TB3252

Using the MSSP in I2C Master Mode

Introduction

The Master Synchronous Serial Port (MSSP) is an integrated serial communications module. The MSSP contains two sub-modules:

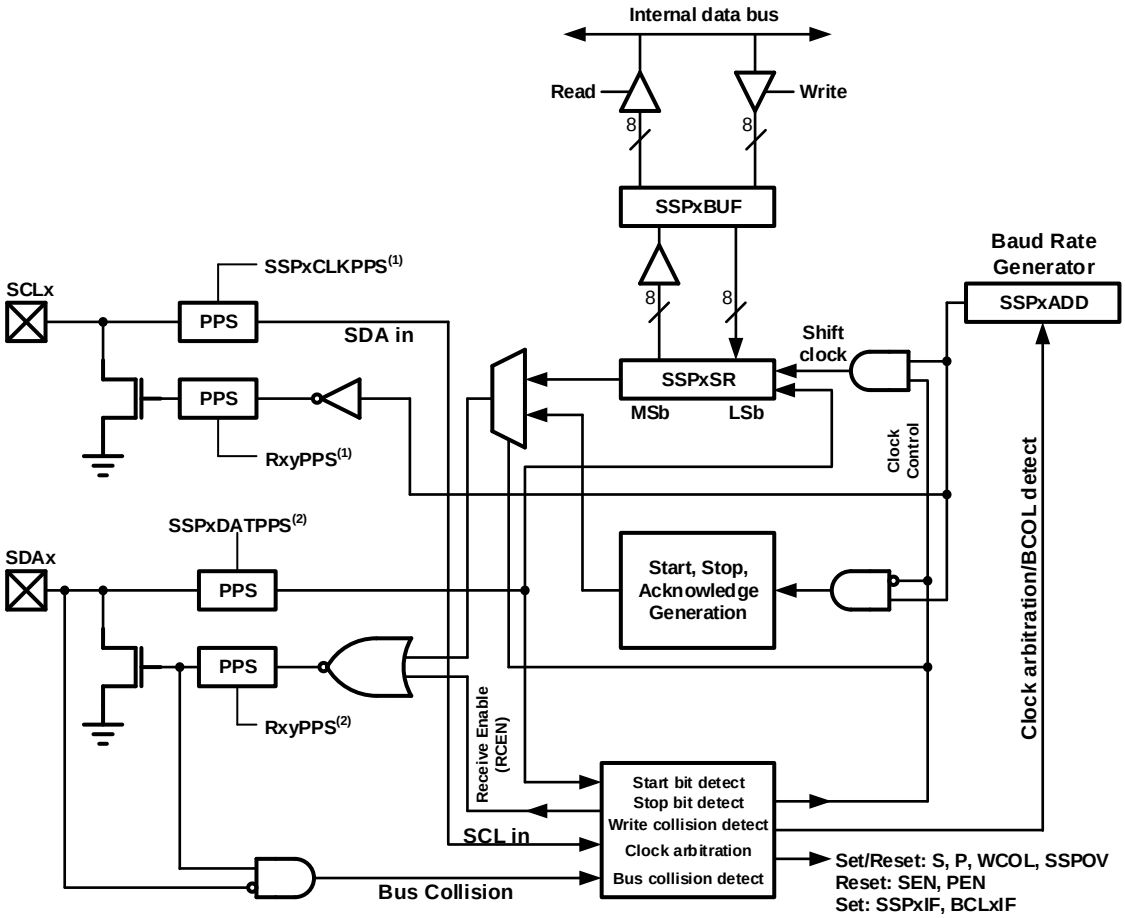
- SPI - Serial Peripheral Interface
- I^2C - Inter-Integrated Circuit

The Inter-Integrated Circuit, commonly referred to as I^2C, is a synchronous, two-wire and bidirectional serial communications bus. The I^2C module can be used to communicate with other I^2C-compatible EEPROMs, display drivers, sensors or other micrcontroller devices. The I^2C module offers the following features:

- Master Mode
- Slave Mode
- Multi-Master Support
- Selectable $\overline{ACK}$/NACK Response
- 7-Bit and 10-Bit Addressing Modes with Address Masking
- Interrupts with Interrupt Masking
- Slave Clock Stretching
- Bus Collision Detection
- Write Collision Detection
- General Call Addressing
- Selectable SDA Hold Time

This application note discusses the MSSP operating in the I^2C Master mode. Figure 1 shows a block diagram of the MSSP module in I^2C Master mode.

Figure 1. MSSP Block Diagram (I^2C Master Mode)



Note 1: SCL pin selections must be the same for input and output.
2: SDA pin selections must be the same for input and output.

Table of Contents

1. I²C Specification

The I²C Specification was developed by Philips Semiconductors to communicate between devices connected to a two-wire bus. Philips recognized that there were many similarities between consumer electronics, industrial electronics and telecommunications designs. These designs often contained similar components, such as Analog-to-Digital Converters (ADCs), Liquid Crystal Displays (LCDs) or external memory modules. Philips determined that they could simplify the system design and maximize hardware efficiency by creating a communications scheme that could be used to transfer data between any device connected to a common bus. The I²C Specification allowed system designers to use devices from multiple manufacturers, or use one device in several applications. The Specification also solved interfacing issues by creating a standard protocol that is now held as an industry standard, meaning any I²C device could communicate with any other I²C device without having to change the hardware or firmware of either device.

The I²C Specification defines the bus as a two-wire, bidirectional communications network. One line carries the Serial Data (SDA) signal, and the other line carries the Serial Clock (SCL) signal. Each I²C device connected to a bus has a unique address, either 7 bits or 10 bits in length. An I²C device may operate as a bus master, a bus slave, or both, depending on the device and the application in which the device is used.

1.1 I²C Terminology

To properly understand the language used in the Specification, Table 1-1 shows a list of commonly used terms found throughout the Specification.

Table 1-1. I²C Bus Terminology

Term	Description
Transmitter	The device that shifts data out onto the bus.
Receiver	The device that shifts data in from the bus.
Master	The device that generates the clock signal (SCL), initiates data transfer and terminates transmission.
Slave	The device addressed by the master.
Multi-Master	A bus that contains at least two master devices, both of which can initiate communications.
Arbitration	A procedure that ensures that only one master device at a time can control the bus.
Synchronization	A procedure that synchronizes the clock signals of two or more devices on the bus.
Idle	Both SDA and SCL signals are at a logic high state and there is no activity on the bus.
Active	The state of the bus during which communication takes place.
Write Request	A master device transmits a slave address with the R/$\overline{W}$ bit clear with the intention of transmitting data to a slave.
Read Request	A master device transmits a slave address with the R/$\overline{W}$ bit set with the intention of receiving data from a slave.
Clock Stretching	Occurs when a device pulls the SCL line low to effectively pause communications.
Bus Collision	The condition in which the *expected* SDA logic level is high, but is actually sampled as a logic low.
Bus Timeout	The condition in which a device is holding the bus for longer than a specified period.

1.2 Data Transfer Rates

The I²C Specification defines data transfer rates as follows:

- Standard-mode - transfer rates up to 100 kbits/s
- Fast-mode - transfer rates up to 400 kbits/s
- Fast-mode Plus - transfer rates up to 1 Mbit/s
- High-Speed mode - transfer rates up to 3.4 Mbits/s

The MSSP module supports both Standard-mode and Fast-mode transfer rates.

1.3 External Pull-up Resistor Selection

The I²C Specification proposes two methods to determine the correct pull-up resistor size.

The first method calculates the maximum pull-up resistor size as a function of bus capacitance and rise time (see Equation 1-1). Bus capacitance is the total capacitance of the bus wires/traces, bus connection points and bus pins, all of which must be considered when calculating the total bus capacitance. Rise time is the period in which the signal transitions from $V_{IL(MAX)}$ (0.3*V_{DD}) to $V_{IH(MIN)}$(0.7*V_{DD}). Rise time values are typically located in the device's data sheet.

Bus capacitance should be measured to achieve the most accurate pull-up values, but an estimated value, or the maximum allowable capacitance as defined by the I²C specification, may also be used. The maximum allowable bus capacitance is specified to limit rise time decreases and allow operation at the rated frequency. The bus may operate at higher than allowable bus capacitance levels, but at a lower frequency.

Equation 1-1. Maximum Pull-up Resistor Size

$$R_{p\ (max)} = \frac{t_{rise}}{0.8473 * C_{bus}}$$

$R_{p\ (max)} = Maximum\ \ pull-up\ \ value$

$t_{rise} = Maximum\ \ rise\ time$

$C_{bus} = \ Total\ \ bus\ \ capacitance$

The second method calculates the minimum pull-up resistor size as a function of V_{DD} (see Equation 1-2). The supply voltage limits the minimum resistor value due to the specified minimum sink current of 3 mA for Standard-mode (100 kHz) or Fast-mode (400 kHz).

Equation 1-2. Minimum Pull-up Resistor Size

$$R_{p\ (min)} = \frac{V_{DD} - V_{OL(max)}}{I_{OL}}$$

$R_{p\ (min)} = Minimum\ \ pull-up\ \ value$

$V_{DD} = Supply\ \ \ voltage$

$V_{OL\ (max)} = Maximum\ \ output\ low\ voltage$

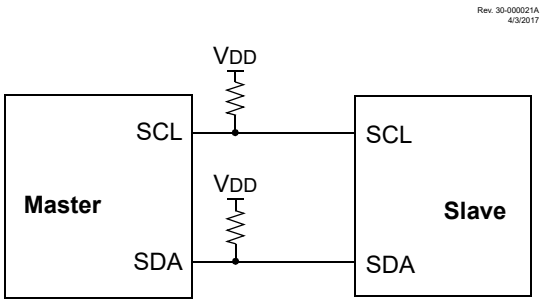
$I_{OL} = Minimum\ \ sink\ \ current$

2. I²C Mode Overview

The MSSP's I²C module provides a synchronous serial interface between the microcontroller and other I²C-compatible devices using a two-wire bus network. The two signals connections, Serial Clock (SCL) and Serial Data (SDA), are bidirectional open-drain lines, each requiring pull-up resistors to the supply voltage. Pulling the signal line to ground is considered a logic '0', while allowing the signal line to float is considered a logic '1'.

Figure 2-1 shows a typical connection between a master and a slave.

Figure 2-1. I²C Master/Slave Connection



Important: Due to the variety of device technologies (e.g., CMOS, NMOS), the logic voltage levels (logic low (0), logic high (1)) are not fixed, but rather are proportional to the bus supply voltage.

According to the I²C Specification, a logic input low level (V_{IL}) is up to 30% of V_{DD} ($V_{IL} \leq 0.3V_{DD}$), while a logic input high level (V_{IH}) is between 70% and 100% of V_{DD} ($V_{IH} \geq 0.7V_{DD}$). Some legacy devices may use the previously defined fixed levels of V_{IL} = 1.5V and V_{IH} = 3.0V. However, all new I²C-compatible devices must adhere to the 30/70% specification.

All I²C communication is performed using an 8-bit data word and a 1-bit Acknowledge condition. All transactions are initiated and terminated by the master device. Address and data are transmitted starting with the Most Significant bit (MSb). Depending on the direction of the data being transferred, there are four main operations performed in I²C mode:

- Master Transmit - master is sending data to a slave
- Master Receive - master is receiving data from a slave
- Slave Transmit - slave is sending data to a master
- Slave Receive - slave is receiving data from a master

The I²C Specification also defines three message protocols:

- Single message where a master writes data to a slave
- Single message where a master reads data from a slave
- Combined message where a master initiates a minimum of two writes, two reads, or a combination of reads and writes, to one or more slaves.

Communication begins when a master device transmits a Start condition, followed by the address of the slave it intends to communicate with. Bit 0 in the 7-bit address byte, or Bit 0 of the high address byte in 10-bit Addressing mode, is reserved as the Read/$\overline{\text{Write}}$ Information (R/$\overline{\text{W}}$) bit. The R/$\overline{\text{W}}$ bit determines whether the master intends to write data to a slave (R/$\overline{\text{W}}$ = 0) or receive data from the slave (R/$\overline{\text{W}}$ = 1).

If the requested slave device exists on the bus, it will respond with an Acknowledge sequence ($\overline{\text{ACK}}$). The $\overline{\text{ACK}}$ sequence takes place during the 9th clock pulse and indicates to the transmitting device that the receiving device is active and ready for communication.

In Master Transmit mode, the master will continue to send data to the slave and the receiver will continue to respond with an $\overline{\text{ACK}}$, as long as the data is considered valid. In Master Receive mode, the master will continue to receive

data from the slave and will respond to the slave with an $\overline{ACK}$. When the master transmits or receives the last byte of data, it can end communication by issuing a Stop condition, or it can issue a Restart condition if it intends to continue to communicate with the bus.

The I^2C Specification allows for a multi-master bus, meaning that there can be several master devices connected to a single bus. A master can select a slave device by transmitting a unique address on the bus. When the address matches a slave's address, the slave responds with an $\overline{ACK}$ and communication between the master and slave can commence. All other devices on the bus must ignore any transactions not intended for them.

2.1 I^2C Operation

All MSSP I^2C communication is byte-oriented and shifted out MSb first. Eight Special Function Registers (SFRs) and two interrupt flags interface the module with the microcontroller and user software. Two pins, Serial Data (SDA) and Serial Clock (SCL), are used by the module to communicate with other external I^2C devices.

2.1.1 Byte Format

All I^2C communication is done in 9-bit segments. A byte is sent from a master to a slave or vice versa, followed by an Acknowledge sequence sent back. After the eighth falling edge of SCL, the device transmitting data on SDA changes that pin from an output to an input and reads the $\overline{ACK}$ value on the ninth clock pulse.

The clock signal is provided by the master. Data is valid to change while SCL is low and sampled on the rising edge of SCL. Changes on the SDA line while SCL is high define special bus conditions, such as a Start or Stop condition.

2.1.2 SDA and SCL Pins

Selection of any I^2C mode with the SSPEN bit set (SSPEN = `1`) forces the SCL and SDA pins to be open-drain. These pins must be configured as inputs by setting the appropriate TRIS bits.

Important: Any device pin can be selected for SDA and SCL functions with the PPS peripheral. These functions are bidirectional. The SDA input is selected with the SSPxDATPPS registers. The SCL input is selected with the SSPxCLKPPS registers. Outputs are selected with the RxyPPS registers. It is the user's responsibility to make the selections so that both the input and the output for each function is on the same pin.

2.1.2.1 SDA Hold Time

The hold time of the SDA pin is selected by the SDA Hold Time Selection (SDAHT) bit. Hold time is the time SDA is held valid after the falling edge of SCL. Setting the SDAHT bit selects a longer 300 ns minimum hold time and may help buses with large capacitance.

2.1.3 Clock Stretching

Clock stretching occurs when a device on the bus holds the SCL line low, effectively pausing communication. The slave may stretch the clock to allow more time to handle data or prepare a response for the master device. A master device is not concerned with stretching, as anytime it is active on the bus and not transferring data, it is stretching. Any stretching done by a slave is invisible to the master software and handled by the hardware that generates SCL.

The CKP bit is used to control clock stretching in Slave mode software. Any time the CKP bit is cleared, the module will wait for the SCL line to go low and then hold it. Setting CKP will release SCL and allow more communication. The CKP bit is unused in Master mode.

2.1.4 Arbitration

Each master device must monitor the bus for Start and Stop conditions. If the device detects that the bus is busy, it cannot begin a new message until the bus returns to an Idle state.

However, two master devices may try to initiate a transmission on or about the same time. When this occurs, the process of arbitration begins. Each transmitter checks the level of the SDA data line and compares it to the level that it expects to find. The first transmitter to observe that the two levels do not match, loses arbitration and must stop transmitting on the SDA line.

For example, if one transmitter holds the SDA line to a logical one (lets SDA float) and a second transmitter holds it to a logical zero (pulls SDA low), the result is that the SDA line will be low. The first transmitter then observes that the level of the line is different than expected and concludes that another transmitter is communicating.

The first transmitter to notice this difference is the one that loses arbitration and must stop driving the SDA line. If this transmitter is also a master device, it also must stop driving the SCL line. It then can monitor the lines for a Stop condition before trying to reissue its transmission. In the meantime, the other device that has not noticed any difference between the expected and actual levels on the SDA line continues with its original transmission. It can do so without any complications, because so far, the transmission appears exactly as expected with no other transmitter disturbing the message.

Slave Transmit mode can also be arbitrated, when a master addresses multiple slaves, but this is less common.

2.1.5 Start Condition

The I^{2}C Specification defines a Start condition as a transition of SDA, from a high-to-low state, while the SCL line is high. A Start condition is always generated by the master and signifies the transition of the bus from an Idle to an Active state. Figure 2-2 shows wave forms for the Start and Stop conditions.

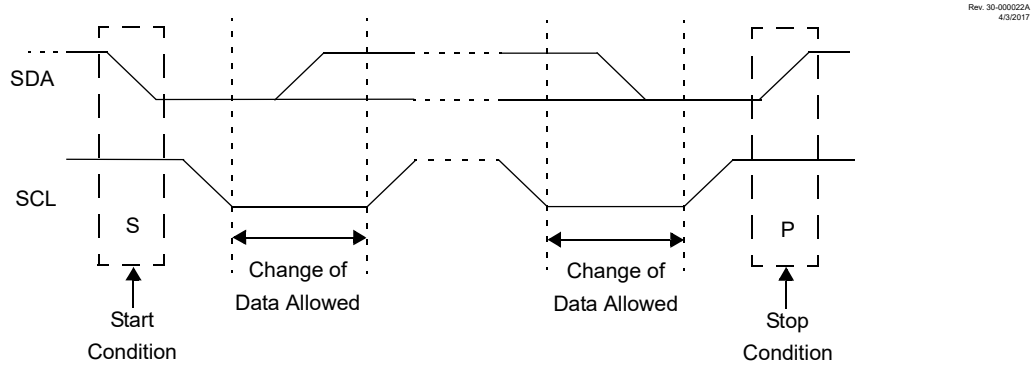
A bus collision can occur on a Start condition if the module samples the SDA line low, before asserting it low. This does not conform to the I^{2}C Specification that states no bus collision can occur on a Start.

2.1.6 Stop Condition

A Stop condition is a transition of the SDA line from a low-to-high state, while the SCL line is high.

Important: At least one SCL low time must appear before a Stop is valid. Therefore, if the SDA line goes low then high again while the SCL line stays high, only the Start condition is detected.

Figure 2-2. I^{2}C Start and Stop Conditions

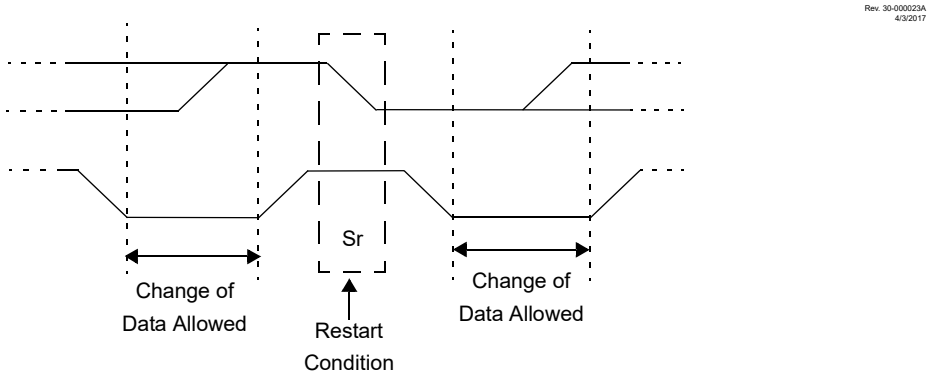


2.1.7 Restart Condition

A Restart condition is valid any time that a Stop would be valid. A master can issue a Restart if it wishes to hold the bus after terminating the current transfer. A Restart has the same effect on the slave that a Start would, resetting all slave logic and preparing it to clock in an address. The master may want to address the same or another slave. Figure 2-3 shows the waveform for a Restart condition.

In 10-bit Addressing Slave mode, a Restart is required for the master to clock data out of the addressed slave. Once a slave has been fully addressed, matching both high and low address bytes, the master can issue a Restart and the high address byte with the R/$\overline{W}$ bit set. The slave logic will then hold the clock and prepare to clock out data.

Figure 2-3. I²C Restart Condition



2.1.8 Acknowledge Sequence

The ninth SCL pulse for any transferred byte in I²C is dedicated as an Acknowledge sequence ($\overline{ACK}$). It allows receiving devices to respond back to the transmitter by pulling the SDA line low. The transmitter must release control of the line during this time to shift in the response. The Acknowledge ($\overline{ACK}$) is an active-low signal. Pulling the SDA line low indicates to the transmitter that the device has received the transmitted data and is ready to receive more.

The result of an $\overline{ACK}$ is placed in the Acknowledge Status (ACKSTAT) bit.

The master software allows the user to select the $\overline{ACK}$ value sent back to the transmitter. The Acknowledge Data (ACKDT) bit is set/cleared to determine the response.

2.2 I²C Master Mode Operation

Master mode is enabled by configuring the MSSP Mode Select (SSPM) bits and setting the MSSP Enable (SSPEN) bit. The SDA and SCL pins must be configured as inputs by setting the associated TRIS bits (TRISxy = 1). MSSP hardware will automatically override the TRIS controls when necessary to drive the pins low.

Master mode operation is supported by interrupt generation on the detection of certain events. The following events will cause the MSSP Interrupt Flag (SSPxIF) bit to be set:

- Start condition detected
- Stop condition detected
- Data byte transmitted/received
- Acknowledge sequence transmitted/received
- Restart condition detected

The Start (S) and Stop (P) bits are cleared by a Reset event or when the MSSP module is disabled. Control of the bus may be taken when the Stop bit is set (P = 1) or when the bus is Idle.

Important:

1. The MSSP module, when configured in I²C Master mode, does not allow queuing of events. For example, user software is not permitted to initiate a Start condition and immediately write the SSPxBUF register to initiate transmission before the Start condition is complete. In this case, the SSPxBUF will not be written to and the Write Collision Detect (WCOL) bit will be set, indicating that the write did not occur.
2. Master mode suspends Start/Stop condition detection when transmitting the Start/Stop conditions by means of the SEN/PEN control bits. The SSPxIF bit is set at the end of the Start/Stop condition, when hardware clears the associated SEN/PEN control bit.

The master device generates all of the serial clock pulses and the Start and Stop conditions. A transfer typically begins with a Start condition and ends with a Stop condition. A Restart condition may be used in place of a Stop

condition, if the master device wishes to hold the bus to continue communication with the same or other slave devices.

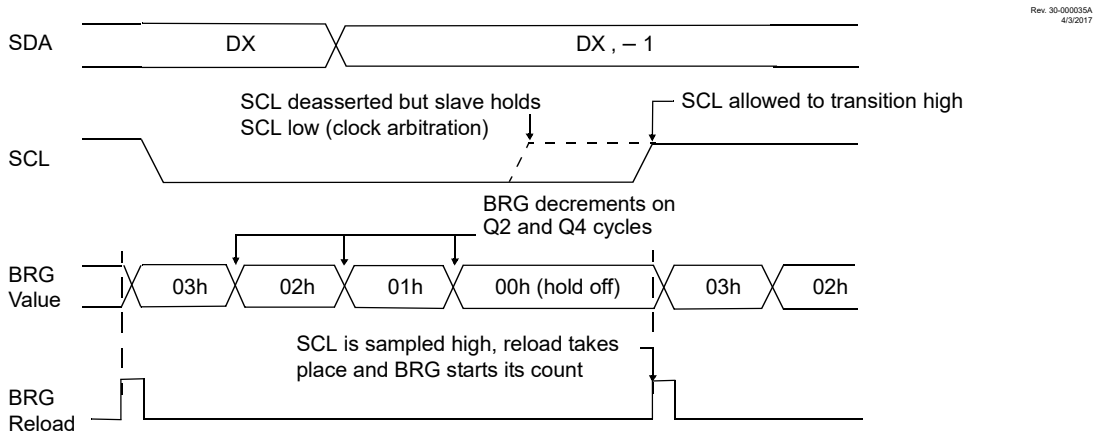
In Master Transmit mode, data is transmitted over the SDA line, while the clock is transmitted over the SCL line. The first byte transmitted by the master contains the 7-bit address, or upper byte of a 10-bit address, and the R/$\overline{W}$ bit. In Transmit mode, the R/$\overline{W}$ bit is clear (R/$\overline{W}$ = 0). Data is transmitted eight bits at a time, and after each byte has been sent, the master releases the clock line and waits for the reception of an Acknowledge sequence from the slave. Once the master has received all of the data it requests, it can issue a Stop condition to terminate transmission, or issue a Restart condition if it wishes to hold control of the bus.

In Master Receive mode, the first byte transmitted is the 7-bit address, or upper byte of the 10-bit address, and the R/$\overline{W}$ bit. In 7-bit Receive mode, the R/$\overline{W}$ bit is set (R/$\overline{W}$ = 1). In 10-bit Receive mode, the R/$\overline{W}$ bit contained in the first transmitted address byte is clear (R/$\overline{W}$ = 0). After the successful transmission of the upper and lower address bytes, the master issues a Restart condition and transmits the upper address byte again, this time with the R/$\overline{W}$ bit set (R/$\overline{W}$ = 1). In Receive mode, the master receives data over the SDA line, but continues to generate the clock signal over the SCL line. After the master receives each byte, it typically transmits an $\overline{ACK}$ sequence back to the slave. Once the master has received the final byte, it can respond with either an $\overline{ACK}$ or NACK sequence and issue either a Stop or Restart condition.

2.2.1 Clock Arbitration

Clock arbitration occurs when the master, during any Receive, Transmit or Repeated Start/Stop condition, releases the SCL pin (SCL allowed to float high). When the SCL pin is allowed to float high, the Baud Rate Generator (BRG) is suspended from counting until the SCL pin is actually sampled high. When the SCL pin is sampled high, the Baud Rate Generator is reloaded with the contents of SSPxADD and begins counting. This ensures that the SCL high time will always be at least one BRG rollover count in the event that the clock is held low by an external device as shown in Figure 2-4.

Figure 2-4. Baud Rate Generator Timing with Clock Arbitration



2.2.2 WCOL Status Flag

If software writes to the SSPxBUF register, when a Start, Restart, Stop, Receive or Transmit sequence is in progress, the Write Collision Detect (WCOL) bit is set and the contents of the buffer are unchanged (the write does not occur). Any time the WCOL bit is set, it indicates that an action on SSPxBUF was attempted while the module was not Idle.

Important: Since queuing of events is not allowed, writing to the lower five bits of SSPxCON2 is disabled until the Start condition is complete.

2.2.3 I²C Master Mode Start Condition Timing

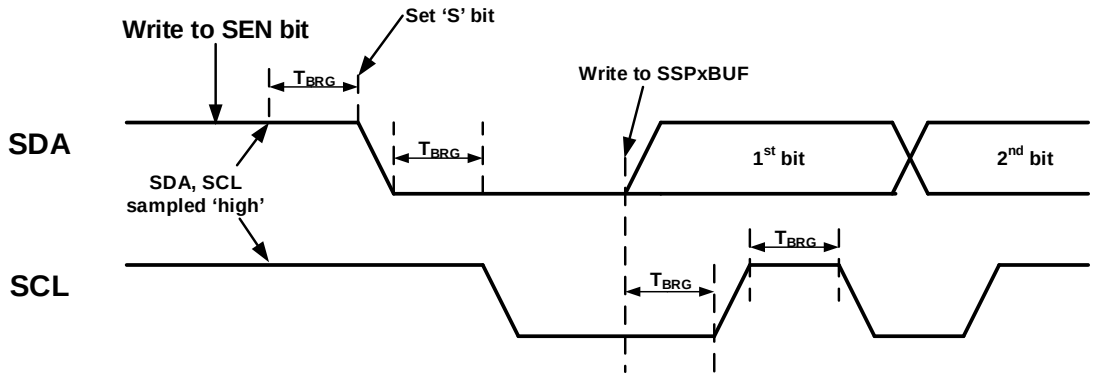
To initiate a Start condition (see Figure 2-5), the user sets the Start Condition Enable (SEN) bit. If the SDA and SCL pins are sampled high, the Baud Rate Generator is reloaded with the contents of SSPxADD and starts its count. If

SCL and SDA are both sampled high when the Baud Rate Generator times out (T_{BRG}), the SDA pin is driven low. The action of the SDA being driven low, while SCL is high, is the Start condition and causes the Start (S) bit to be set. Following this, the Baud Rate Generator is reloaded with the contents of SSPxADD and resumes its count. When the Baud Rate Generator times out (T_{BRG}), the SEN bit will be automatically cleared by hardware; the Baud Rate Generator is suspended, leaving the SDA line held low and the Start condition is complete.

Important:

1. If at the beginning of the Start condition, the SDA and SCL pins are already sampled low, or if during the Start condition, the SCL line is sampled low before the SDA line is driven low, a bus collision occurs. When this collision occurs, the Bus Collision Interrupt Flag (BCLxIF) is set, the Start condition is aborted and the I²C module is reset into its Idle state.
2. The Philips I²C Specification states that a bus collision cannot occur on a Start.

Figure 2-5. First Start Bit Timing



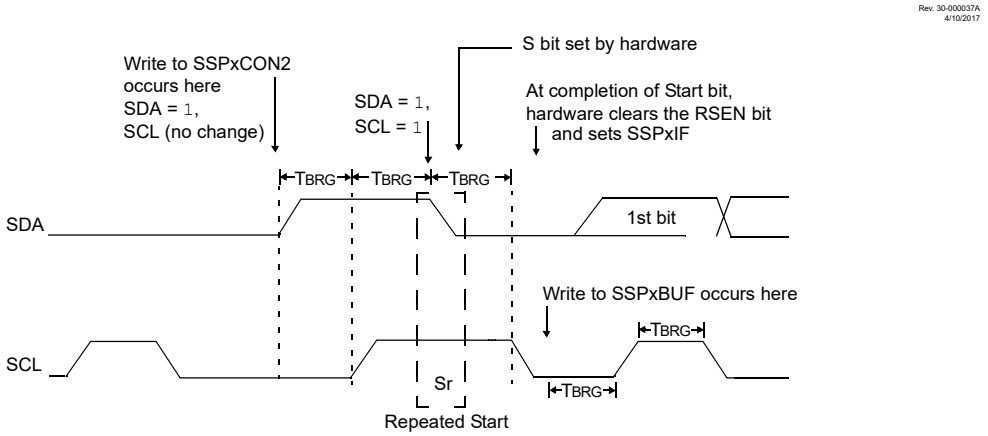
2.2.4 I²C Master Mode Repeated Start Condition Timing

A Repeated Start condition (see Figure 2-6) occurs when the Repeated Start Condition Enable (RSEN) bit is programmed high and the master state machine is no longer active. When the RSEN bit is set, the SCL pin is asserted low. When the SCL pin is sampled low, the Baud Rate Generator is loaded and begins counting. The SDA pin is released (brought high) for one Baud Rate Generator count (T_{BRG}). When the Baud Rate Generator times out, if SDA is sampled high, the SCL pin will be deasserted (brought high). When SCL is sampled high, the Baud Rate Generator is reloaded and begins counting. SDA and SCL must be sampled high for one T_{BRG}. This action is then followed by assertion of the SDA pin (SDA = 0) for one T_{BRG}, while SCL is high. After the BRG sample time has passed, SCL is asserted low. Following this, the RSEN bit will be automatically cleared and the Baud Rate Generator will not be reloaded, leaving the SDA pin held low. As soon as a Start condition is detected on the SDA and SCL pins, the S bit will be set. The SSPxIF bit will not be set until the Baud Rate Generator has timed out.

Important:

1. If RSEN is programmed while any other event is in progress, it will not take effect.
2. A bus collision during the Repeated Start condition occurs if:
   - SDA is sampled low when SCL goes from low-to-high.
   - SCL goes low before SDA is asserted low. This may indicate that another master is attempting to transmit a data '1'.

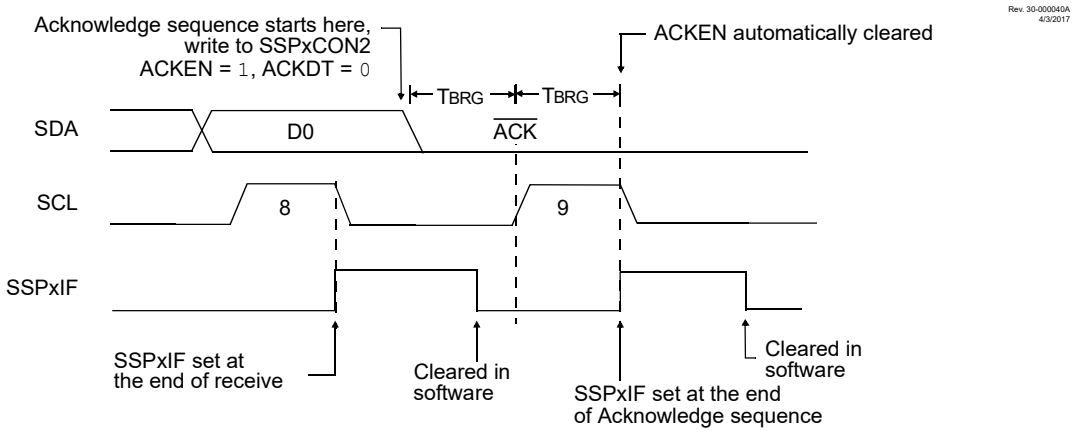
Figure 2-6. Repeated Start Condition Waveform



2.2.5 Acknowledge Sequence Timing

An Acknowledge sequence (see Figure 2-7) is enabled by setting the Acknowledge Sequence Enable (ACKEN) bit. When this bit is set, the SCL pin is pulled low and the contents of the Acknowledge Data (ACKDT) bit are presented on the SDA pin. If the user wishes to generate an Acknowledge, then the ACKDT bit should be cleared. If not, the user should set the ACKDT bit before starting an Acknowledge sequence. The Baud Rate Generator then counts for one rollover period (T_{BRG}) and the SCL pin is deasserted (pulled high). When the SCL pin is sampled high (clock arbitration), the Baud Rate Generator counts for T_{BRG}. The SCL pin is then pulled low. Following this, the ACKEN bit is automatically cleared, the Baud Rate Generator is turned off and the MSSP module then goes into Idle mode.

Figure 2-7. Acknowledge Sequence Waveform

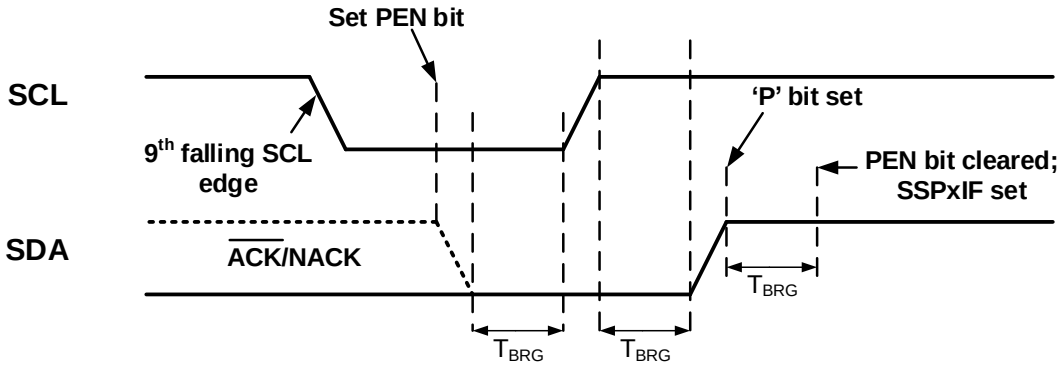


Note: TBRG = one Baud Rate Generator period.

2.2.6 Stop Condition Timing

A Stop condition (see Figure 2-8) is asserted on the SDA pin at the end of a receive/transmit by setting the Stop Condition Enable (PEN) bit. At the end of a receive/transmit, the SCL line is held low after the falling edge of the ninth clock. When the PEN bit is set, the master will assert the SDA line low. When the SDA line is sampled low, the Baud Rate Generator is reloaded and counts down to '0'. When the Baud Rate Generator times out, the SCL pin will be brought high and one T_{BRG} (Baud Rate Generator rollover count) later, the SDA pin will be deasserted. When the SDA pin is sampled high while SCL is high, the Stop (P) bit is set by hardware. One T_{BRG} later, the PEN bit is cleared and the SSPxIF bit is set.

Figure 2-8. Stop Condition in Receive or Transmit Mode



2.2.7 Sleep Operation

While in Sleep mode, the I²C slave module can receive addresses or data. When an address match or complete byte transfer occurs during Sleep, it will wake the processor, if the MSSP Interrupt Enable (SSPxIE) bit is enabled.

2.2.8 Effects of a Reset

A Reset disables the MSSP module and terminates the current transfer.

2.3 I²C Master Mode Transmission

Transmission of a data byte, a 7-bit address or the other half of a 10-bit address is accomplished by simply writing a value to the SSPxBUF register. This action will set the Buffer Full Status (BF) bit and allow the Baud Rate Generator to begin counting and start the next transmission. Each bit of address/data will be shifted out onto the SDA pin after the falling edge of SCL is asserted. SCL is held low for one Baud Rate Generator rollover count (T_{BRG}). Data should be valid before SCL is released high. When the SCL pin is released high, it is held that way for T_{BRG}. The data on the SDA pin must remain stable for that duration and some hold time after the next falling edge of SCL.

After the eighth bit is shifted out (the falling edge of the eighth clock), the BF flag is cleared and the master releases SDA. This allows the slave device being addressed to respond with an $\overline{ACK}$ sequence during the ninth bit time, if an address match occurred, or if data was received properly. The status of $\overline{ACK}$ is written into the Acknowledge Status (ACKSTAT) bit on the rising edge of the ninth clock. If the master receives an $\overline{ACK}$, the ACKSTAT bit is cleared by hardware. If a NACK is received, hardware sets the ACKSTAT bit. After the ninth falling clock edge, the SSPxIF bit is set and the master clock (Baud Rate Generator) is suspended until the next data byte is loaded into the SSPxBUF, leaving SCL low and SDA unchanged (see Figure 2-9).

After the write to the SSPxBUF, each bit of the address will be shifted out on the falling edge of SCL until all seven address bits and the R/$\overline{W}$ bit are completed. On the falling edge of the eighth clock, the master will release the SDA pin, allowing the slave to respond with an $\overline{ACK}$. On the falling edge of the ninth clock, the master will sample the SDA pin to see if the address was recognized by a slave. The status of the $\overline{ACK}$ bit is loaded into the ACKSTAT bit. Following the falling edge of the ninth clock transmission of the address, the SSPxIF is set, the BF flag is cleared and the Baud Rate Generator is turned off until another write to the SSPxBUF takes place, holding SCL low and allowing SDA to float.

2.3.1 BF Status Flag

In Transmit mode, the Buffer Full Status (BF) bit is set automatically by hardware when software writes to SSPxBUF, and is cleared by hardware after all eight bits are shifted out.

2.3.2 WCOL Status Flag

If the user writes the SSPxBUF when a transmit is already in progress (e.g., SSPSR is still shifting out a data byte), the Write Collision Detect (WCOL) bit is set and the contents of the buffer are unchanged (the write does not occur).

The WCOL bit must be cleared by software before the next transmission.

2.3.3 ACKSTAT Status Flag

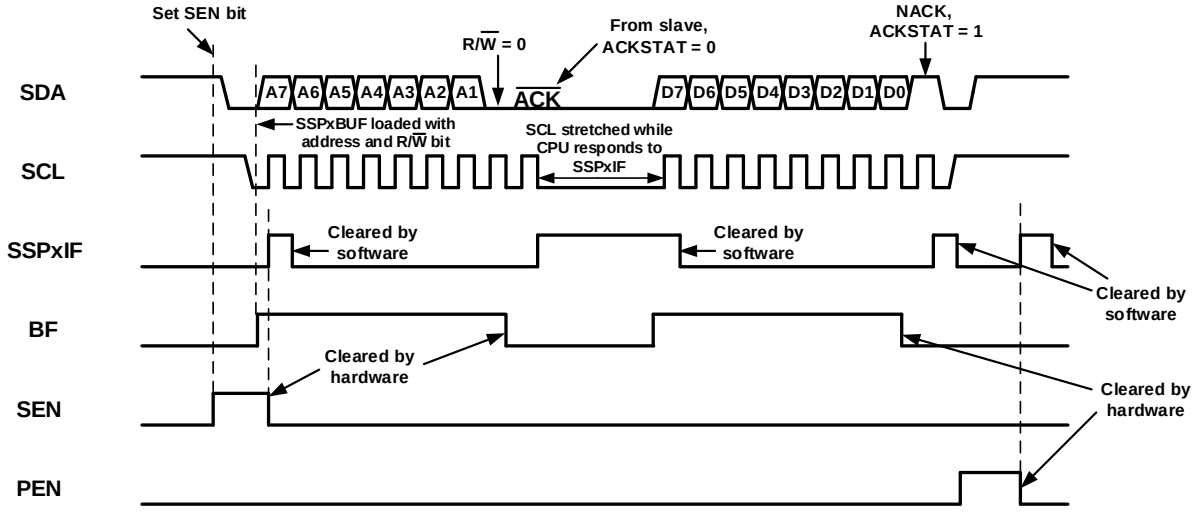
In Transmit mode, the Acknowledge Status (ACKSTAT) bit is cleared by module hardware when the slave has sent an Acknowledge ($\overline{ACK}$ = 0), and is set by hardware when the slave issues a NACK. A slave sends an $\overline{ACK}$ when it has recognized its address (including a General Call), or when the slave has properly received its data.

2.3.4 Master Mode 7-Bit Transmit Sequence

The following steps highlight a typical 7-bit transmit sequence:

1. Software sets the SEN bit, master hardware generates a Start condition.
2. Upon the completion of the Start condition, hardware sets SSPxIF.
3. Software clears the SSPxIF bit.
4. Software loads SSPxBUF with the 7-bit slave address and R/$\overline{W}$ bit. In Master Transmit mode, the R/$\overline{W}$ value is '0'.
5. The address is shifted out on the SDA pin until all eight bits have been transmitted.
6. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
7. Master hardware sets the SSPxIF bit. If the SSPxIE bit is also set, an interrupt is generated. SSPxIF must be cleared by software.
8. Software loads SSPxBUF with a data byte.
9. Data is shifted out until all eight bits have been transmitted.
10. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
11. On the ninth falling clock edge, SSPxIF is set by hardware. If SSPxIE is also set, an interrupt is generated. SSPxIF must be cleared in software.
12. Repeat steps 8 - 11 until all data has been transmitted.
13. Software generates a Stop or Restart condition by setting the PEN or RSEN bits, respectively. Once the Stop/ Restart condition is complete, hardware sets SSPxIF.

Figure 2-9. I²C Master Mode Waveform (Transmission, 7-bit Address)

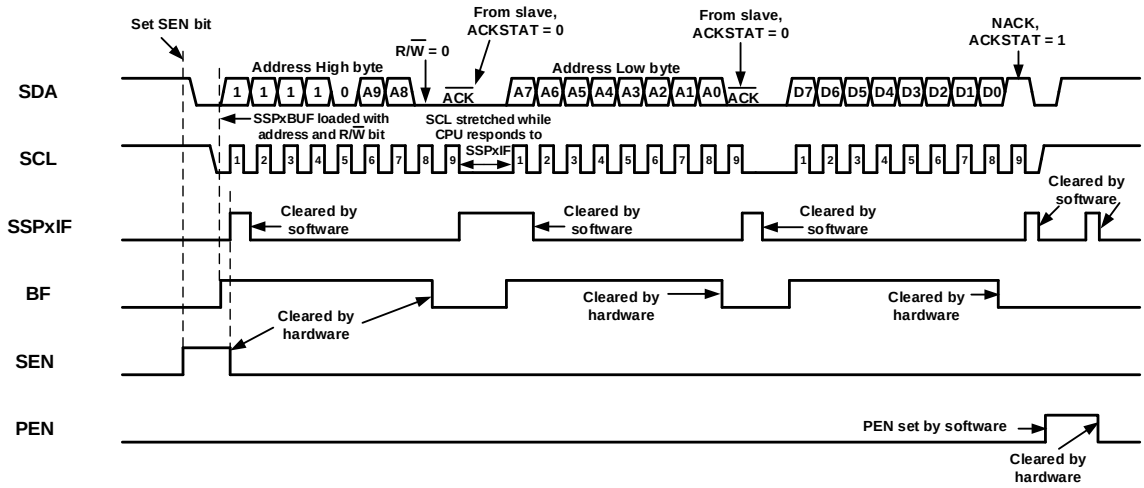


2.3.5 Master Mode 10-Bit Transmit Sequence

The following steps highlight a typical 10-bit transmit sequence:

1. Software sets the SEN bit, master hardware generates a Start condition.
2. Upon the completion of the Start condition, hardware sets SSPxIF.
3. Software clears the SSPxIF bit.
4. Software loads SSPxBUF with the 10-bit high address byte and R/$\overline{W}$ bit. In 10-bit Master Transmit mode, the R/$\overline{W}$ bit value is '0'.
5. The high address byte is shifted out on the SDA line until all eight bits have been transmitted.
6. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
7. Master hardware sets the SSPxIF bit. If the SSPxIE bit is also set, an interrupt is generated. SSPxIF must be cleared by software.
8. Software loads SSPxBUF with the 10-bit low address byte.
9. The low address byte is shifted out on the SDA line until all eight bits have been transmitted.
10. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
11. Master hardware sets the SSPxIF bit. If the SSPxIE bit is also set, an interrupt is generated. SSPxIF must be cleared by software.
12. Software loads SSPxBUF with a data byte.
13. Data is shifted out until all eight bits have been transmitted.
14. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
15. On the ninth falling clock edge, SSPxIF is set by hardware. If SSPxIE is also set, an interrupt is generated. SSPxIF must be cleared in software.
16. Repeat steps 12 - 15 until all data has been transmitted.
17. Software generates a Stop or Restart condition by setting the PEN or RSEN bits, respectively. Once the Stop/ Restart condition is complete, hardware sets SSPxIF.

Figure 2-10. I²C Master Mode Waveform (Transmission, 10-bit Address)



2.4 I²C Master Mode Reception

Master mode reception (see Figure 2-11) is enabled by setting the Receive Enable (RCEN) bit.

Important: The MSSP module must be in an Idle state before the RCEN bit is set or the RCEN bit will be disregarded.

The Baud Rate Generator begins counting and on each rollover, the state of the SCL pin changes (high-to-low/low-to-high) and data is shifted into the SSPxSR. After the falling edge of the eighth clock all the following events occur:

- RCEN is automatically cleared by hardware.
- The contents of the SSPxSR are loaded into the SSPxBUF.
- The BF flag bit is set.
- The SSPxIF flag bit is set.
- The Baud Rate Generator is suspended from counting.
- The SCL pin is held low.

The MSSP is now in an Idle state awaiting the next command. When the buffer is read by software, the BF flag bit is automatically cleared. The Master can then send an Acknowledge sequence at the end of reception by setting the Acknowledge Sequence Enable (ACKEN) bit.

2.4.1 BF Status Flag

In receive operation, the BF bit is set when an address or data byte is loaded into SSPxBUF from SSPxSR. BF is cleared when the SSPxBUF register is read.

2.4.2 WCOL Status Flag

If software writes the SSPxBUF when a receive is already in progress (i.e., SSPxSR is still shifting in a data byte), the WCOL bit is set and the contents of the buffer are unchanged (the write does not occur). WCOL must be cleared in software.

2.4.3 SSPOV Status Flag

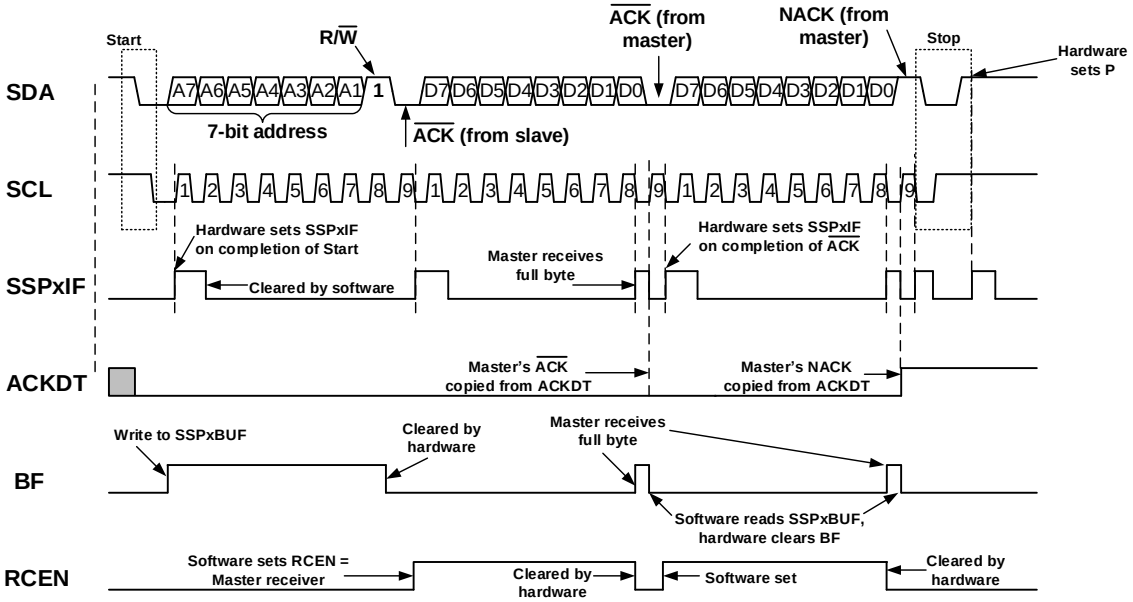
In receive operation, the MSSP Overflow (SSPOV) bit is set when eight bits are received into SSPxSR while the BF flag bit is already set from a previous reception. Software must clear SSPOV.

2.4.4 Master Mode 7-Bit Receive Sequence

The following steps highlight a typical 7-bit receive sequence:

1. Software sets the SEN bit, master hardware generates a Start condition.
2. Upon the completion of the Start condition, hardware sets SSPxIF.
3. Software clears the SSPxIF bit.
4. Software loads SSPxBUF with the 7-bit slave address and R/$\overline{W}$ bit. In 7-bit Master Receive mode, the R/$\overline{W}$ value is ‘1’.
5. The address is shifted out on the SDA pin until all eight bits have been transmitted.
6. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
7. Master hardware sets the SSPxIF bit. If the SSPxIE bit is also set, an interrupt is generated. SSPxIF must be cleared by software.
8. Software sets the RCEN bit, and the master clocks in a byte from the slave.
9. After the eight falling clock edge, the byte is transferred from the SSPxSR and sets the SSPxIF and BF bits. RCEN is cleared by hardware.
10. Software clears SSPxIF and reads the received byte from SSPxBUF, clearing the BF bit.
11. Software clears the ACKDT bit and initiates an $\overline{ACK}$ sequence by setting the ACKEN bit.
12. Master hardware transmits the $\overline{ACK}$ sequence.
13. On the ninth falling clock edge, SSPxIF is set. Software must clear SSPxIF.
14. Repeat steps 8 - 13 until all bytes have been received from the slave.
15. Master software can end communication by performing one of the following:
    - Software sets the ACKDT bit and transmits a NACK sequence by setting the ACKEN bit.
    - Software sets the PEN bit and hardware transmits a Stop condition.
    - Software sets the RSEN bit and hardware issues a Restart condition.

Figure 2-11. I²C Master Mode Waveform (Reception, 7-bit Address)



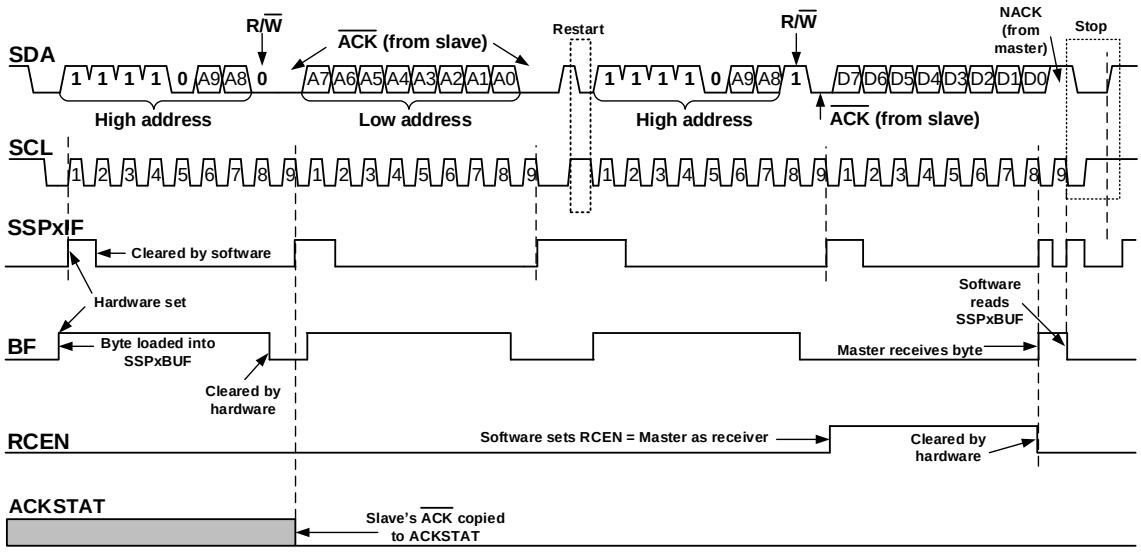
2.4.5 Master Mode 10-Bit Receive Sequence

The following steps highlight a typical 10-bit receive sequence:

1. Software sets the SEN bit, master hardware generates a Start condition.
2. Upon the completion of the Start condition, hardware sets SSPxIF.
3. Software clears the SSPxIF bit.
4. Software loads SSPxBUF with the 10-bit high address byte and R/$\overline{W}$ bit. In 10-bit Master Receive mode, the first time the high address byte is transmitted, the R/$\overline{W}$ bit value is '0' (see Figure 2-12).
5. The high address byte is shifted out on the SDA line until all eight bits have been transmitted.
6. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
7. Master hardware sets the SSPxIF bit. If the SSPxIE bit is also set, an interrupt is generated. SSPxIF must be cleared by software.
8. Software loads SSPxBUF with the 10-bit low address byte.
9. The low address byte is shifted out on the SDA line until all eight bits have been transmitted.
10. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
11. Master hardware sets the SSPxIF bit. If the SSPxIE bit is also set, an interrupt is generated. SSPxIF must be cleared by software.
12. Software sets the RSEN bit, hardware issues a Restart condition.
13. Software loads SSPxBUF with the 10-bit high address byte and R/$\overline{W}$ bit. This second stage of addressing in 10-bit Master Receive mode requires the R/$\overline{W}$ bit value to be '1'.
14. The high address byte is shifted out on the SDA line until all eight bits have been transmitted.
15. Master hardware clocks in the $\overline{ACK}$ value from the slave and copies the value into the ACKSTAT bit.
16. Master hardware sets the SSPxIF bit. If the SSPxIE bit is also set, an interrupt is generated. SSPxIF must be cleared by software.
17. Software sets the RCEN bit, and hardware clocks in a byte from the slave.
18. After the eight falling clock edge, the byte is transferred from the SSPxSR and sets the SSPxIF and BF bits. RCEN is cleared by hardware.

19. Software clears SSPxIF and reads the received byte from SSPxBUF, clearing the BF bit.
20. Software clears the ACKDT bit and initiates an $\overline{ACK}$ sequence by setting the ACKEN bit.
21. Master hardware transmits the $\overline{ACK}$ sequence.
22. On the ninth falling clock edge, SSPxIF is set. Software must clear SSPxIF.
23. Repeat steps 17 - 22 until all bytes have been received from the slave.
24. Master software can end communication by performing one of the following:
    – Software sets the ACKDT bit and transmits a NACK sequence by setting the ACKEN bit.
    – Software sets the PEN bit and hardware transmits a Stop condition.
    – Software sets the RSEN bit and hardware issues a Restart condition.

Figure 2-12. I²C Master Mode Waveform (Reception, 10-bit Address)



2.5 Baud Rate Generator

The MSSP module has a Baud Rate Generator available for clock generation in both I²C and SPI Master modes. The Baud Rate Generator (BRG) reload value is placed in the SSPxADD register. When a write occurs to SSPxBUF, the Baud Rate Generator will automatically begin counting down. Equation 2-1 shows how the value for SSPxADD is calculated.

Once the given operation is complete, the internal clock will automatically stop counting and the clock pin will remain in its last state.

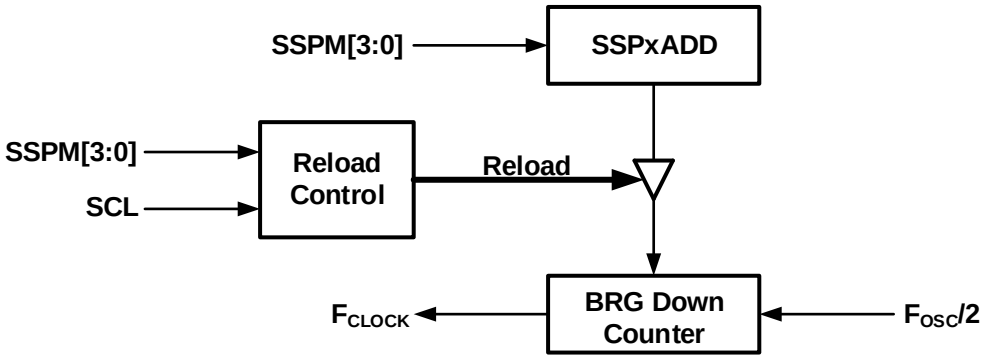
An internal signal “Reload”, shown in Figure 2-13, triggers the value from SSPxADD to be loaded into the BRG counter. This occurs twice for each oscillation of the module clock line. The logic dictating when the reload signal is asserted depends on the mode in which the MSSP is being operated.

Table 2-1 illustrates clock rates based on instruction cycles and the BRG value loaded into SSPxADD.

Equation 2-1. MSSP Baud Rate Generator Frequency

$$F_{CLOCK} = \frac{F_{OSC}}{4 \times (SSPxADD + 1)}$$

Figure 2-13. Baud Rate Generator Block Diagram



Important: Values of 0x00, 0x01 and 0x02 are not valid for SSPxADD when used as a Baud Rate Generator for I^2C. This is an implementation limitation.

Table 2-1. MSSP Clock Rate w/BRG

F_{OSC}	F_{CY}	BRG Value	F_{CLOCK} (2 Rollovers of BRG)
32 MHz	8 MHz	13h	400 kHz
32 MHz	8 MHz	19h	308 kHz
32 MHz	8 MHz	4Fh	100 kHz
16 MHz	4 MHz	09h	400 kHz
16 MHz	4 MHz	0Ch	308 kHz
16 MHz	4 MHz	27h	100 kHz
4 MHz	1 MHz	09h	100 kHz

Note: Refer to the **"Electrical Specifications"** chapter of the data sheet, to ensure the system is designed to support all requirements.

3. Master Mode Code Example

Example 3.1 shows the use of the MSSP in 7-bit I²C Master mode. The example uses bit polling to determine when to issue a Start condition, load data and other I²C functions. The functions used in the code example are used to read and write a single data byte to a specific address location within the slave's memory. In the '`I2C_WriteByte()`' routine, the byte following the address byte represents the register or memory address within the slave that will receive the data. In the '`I2C_ReadByte()`' routine, the byte following the address is also the register or memory address within the slave's memory that will be read by the master. In this routine, once the slave acknowledges the register address, the master must issue a Restart condition and transmit the slave address again with the R/$\overline{W}$ bit set.

Important: For this code example, the slave software must be written such that the slave understands that the first data byte is actually a register address and not actual data.

Example 3-1. MSSP in I²C Master Mode

```
// main() routine
uint8_t slaveAddress = 0x30;                    // 7-bit slave address
uint8_t regAddress = 0x00;                      // Register location
uint8_t writeVal = 0xCB;                        // Value to write
uint8_t readVal = 0x00;                         // Value from slave

void main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        I2C_WriteByte(slaveAddress, regAddress, writeVal);   // Write to slave
        __delay_ms(100);                                     // Short delay
        readVal = I2C_ReadByte(slaveAddress, regAddress);    // Read back data
        __delay_ms(1000);                                    // 1 second delay
    }
}

// I2C initialize routine
void I2C_Initialize(void)
{
    SSP1STAT = 0x80;                        // Sample end of data output
    SSP1CON1 = 0x08;                        // SCL =FOSC/4(SSPxADD+1)
    SSP1CON3 = 0x00;
    SSP1ADD = 0x4F;                         // 0x4F = 100 kHz Clock @ 32 MHz
    PIR3bits.SSP1IF = 0;                    // Clear the interrupt flag
    PIE3bits.SSP1IE = 0;                    // Disable the interrupt
    SSP1CON1bits.SSPEN = 1;                 // Enable MSSP
}

// Master write routine
void I2C_WriteByte(uint8_t deviceADDR, uint8_t registerADDR, uint8_t val)
{
    while (!Idle);                          // Wait until MSSP module is idle
    I2C_Start = 1;                          // Set START bit
    while (I2C_Start);                      // Wait until hardware clears SEN
    SSP1BUF = deviceADDR;                   // Load device address (write)
    while (!Idle);                          // Wait until MSSP module is idle
    SSP1BUF = registerADDR;                 // Send register address (write)
    while (!Idle);                          // Wait until module is idle
    SSP1BUF = val;                          // Send data to slave
    while (!Idle);                          // Wait until module is idle
    I2C_Stop = 1;                           // Set STOP bit
    while (!Idle);                          // Wait until module is idle
}

// Master read routine
uint8_t I2C_ReadByte(uint8_t deviceADDR, uint8_t registerADDR)
{
```

```
    uint8_t data;                       // Data byte
    while (!Idle);                      // Wait until module is idle
    I2C_Start = 1;                      // Set START bit
    while (I2C_Start);                  // Wait until hardware clears SEN
    SSP1BUF = deviceADDR;               // Load device address (write)
    while (!Idle);                      // Wait until module is idle
    SSP1BUF = registerADDR;             // Send register address (write)
    while (!Idle);                      // Wait until module is idle

    I2C_Restart = 1;                    // Restart condition
    while (I2C_Restart);

    SSP1BUF = (deviceADDR | 0x01);      // Send device address (read)
    while (!Idle);                      // Wait until module is idle
    SSP1CON2bits.RCEN = 1;              // Set RCEN (Master receiver)
    while (!Idle);                      // Wait until module is idle
    data = SSP1BUF;                     // Read SSPBUF
    SSP1CON2bits.ACKDT = 1;             // ACK bit, 1 = Not acknowledge
    SSP1CON2bits.ACKEN = 1;             // Start ACK sequence
    while (SSP1CON2bits.ACKEN);         // Wait for end of ACK sequence
    SSP1CON2bits.ACKDT = 0;             // ACK bit, 0 = acknowledge
    I2C_Stop = 1;                       // Set STOP bit
    while (!Idle);                      // Wait until module is idle
    return data;                        // Return the data
}
```

4. Conclusion

The Master Synchronous Serial Port (MSSP) is an integrated serial communications module that contains two sub-modules: the SPI and the I^2C. The I^2C can be configured to operate as a bus master, a bus slave or both, in Multi-Master mode. This technical brief highlights the use of the MSSP in I^2C Master mode and includes basic I^2C Specification information, terminology, set-up information and a software example.

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

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

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

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

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

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

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

ISBN: 978-1-5224-5879-1

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820