLREL:** SCL Release Control bit
In I²C Slave mode only
Module Reset and (ON = 0) sets SCLREL = 1
If STREN = 0:
1 = Release clock
0 = Force clock low (clock stretch)

Note: Automatically cleared to ‘0’ at beginning of slave transmission.

If STREN = 1:
1 = Release clock
0 = Holds clock low (clock stretch). User may program this bit to ‘0’ to force a clock stretch at the next SCL low.

Note: Automatically cleared to ‘0’ at beginning of slave transmission; automatically cleared to ‘0’ at end of slave reception.

Register 24-1: I2CxCON: I²C Control Register (Continued)

bit 11 **STRICT:** Strict I²C Reserved Address Rule Enable bit
1 = Strict reserved addressing is enforced. Device doesn't respond to reserved address space or generate addresses in reserved address space.
0 = Strict I²C Reserved Address Rule not enabled

bit 10 **A10M:** 10-bit Slave Address Flag bit
1 = I2CxADD is a 10-bit slave address
0 = I2CADD is a 7-bit slave address

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

bit 8 **SMEN:** SMBus Input Levels Disable bit
1 = Enable input logic so that thresholds are compliant with SMBus specification
0 = Disable SMBus specific inputs

bit 7 **GCEN:** General Call Enable bit
In I²C Slave mode only
1 = Enable interrupt when a general call address is received in I2CSR. Module is enabled for reception.
0 = General call address disabled.

bit 6 **STREN:** SCL Clock Stretch Enable bit
In I²C Slave mode only; used in conjunction with SCLREL bit.
1 = Enable clock stretching
0 = Disable clock stretching

bit 5 **ACKDT:** Acknowledge Data bit
In I²C Master mode only; applicable during master receive. Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.
1 = A NACK is sent
0 = ACK is sent

bit 4 **ACKEN:** Acknowledge Sequence Enable bit
In I²C Master mode only; applicable during master receive
1 = Initiate Acknowledge sequence on SDA and SCL pins, and transmit ACKDT data bit; cleared by module
0 = Acknowledge sequence idle

bit 3 **RCEN:** Receive Enable bit
In I²C Master mode only
1 = Enables Receive mode for I²C, automatically cleared by module at end of 8-bit receive data byte
0 = Receive sequence not in progress

bit 2 **PEN:** Stop Condition Enable bit
In I²C Master mode only
1 = Initiate Stop condition on SDA and SCL pins; cleared by module
0 = Stop condition idle

bit 1 **RSEN:** Restart Condition Enable bit
In I²C Master mode only
1 = Initiate Restart condition on SDA and SCL pins; cleared by module
0 = Restart condition idle

bit 0 **SEN:** Start Condition Enable bit
In I²C Master mode only
1 = Initiate Start condition on SDA and SCL pins; cleared by module
0 = Start condition idle

Register 24-2: I2CxCONCLR: I²C 'x' Control Clear Register

Write clears selected bits in I2CxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxCON**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxCONCLR = 0x00008001` will clear bits 15 and 0 in I2CxCON register.

Register 24-3: I2CxCONSET: I²C 'x' Control Set Register

Write sets selected bits in I2CxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxCON**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxCONSET = 0x00008001` will set bits 15 and 0 in I2CxCON register.

Register 24-4: I2CxCONINV: I²C 'x' Control Invert Register

Write inverts selected bits in I2CxCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxCON**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxCON register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxCONINV = 0x00008001` will invert bits 15 and 0 in I2CxCON register.

Register 24-5: I2CxSTAT: I^{2}C Status Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R-0	R-0	r-x	r-x	r-x	R/W-0	R-0	R-0
ACKSTAT	TRSTAT	—	—	—	BCL	GCSTAT	ADD10
bit 15							bit 8

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

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16 **Reserved:** Write ‘0’; ignore read

bit 15 **ACKSTAT:** Acknowledge Status bit
In both I^{2}C Master and Slave modes; applicable to both transmit and receive.
1 = Acknowledge was not received
0 = Acknowledge was received

bit 14 **TRSTAT:** Transmit Status bit
In I^{2}C Master mode only; applicable to Master Transmit mode.
1 = Master transmit is in progress (8 bits + $\overline{ACK}$)
0 = Master transmit is not in progress

bit 13-11 **Reserved:** Write ‘0’; ignore read

bit 10 **BCL:** Master Bus Collision Detect bit
Cleared when the I^{2}C module is disabled (ON = 0).
1 = A bus collision has been detected during a master operation
0 = No collision has been detected

bit 9 **GCSTAT:** General Call Status bit
Cleared after Stop detection.
1 = General call address was received
0 = General call address was not received

bit 8 **ADD10:** 10-bit Address Status bit
Cleared after Stop detection.
1 = 10-bit address was matched
0 = 10-bit address was not matched

bit 7 **IWCOL:** Write Collision Detect bit
1 = An attempt to write the I2CxTRN register collided because the I^{2}C module is busy.
Must be cleared in software.
0 = No collision

Register 24-5: I2CxSTAT: I²C Status Register (Continued)

bit 6 **I2COV:** I²C Receive Overflow Status bit
1 = A byte is received while the I2CxRCV register is still holding the previous byte. I2COV is a "don't care" in Transmit mode. Must be cleared in software.
0 = No overflow

bit 5 **D/$\overline{A}$:** Data/Address bit
Valid only for Slave mode operation.
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit
Updated when Start, Reset or Stop detected; cleared when the I²C module is disabled (ON = 0).
1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last

bit 3 **S:** Start bit
Updated when Start, Reset or Stop detected; cleared when the I²C module is disabled (ON = 0).
1 = Indicates that a start (or restart) bit has been detected last
0 = Start bit was not detected last

bit 2 **R/$\overline{W}$:** Read/Write Information bit
Valid only for Slave mode operation.
1 = Read – indicates data transfer is output from slave
0 = Write – indicates data transfer is input to slave

bit 1 **RBF:** Receive Buffer Full Status bit
1 = Receive complete; I2CxRCV is full
0 = Receive not complete; I2CxRCV is empty

bit 0 **TBF:** Transmit Buffer Full Status bit
1 = Transmit in progress; I2CxTRN is full (8-bits of data)
0 = Transmit complete; I2CxTRN is empty

Register 24-6: I2CxSTATCLR: I²C 'x' Status Clear Register

Write clears selected bits in I2CxSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxSTAT**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxSTATCLR = 0x00008001` will clear bits 15 and 0 in I2CxSTAT register.

Register 24-7: I2CxSTATSET: I²C 'x' Status Set Register

Write sets selected bits in I2CxSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxSTAT**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxSTATSET = 0x00008001` will set bits 15 and 0 in I2CxSTAT register.

Register 24-8: I2CxSTATINV: I²C 'x' Status Invert Register

Write inverts selected bits in I2CxSTAT, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxSTAT**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxSTAT register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxSTATINV = 0x00008001` will invert bits 15 and 0 in I2CxSTAT register.

Register 24-9: I2CxADD: I^{2}C Slave Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	ADD<9:8>	
bit 15							bit 8

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

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-10 **Reserved:** Write ‘0’; ignore read

bit 9-0 **ADD<9:0>:** I^{2}C Slave Device Address bits
Either Master or Slave mode

Register 24-10: I2CxADDCLR: I²C 'x' Slave Address Clear Register

Write clears selected bits in I2CxADD, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxADD**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxADD register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxADDCLR = 0x00008001` will clear bits 15 and 0 in I2CxADD register.

Register 24-11: I2CxADDSET: I²C 'x' Slave Address Set Register

Write sets selected bits in I2CxADD, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxADD**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxADD register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxADDSET = 0x00008001` will set bits 15 and 0 in I2CxADD register.

Register 24-12: I2CxADDINV: I²C 'x' Slave Address Invert Register

Write inverts selected bits in I2CxADD, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxADD**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxADD register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxADDINV = 0x00008001` will invert bits 15 and 0 in I2CxADD register.

Register 24-13: I2CxMSK: I²C Address Mask Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	MSK<9:8>	
bit 15							bit 8

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

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-10 **Reserved:** Write ‘0’; ignore read

bit 9-0 **MSK<9:0>:** I²C Address Mask bits

1 = Forces a “don’t care” in the particular bit position on the incoming address match sequence.
0 = Address bit position must match the incoming I²C address match sequence.

Note: MSK<9:8> and MSK<0> are only used in I²C 10-bit mode.

Register 24-14: I2CxMSKCLR: I²C 'x' Address Mask Clear Register

Write clears selected bits in I2CxMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxMSK**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxMSKCLR = 0x00008001` will clear bits 15 and 0 in I2CxMSK register.

Register 24-15: I2CxMSKSET: I²C 'x' Address Mask Set Register

Write sets selected bits in I2CxMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxMSK**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxMSKSET = 0x00008001` will set bits 15 and 0 in I2CxMSK register.

Register 24-16: I2CxMSKINV: I²C 'x' Address Mask Invert Register

Write inverts selected bits in I2CxMSK, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxMSK**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxMSK register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxMSKINV = 0x00008001` will invert bits 15 and 0 in I2CxMSK register.

Register 24-17: I2CxBRG: I^{2}C Baud Rate Generator Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	I2CxBRG<11:8>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2CxBRG<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-12 **Reserved:** Write ‘0’; ignore read

bit 11-0 **I2CxBRG<11:0>:** I^{2}C Baud Rate Generator Value bits
A divider function of the Peripheral Clock.

Register 24-18: I2CxBRGCLR: I²C 'x' Baud Rate Generator Clear Register

Write clears selected bits in I2CxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxBRG**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxBRGCLR = 0x00008001` will clear bits 15 and 0 in I2CxBRG register.

Register 24-19: I2CxBRGSET: I²C 'x' Baud Rate Generator Set Register

Write sets selected bits in I2CxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxBRG**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxBRGSET = 0x00008001` will set bits 15 and 0 in I2CxBRG register.

Register 24-20: I2CxBRGINV: I²C 'x' Baud Rate Generator Invert Register

Write inverts selected bits in I2CxBRG, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxBRG**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxBRG register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxBRGINV = 0x00008001` will invert bits 15 and 0 in I2CxBRG register.

Register 24-21: I2CxTRN: I^{2}C Transmit Data Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2CTXDATA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **I2CTXDATA<7:0>:** I^{2}C Transmit Data Buffer bits

Register 24-22: I2CxTRNCLR: I²C 'x' Transmit Data Clear Register

Write clears selected bits in I2CxTRN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in I2CxTRN**

A write of '1' in one or more bit positions clears the corresponding bit(s) in I2CxTRN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxTRNCLR = 0x00008001` will clear bits 15 and 0 in I2CxTRN register.

Register 24-23: I2CxTRNSET: I²C 'x' Transmit Data Set Register

Write sets selected bits in I2CxTRN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in I2CxTRN**

A write of '1' in one or more bit positions sets the corresponding bit(s) in I2CxTRN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxTRNSET = 0x00008001` will set bits 15 and 0 in I2CxTRN register.

Register 24-24: I2CxTRNINV: I²C 'x' Transmit Data Invert Register

Write inverts selected bits in I2CxTRN, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in I2CxTRN**

A write of '1' in one or more bit positions inverts the corresponding bit(s) in I2CxTRN register and does not affect unimplemented or read-only bits. A write of '0' will not affect the register.

Example: `I2CxTRNINV = 0x00008001` will invert bits 15 and 0 in I2CxTRN register.

Register 24-25: I2CxRCV: I²C Receive Data Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
I2CRXDATA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-8 **Reserved:** Write ‘0’; ignore read

bit 7-0 **I2CRXDATA<7:0>:** I²C Receive Data Buffer bits

Register 24-26: IFS0: Interrupt Flag Status Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIF	I2C1SIF	I2C1BIF	U1TXIF	U1RXIF	U1EIF	SPI1RXIF	SPI1TXIF
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIF	OC5IF	IC5IF	T5IF	INT4IF	OC4IF	IC4IF	T4IF
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IF	OC3IF	IC3IF	T3IF	INT2IF	OC2IF	IC2IF	T2IF
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	RW-0	RW-0	RW-0
INT1IF	OC1IF	IC1IF	T1IF	INT0IF	CS1IF	CS0IF	CTIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **I2C1MIF**: I^2C Master Interrupt Flag Status bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 30 **I2C1SIF**: I^2C Slave Interrupt Flag Status bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

bit 29 **I2C1BIF**: I^2C Bus Collision Interrupt Flag Status bit
1 = Interrupt request has occurred
0 = No interrupt request has a occurred

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^2C™.

Register 24-27: IEC0: Interrupt Enable Control Register 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
I2C1MIE	I2C1SIE	I2C1BIE	U1TXIE	U1RXIE	U1EIE	SPI1RXIE	SPI1TXIE
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPI1EIE	OC5IE	IC5IE	T5IE	INT4IE	OC4IE	IC4IE	T4IE
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT3IE	OC3IE	IC3IE	T3IE	INT2IE	OC2IE	IC2IE	T2IE
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT1IE	OC1IE	IC1IE	T1IE	INT0IE	CS1IE	CS0IE	CTIE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31 **I2C1MIE**: I^{2}C Master Interrupt Enable Control bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 30 **I2C1SIE**: I^{2}C Slave Interrupt Enable Control bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 29 **I2C1BIE**: I^{2}C Bus Collision Interrupt Enable Control bit
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^{2}C™.

Register 24-28: IPC6: Interrupt Priority Control Register 6(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	AD1IP<2:0>			AD1IS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	CNIP<2:0>			CNIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C1IP<2:0>			I2C1IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U1IP<2:0>			U1IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **I2C1IP<2:0>:** I^2C 1 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled.

bit 9-8 **I2C1IS<1:0>:** I^2C 1 Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^2C™.

Register 24-29: IPC8: Interrupt Priority Control Register 8(1)

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	RTCCIP<2:0>			RTCCIS<1:0>	
bit 31							bit 24

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FCSMIP<2:0>			FCSMIS<1:0>	
bit 23							bit 16

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	I2C2IP<2:0>			I2C2IS<1:0>	
bit 15							bit 8

r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	U2IP<2:0>			U2IS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 12-10 **I2C2IP<2:0>:** I^2C 2 Interrupt Priority bits
- 111 = Interrupt Priority is 7
- 110 = Interrupt Priority is 6
- 101 = Interrupt Priority is 5
- 100 = Interrupt Priority is 4
- 011 = Interrupt Priority is 3
- 010 = Interrupt Priority is 2
- 001 = Interrupt Priority is 1
- 000 = Interrupt is disabled

bit 9-8 **I2C2IS<1:0>:** I^2C 2 Subpriority bits
- 11 = Interrupt Subpriority is 3
- 10 = Interrupt Subpriority is 2
- 01 = Interrupt Subpriority is 1
- 00 = Interrupt Subpriority is 0

Note 1: Shaded bit names in this Interrupt register control other PIC32MX peripherals and are not related to the I^2C™.

24.3 I²C™ BUS CHARACTERISTICS

The I²C bus is a two-wire serial interface. Figure 24-2 shows a schematic of an I²C connection between a PIC32MX device and a 24LC256 I²C serial EEPROM, which is a typical example for any I²C interface.

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

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

In the I²C 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 of operation (transmitter/receiver) during a data transfer. That is, they can be thought of as operating in either of the following two relations:

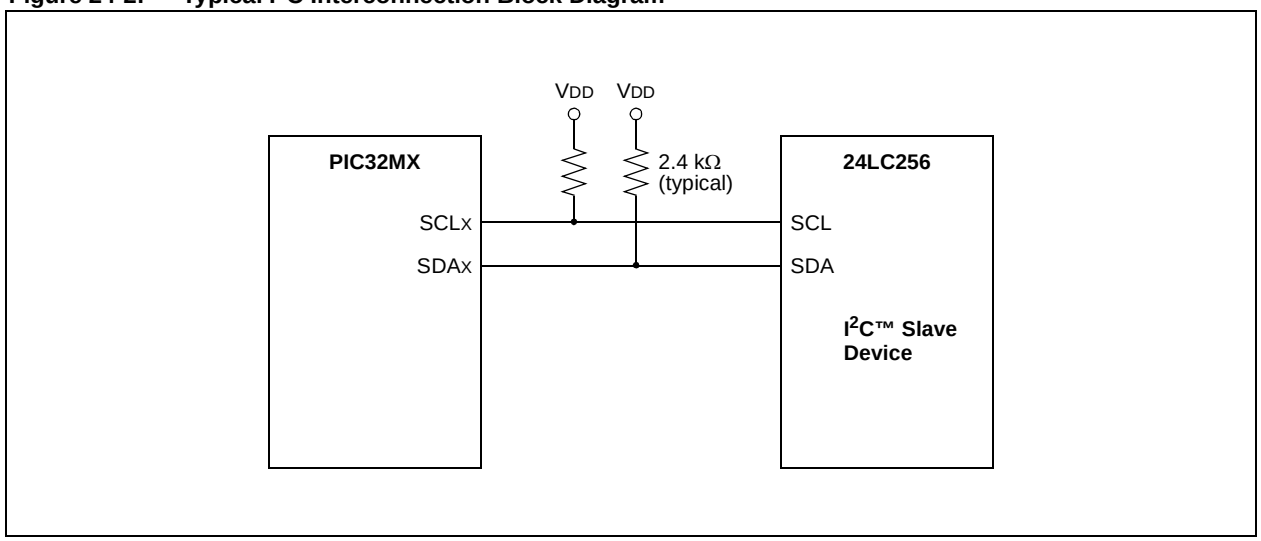
- Master-Transmitter and Slave-Receiver
- Slave-Transmitter and Master-Receiver

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

The following modes and features specified in the V2.1 I²C specifications are not supported:

- HS mode and switching between F/S modes and HS mode
- Start Byte
- CBUS Compatibility
- 2nd byte of General Call Address

Figure 24-2: Typical I²C Interconnection Block Diagram



24.3.1 Bus Protocol

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

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

Accordingly, the following bus conditions have been defined and are shown in Figure 24-3.

24.3.1.1 Start Data Transfer (S)

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

24.3.1.2 Stop Data Transfer (P)

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

24.3.1.3 Repeated Start (R)

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

24.3.1.4 Data Valid (D)

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

24.3.1.5 Acknowledge (A) or Not Acknowledge (N)

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

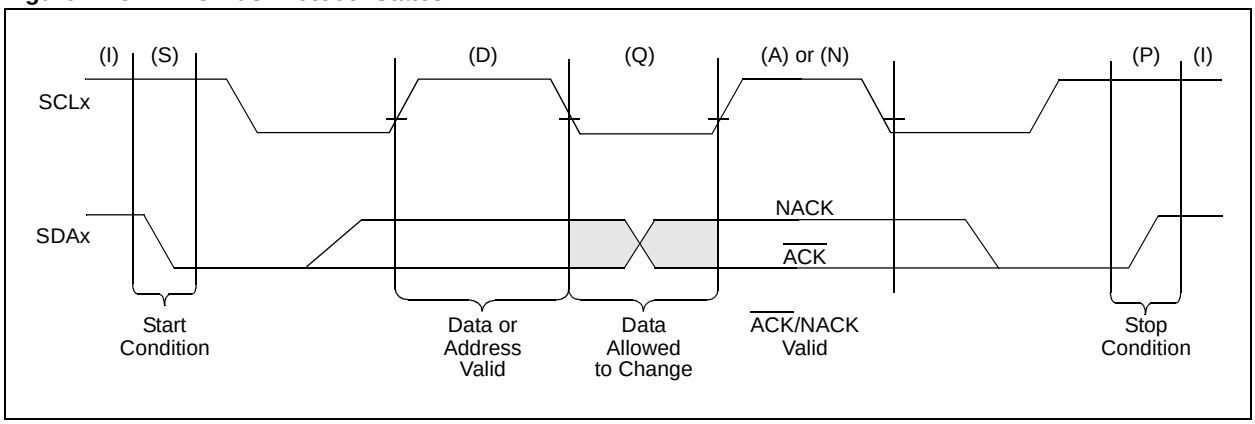
24.3.1.6 Wait/Data Invalid (Q)

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

24.3.1.7 Bus Idle (I)

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

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

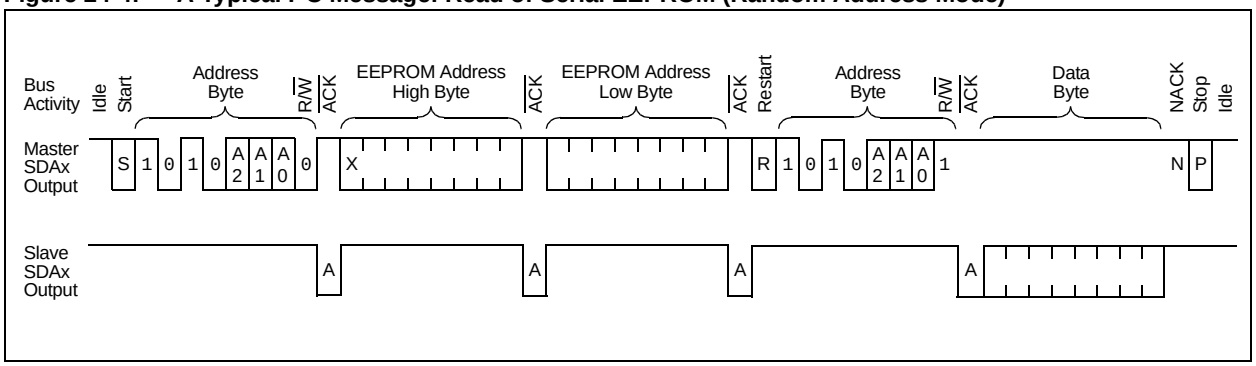


24.3.2 Message Protocol

A typical I^{2}C message is shown in Figure 24-4. In this example, the message will read a specified byte from a 24LC256 I^{2}C serial EEPROM. The PIC32MX device will act as the master and the 24LC256 device will act as the slave.

Figure 24-4 indicates the data as driven by the master device and the data as driven by the slave device, taking into account that the combined SDAx line is a wired AND of the master and slave data. The master device controls and sequences the protocol. The slave device will only drive the bus at specifically determined times.

Figure 24-4: A Typical I^{2}C Message: Read of Serial EEPROM (Random Address Mode)



24.3.2.1 Start Message

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

24.3.2.2 Address Slave

In Figure 24-4, the first byte is the device address byte, that must be the first part of any I^{2}C message. It contains a device address and a R/$\overline{W}$ bit. For additional information on address byte formats, refer to **Appendix A** (check the Microchip web site, www.microchip.com, for availability). Note that R/$\overline{W}$ = 0 for this first address byte, indicating that the master will be a transmitter and the slave will be a receiver.

24.3.2.3 Slave Acknowledge

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

24.3.2.4 Master Transmit

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

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

24.3.2.6 Slave Reply

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

24.3.2.7 Master Acknowledge

During reads, a master must terminate data requests to the slave by Not Acknowledging (generating a “NACK”) on the last byte of the message. Data is acked for each byte, except for the last byte.

24.3.2.8 Stop Message

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

24.4 ENABLING I²C™ OPERATION

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

The I²C module fully implements all master and slave functions. When the module is enabled, the master and slave functions are active simultaneously and will respond according to the software or bus events.

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

24.4.1 Enabling I²C I/O

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

24.4.2 I²C Interrupts

The I²C module generates three interrupt signals: slave interrupt (I2CxSIF), master interrupt (I2CxMIF) and bus collision interrupt (I2CxBIF). The slave interrupt, master interrupt and bus collision interrupt signals are pulsed high for at least one PBCLK on the falling edge of the 9th clock pulse of the SCL clock. These interrupts will set the corresponding interrupt flag bit and will interrupt the CPU if the corresponding interrupt enable bit is set and the corresponding interrupt priority is high enough.

Master mode operations that generate a master interrupt (I2CxMIF) are as follows:

1. Start Condition
   - 1 BRG (Baud Rate Generator) time after falling edge of SDA
2. Repeated Start Sequence
   - 1 BRG time after falling edge of SDA
3. Stop Condition
   - 1 BRG time after the rising edge of SDA
4. Data transfer byte received
   - 8th falling edge of SCL
   (After receiving eight bits of data from slave)
5. During SEND $\overline{ACK}$ sequence
   - 9th falling edge of SCL
   (After sending $\overline{ACK}$ or NACK to slave)
6. Data transfer byte transmitted
   - 9th falling edge of SCL
   (Regardless of receiving $\overline{ACK}$ from slave)
7. During a slave-detected Stop
   - When slave sets P bit

Slave mode operations that generate a slave interrupt (I2CxSIF) are as follows:

1. Detection of a valid device address (including general call)
   - 9th falling edge of SCL
   (After sending ACK to master. Address must match unless STRICT = 1 or GCEN = 1)
2. Reception of data
   - 9th falling edge of SCL
   (After sending the ACK to master)
3. Request to transmit data
   - 9th falling edge of SCL
   (Regardless of receiving ACK from master)

Bus Collision events that generate an interrupt (I2CxBIF) are as follows:

1. During Start sequence
   - SDA sampled before start condition
2. During Start sequence
   - SCL = 0 before SDA = 0
3. During Start sequence
   - SDA = 0 before BRG time out
4. During a Repeated Start sequence
   - If SDA is sampled 0 when SCL goes high
5. During a Repeated Start sequence
   - If SCL goes low before SDA goes low
6. During a Stop sequence
   - If SDA is sampled low after allowing it to float
7. During a Stop sequence
   - If SCL goes low before SDA goes high

24.4.3 I²C Transmit and Receive Registers

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

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

If the module receives another complete byte before the software reads the previous byte from the I2CxRCV register, a receiver overflow occurs and sets the I2COV bit (I2CxSTAT<6>). The byte in the I2CxRSR is lost.

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

24.4.4 I²C Baud Rate Generator

The Baud Rate Generator used for I²C Master mode operation is used to set the SCL clock frequency for 100 kHz, 400 kHz and 1 MHz. The Baud Rate Generator re-load value is contained in the I2CxBRG register. The Baud Rate Generator will automatically begin counting on a write to the I2CxTRN. Once the given operation is complete (i.e., transmission of the last data bit is followed by ACK) the internal clock will automatically stop counting and the SCL pin will remain in its last state.

24.4.5 Baud Rate Generator in I²C Master Mode

In I²C Master mode, the reload value for the BRG is located in the I2CxBRG register. When the BRG is loaded with this value, the BRG counts down to zero and stops until another reload has taken place. In I²C Master mode, the BRG is not reloaded automatically. If Clock Arbitration is taking place, for instance, the BRG will be reloaded when the SCL pin is sampled high (see Figure 24-6). Table 24-1 shows device frequency vs. I2CxBRG setting for standard baud rates.

Note:	I2CxBRG values of 0x0 and 0x1 are expressly forbidden. The user should never program the I2CxBRG with a value of 0x0 or 0x1, as indeterminate results may occur.

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

Equation 24-1: Baud Rate Generator Reload Value Calculation

$$FSCK = (PBCLK) / ((I2CxBRG+2) * 2)$$

$$I2CBRG = (PBCLK / (2 * FSCK)) - 2$$

Table 24-1: I²C Clock Rate w/BRG

PBCLK	I2CxBRG	Approx. Fsck (2 roll-overs of BRG)
50 MHz	0x03C	400 kHz
50 MHz	0x0F8	100 kHz
40 MHz	0x030	400 kHz
40 MHz	0x0C6	100 kHz
30 MHz	0x023	400 kHz
30 MHz	0x094	100 kHz
20 MHz	0x017	400 kHz
20 MHz	0x062	100 kHz
10 MHz	0x00A	400 kHz
10 MHz	0x030	100 kHz

Figure 24-5: Baud Rate Generator Block Diagram

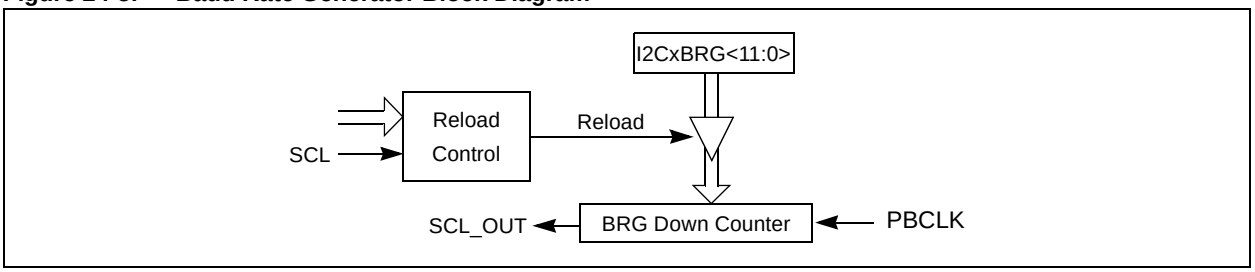
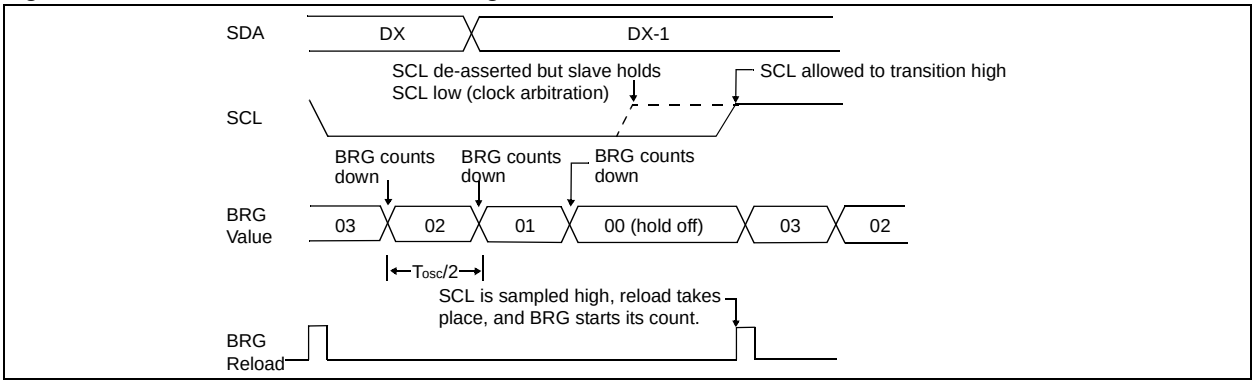


Figure 24-6: Baud Rate Generator Timing With Clock Arbitration



24.5 COMMUNICATING AS A MASTER IN A SINGLE MASTER ENVIRONMENT

The I^2C module's typical operation in a system is using the I^2C to communicate with an I^2C peripheral, such as an I^2C serial memory. In an I^2C system, the master controls the sequence of all data communication on the bus. In this example, the PIC32MX and its I^2C module have the role of the single master in the system. As the single master, it is responsible for generating the SCLx clock and controlling the message protocol.

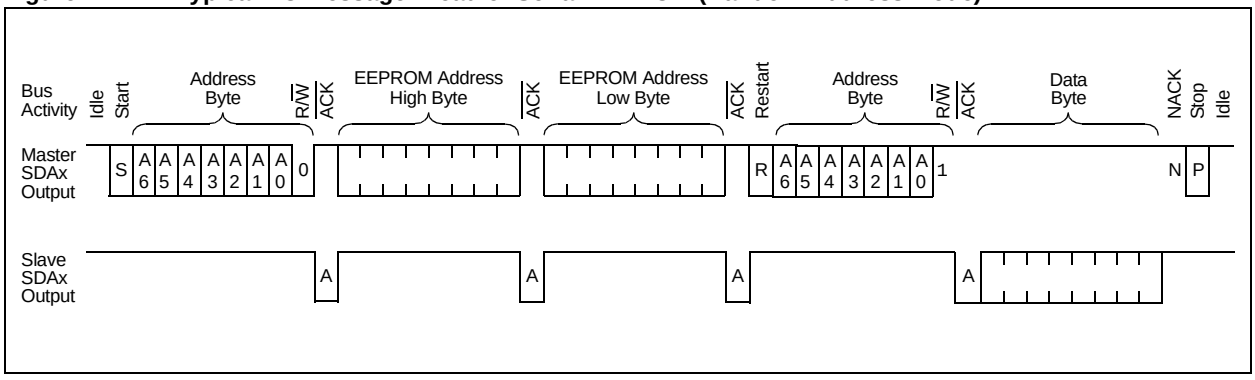
In the I^2C module, the module controls individual portions of the I^2C message protocol; however, sequencing of the components of the protocol to construct a complete message is a software task.

For example, a typical operation in a single master environment may be to read a byte from an I^2C serial EEPROM. This example message is depicted in Figure 24-7.

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

1. Turn on the module by setting ON bit (I2CxCON<15>) to '1'.
1. Assert a Start condition on SDAx and SCLx.
2. Send the I^2C device address byte to the slave with a write indication.
3. Wait for and verify an Acknowledge from the slave.
4. Send the serial memory address high byte to the slave.
5. Wait for and verify an Acknowledge from the slave.
6. Send the serial memory address low byte to the slave.
7. Wait for and verify an Acknowledge from the slave.
8. Assert a Repeated Start condition on SDAx and SCLx.
9. Send the device address byte to the slave with a read indication.
10. Wait for and verify an Acknowledge from the slave.
11. Enable master reception to receive serial memory data.
12. Generate an $\overline{ACK}$ or NACK condition at the end of a received byte of data.
13. Generate a Stop condition on SDAx and SCLx.

Figure 24-7: 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, an Acknowledge generator and a Baud Rate Generator. Generally, the software will write to a control register to start a particular step, then wait for an interrupt or poll status to wait for completion.

Subsequent sections detail each of these operations.

Note:	The I^2C module does not allow queueing of events. For instance, the software is not allowed to initiate a Start condition and then immediately write the I2CxTRN register to initiate transmission before the Start condition is complete. In this case, the I2CxTRN will not be written to and the IWCOL bit will be set, indicating that this write to the I2CxTRN did not occur.

24.5.1 Generating Start Bus Event

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

Figure 24-8 shows the timing of the Start condition.

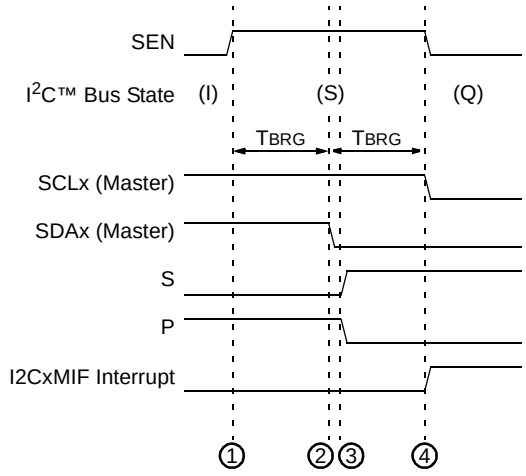
- Slave logic detects the Start condition, sets the S bit (I2CxSTAT<3>) and clears the P bit (I2CxSTAT<4>).
- The SEN bit is automatically cleared at completion of the Start condition.
- I2CxMIF interrupt is generated at completion of the Start condition.
- After the Start condition, the SDAx line and SCLx line are left low (Q state).

24.5.1.1 IWCOL Status Flag

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

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

Figure 24-8: Master Start Timing Diagram



1. Writing SEN = 1 initiates a master Start event. Baud Rate Generator starts.
2. Baud Rate Generator times out. Master module drives SDAx low. Baud Rate Generator restarts.
3. Slave module detects Start and sets S = 1 and P = 0.
4. Baud Rate Generator times out. Master module drives SCLx low, generates interrupt and clears SEN.

24.5.2 Sending Data to a Slave Device

Figure 24-9 shows the timing diagram of master to slave transmission. Transmission of a data byte, a 7-bit device address byte or the second byte of a 10-bit address is accomplished by simply writing the appropriate value to the I2CxTRN register. Loading this register will start the following process:

- The software loads the I2CxTRN with the data byte to transmit.
- Writing I2CxTRN sets the buffer full flag bit, TBF (I2CxSTAT<0>).
- The data byte is shifted out the SDAx pin until all 8 bits are transmitted. Each bit of address/data will be shifted out onto the SDAx pin after the falling edge of SCLx.
- On the ninth SCLx clock, the module shifts in the $\overline{ACK}$ bit from the slave device and writes its value into the ACKSTAT bit (I2CxSTAT<15>).
- The module generates the I2CxMIF interrupt at the end of the ninth SCLx clock cycle.

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

24.5.2.1 Sending a 7-Bit Address to the Slave

Sending a 7-bit device address involves sending one byte to the slave. A 7-bit address byte must contain the 7 bits of the I^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 reception) or a read from the slave (slave transmission and master reception).

24.5.2.2 Sending a 10-Bit Address to the Slave

Sending a 10-bit device address involves sending 2 bytes to the slave. The first byte contains 5 bits of the I^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.

24.5.2.3 Receiving Acknowledge From the Slave

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

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

The status of $\overline{ACK}$ is written into the Acknowledge Status bit, ACKSTAT (I2CxSTAT<15>), on the falling edge of the ninth SCLx clock. After the ninth SCLx clock, the module generates the I2CxMIF interrupt and enters an Idle state until the next data byte is loaded into I2CxTRN.

24.5.2.4 ACKSTAT Status Flag

The ACKSTAT bit (I2CxSTAT<15>) is updated in both Master and Slave modes on the 9th SCL clock irrespective of Transmit or Receive modes. ACKSTAT is cleared when acknowledged ($\overline{ACK}$ = 0 i.e.,SDA is 0 on the 9th clock pulse), and is set when not acknowledged ($\overline{ACK}$ = 1, i.e., SDA is 1 on the 9th clock pulse) by the peer.

24.5.2.5 TBF Status Flag

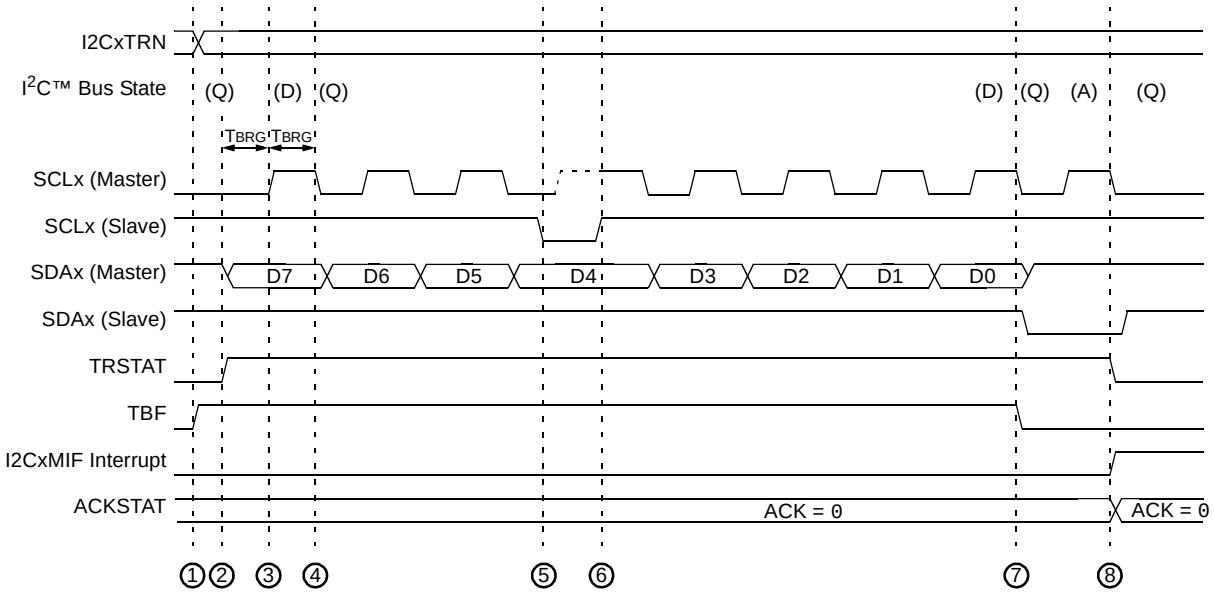
When transmitting, the TBF bit (I2CxSTAT<0>) is set when the CPU writes to I2CXTRN and is cleared when all 8 bits are shifted out.

24.5.2.6 IWCOL Status Flag

If the software writes the I2CxTRN when a transmit is already in progress (i.e., the module is still shifting out a data byte), then IWCOL is set and the contents of the buffer are unchanged (the write doesn't occur). IWCOL must be cleared in software.

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

Figure 24-9: Master Transmission Timing Diagram



1. Writing the I2CxTRN register will start a master transmission event. TBF bit is set.
2. Baud Rate Generator starts. The MSB of the I2CxTRN drives SDAx. SCLx remains low. TRSTAT bit is set.
3. Baud Rate Generator times out. SCLx released. Baud Rate Generator restarts.
4. Baud Rate Generator times out. SCLx driven low. After SCLx detected low, the next bit of I2CxTRN drives SDAx.
5. While SCLx is low, the slave can also pull SCLx low to initiate a wait (clock stretch).
6. Master has already released SCLx and slave can release to end wait. Baud Rate Generator restarts.
7. At falling edge of 8th SCLx clock, master releases SDAx. TBF bit is cleared. Slave drives $\overline{\text{ACK}}$/NACK.
8. At falling edge of 9th SCLx clock, master generates interrupt. SCLx remains low until next event. Slave releases SDAx. TRSTAT bit is clear.

24.5.3 Receiving Data from a Slave Device

Figure 24-10 shows the timing diagram of master reception. The master can receive data from a slave device after the master has transmitted the slave address with an R/$\overline{W}$ bit value of '1'. This is enabled by setting the Receive Enable bit, RCEN (I2CxCON<3>). The master logic begins to generate clocks, and before each falling edge of the SCLx, the SDAx line is sampled and data is shifted into the I2CxRSR.

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

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

- The RCEN bit is automatically cleared.
- The contents of the I2CxRSR transfer into the I2CxRCV.
- The RBF flag bit is set.
- The module generates the I2CxMIF interrupt.

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

24.5.3.1 RBF Status Flag

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

24.5.3.2 I2COV Status Flag

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

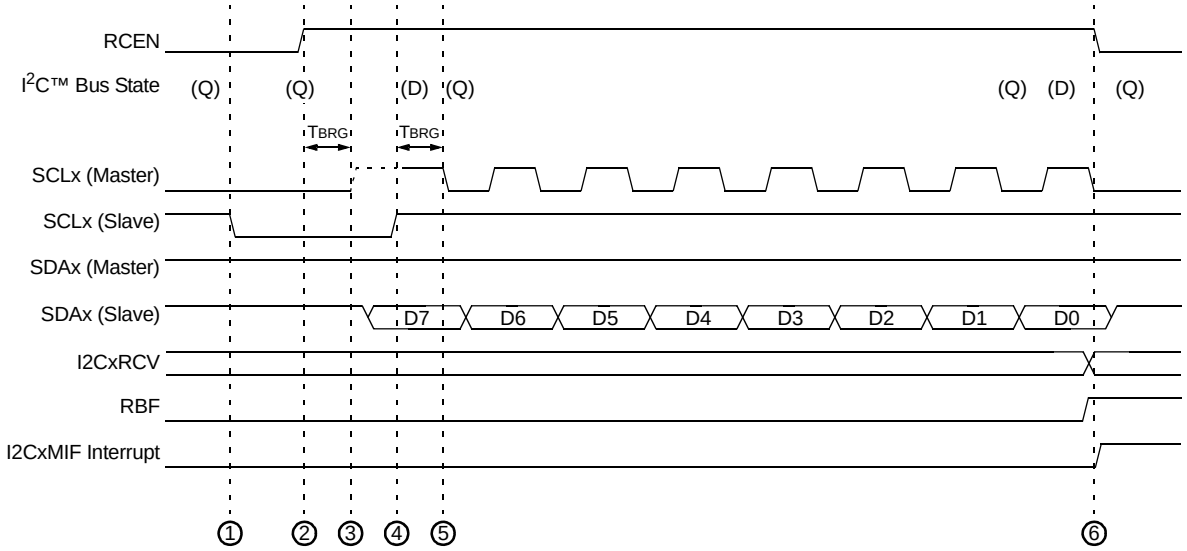
Leaving I2COV set does not inhibit further reception. If RBF is cleared by reading the I2CxRCV and the I2CxRSR receives another byte, that byte will be transferred to the I2CxRCV.

24.5.3.3 IWCOL Status Flag

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

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

Figure 24-10: Master Reception Timing Diagram



1. Typically, the slave can pull SCLx low (clock stretch) to request a wait to prepare data response. The slave will drive the MSB of the data response on SDAx when ready.
2. Writing the RCEN bit will start a master reception event. The Baud Rate Generator starts. SCLx remains low.
3. Baud Rate Generator times out. Master attempts to release SCLx.
4. When slave releases SCLx, Baud Rate Generator restarts.
5. Baud Rate Generator times out. MSB of response shifted to I2CxRSR. SCLx driven low for next baud interval.
6. At falling edge of 8th SCLx clock, I2CxRSR transferred to I2CxRCV. Module clears RCEN bit. RBF bit is set. Master generates interrupt.

24.5.4 Acknowledge Generation

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

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

Figure 24-11 shows an $\overline{ACK}$ sequence and Figure 24-12 shows a NACK sequence. The Acknowledge Data bit, ACKDT (I2CxCON<5>), specifies $\overline{ACK}$ or NACK.

After two baud periods, the ACKEN bit is automatically cleared and the module generates the I2CxMIF interrupt.

24.5.4.1 IWCOL Status Flag

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

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

Figure 24-11: Master Acknowledge ($\overline{ACK}$) Timing Diagram

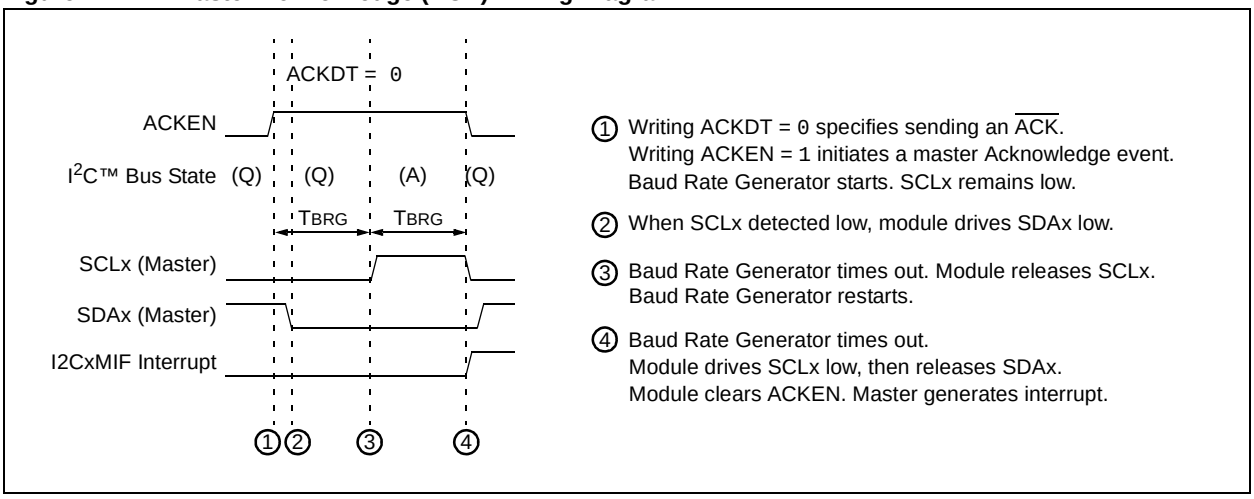
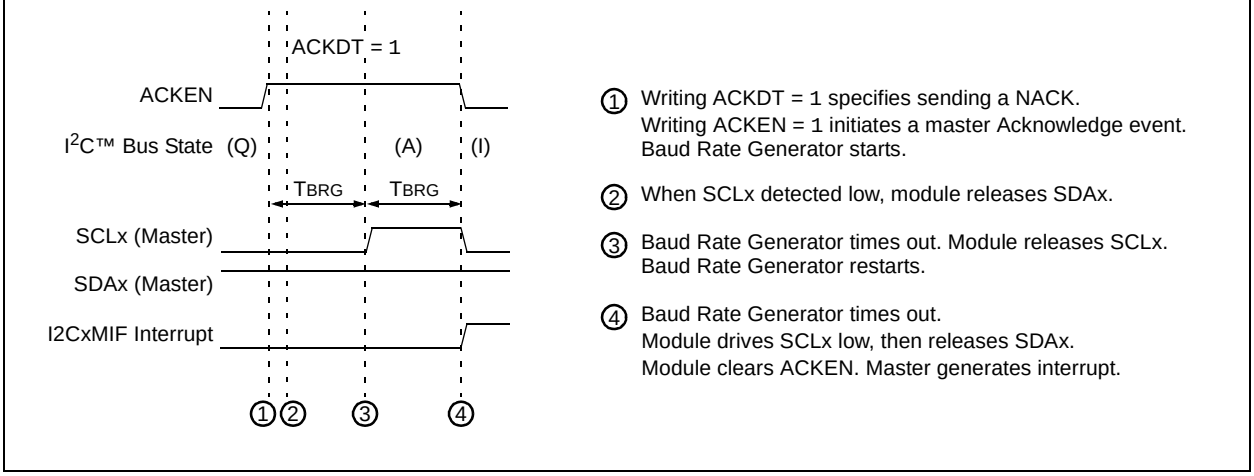


Figure 24-12: Master Not Acknowledge (NACK) Timing Diagram



24.5.5 Generating Stop Bus Event

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

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

When the PEN bit is set, the master generates the Stop sequence as shown in Figure 24-13.

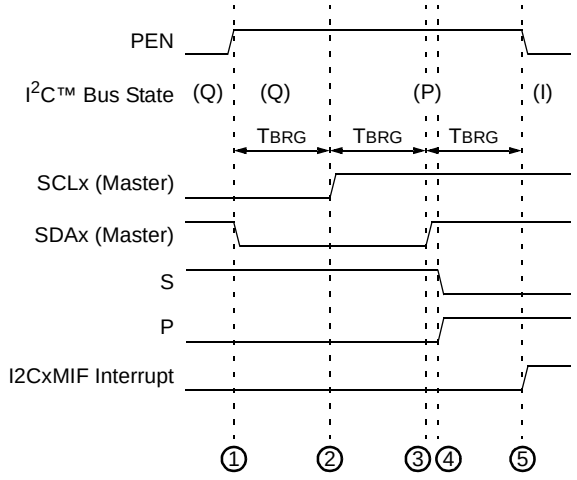
- The slave detects the Stop condition, sets the P bit (I2CxSTAT<4>) and clears the S bit (I2CxSTAT<3>).
- The PEN bit is automatically cleared.
- The module generates the I2CxMIF interrupt.

24.5.5.1 IWCOL Status Flag

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

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

Figure 24-13: Master Stop Timing Diagram



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

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

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

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

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

24.5.6 Generating Repeated Start Bus Event

Setting the Repeated Start Enable bit, RSEN (I2CxCON<1>), enables generation of a master Repeated Start sequence (see Figure 24-14).

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

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

The following is the Repeated Start sequence:

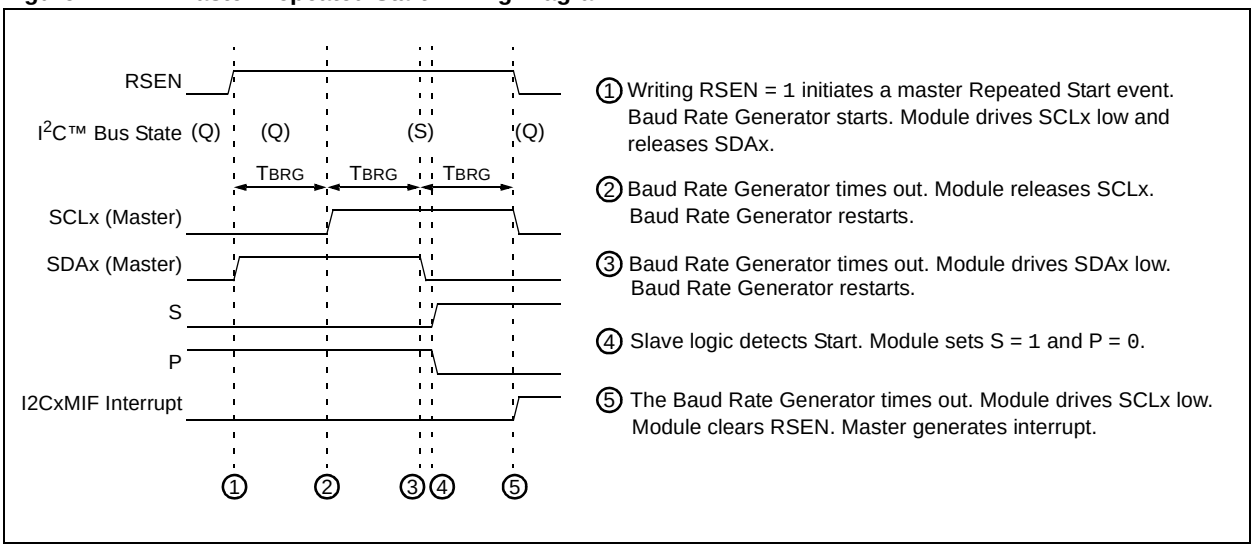
- The slave detects the Start condition, sets the S bit (I2CxSTAT<3>) and clears the P bit (I2CxSTAT<4>).
- The RSEN bit is automatically cleared.
- The module generates the I2CxMIF interrupt.

24.5.6.1 IWCOL Status Flag

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

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

Figure 24-14: Master Repeated Start Timing Diagram



24.5.7 Building Complete Master Messages

As described at the beginning of **Section 24.5 "Communicating as a Master in a Single Master Environment"**, the software is responsible for constructing messages with the correct message protocol. The module controls individual portions of the I^2C message protocol; however, sequencing of the components of the protocol to construct a complete message is a software task.

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

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

Table 24-2: Master Message Protocol States

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

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

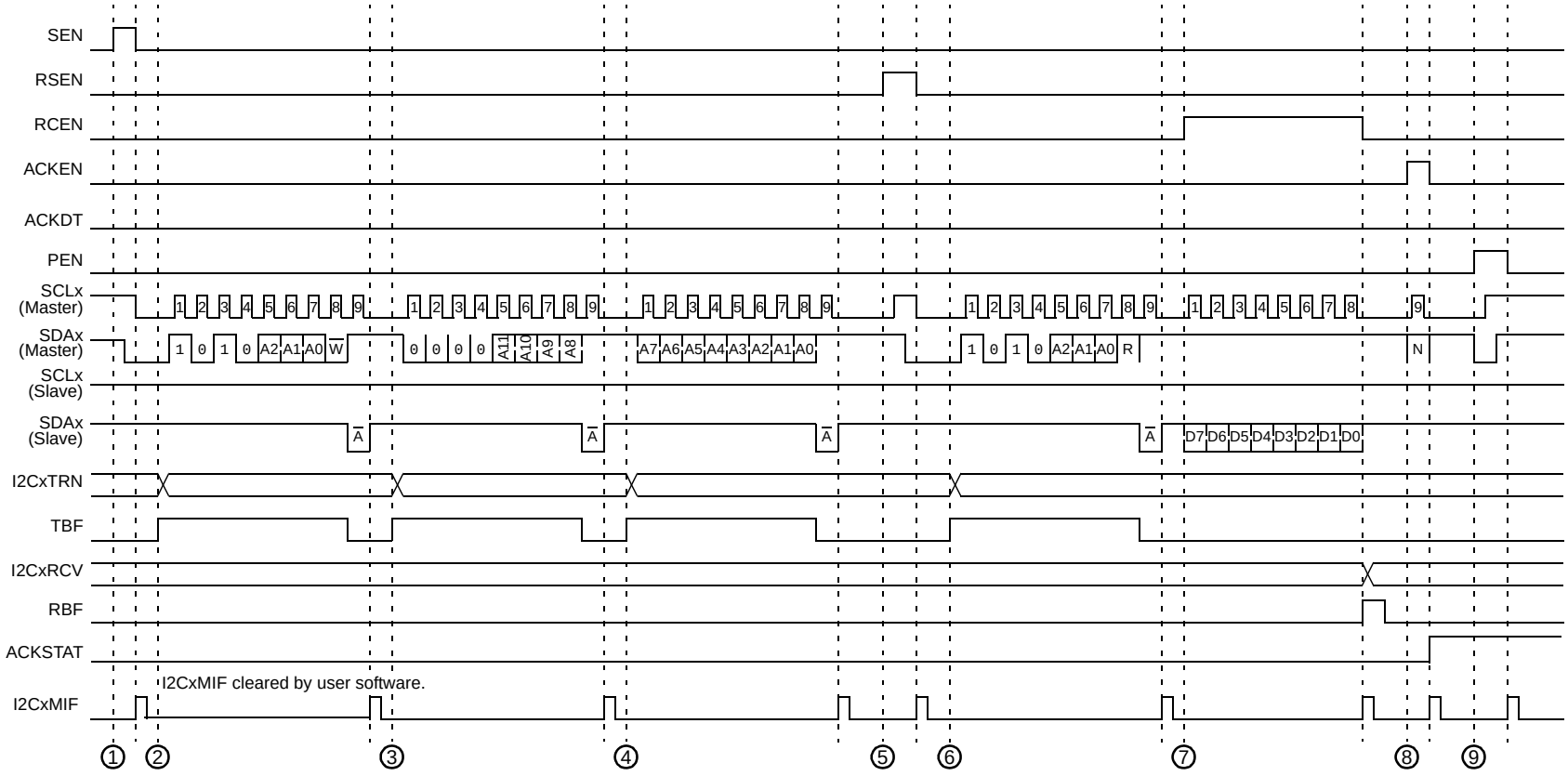
The software will begin a message by issuing a START command. The software will record the state number corresponding to the Start.

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

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

Figure 24-15 provides a more detailed examination of the same message sequence shown in Figure 24-7. Figure 24-16 shows some simple examples of messages using 7-bit addressing format. Figure 24-17 shows an example of a 10-bit addressing format message sending data to a slave. Figure 24-18 shows an example of a 10-bit addressing format message receiving data from a slave.

Figure 24-15: Master Message (Typical I²C Message: Read of Serial EEPROM)



① Setting the SEN bit starts a Start event.

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

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

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

⑤ Setting the RSEN bit starts a Repeated Start event.

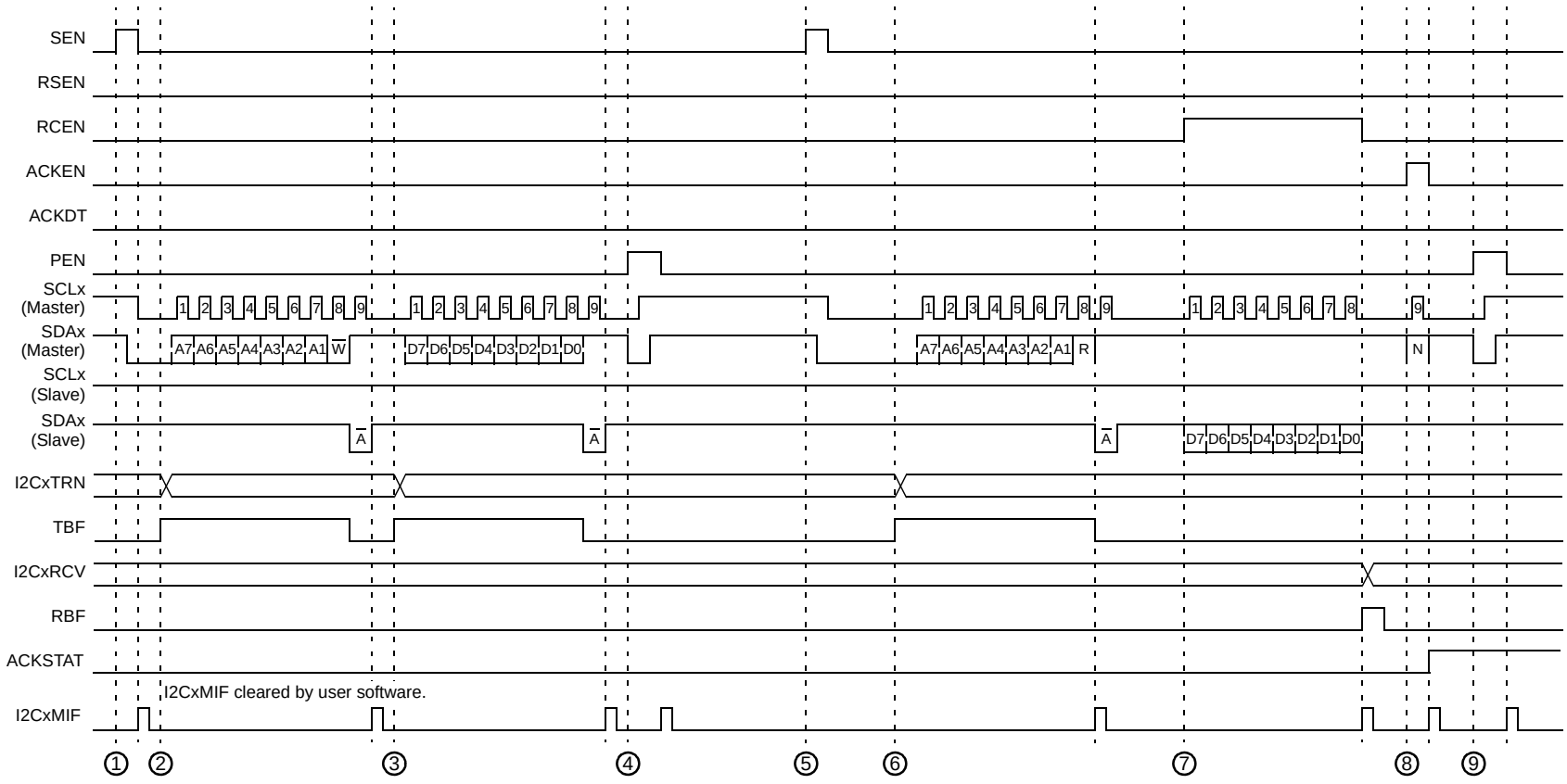
⑥ Writing the I2CxTRN register starts a master transmission. The data is a re-send of the serial EEPROM device address byte, but with R/$\overline{W}$ bit set, indicating a read.

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

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

⑨ Setting the PEN bit starts a master Stop event.

Figure 24-16: Master Message (7-Bit Address: Transmission And Reception)



① Setting the SEN bit starts a Start event.

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

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

④ Setting the PEN bit starts a master Stop event.

⑤ Setting the SEN bit starts a Start event.

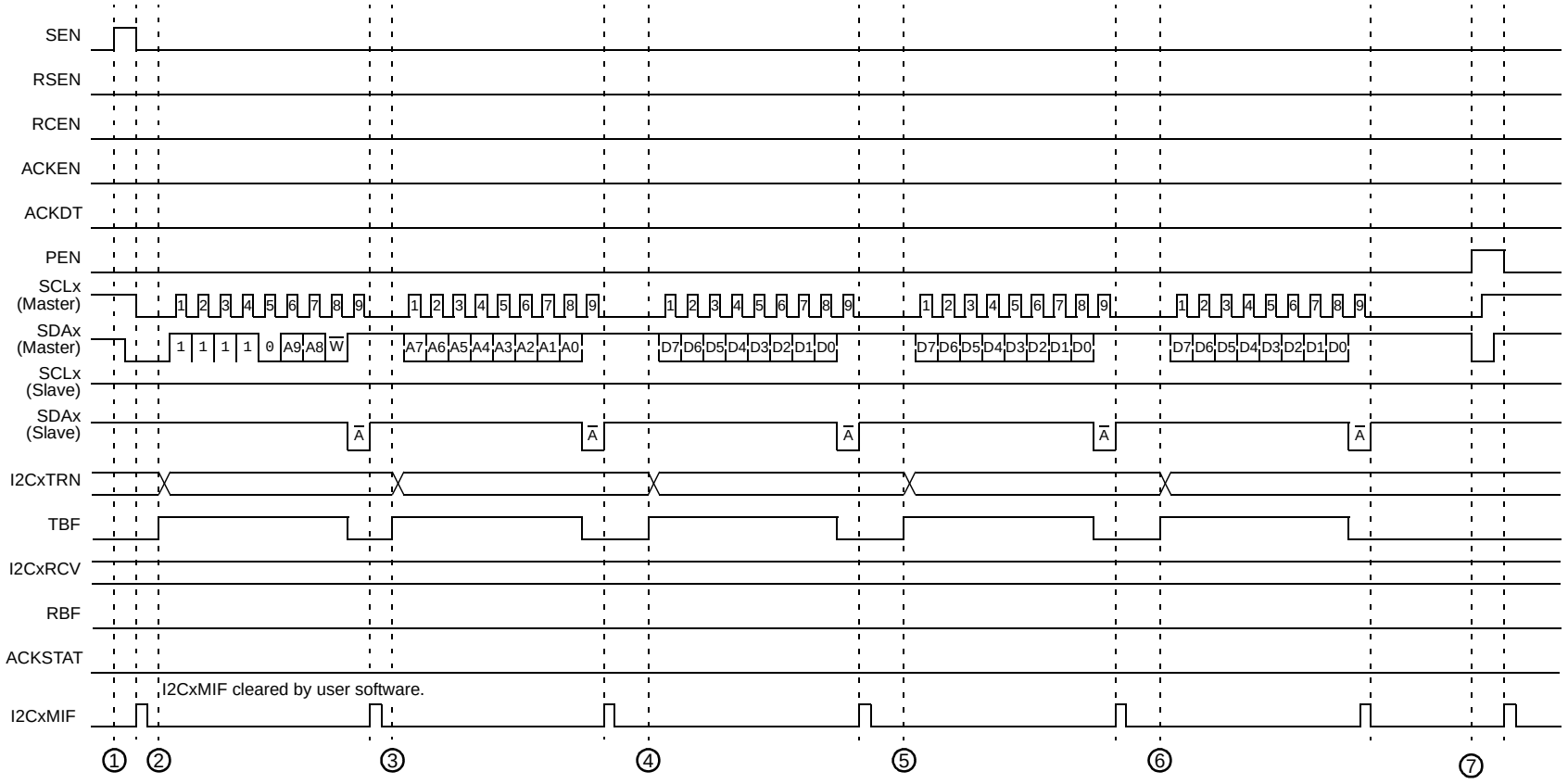
⑥ Writing the I2CxTRN register starts a master transmission. The data is the address byte with R/$\overline{W}$ bit set.

⑦ Setting the RCEN bit starts a master reception.

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

⑨ Setting the PEN bit starts a master Stop event.

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



① Setting the SEN bit starts a Start event.

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

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

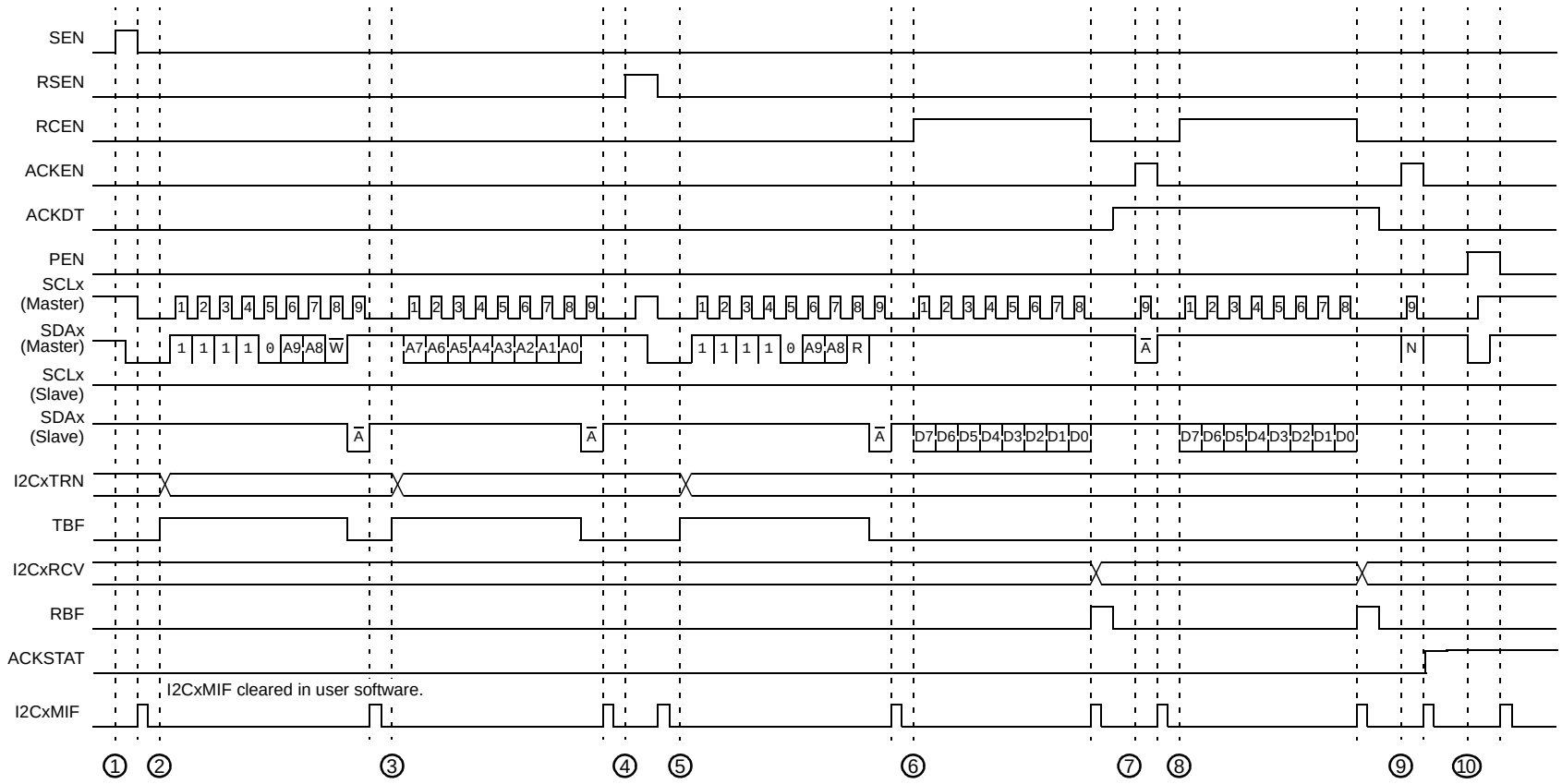
④ Writing the I2CxTRN register starts a master transmission. The data is the first byte of the message data.

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

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

⑦ Setting the PEN bit starts a master Stop event.

Figure 24-18: Master Message (10-Bit Reception)



① Setting the SEN bit starts a Start event.

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

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

④ Setting the RSEN bit starts a master Restart event.

⑤ Writing the I2CxTRN register starts a master transmission. The data is a re-send of the first byte with the R/$\overline{W}$ bit set.

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

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

⑧ Setting the RCEN bit starts a master reception.

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

⑩ Setting the PEN bit starts a master Stop event.

24.6 COMMUNICATING AS A MASTER IN A MULTI-MASTER ENVIRONMENT

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

24.6.1 Multi-Master Operation

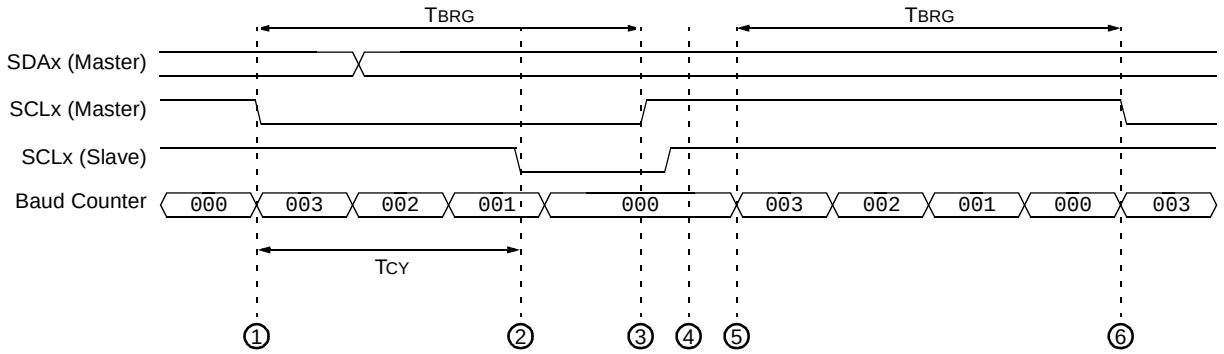
The master module has no special settings to enable multi-master operation. The module performs clock synchronization and bus arbitration at all times. If the module is used in a single master environment, clock synchronization will only occur between the master and slaves, and bus arbitration will not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

24.6.2 Master Clock Synchronization

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

Clock synchronization occurs when the master deasserts the SCLx pin (SCLx intended to float high). When the SCLx pin is released, the BRG is suspended from counting until the SCLx pin is actually sampled high. When the SCLx pin is sampled high, the BRG is reloaded with the contents of I2CxBR<11:0> and begins counting. This ensures that the SCLx high time will always be at least one BRG rollover count in the event that the clock is held low by an external device, as shown in Figure 24-19.

Figure 24-19: Baud Rate Generator Timing with Clock Synchronization



1. The baud counter decrements twice per TCY. On rollover, the master SCLx will transition.
2. The slave has pulled SCLx low to initiate a wait.
3. At what would be the master baud counter rollover, detecting SCLx low holds counter.
4. Logic samples SCLx once per TCY. Logic detects SCLx high.
5. The baud counter rollover occurs on next cycle.
6. On next rollover, the master SCLx will transition.

24.6.3 Bus Arbitration and Bus Collision

Bus arbitration supports multi-master system operation.

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

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

The first master will set the Bus Collision bit, BCL (I2CxSTAT<10>), and generate a bus collision interrupt. The master module will reset the I^2C port to its Idle state.

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

The states where arbitration can be lost are:

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

24.6.4 Detecting Bus Collisions and Re-sending Messages

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

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

24.6.5 Bus Collision During a Start Condition

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

24.6.6 Bus Collision During a Repeated Start Condition

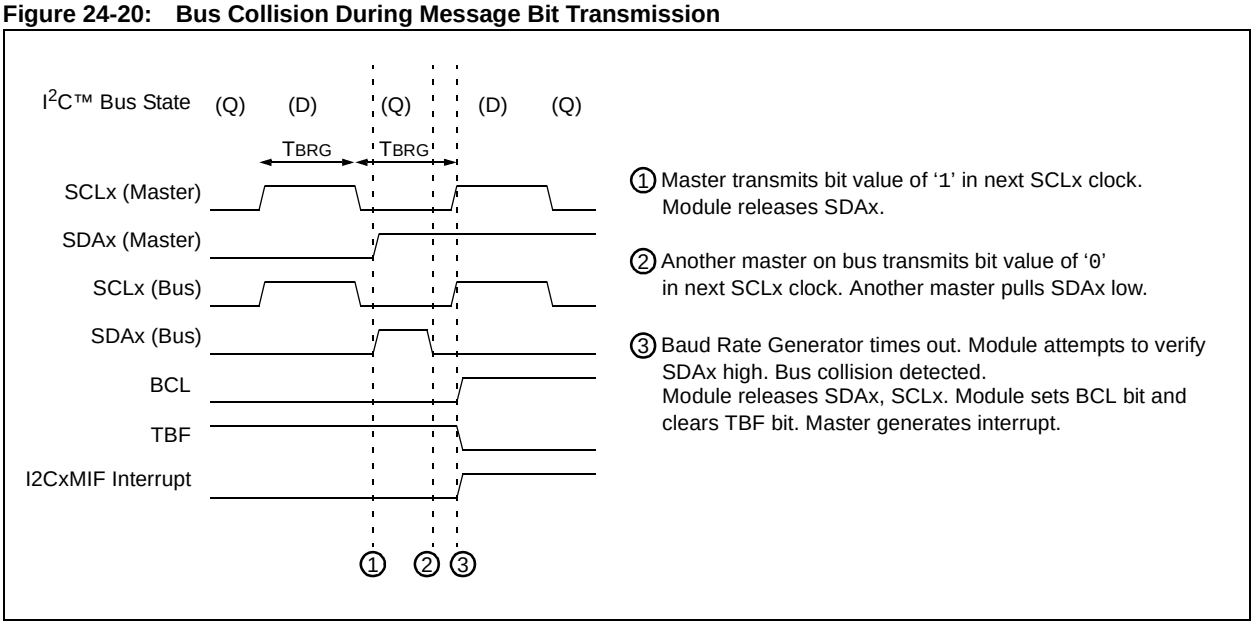
Should two masters not collide throughout an address byte, a bus collision may occur when one master attempts to assert a Repeated Start while another transmits data. In this case, the master generating the Repeated Start will lose arbitration and generate a bus collision interrupt.

24.6.7 Bus Collision During Message Bit Transmission

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

If the software is properly checking the bus state, it is unlikely that a bus collision will occur on a Start condition. However, because another master can, at a very similar time, check the bus and initiate its own Start condition, it is likely that SDAx arbitration will occur and synchronize the Start of two masters. In this condition, both masters will begin and continue to transmit their messages until one master loses arbitration on a message bit. Remember that the SCLx clock synchronization will keep the two masters synchronized until one loses arbitration. Figure 24-20 shows an example of message bit arbitration.

Figure 24-20: Bus Collision During Message Bit Transmission



24.6.8 Bus Collision During a Stop Condition

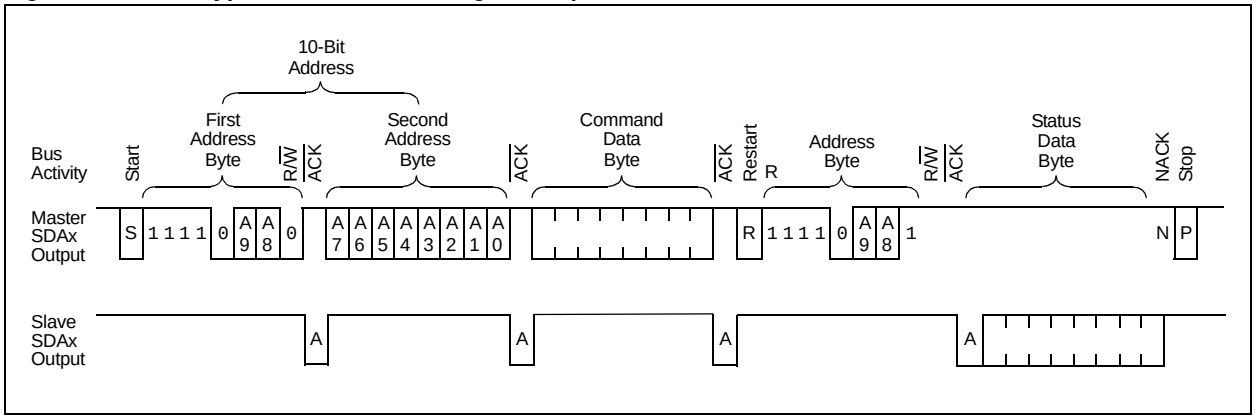
If the master software loses track of the state of the I^{2}C bus, there are conditions which cause a bus collision during a Stop condition. In this case, the master generating the Stop condition will lose arbitration and generate a bus collision interrupt.

24.7 COMMUNICATING AS A SLAVE

In some systems, particularly where multiple processors communicate with each other, the PIC32MX device may communicate as a slave (see Figure 24-21). 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^2C protocol. The slave module replies to the master at the appropriate times as defined by the protocol.

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

Figure 24-21: A Typical Slave I^2C Message: Multiprocessor Command/Status



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

24.7.1 Sampling Receive Data

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

24.7.2 Detecting Start and Stop Conditions

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

24.7.3 Detecting the Address

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

24.7.3.1 Slave Address Masking

The I2CxMSK register masks address bit positions, designating them as “don’t care” bits for both 10-Bit and 7-Bit Addressing modes. When a bit in the I2CxMSK register is set (= 1), it means “don’t care”. The slave module will respond when the bit in the corresponding location of the address is a ‘0’ or ‘1’. For example, in 7-Bit Slave mode with I2CxMSK = 0110000, the module will Acknowledge addresses ‘0010000’ and ‘0100000’ as valid.

24.7.3.2 Limitations of Address Mask

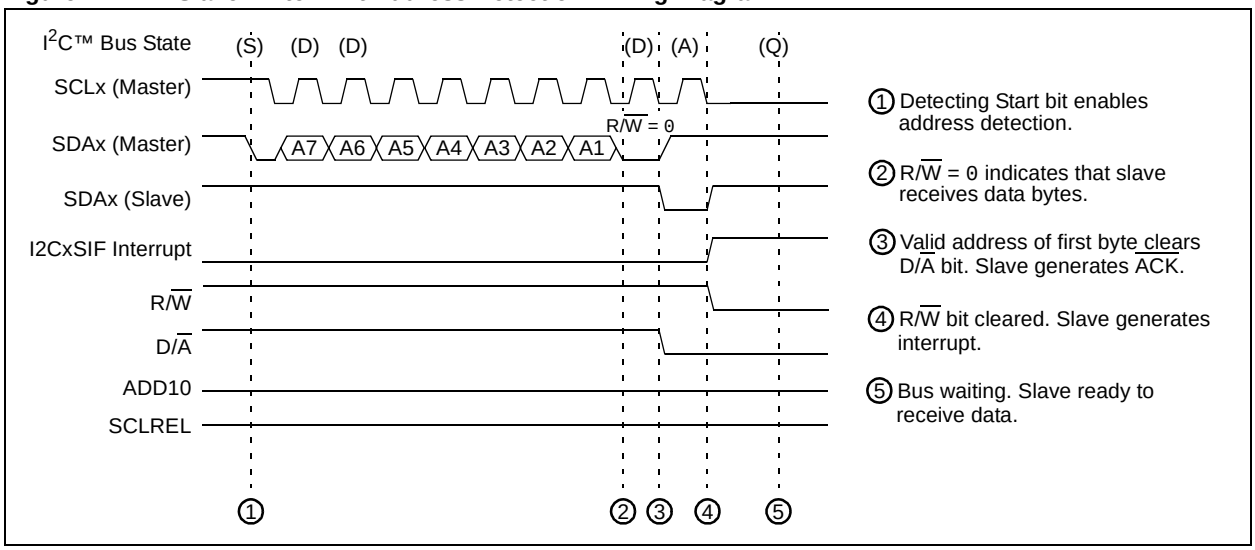
By default, the device will respond or generate addresses in the reserved address space with the address mask enabled (see Table 24-3 for the reserved address spaces). When using the address mask and the STRICT (I2CxCON<11>) bit is cleared, reserved addresses may be acknowledged. If the user wants to enforce the reserved address space, the STRICT bit must be set to a ‘1’. Once the bit is set, the device will not acknowledge reserved addresses regardless of the address mask settings.

24.7.3.3 7-BIT ADDRESS and SLAVE WRITE

Following the Start condition, the module shifts 8 bits into the I2CxRSR register (see Figure 24-22). The value of register I2CxRSR<7:1> is evaluated against that of the I2CxADD<6:0> and I2CxMSK<6:0> registers on the falling edge of the eighth clock (SCLx). If the address is valid (i.e., an exact match between unmasked bit positions), the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D/$\overline{\text{A}}$ and R/$\overline{\text{W}}$ bits are cleared.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.
4. The module will wait for the master to send data.

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



24.7.3.4 7-Bit Address and Slave Read

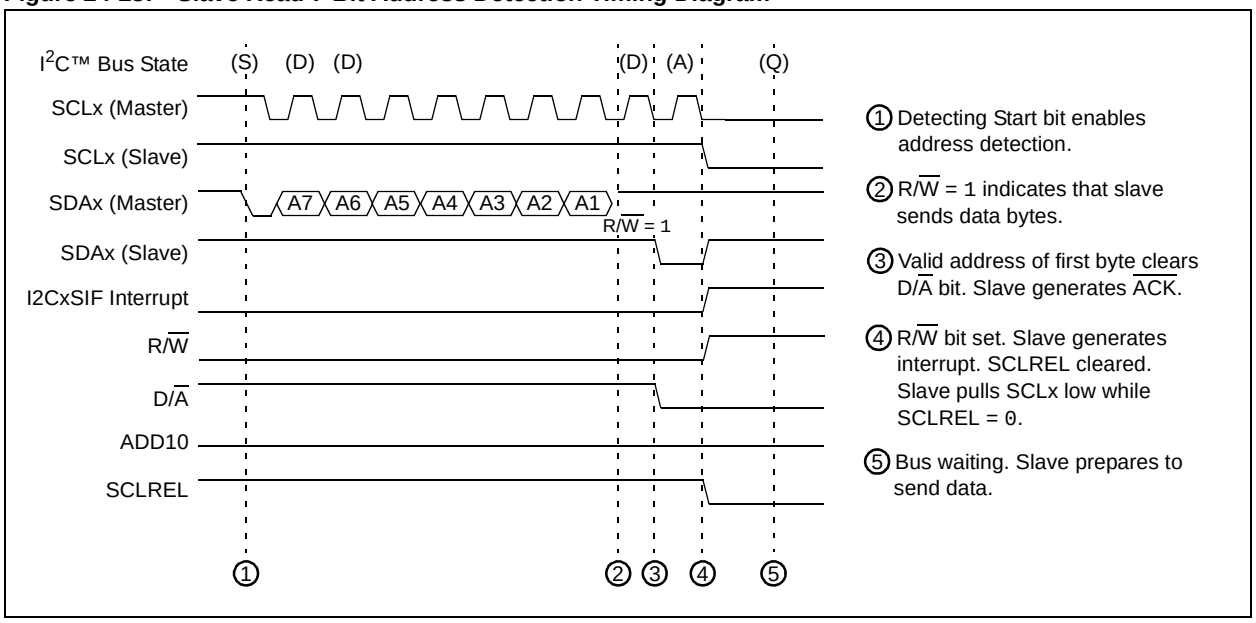
When a slave read is specified by having R/$\overline{\text{W}}$ = 1 in a 7-bit address byte, the process of detecting the device address is similar to that for a slave write (see Figure 24-23). If the addresses match, the following events occur:

1. An $\overline{\text{ACK}}$ is generated.
2. The D/$\overline{\text{A}}$ bit is cleared and the R/$\overline{\text{W}}$ bit is set.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.

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

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

Figure 24-23: Slave Read 7-Bit Address Detection Timing Diagram



24.7.3.5 10-bit Addressing Mode

Figure 24-24 shows the sequence of address bytes on the bus in 10-bit Address mode. In this mode, the slave must receive two device address bytes (see Figure 24-25). The five Most Significant bits (MSbs) of the first address byte specify a 10-bit address. The R/$\overline{W}$ bit of the address must specify a write, causing the slave device to receive the second address byte. For a 10-bit address, the first byte would equal '11110 A9 A8 0', where 'A9' and 'A8' are the two MSbs of the address.

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

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

1. An $\overline{ACK}$ is generated.
2. The D/$\overline{A}$ and R/$\overline{W}$ bits are cleared.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.

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

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

1. An $\overline{ACK}$ is generated.
2. The ADD10 bit is set.
3. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.
4. The module will wait for the master to send data or initiate a Repeated Start condition.

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

Figure 24-24: 10-bit Address Sequence

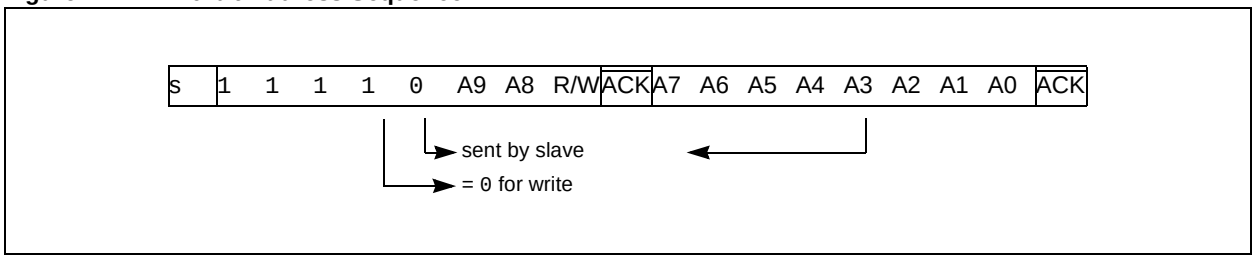
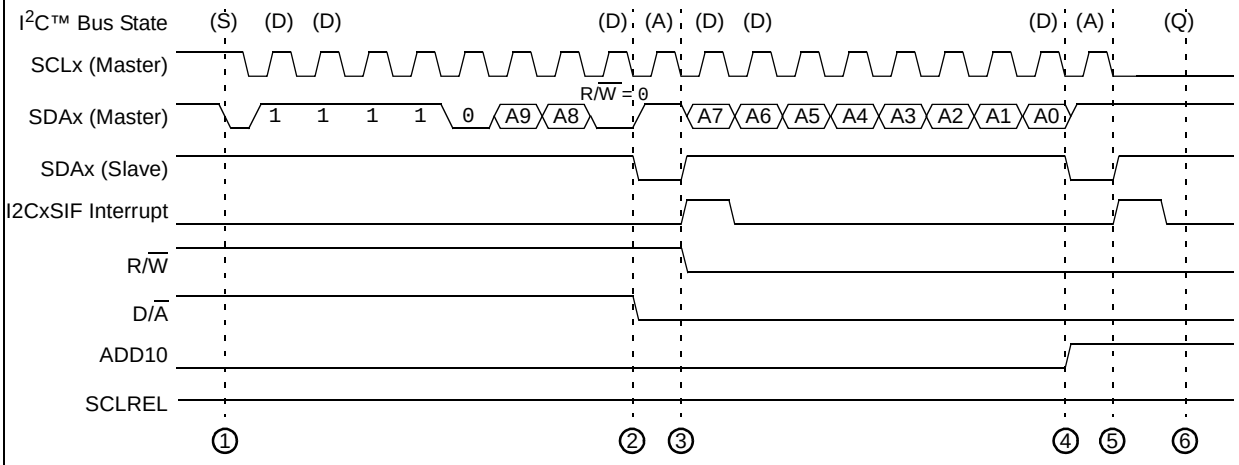


Figure 24-25: 10-Bit Address Detection Timing Diagram



1. Detecting Start bit enables address detection.
2. Address match of first byte clears D/$\overline{A}$ bit and causes slave logic to generate $\overline{ACK}$.
3. Reception of first byte clears R/$\overline{W}$ bit. Slave logic generates interrupt.
4. Address match of first and second byte sets ADD10 and causes slave logic to generate $\overline{ACK}$.
5. Reception of second byte completes 10-bit address. Slave logic generates interrupt.
5. Bus waiting. Slave ready to receive data.

24.7.3.6 General Call Operation

The addressing procedure for the I²C bus is such that the first byte (or first two bytes in case of 10-bit Addressing mode) after a Start condition usually determines which slave device the master is addressing. The exception is the general call address, which can address all devices. When this address is used, all enabled devices should respond with an Acknowledge. The general call address is one of eight addresses reserved for specific purposes by the I²C protocol. It consists of all zeros with R/$\overline{W}$ = 0. The general call is always a slave write operation.

The general call address is recognized when the General Call Enable bit, GCEN (I2CxCON<7>), is set (see Figure 24-26). Following a Start bit detect, eight bits are shifted into the I2CxRSR and the address is compared against the I2CxADD and the general call address.

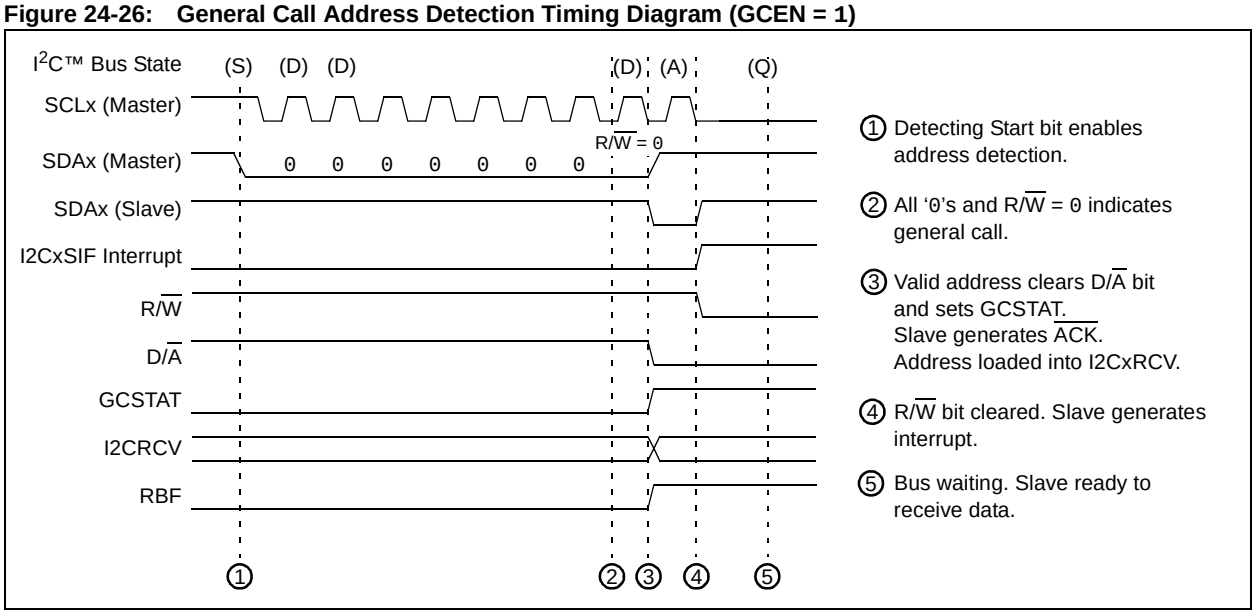
If the general call address matches, the following events occur:

1. An $\overline{ACK}$ is generated.
2. Slave module will set the GCSTAT bit (I2CxSTAT<9>).
3. The D/$\overline{A}$ and R/$\overline{W}$ bits are cleared.
4. The module generates the I2CxSIF interrupt on the falling edge of the ninth SCLx clock.
5. The I2CxRSR is transferred to the I2CxRCV and the RBF flag bit is set (during the eighth bit).
6. The module will wait for the master to send data.

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

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

Figure 24-26: General Call Address Detection Timing Diagram (GCEN = 1)



24.7.3.7 STRICT ADDRESS SUPPORT

When the STRICT (I2CxCON<11>) control bit is set, it enables the module to enforce all reserved addressing and will not acknowledge any addresses if they fall within the reserved address table.

24.7.3.8 When an Address is Invalid

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

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

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

24.7.3.9 Addresses Reserved From Masking

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

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

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

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

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

24.7.4 Receiving Data From a Master Device

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

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

1. The module begins to generate an $\overline{ACK}$ or NACK.
2. The RBF bit is set to indicate received data.
3. The I2CxRSR byte is transferred to the I2CxRCV register for access by the software.
4. The D/$\overline{A}$ bit is set.
5. A slave interrupt is generated. Software may check the status of the I2CxSTAT register to determine the cause of the event and then clear the I2CxSIF flag.
6. The module will wait for the next data byte.

24.7.4.1 Acknowledge Generation

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

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

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

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

Table 24-4: Data Transfer Received Byte Actions

Status Bits as Data Byte Received		Transfer I2CxRSR to I2CxRCV	Generate ACK	Generate I2CxSIF Interrupt (interrupt occurs if enabled)	Set RBF	Set I2COV
RBF	I2COV					
0	0	Yes	Yes	Yes	Yes	No change
1	0	No	No	Yes	No change	Yes
1	1	No	No	Yes	No change	Yes
0	1	Yes	No	Yes	Yes	No change

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

24.7.4.2 Wait States During Slave Receptions

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

The STREN bit (I2CxCON<6>) enables a bus wait to occur on slave receptions. When STREN = 1 at the falling edge of the 9th SCLx clock of a received byte, the slave module clears the SCLREL bit. Clearing the SCLREL bit causes the slave module to pull the SCLx line low, initiating a wait. The SCLx clock of the master and slave will synchronize, as shown in **Section 24.6.2 "Master Clock Synchronization"**.

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

24.7.4.3 Example Messages of Slave Reception

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

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

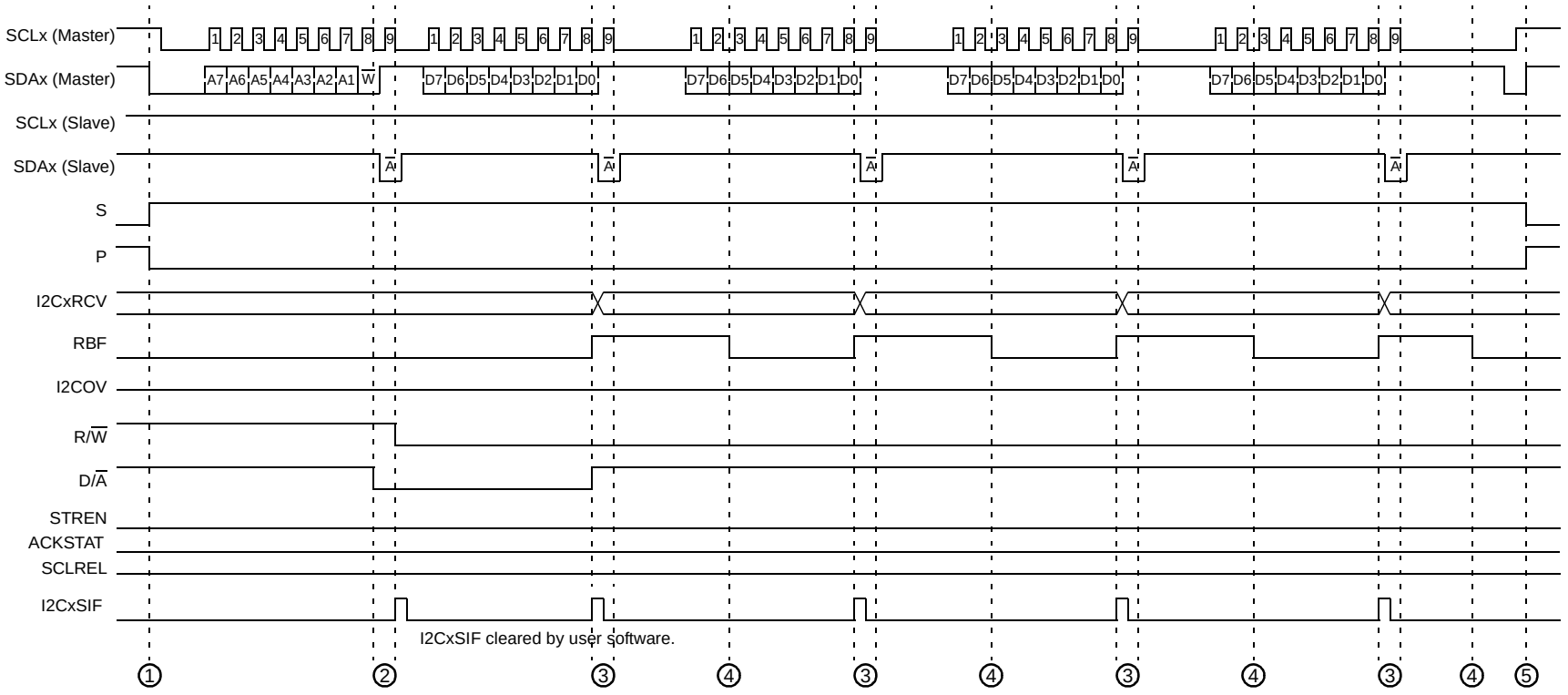
Figure 24-27 shows a simple receive message. Because it is a 7-bit address message, only one interrupt occurs for the address bytes. Then, interrupts occur for each of four data bytes. At an interrupt, the software may monitor the RBF, D/$\overline{A}$ and R/$\overline{W}$ bits to determine the condition of the byte received.

Figure 24-28 shows a similar message using a 10-bit address. In this case, two bytes are required for the address.

Figure 24-29 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 re-send the previous byte. The I2COV bit indicates that the buffer has overrun. The I2CxRCV buffer retains the contents of the first byte. On reception of the third byte, the buffer is still full, and again, the module will NACK the master. After this, the software finally reads the buffer. Reading the buffer will clear the RBF bit, however, the I2COV bit remains set. The software must clear the I2COV bit. The next received byte will be moved to the I2CxRCV buffer and the module will respond with an $\overline{ACK}$.

Figure 24-30 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 SCLx bus line low. As shown with the second received data byte, if the software can read the buffer and clear the RBF before the falling edge of the 9th clock, the clock stretching will not occur. The software can also suspend the bus at any time. By clearing the SCLREL bit, the module will pull the SCLx line low after it detects the bus SCLx low. The SCLx line will remain low, suspending transactions on the bus until the SCLREL bit is set.

Figure 24-27: Slave Message (Write Data to Slave: 7-Bit Address; Address Matches; A10M = 0; GCEN = 0; STRICT = 0)



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

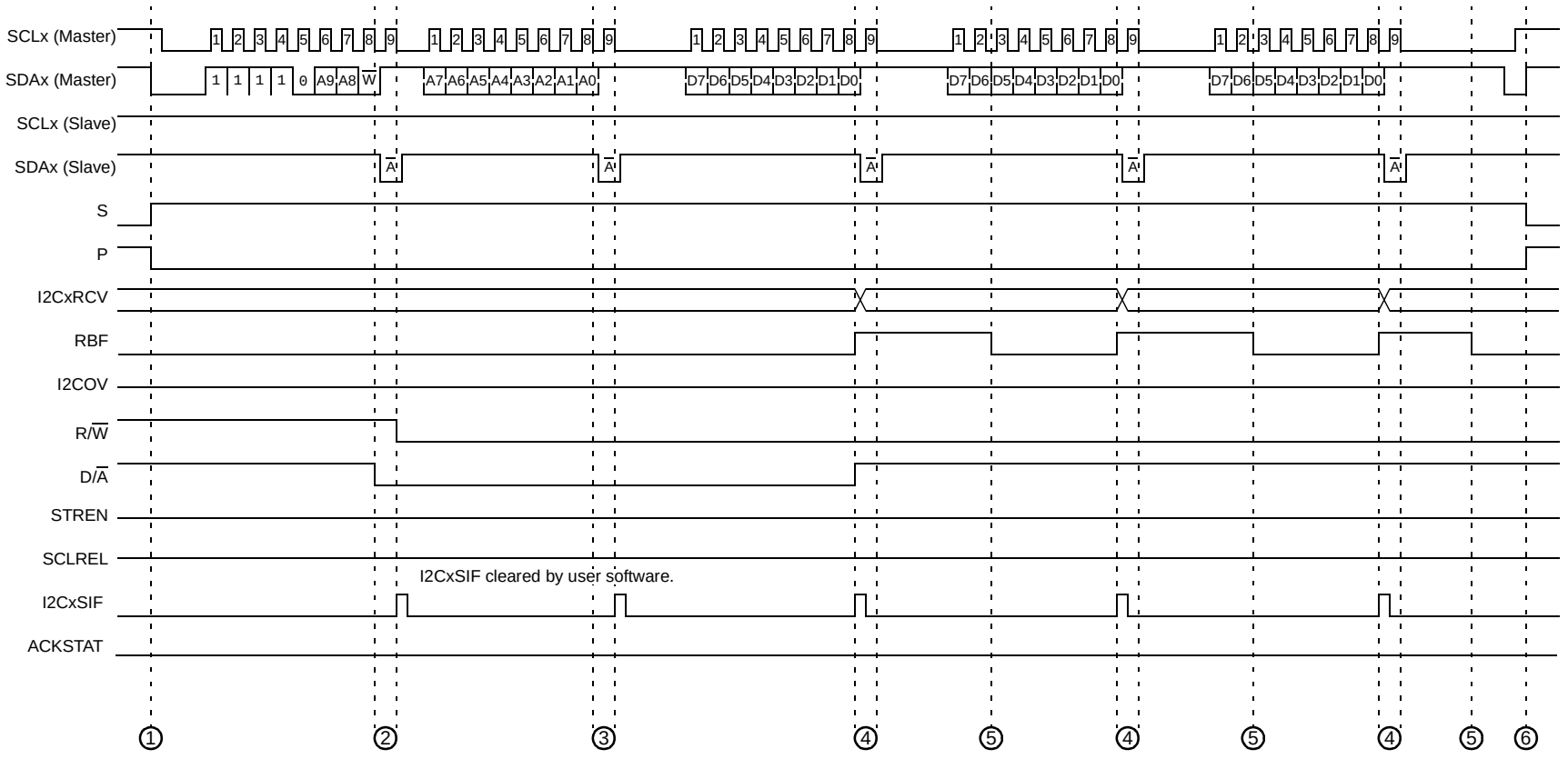
② Slave receives address byte. Address matches. Slave Acknowledges and generates interrupt. Address byte is moved to I2CxRCV register and must be read by user software to prevent buffer overflow.

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

④ Software reads I2CxRCV register. RBF bit clears.

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

Figure 24-28: Slave Message (Write Data to Slave: 10-Bit Address; Address Matches; A10M = 1; GCEN = 0; STRICT = 0)



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

② Slave receives address byte. High-order address matches. Slave Acknowledges and generates interrupt. Address byte not moved to I2CxRCV register.

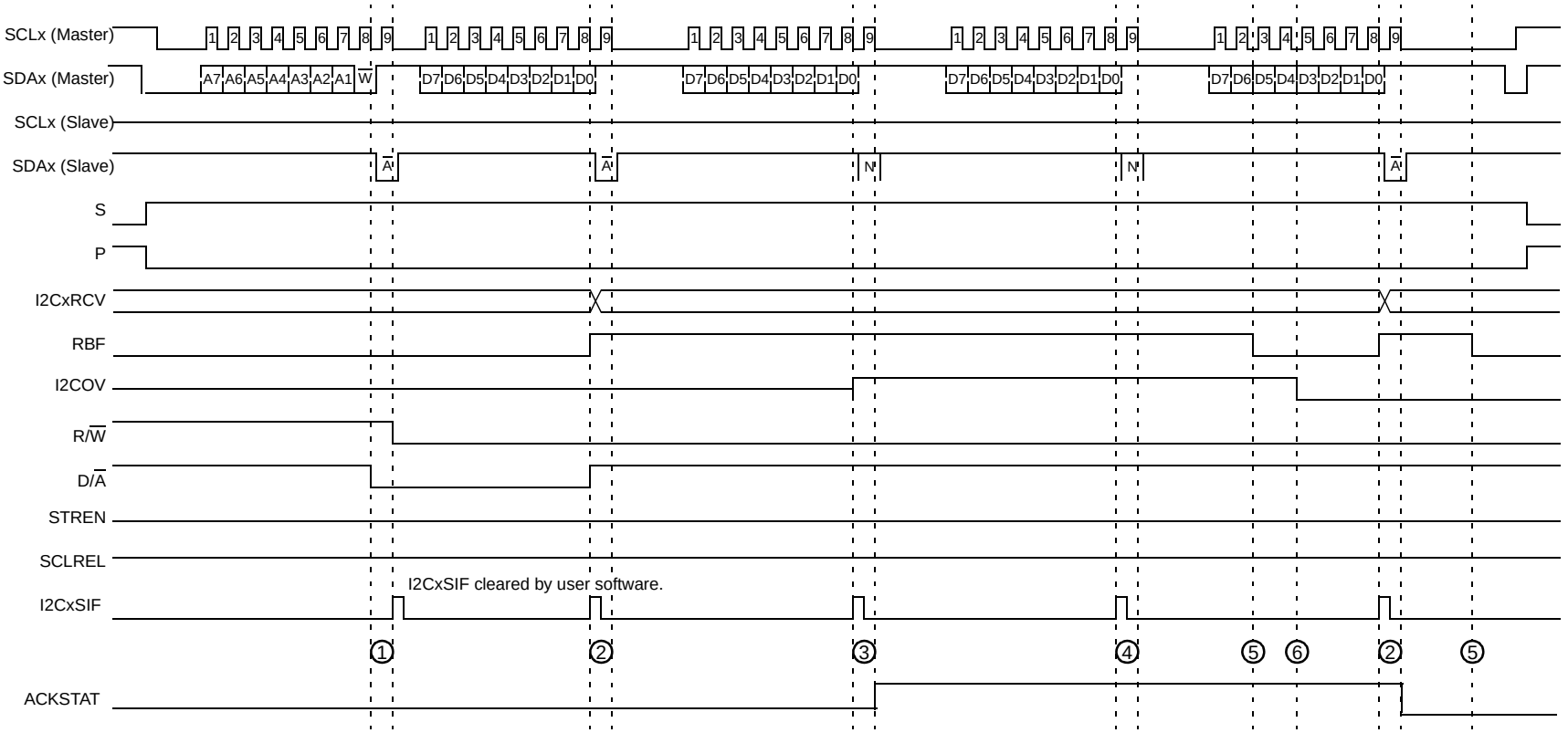
③ Slave receives address byte. Low-order address matches. Slave Acknowledges and generates interrupt. Address byte not moved to I2CxRCV register.

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

⑤ Software reads I2CxRCV register. RBF bit clears.

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

Figure 24-29: Slave Message (Write Data to Slave: 7-Bit Address; Buffer Overrun; A10M = 0; GCEN = 0; STRICT = 0)



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

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

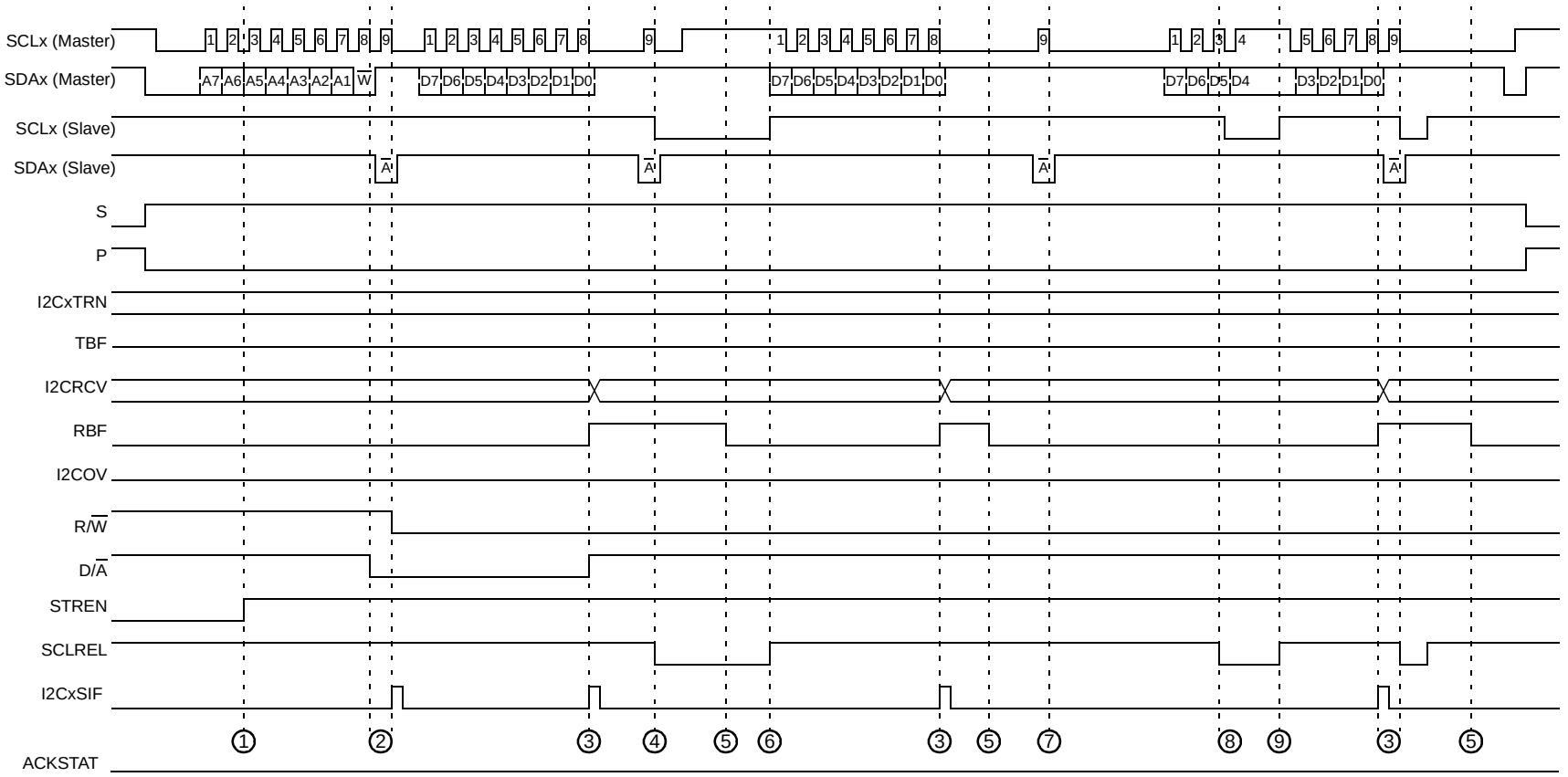
③ Next byte received before I2CxRCV read by software. I2CxRCV register unchanged. I2COV overflow bit set. Slave generates interrupt. Slave sends NACK for reception.

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

⑥ Software reads I2CxRCV register. RBF bit clears.

⑦ Software clears I2COV bit.

Figure 24-30: Slave Message (Write Data to Slave: 7-Bit Address; Clock Stretching Enabled; A10M = 0; GCEN = 0; STRICT = 0)



① Software sets the STREN bit to enable clock stretching.

② Slave receives address byte.

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

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

⑤ Software reads I2CxRCV register. RBF bit clears.

⑥ Software sets SCLREL bit to release clock.

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

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

⑨ Software may set SCLREL to release a clock hold.

24.7.5 Sending Data to a Master Device

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

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

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

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

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

A slave interrupt is generated on the falling edge of the ninth SCLx clock. Software must check the status of the I2CxSTAT register and clear the I2CxSIF flag.

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

24.7.5.1 Wait States During Slave Transmissions

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

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

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

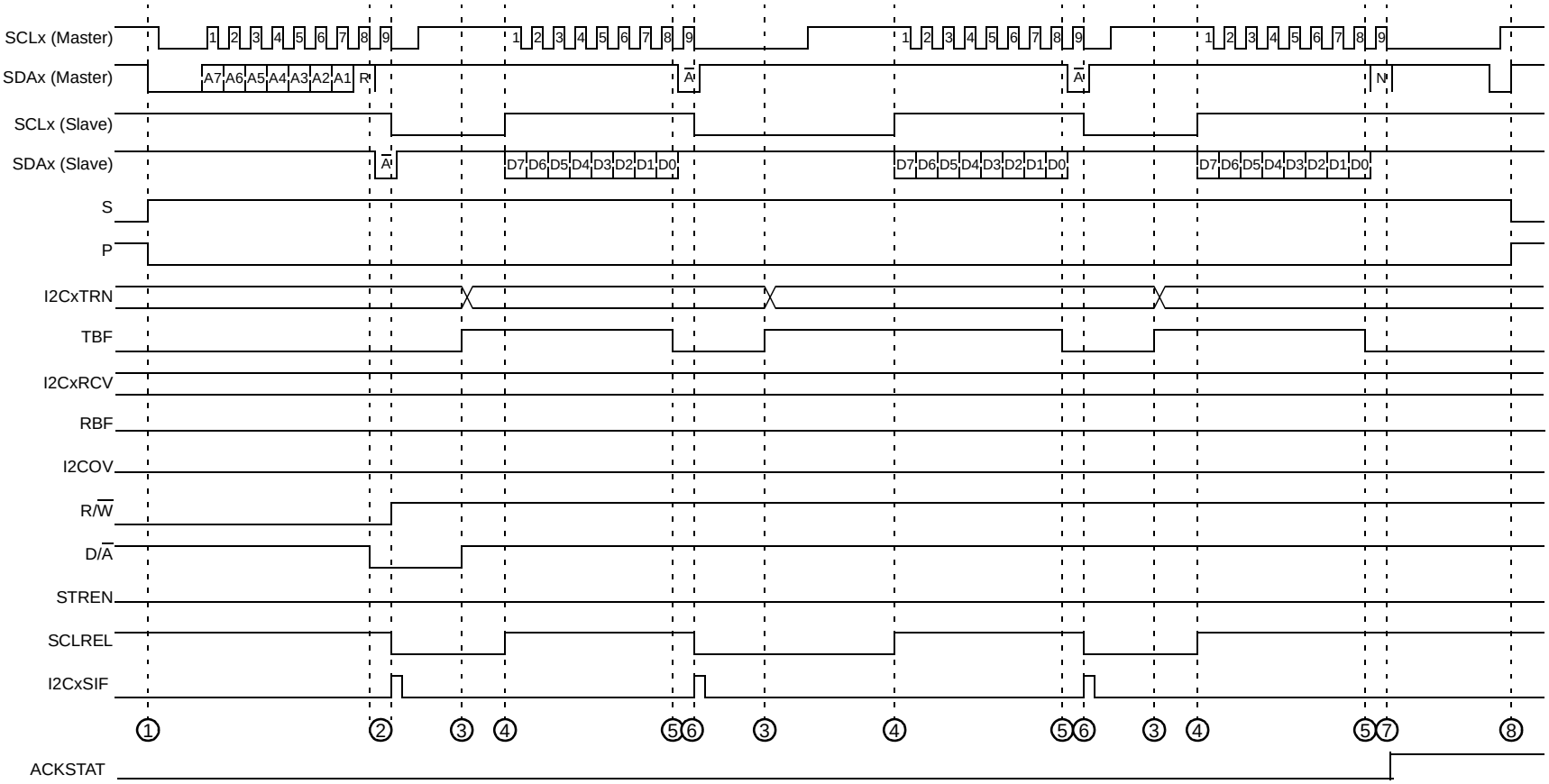
When the software loads the I2CxTRN and is ready to resume transmission, the software sets SCLREL. This causes the slave module to release the SCLx line and the master resumes clocking.

24.7.5.2 Example Messages of Slave Transmission

Slave transmissions for 7-bit address messages are shown in Figure 24-31. When the address matches and the R/$\overline{W}$ bit of the address indicates a slave transmission, the module will automatically initiate clock stretching by clearing the SCLREL bit and generates an interrupt to indicate a response byte is required. The software will write the response byte into the I2CxTRN register. As the transmission completes, the master will respond with an Acknowledge. If the master replies with an $\overline{ACK}$, the master expects more data and the module will again clear the SCLREL bit and generate another interrupt. If the master responds with a NACK, no more data is required and the module will not stretch the clock nor generate an interrupt.

Slave transmissions for 10-bit address messages require the slave to first recognize a 10-bit address. Because the master must send two bytes for the address, the R/$\overline{W}$ bit in the first byte of the address specifies a write. To change the message to a read, the master will send a Repeated Start and repeat the first byte of the address with the R/$\overline{W}$ bit specifying a read. At this point, the slave transmission begins as shown in Figure 24-32.

Figure 24-31: 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 I2CxRCV register. R/W = 1 to indicate read from slave. SCLREL = 0 to suspend master clock.

③ Software writes I2CxTRN with response data. TBF = 1 indicates that buffer is full. Writing I2CxTRN sets D/A, indicating data byte.

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

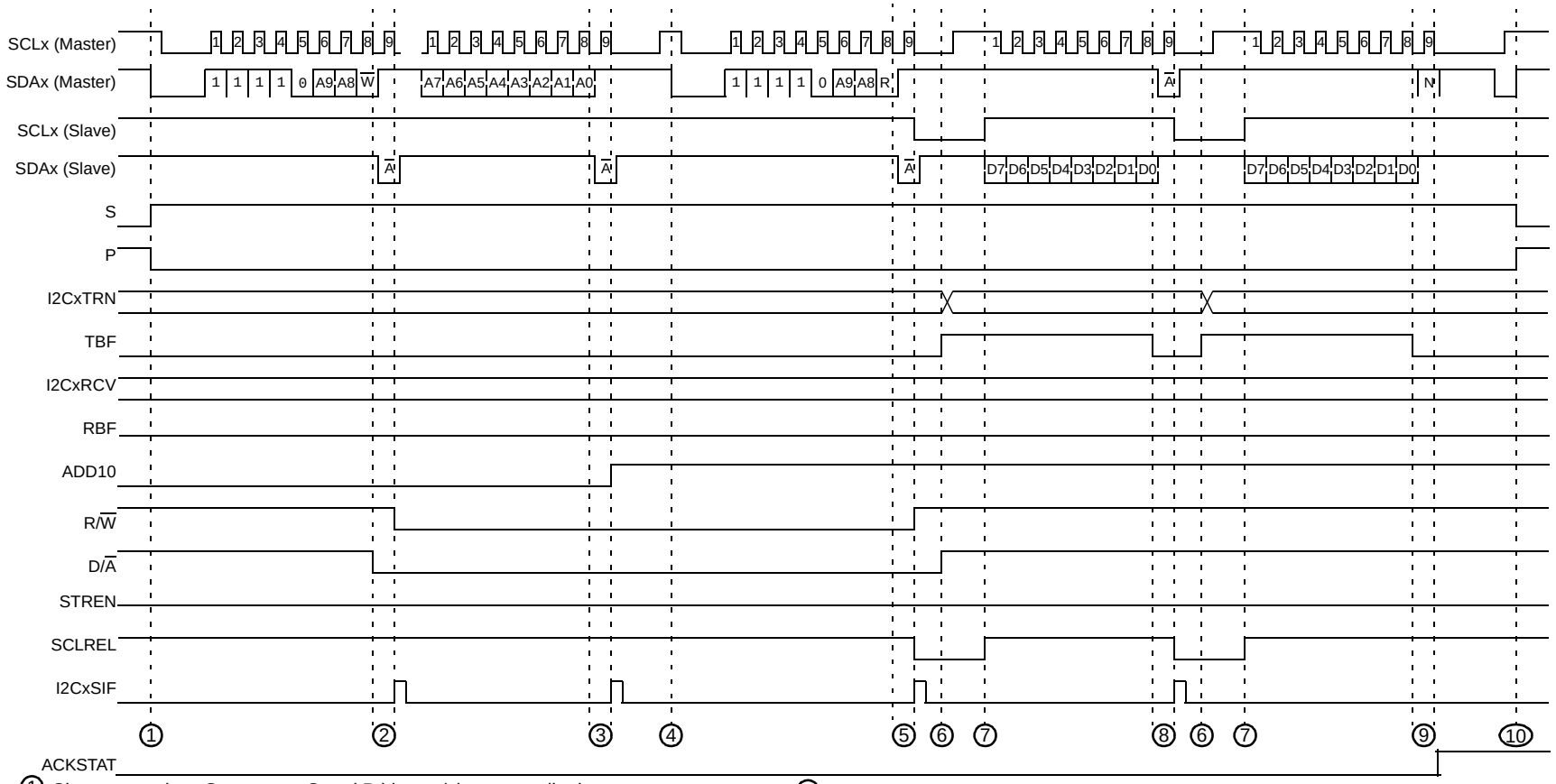
⑤ After last bit, module clears TBF bit, indicating buffer is available for next byte.

⑥ At end of 9th clock, if master sent ACK, module clears SCLREL to suspend clock. Slave generates interrupt.

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

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

Figure 24-32: 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 re-send of first address byte. Read indicated. Slave suspends clock.
6. Software writes I2CxTRN with response data.
7. Software sets SCLREL to release clock hold. Master resumes clocking and slave transmits data byte.
8. At end of 9th clock, if master sent $\overline{\text{ACK}}$, module clears SCLREL to suspend clock. Slave generates interrupt.
9. At end of 9th clock, if master sent NACK, no more data expected. Module does not suspend clock or generate interrupt.
10. Slave recognizes Stop event; S and P bits set/clear accordingly.

24.8 CONNECTION CONSIDERATIONS FOR I²C BUS

Because the I²C bus is a wired AND bus connection, pull-up resistors on the bus are required, shown as RP in Figure 24-33. Series resistors, shown as RS, are optional and used to improve ESD susceptibility. The values of resistors, RP and RS, depend on the following parameters:

- Supply voltage
- Bus capacitance
- Number of connected devices (input current + leakage current)
- Input level selection (I²C or SMBus)

To get accurate SCK clock, the rise time should be as small as possible. The limitation factor is the maximum current sink available on the SCK pad. The following example calculates the Rp min based on 3.3V supply and 6.6 mA sink current at VOLMAX = 0.4V:

Equation 24-2:

$$R_{PMIN} = (V_{DDMAX} - V_{OLMAX})/I_{OL} = (3.3V - 0.4V)/6.6\ mA = 439\Omega$$

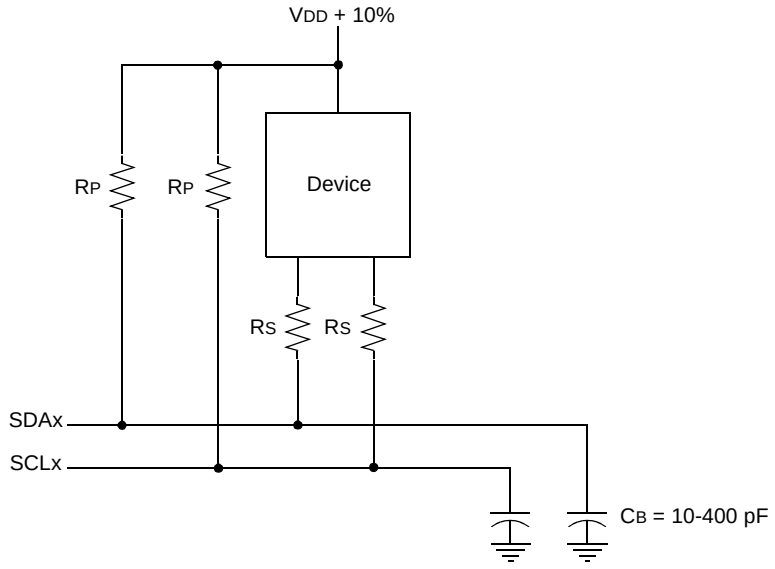
The maximum value for RS is determined by the desired noise margin for the low level. RS cannot drop enough voltage to make the device VOL plus the voltage across RS more than the maximum VIL. Mathematically:

Equation 24-3:

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

The SCLx clock input must have a minimum high and low time for proper operation. The high and low times of the I²C specification, as well as the requirements of the I²C module, are shown in the Electrical Characteristics section in the device data sheet (DS61143).

Figure 24-33: Sample Device Configuration for I²C Bus



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

24.8.1 Integrated Signal Conditioning and Slope Control

The SCLx and SDAx 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 (I2CxCON<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_{ILMAX} and V_{IHMIN}. In a normal I^2C system, V_{ILMAX} is 0.3 V_{DD}; V_{IHMIN} is 0.7 V_{DD}. By contrast, in an SMBus (System Management Bus) system, V_{ILMAX} is set at 0.8V, while V_{IHMIN} is set at 2.1V.

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

24.9 I²C™ OPERATION IN POWER-SAVE MODES AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

PIC32MX based devices have two Power-Saving modes:

- IDLE mode: core and selected peripherals are shut down
- SLEEP mode: entire device is shut down

24.9.1 SLEEP in Master Mode Operation

When the device enters SLEEP mode, all clock sources to the module are shut down. The Baud Rate Generator stops because the clocks stop. It may have to be reset to prevent partial clock detection.

If SLEEP occurs in the middle of a transmission, and the master state machine is partially into a transmission as the clocks stop, the Master mode transmission is aborted.

There is no automatic way to prevent SLEEP entry if a transmission or reception is pending. The user software must synchronize SLEEP entry with I²C operation to avoid aborted transmissions.

Register contents are not affected by going into SLEEP mode or coming out of SLEEP mode.

24.9.2 SLEEP in Slave Mode Operation

The I²C module can still function in Slave mode operation while the device is in SLEEP.

When operating in Slave mode and the device is put into SLEEP, the master-generated clock will run the slave state machine. This feature provides an interrupt to the device upon reception of the address match in order to wake up the device.

Register contents are not affected by going into SLEEP mode or coming out of SLEEP mode.

It is an error condition to set SLEEP in the middle of a slave data transmit operation; indeterminate results may occur.

24.9.3 IDLE Mode

When the device enters IDLE mode, all PBCLK clock sources remain functional. If the module intends to power down, it disables its own clocks.

For the I²C, the I2CxSIDL bit (I2CxCON<13>) selects whether the module will stop on IDLE or continue on IDLE. If I2CxSIDL = 0, the module will continue operation in IDLE mode. If I2CxSIDL = 1, the module will stop on IDLE.

The I²C module will perform the same procedures for stop on IDLE mode as for SLEEP mode. The module state machines must be reset.

24.9.4 Operation in DEBUG Mode

The FRZ bit (I2CxCON<14>) determines whether the I²C module will run or stop while the CPU is executing Debug Exception code (i.e., the application is halted) in DEBUG mode. When FRZ = 0, the I²C module continues to run even when the application is halted in DEBUG mode. When FRZ = 1 and the application is halted in DEBUG mode, the module will freeze its operations and make no changes to the state of the I²C module. The module will resume its operation after the CPU resumes execution.

Note:	The FRZ bit is readable and writable only when the CPU is executing in Debug Exception mode. In all other modes, the FRZ bit reads as '0'. If FRZ bit is changed during DEBUG mode, the new value does not take effect until the current Debug Exception mode is exited and re-entered. During the Debug Exception mode, the FRZ bit reads the state of the peripheral when entering DEBUG mode.

24.10 EFFECTS OF A RESET

A Reset (POR, WDT, etc.) disables the I^2C module and terminates any active or pending message activity. See the register definitions of I2CxCON and I2CxSTAT for the Reset conditions of those registers.

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

24.11 PIN CONFIGURATION IN I^2C MODE

In I^2C mode, pin SCL is clock and pin SDA is data. The module will override the data direction bits (TRIS bits) for these pins. The pins that are used for I^2C modes are configured as open drain. Table 24-5 lists the pin usage in different modes.

Table 24-5: Required IO Pin Resources

IO Pin Name	Master Mode	Slave Mode
SDAx	Yes	Yes
SCLx	Yes	Yes

Note: "No" indicates the pin is not required and can be used as a general purpose IO pin.

24.12 DESIGN TIPS

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

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

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

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

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

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

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

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

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

Answer: In a multi-master system, other masters may cause bus collisions. In the 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 re-sent from the beginning.

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

Answer: Look at the S bit. If S = 0, the bus is Idle.

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

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

24.13 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 PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the I^2C module include the following:

Title	Application Note #
Use of the SSP Module in the I ^{2}C™ Multi-Master Environment	AN578
Using the PIC® Microcontroller SSP for Slave I^2C™ Communication	AN734
Using the PIC® Microcontroller 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 PIC32MX family of devices.

24.14 REVISION HISTORY

Revision A (October 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (July 2008)

Revised Figure 24-1; Section 24.2 (I2CxMIF); Register 24-1, bits 13 and 14; Revised Register 24-26-24-29; Revised Table 24-1, I2CxCON; Change Reserved bits from “Maintain as” to “Write”; Added Note to ON bit (I2CxCON Register); Delete Section 24.12 (Electrical Characteristics).

NOTES:

