TB3251

Using the MSSP in I2C Slave Mode

Introduction

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

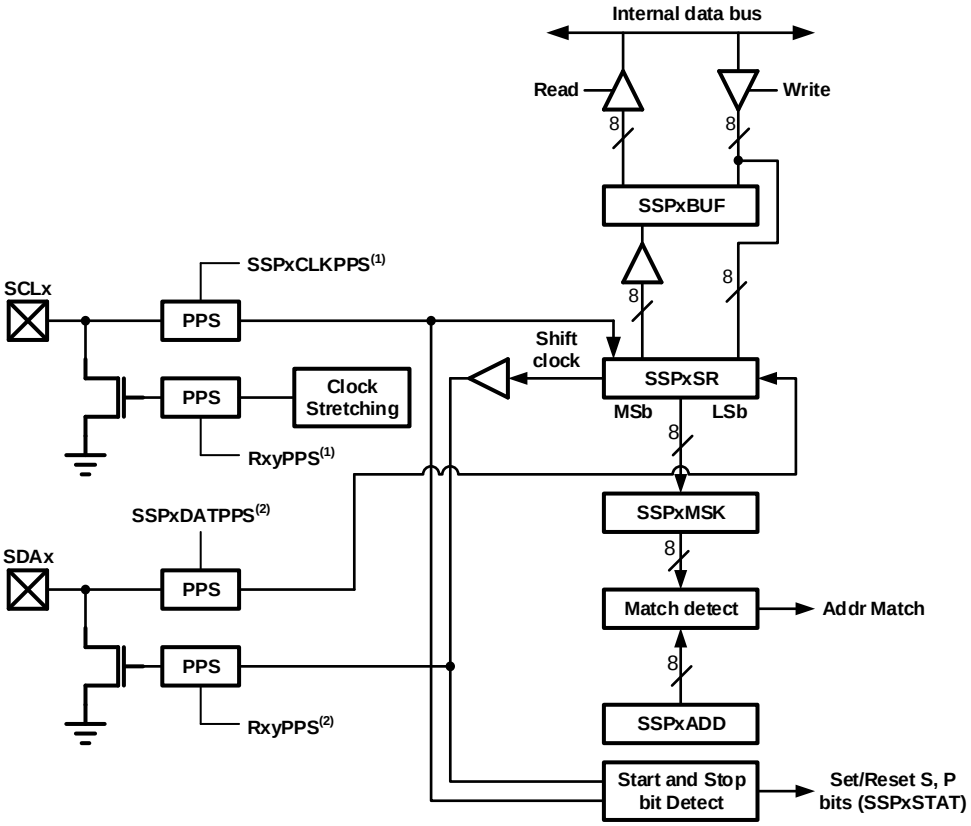
- SPI - Serial Peripheral Interface
- I^2C - Inter-Integrated Circuit

The Inter-Integrated Circuit, commonly referred to as I^2C, is a synchronous, two-wire, bidirectional serial communications bus. The I^2C module can be used to communicate with other I^2C-compatible devices such as EEPROMs, display drivers, sensors or other mircontroller 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 technical brief discusses the MSSP operating in the I^2C Slave mode. Figure 1 shows a block diagram of the MSSP module in I^2C Slave mode.

Figure 1. MSSP Block Diagram (I²C Slave Mode)



Note 1: SDA pin selections must be the same for input and output.
2: SCL 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 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 can communicate with any other I²C device changing 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

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

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

$V_{DD} = Supply\ \ voltage$

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

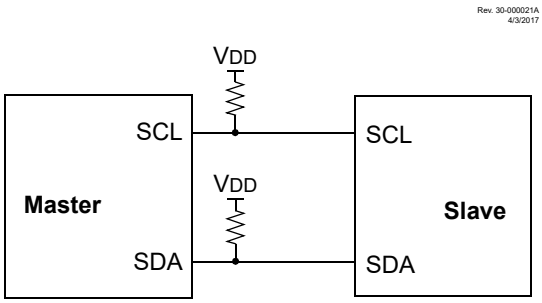
$I_{OL} = Minimum\ sink\ current$

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 signal 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 stretching in 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.

2.1.4 Start Condition

The I^2C Specification defines a Start condition as a transition of SDA, from a High to a 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 Start and Stop conditions.

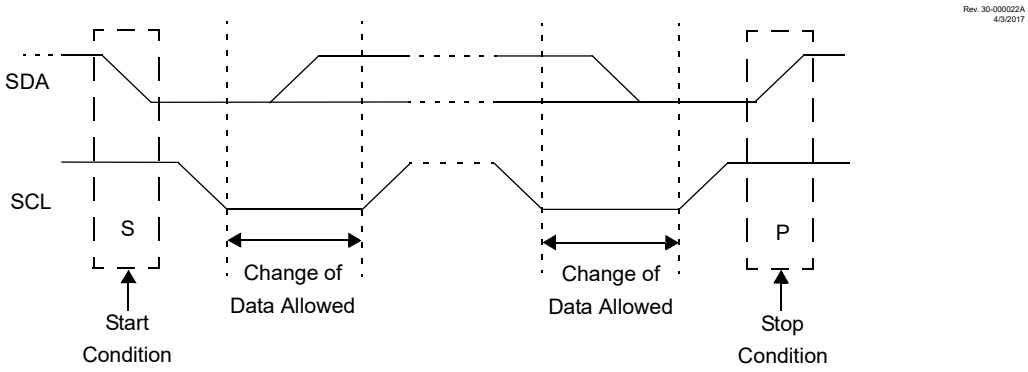
A bus collision can occur on a Start condition if the module samples the SDA line low, before asserting it low. This does not conform to the I^2C Specification that states no bus collision can occur on a Start.

2.1.5 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. I²C Start and Stop Conditions



2.1.6 Start/Stop Condition Interrupt Masking

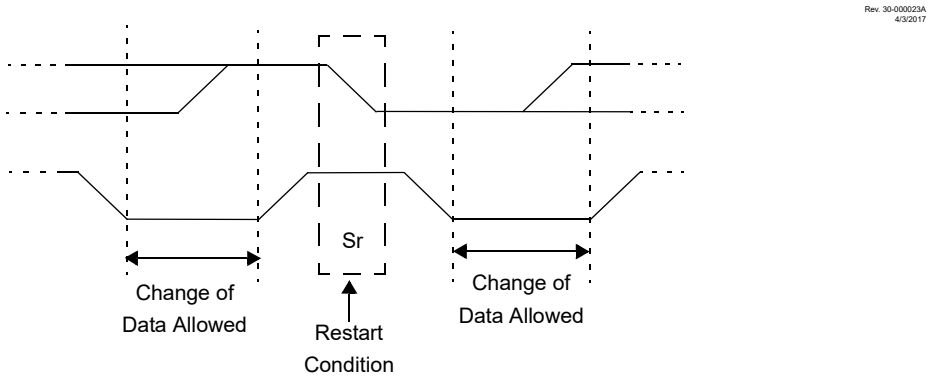
The Start Condition Interrupt Enable (SCIE) and Stop Condition Interrupt Enable (PCIE) bits can enable the generation of an interrupt in Slave modes that do not typically support this function. These bits will have no effect in Slave modes where interrupt on Start and Stop detect are already enabled.

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 slave software, when the Address Hold Enable (AHEN) and Data Hold Enable (DHEN) bits are set, 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.

The slave hardware will generate an $\overline{ACK}$ response under most circumstances. However, if the BF bit or the Receive Overflow Indicator (SSPOV) bits are set when a byte is received, then the $\overline{ACK}$ will not be sent by the slave.

When the module is addressed, after the eighth falling edge of SCL on the bus, the Acknowledge Time Status (ACKTIM) bit is set. The ACKTIM bit indicates the acknowledge time of the active bus. The ACKTIM bit is only active when either the AHEN bit or DHEN bit is enabled.

2.2 I²C Slave Mode Operation

The MSSP Slave mode operates in one of four modes selected by the MSSP Mode Select (SSPM) bits. The modes can be divided into 7-bit and 10-bit Addressing modes. The 10-bit Addressing mode operates the same as the 7-bit, with some additional overhead for handling the larger addresses.

Modes with Start and Stop condition interrupts operate the same as the other modes with SSPxIF, additionally getting set upon detection of a Start, Restart or Stop condition.

2.2.1 Slave Mode Addresses

The SSPxADD register contains the Slave mode address. The first byte received after a Start or Restart condition is compared against the value stored in this register. If the byte matches, the value is loaded into the SSPxBUF register and an interrupt is generated. If the value does not match, the module goes Idle and no indication is given to the software that anything happened.

The SSPxMSK register affects the address matching process. See the “SSP Mask Register” section for more information.

2.2.1.1 I²C Slave 7-bit Addressing Mode

In 7-bit Addressing mode, the LSb of the received data byte is ignored when determining if there is an address match.

2.2.1.2 I²C Slave 10-bit Addressing Mode

In 10-bit Addressing mode, the first received byte is compared to the binary value of ‘1 1 1 1 0 A9 A8 0’. A9 and A8 are the two MSbs of the 10-bit address and stored in bits 2 and 1 of the SSPxADD register.

After the acknowledge of the high byte, the Update Address (UA) bit is set and SCL is held low until the user updates SSPxADD with the low address. The low address byte is clocked in and all eight bits are compared to the low address value in SSPxADD. Even if there is not an address match; SSPxIF and UA are set, and SCL is held low until SSPxADD is updated to receive a high byte again. When SSPxADD is updated, the UA bit is cleared. This ensures the module is ready to receive the high address byte on the next communication.

A high and low address match as a write request is required at the start of all 10-bit addressing communication. A transmission can be initiated by issuing a Restart, once the slave is addressed and clocking in the high address with the R/$\overline{W}$ bit set. The slave hardware will then acknowledge the read request and prepare to clock out data. This is only valid for a slave, after it has received a complete high and low address byte match.

2.2.2 Clock Stretching

When a slave device has not completed processing data, it can delay the transfer of more data through the process of clock stretching. An addressed slave device may hold the SCL clock line low after receiving or sending a bit, indicating that it is not yet ready to continue. The master that is communicating with the slave will attempt to raise the SCL line in order to transfer the next bit, but will detect that the clock line has not yet been released. Because the SCL connection is open-drain, the slave has the ability to hold that line low until it is ready to continue communicating.

Clock stretching allows receivers that cannot keep up with a transmitter to control the flow of incoming data.

2.2.2.1 Normal Clock Stretching

Following an $\overline{ACK}$, if the R/$\overline{W}$ bit is set (a read request), the slave hardware will clear CKP. This allows the slave time to update SSPxBUF with data to transfer to the master. If the Stretch Enable (SEN) bit is set, the slave hardware will

always stretch the clock after the $\overline{ACK}$ sequence. Once the slave is ready; CKP is set by software and communication resumes.

2.2.2.2 10-bit Addressing Mode

In 10-bit Addressing mode, when the UA bit is set, the clock is always stretched. This is the only time the SCL is stretched without CKP being cleared. SCL is released immediately after a write to SSPxADD.

2.2.2.3 Byte NACKing

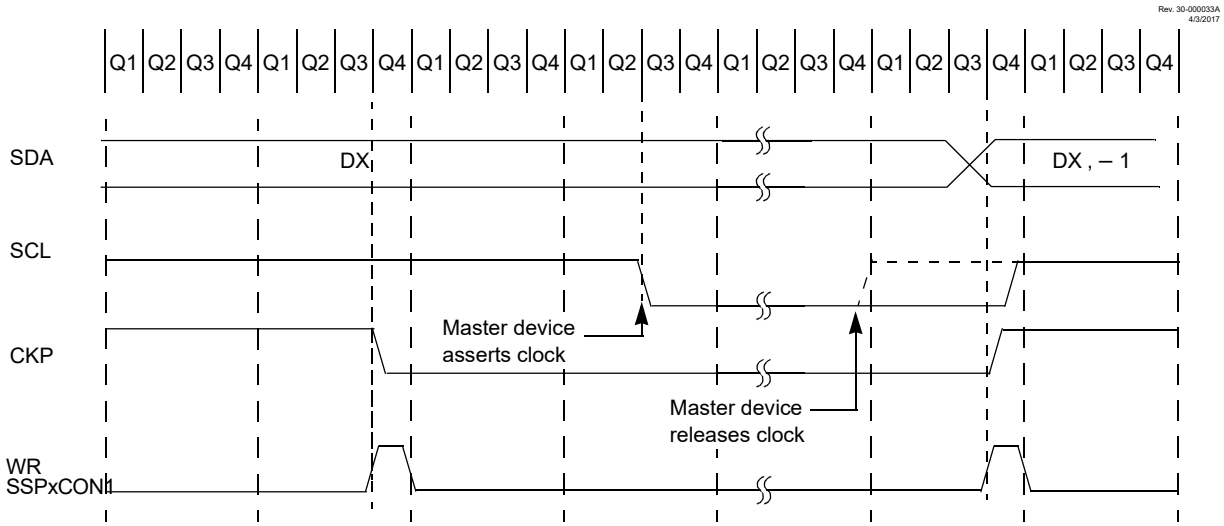
When the AHEN bit is set, CKP is cleared by hardware after the eighth falling edge of SCL for a received matching address byte. When the DHEN bit is set, CKP is cleared after the eighth falling edge of SCL for received data.

Stretching after the eighth falling edge of SCL allows the slave to look at the received address or data and decide if it wants to acknowledge ($\overline{ACK}$) the received address or data, or not acknowledge (NACK) the address or data.

2.2.3 Clock Synchronization and the CKP bit

Any time the CKP bit is cleared, the module will wait for the SCL line to go low and then hold it. However, clearing the CKP bit will not assert the SCL output low until the SCL output is already sampled low. Therefore, the CKP bit will not assert the SCL line until an external I^2C master device has already asserted the SCL line. The SCL output will remain low until the CKP bit is set and all other devices on the I^2C bus have released SCL. This ensures that a write to the CKP bit will not violate the minimum high time requirement for SCL (see Figure 2-4).

Figure 2-4. Clock Synchronization Timing

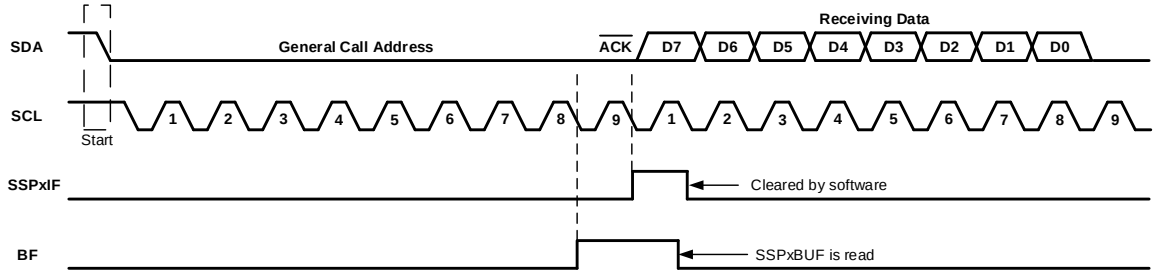


2.2.4 General Call Address Support

The addressing procedure for the I^2C bus is such that the first byte after the Start condition usually determines which device will be the slave addressed by the master device. The exception is the general call address that can address all devices. When this address is used, all devices should, in theory, respond with an $\overline{ACK}$.

The general call address is a reserved address in the I^2C Specification, defined as address 0x00. When the General Call Enable (GCEN) bit is set, the slave module will automatically $\overline{ACK}$ the reception of this address, regardless of the value stored in SSPxADD. After the slave clocks in an address of all zeros with the R/$\overline{W}$ bit clear, an interrupt is generated and slave software can read SSPxBUF and respond. Figure 2-5 shows a general call reception sequence.

Figure 2-5. Slave Mode General Call Address Sequence



In 10-bit Address mode, the UA bit will not be set on the reception of the general call address. The slave will prepare to receive the second byte as data, just as it would in 7-bit mode.

If the AHEN bit is set, just as with any other address reception, the slave hardware will stretch the clock after the eighth falling edge of SCL. The slave must then set its Acknowledge Sequence Enable (ACKEN) bit and release the clock with communication progressing as it would normally.

2.2.5 SSP Mask Register

The MSSP Mask register (SSPxMSK) is available in I^{2}C Slave mode as a mask for the value held in the SSPSR register during an address comparison operation. A zero ('0') bit in the SSPxMSK register has the effect of making the corresponding bit of the received address a "don't care".

This register is Reset to all '1's upon any Reset condition and, therefore, has no effect on standard MSSP operation until written with a mask value.

SSPxMSK is active during:

- 7-bit Address mode: address compare of A[7:1].
- 10-bit Address mode: address compare of A[7:0] only. The MSSP mask has no effect during the reception of the first (high) byte of the address.

2.2.6 Slave Reception

When the R/$\overline{W}$ bit of a matching received address byte is clear, the R/$\overline{W}$ bit is cleared. The received address is loaded into the SSPxBUF register and acknowledged.

When the overflow condition exists for a received address, a Not Acknowledge (NACK) is transmitted and the Receive Overflow Indicator (SSPOV) bit is set. The Buffer Override Enable (BOEN) bit modifies this operation.

An MSSP interrupt is generated for each transferred data byte. The SSPxIF flag bit must be cleared by software.

When the SEN bit is set, SCL will be held low (clock stretch) following each received byte. The clock must be released by setting the CKP bit, but with some exceptions in 10-bit mode. See "10-Bit Addressing Mode" for more details.

2.2.6.1 7-bit Addressing Reception

The following describes a standard sequence of events for the MSSP module configured as an I^{2}C slave in 7-bit Addressing mode. Figure 2-6 and Figure 2-7 are used as a visual reference for this description.

This is a step by step process of what typically must be done to accomplish I^{2}C communication.

1. Start condition detected.
2. The Start (S) bit is set; SSPxIF is set if the Start Condition Interrupt Enable (SCIE) bit is set.
3. Matching address when R/$\overline{W}$ bit clear is received.
4. The slave pulls SDA low, sending an $\overline{ACK}$ to the master and sets SSPxIF bit.
5. Software clears the SSPxIF bit.
6. Software reads received address from SSPxBUF, clearing the BF flag.

7. If SEN = 1; Slave software sets the CKP bit to release the SCL line.
8. The master clocks out a data byte.
9. Slave drives SDA low, sending an $\overline{ACK}$ to the master and setting the SSPxIF bit.
10. Software clears SSPxIF.
11. Software reads the received byte from SSPxBUF, clearing BF.
12. Steps 8-12 are repeated for all received bytes from the master.
13. Master sends Stop condition, setting the Stop (P) bit, and the bus goes Idle.

Figure 2-6. I²C Slave, 7-bit Address, Reception (SEN = 0, AHEN = 0, DHEN = 0)

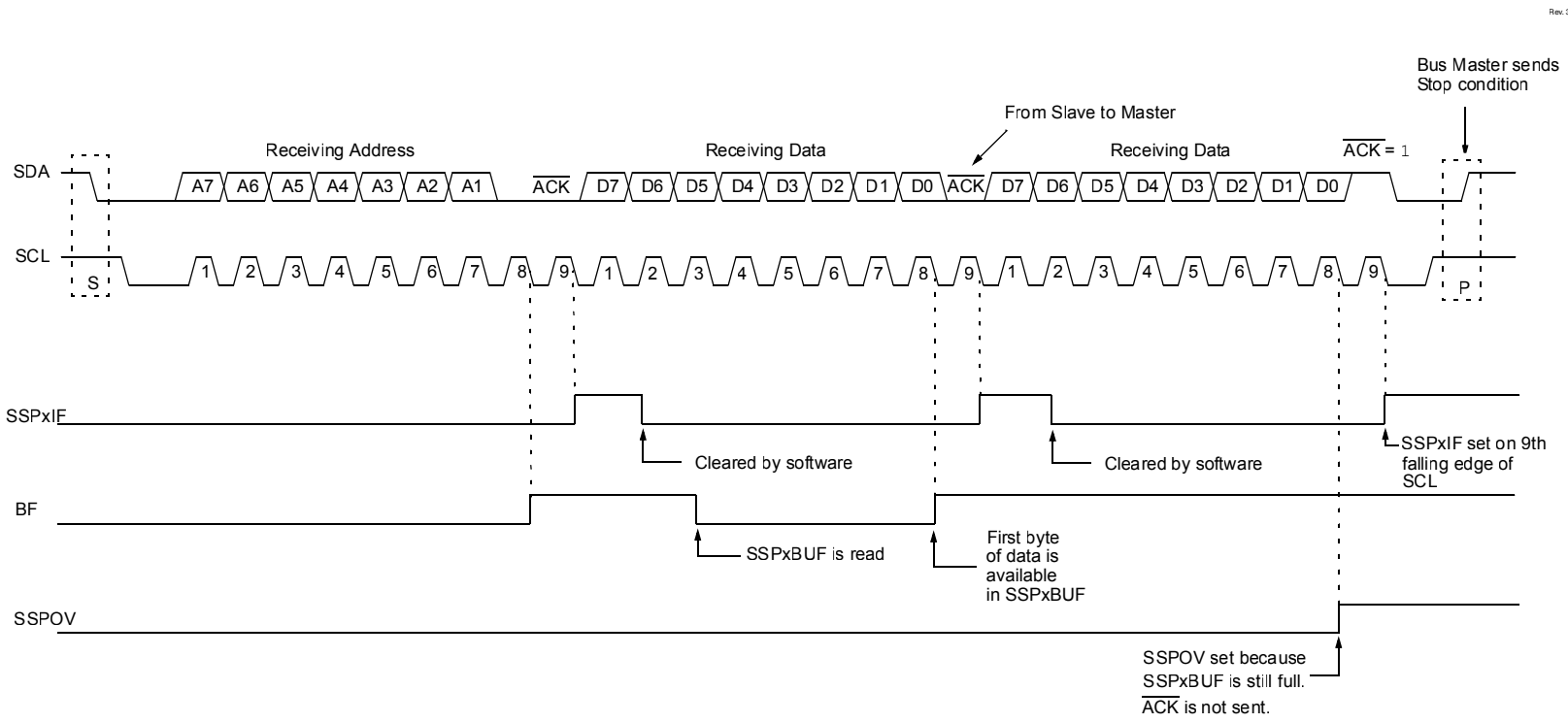
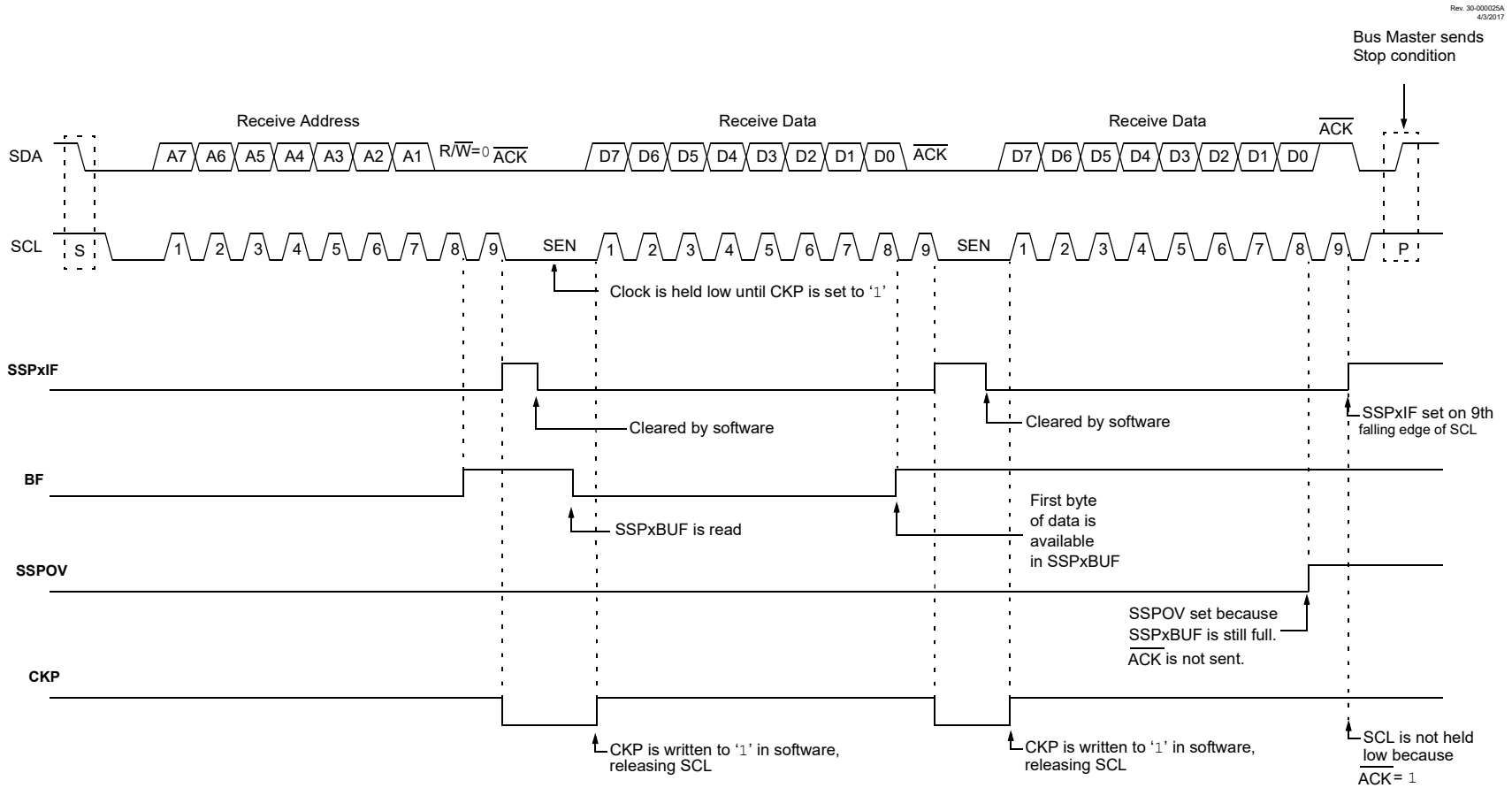


Figure 2-7. I²C Slave, 7-bit Address, Reception (SEN = 1, AHEN = 0, DHEN = 0)



2.2.6.2 7-bit Reception with AHEN and DHEN

Slave device reception, with AHEN and DHEN set, operates the same as without these options with extra interrupts and clock stretching added, after the eighth falling edge of SCL. These additional interrupts allow the slave software to decide whether it wants to $\overline{ACK}$ the receive address or data byte, rather than the hardware. This functionality adds support for PMBus™ that was not present on previous versions of this module.

This list describes the steps that need to be taken by slave software to use these options for I^2C communication. Figure 2-8 displays a module using both address and data holding. Figure 2-9 includes the operation with the SEN bit set.

1. The Start (S) bit is set; SSPxIF is set if SCIE is set.
2. Matching address with the R/$\overline{W}$ bit clear is clocked in. SSPxIF is set and CKP is cleared after the eighth falling edge of SCL.
3. Software clears the SSPxIF.
4. Slave can look at the ACKTIM bit to determine if the SSPxIF was after or before the $\overline{ACK}$.
5. Slave reads the address value from SSPxBUF, clearing the BF flag.
6. Slave transmits an $\overline{ACK}$ to the master by clearing ACKDT.
7. Slave releases the clock by setting CKP.
8. SSPxIF is set after an $\overline{ACK}$, not after a NACK.
9. If SEN = `1`, the slave hardware will stretch the clock after the $\overline{ACK}$.
10. Slave clears SSPxIF.

Important: SSPxIF is still set after the ninth falling edge of SCL, even if there is no clock stretching and BF has been cleared. Only if a NACK is sent to the master is SSPxIF not set.

11. SSPxIF is set and CKP cleared after eighth falling edge of SCL for a received data byte.
12. Slave looks at the ACKTIM bit to determine the source of the interrupt.
13. Slave reads the received data from SSPxBUF, clearing BF.
14. Steps 7-14 are the same for each received data byte.
15. Communication is ended by either the slave sending a NACK, or the master sending a Stop condition. If a Stop is sent and the Stop Condition Interrupt Enable (PCIE) bit is clear, the slave will only know by polling the Stop (P) bit.

Figure 2-8. I²C Slave, 7-bit Address, Reception (SEN = 0, AHEN = 1, DHEN = 1)

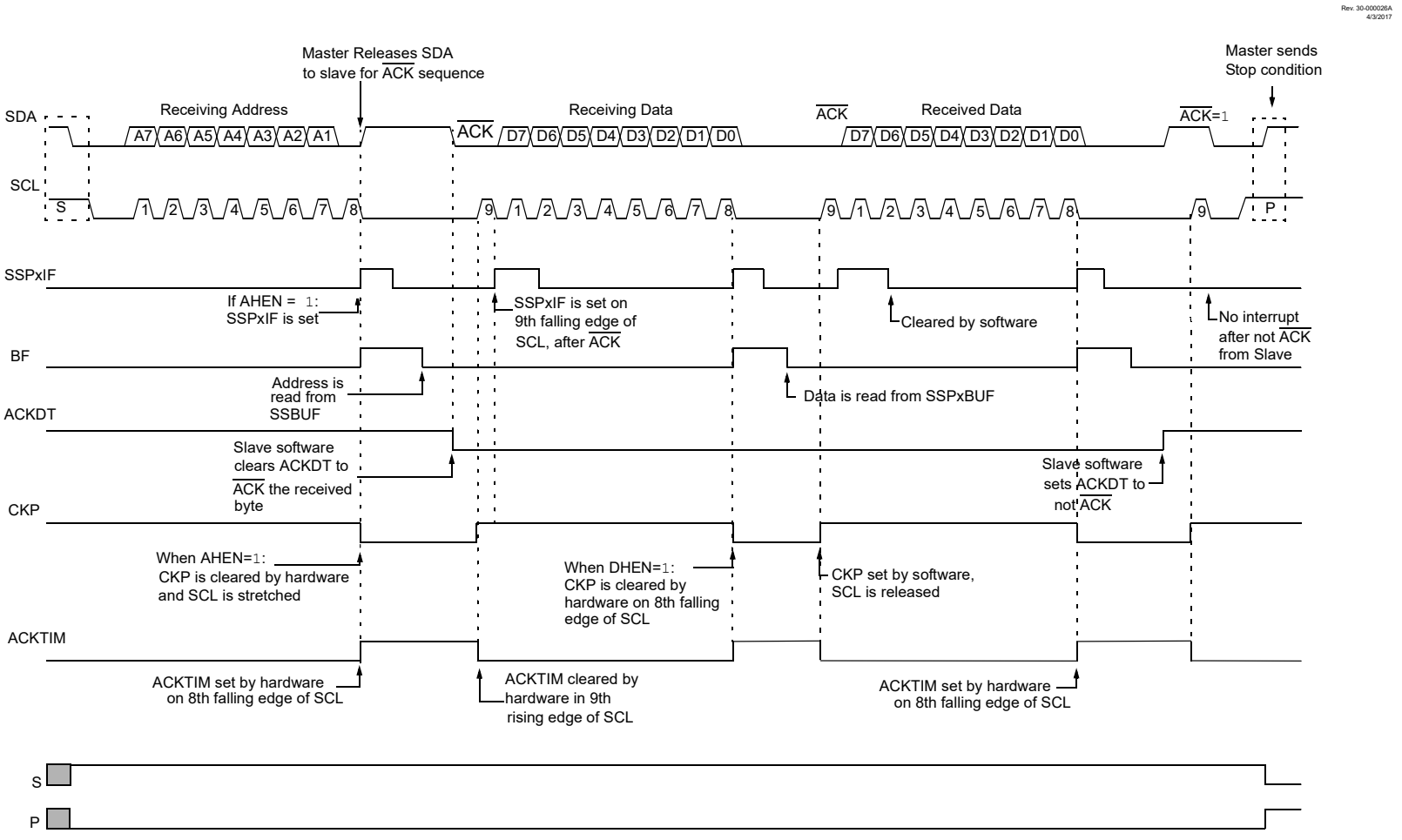
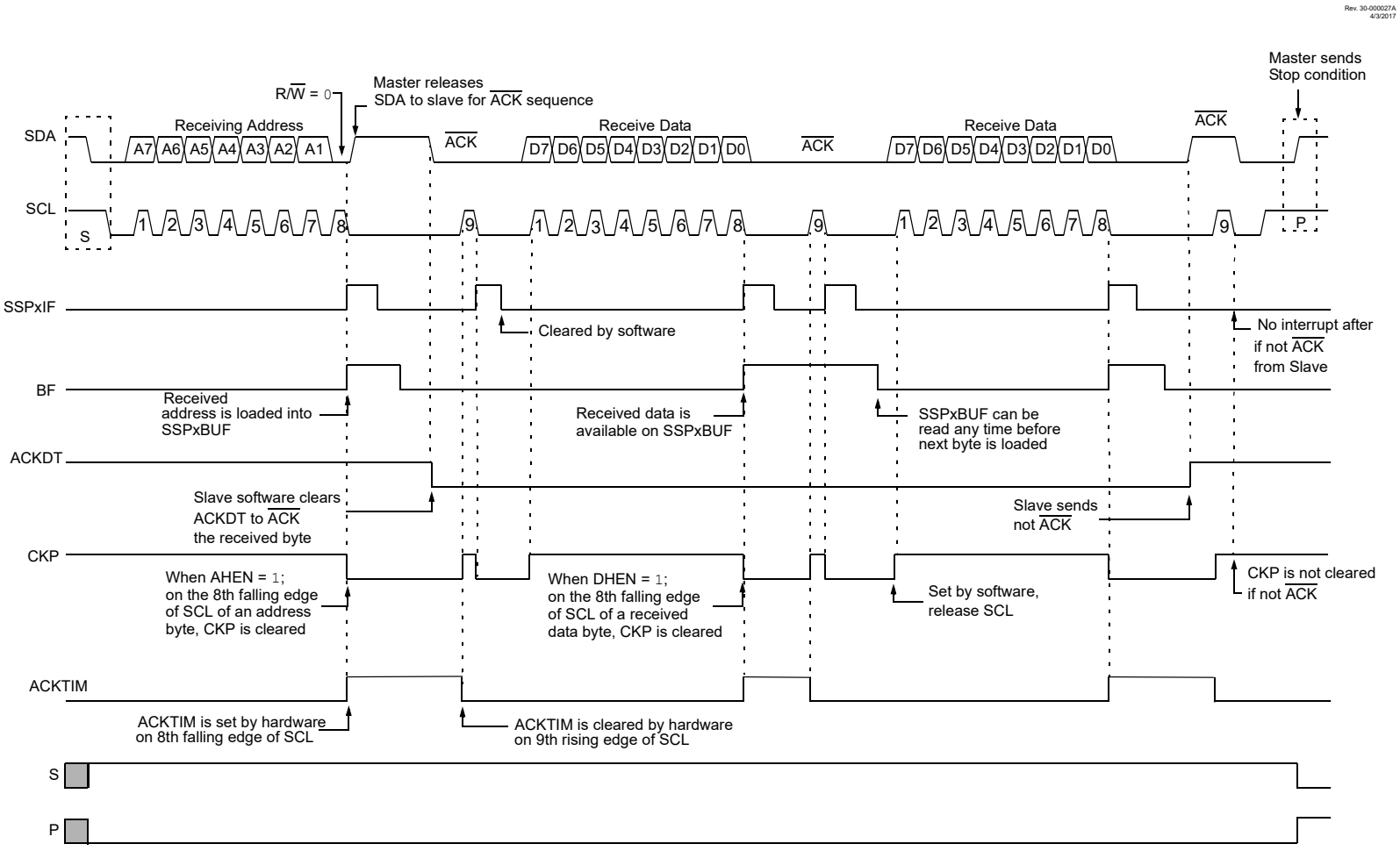


Figure 2-9. I²C Slave, 7-bit Address, Reception (SEN = 1, AHEN = 1, DHEN = 1)



2.2.6.3 Slave Mode 10-bit Address Reception

This section describes a standard sequence of events for the MSSP module configured as an I^2C slave in 10-bit Addressing mode. Figure 2-10 shows a standard waveform for a slave receiver in 10-bit addressing mode with clock stretching enabled.

This is a step-by-step process of what must be done by the slave software to accomplish I^2C communication.

1. Bus starts Idle.
2. Master sends Start condition; S bit is set; SSPxIF is set if SCIE is set.
3. Master sends matching high address with the R/$\overline{W}$ bit clear; the UA bit is set.
4. Slave sends $\overline{ACK}$ and SSPxIF is set.
5. Software clears the SSPxIF bit.
6. Software reads received address from SSPxBUF, clearing the BF flag.
7. Slave loads low address into SSPxADD, releasing SCL.
8. Master sends matching low address byte to the slave; UA bit is set.

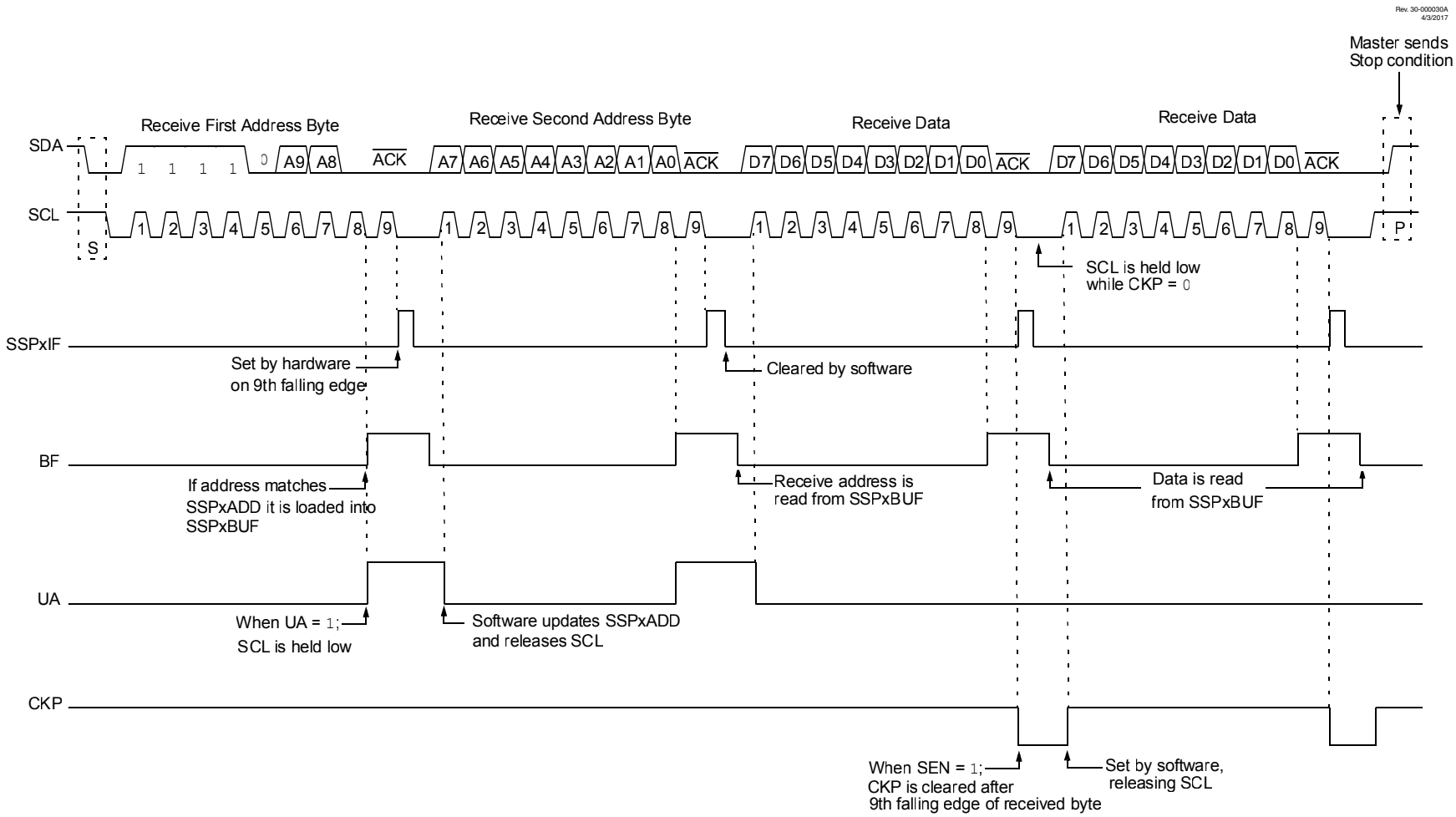
Important: Updates to the SSPxADD register are not allowed until after the $\overline{ACK}$ sequence.

9. Slave sends $\overline{ACK}$ and SSPxIF is set.

Important: If the low address does not match, SSPxIF and UA are still set so that the slave software can set SSPxADD back to the high address. BF is not set because there is no match. CKP is unaffected.

10. Slave clears SSPxIF.
11. Slave reads the received matching address from SSPxBUF, clearing BF.
12. Slave loads high address into SSPxADD.
13. Master clocks a data byte to the slave and clocks out the slaves $\overline{ACK}$ on the ninth SCL pulse; SSPxIF is set.
14. If the SEN bit is set, CKP is cleared by hardware and the clock is stretched.
15. Slave clears SSPxIF.
16. Slave reads the received byte from SSPxBUF, clearing BF.
17. If SEN is set, the slave software sets CKP to release the SCL.
18. Steps 13-17 repeat for each received byte.
19. Master sends Stop to end the transmission.

Figure 2-10. I²C Slave, 10-bit Address, Reception (SEN = 1, AHEN = 0, DHEN = 0)



2.2.6.4 10-bit Addressing with Address or Data Hold

Reception using 10-bit addressing with AHEN or DHEN set is the same as with 7-bit modes. The only difference is the need to update the SSPxADD register using the UA bit. All functionality, specifically when the bit is cleared and the SCL line is held low, is the same. Figure 2-11 can be used as a reference of a slave in 10-bit addressing with AHEN set.

Figure 2-12 shows a standard waveform for a slave transmitter in 10-bit Addressing mode.

Figure 2-11. I²C Slave, 10-bit Address, Reception (SEN = 0, AHEN = 1, DHEN = 0)

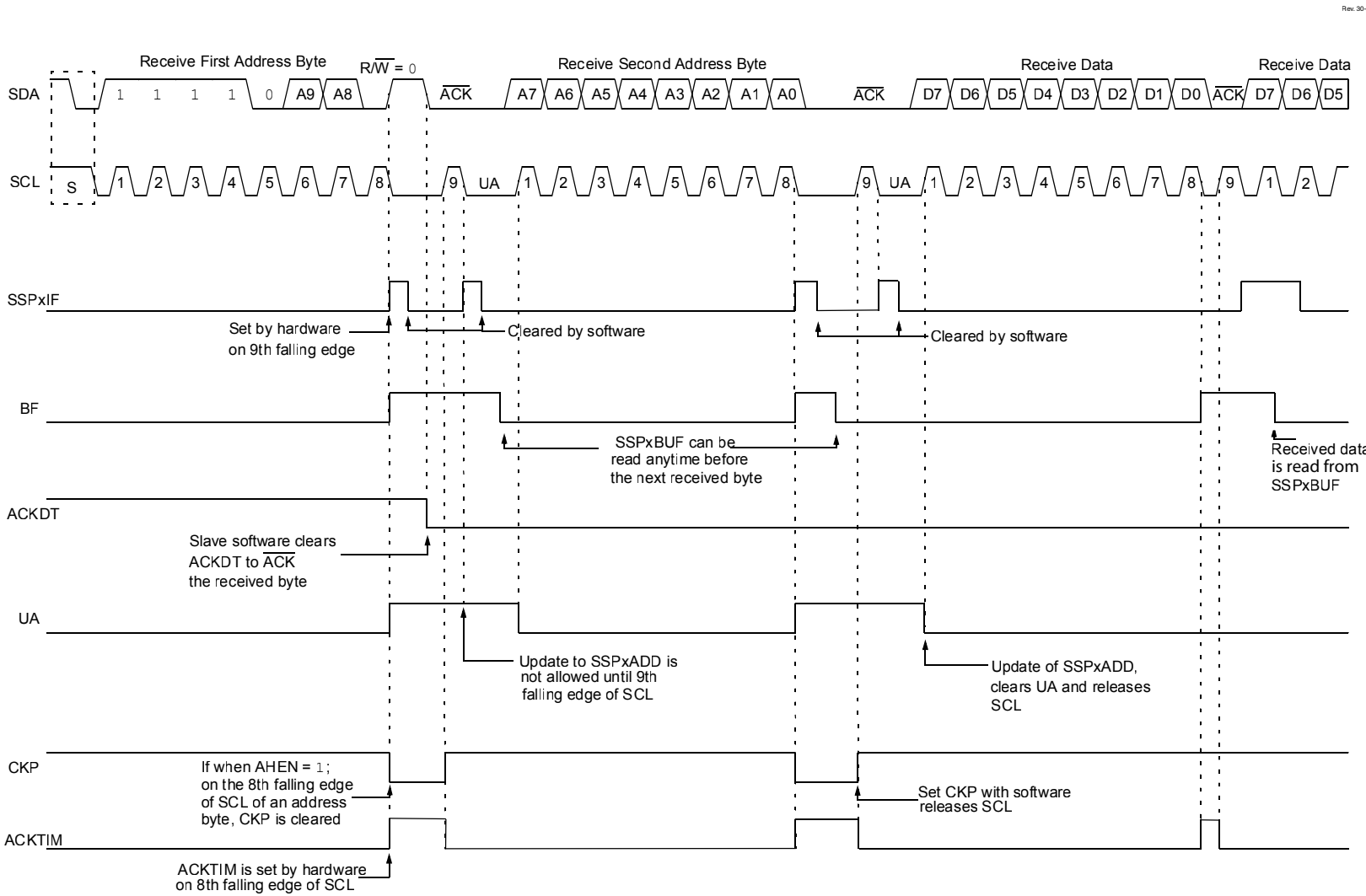
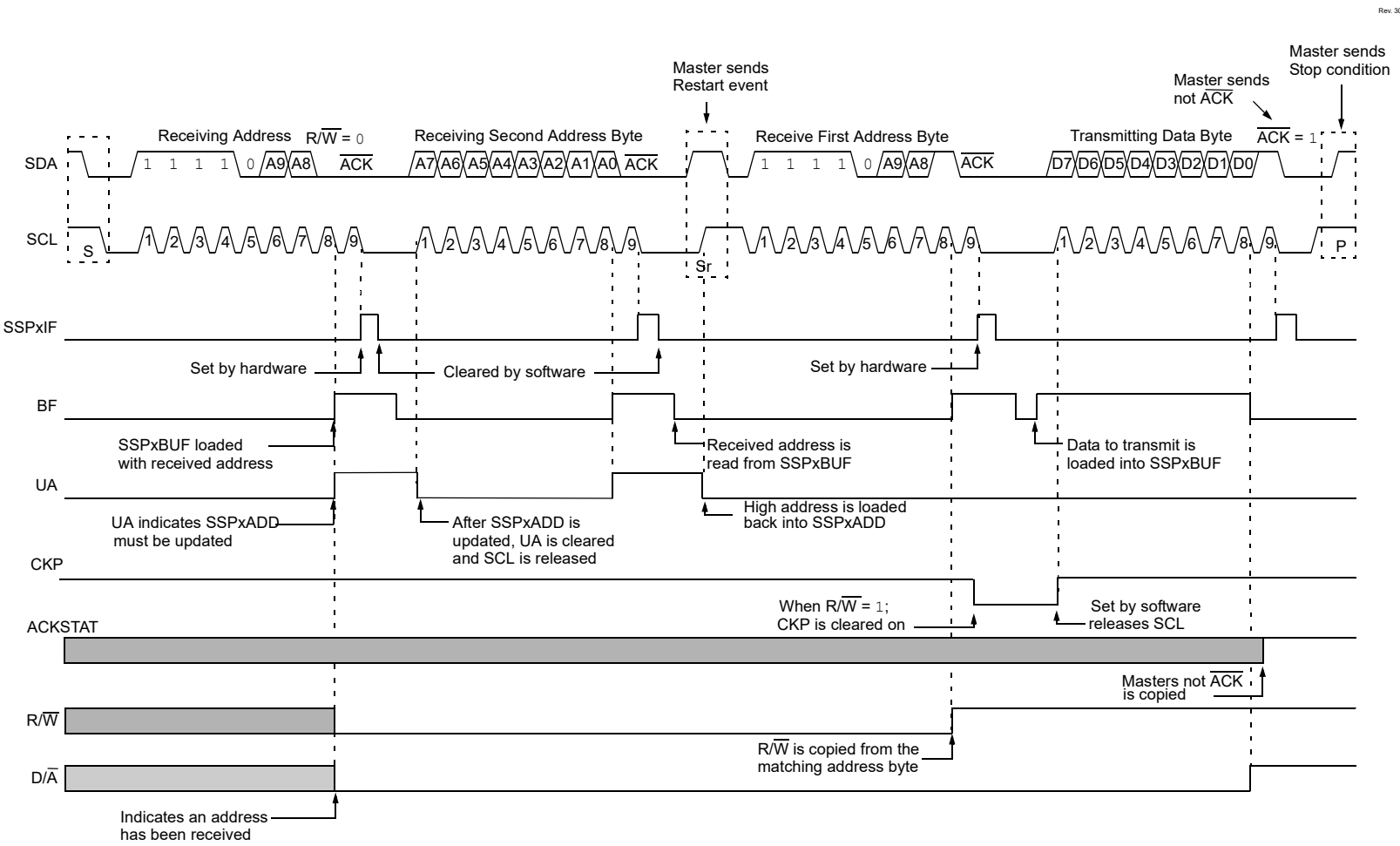


Figure 2-12. I²C Slave, 10-bit Address, Transmission (SEN = 0, AHEN = 0, DHEN = 0)



2.2.7 Slave Transmission

When the R/$\overline{W}$ bit of the incoming address byte is set and an address match occurs, the R/$\overline{W}$ bit is set. The received address is loaded into the SSPxBUF register, and an $\overline{ACK}$ pulse is sent by the slave on the ninth bit.

Following the $\overline{ACK}$, slave hardware clears the CKP bit and the SCL pin is held low (see "Clock Stretching" for more details). By stretching the clock, the master will be unable to assert another clock pulse until the slave is done preparing the transmit data.

The transmit data must be loaded into the SSPxBUF register, which also loads the SSPSR register. Then the SCL pin should be released by setting the CKP bit. The eight data bits are shifted out on the falling edge of the SCL input. This ensures that the SDA signal is valid during the SCL high time.

The $\overline{ACK}$ pulse from the master receiver is latched on the rising edge of the ninth SCL input pulse. This $\overline{ACK}$ value is copied to the ACKSTAT bit. If ACKSTAT is set (NACK), then the data transfer is complete. In this case, when the NACK is latched by the slave, the slave goes Idle and waits for another occurrence of a Start condition. If the SDA line was low ($\overline{ACK}$), the next transmit data must be loaded into the SSPxBUF register. Again, the SCL pin must be released by setting bit CKP.

An MSSP interrupt is generated for each data transfer byte. The SSPxIF bit must be cleared by software and the SSPxSTAT register is used to determine the status of the byte. The SSPxIF bit is set on the falling edge of the ninth clock pulse.

2.2.7.1 Slave Mode Bus Collision

A slave receives a read request and begins shifting data out on the SDA line. If a bus collision is detected and the Slave Mode Bus Collision Detect Enable (SBCDE) bit is set, the Bus Collision Interrupt Flag (BCLxIF) bit of the PIRx register is set. Once a bus collision is detected, the slave goes Idle and waits to be addressed again. User software can use the BCLxIF bit to handle a slave bus collision.

2.2.7.2 7-bit Transmission

A master device can transmit a read request to a slave, and then clock data out of the slave. The list below outlines what software for a slave will need to do to accomplish a standard transmission. Figure 2-13 can be used as a reference to this list.

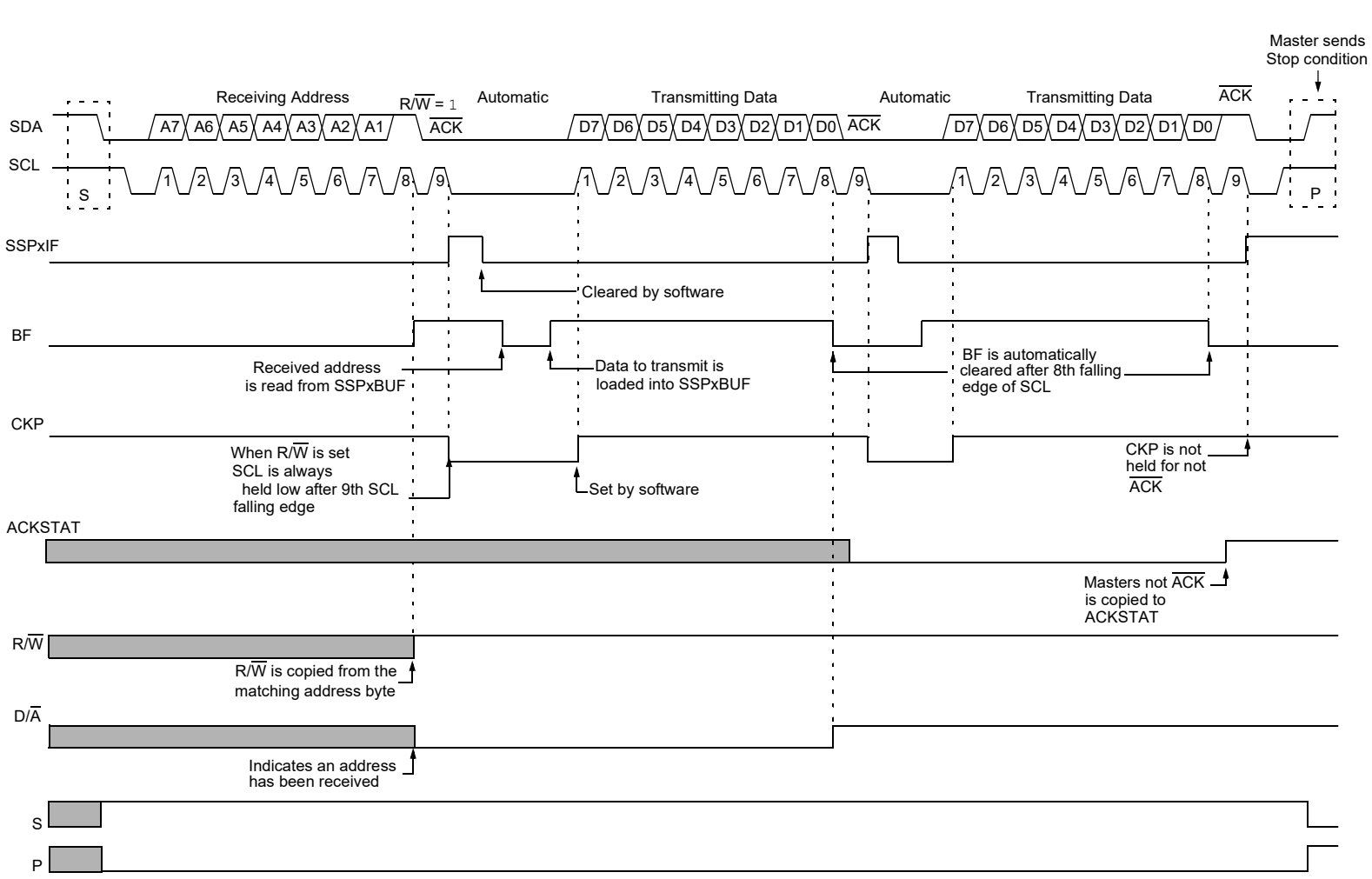
1. Master sends a Start condition.
2. The Start (S) bit is set; SSPxIF is set if SCIE is set.
3. Matching address with R/$\overline{W}$ bit set is received by the Slave, setting SSPxIF bit.
4. Slave hardware generates an $\overline{ACK}$ and sets SSPxIF.
5. The SSPxIF bit is cleared by software.
6. Software reads the received address from SSPxBUF, clearing BF.
7. R/$\overline{W}$ is set so CKP was automatically cleared after the $\overline{ACK}$.
8. The slave software loads the transmit data into SSPxBUF.
9. CKP bit is set by software, releasing SCL, allowing the master to clock the data out of the slave.
10. SSPxIF is set after the $\overline{ACK}$ response from the master is loaded into the ACKSTAT bit.
11. SSPxIF bit is cleared.
12. The slave software checks the ACKSTAT bit to see if the master wants to clock out more data.

Important:

1. If the master $\overline{ACK}$s then the clock will be stretched.
2. ACKSTAT is the only bit updated on the rising edge of the ninth SCL clock instead of the falling edge.

13. Steps 9-13 are repeated for each transmitted byte.
14. If the master sends a not $\overline{ACK}$; the clock is not held, but SSPxIF is still set.
15. The master sends a Restart condition or a Stop.

Figure 2-13. I²C Slave, 7-bit Address, Transmission (AHEN = 0)



2.2.7.3 7-bit Transmission with Address Hold Enabled

Setting the AHEN bit enables additional clock stretching and interrupt generation after the eighth falling edge of a received matching address. Once a matching address has been clocked in, CKP is cleared and the SSPxIF interrupt is set.

Figure 2-14 displays a standard waveform of a 7-bit address slave transmission with AHEN enabled.

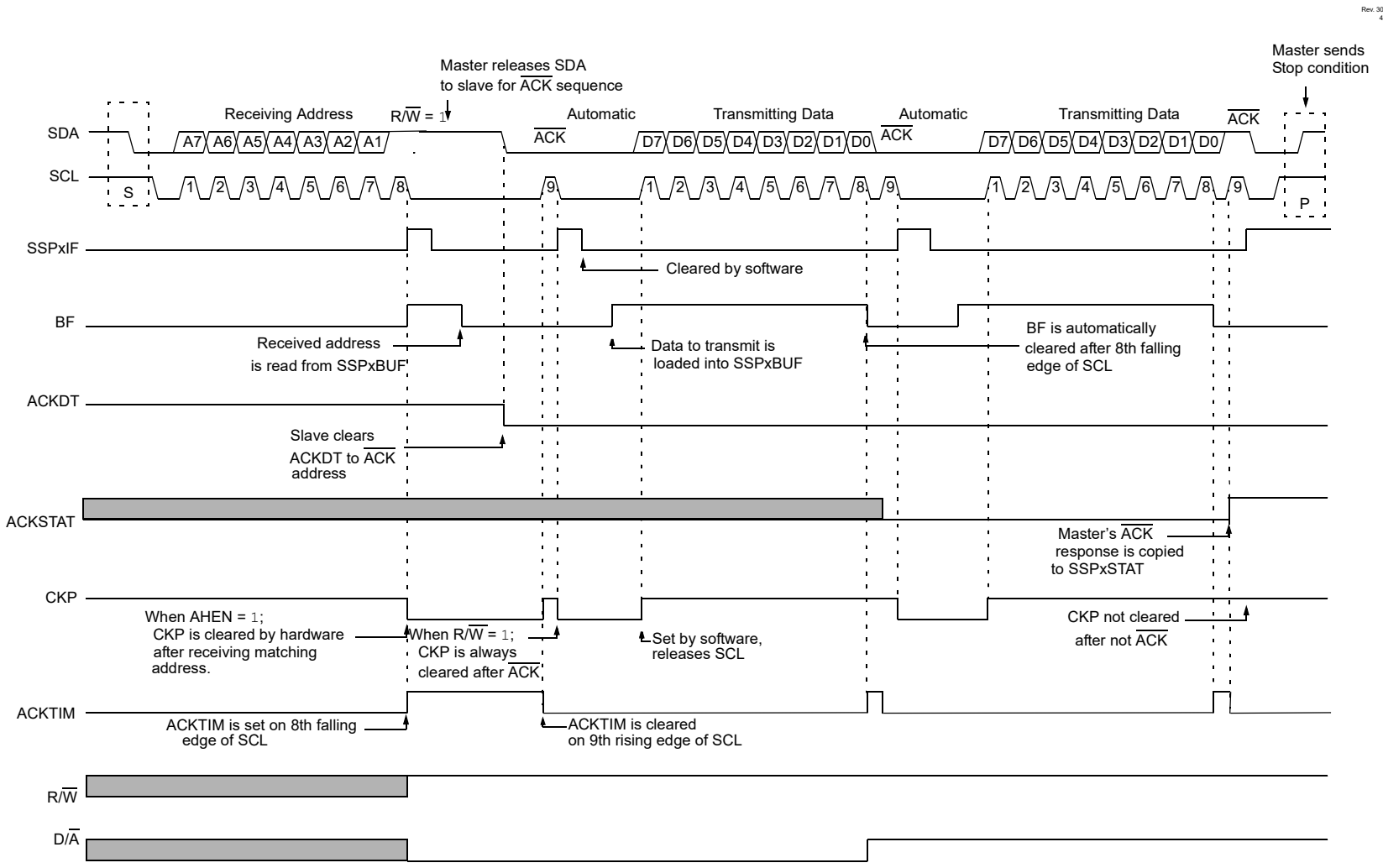
1. Bus starts Idle.
2. Master sends Start condition; the S bit is set; SSPxIF is set if SCIE is set.
3. Master sends matching address with the R/$\overline{W}$ bit set. After the eighth falling edge of the SCL line the CKP bit is cleared and SSPxIF interrupt is generated.
4. Slave software clears SSPxIF.
5. Slave software reads the ACKTIM, R/$\overline{W}$ and D/$\overline{A}$ bits to determine the source of the interrupt.
6. Slave reads the address value from the SSPxBUF register, clearing the BF bit.
7. Slave software decides from this information if it wishes to $\overline{ACK}$ or NACK and sets the ACKDT bit accordingly.
8. Slave software sets the CKP bit, releasing SCL.
9. Master clocks in the $\overline{ACK}$ value from the slave.
10. Slave hardware automatically clears the CKP bit and sets SSPxIF after the $\overline{ACK}$, if the R/$\overline{W}$ bit is set.
11. Slave software clears SSPxIF.
12. Slave loads value to transmit to the master into SSPxBUF, setting the BF bit.

Important: SSPxBUF cannot be loaded until after the $\overline{ACK}$.

13. Slave software sets the CKP bit, releasing the clock.
14. Master clocks out the data from the slave and sends an $\overline{ACK}$ value on the ninth SCL pulse.
15. Slave hardware copies the $\overline{ACK}$ value into the ACKSTAT bit.
16. Steps 10-15 are repeated for each byte transmitted to the master from the slave.
17. If the master sends a not $\overline{ACK}$, the slave releases the bus allowing the master to send a Stop and end the communication.

Important: Master must send a not $\overline{ACK}$ on the last byte to ensure that the slave releases the SCL line to receive a Stop.

Figure 2-14. I²C Slave Waveform (Transmission, 7-bit, AHEN = 1)

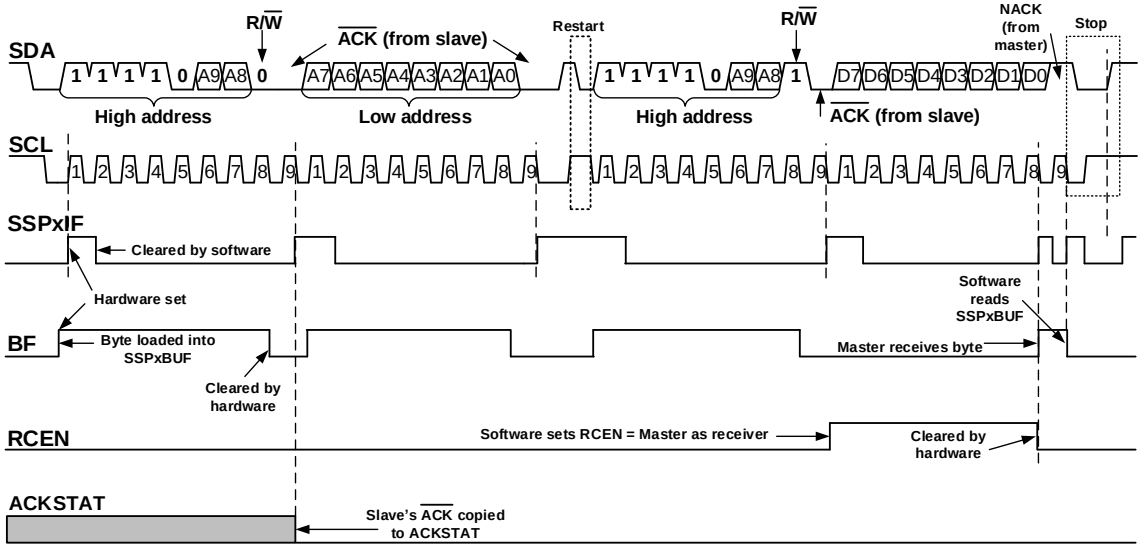


2.2.7.4 10-bit Transmission

The list below outlines the steps for 10-bit transmission in Slave mode:

1. Master issues a Start condition.
2. The Start (S) bit is set, SSPxIF is set; if SSPxIE is set, jump to ISR.
3. Software clears SSPxIF.
4. Slave receives upper address byte of 10-bit address with R/$\overline{W}$ bit clear, hardware sets BF and UA, and clears D/$\overline{A}$. Hardware stretches the clock.
5. Slave hardware compares the received address byte with the address loaded into SSPxADD.
6. Slave transmits $\overline{ACK}$ sequence. If address does not match, slave transmits a NACK sequence.
7. Slave hardware sets SSPxIF.
8. Slave software reads SSPxBUF, hardware clears BF.
9. Software clears SSPxIF.
10. Software loads the lower 10-bit address into SSPxADD, clearing UA and releasing SCL.
11. Slave receives lower address byte, sets BF and UA.
12. Hardware compares the received address byte to the address loaded into SSPxADD.
13. Slave transmits $\overline{ACK}$ sequence, sets SSPxIF.
14. Software reads SSPxBUF, hardware clears BF.
15. Software clears SSPxIF.
16. Software loads SSPxADD with the high byte of the 10-bit address, hardware clears UA.
17. Master issues a Restart condition and transmits high byte of 10-bit address with R/$\overline{W}$ set.
18. Slave receives the high byte of the 10-bit address with the R/$\overline{W}$ bit set, hardware sets BF and R/$\overline{W}$.
19. Hardware compares the received address byte to the address loaded into SSPxADD.
20. Software reads SSPxBUF, clearing BF.
21. Hardware issues an $\overline{ACK}$ sequence, sets SSPxIF.
22. After the 9th falling SCL edge, hardware clears CKP (clock stretch), allowing software to load SSPxBUF with data to transmit, setting BF.
23. Software sets CKP, releasing the clock.
24. Slave transmits full byte, hardware clears BF and sets D/$\overline{A}$.
25. Slave receives $\overline{ACK}$ from master, hardware clears ACKSTAT and sets SSPxIF.
26. Software clears SSPxIF.
27. Slave repeats steps 22 - 26 until all bytes have been transmitted.
28. Slave receives Stop condition, setting the P bit.

Figure 2-15. I²C Slave Mode Waveform (Transmission, 10-bit Address)



3. Slave Mode Code Example

Example 3-1 shows the use of the MSSP in 7-bit I²C Slave mode. The example uses an Interrupt Service Routine (ISR) to determine whether the master intends to read data from or write data to the slave, and react accordingly. A PICkit™ Serial Analyzer tool, which includes a graphical user interface (GUI), was used as the master device for simplicity. The GUI allows a user to read data from or write data to specific address locations within the slave.

Example 3-1. MSSP in I²C Slave Mode

```
#define ARRAY_CNT        32               // Number of bytes in array

uint8_t slaveAddress = 0x30;                    // 7-bit slave address
uint8_t index =           0;                    // Array pointer
uint8_t temp =            0;                    // Temp register
uint8_t regAdd =          1;                    // First data byte was reg add

uint8_t i2cArray[ARRAY_CNT] =
{0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0xFA,
0xEA, 0xDA, 0xCA, 0xBA, 0xFB, 0xFC, 0xFD, 0xFE,
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

void I2C_Initialize(void)
{
    SSP1STATbits.SMP = 1;                       // Disable slew control
    SSP1CON1bits.SSPM = 0b0110;                 // 7-bit slave mode
    SSP1CON2bits.SEN = 1;                       // Enable clock stretching
    SSP1CON3bits.SBCDE = 1;                     // Enable BCLIF
    SSP1ADD = slaveAddress;                     // Load slave address
    SSP1CON1bits.SSPEN = 1;                     // Enable the module

    PIR3bits.BCL1IF = 0;                        // Clear Bus Collision IF
    PIR3bits.SSP1IF = 0;                        // Clear SSP interrupt flag
    PIE3bits.BCL1IE = 1;                        // Enable BCLIF
    PIE3bits.SSP1IE = 1;                        // Enable SSPIF
    INTCONbits.PEIE = 1;                        // Enable periph interrupts
    INTCONbits.GIE = 1;                         // Enable global interrupts
}

void __interrupt() ISR(void)
{
    if(PIR3bits.SSP1IF)                         // Check for SSPIF
    {
        if(SSP1STATbits.R_nW == 1)              // Master read (slave transmit)
        {
            SSP1BUF = i2cArray[index++];        // Load array value
            SSP1CON1bits.CKP = 1;               // Release clock stretch
        }
        if(SSP1STATbits.R_nW == 0)              // Master write (slave receive)
        {
            if(SSP1STATbits.D_nA == 0)          // Last byte was an address
            {
                regAdd = 1;                     // Next byte register address
                temp = SSP1BUF;                 // Clear BF
                SSP1CON1bits.CKP = 1;           // Release clock stretch
            }
            if(SSP1STATbits.D_nA == 1)          // Last byte was data
            {
                if(regAdd == 1)                 // Last byte was register add
                {
                    index = SSP1BUF;            // Load register address
                    regAdd = 0;                 // Next byte will be true data
                }
                else
                {
                    if(index < ARRAY_CNT)               // Within boundaries?
                    {
                        i2cArray[index++] = SSP1BUF;    // Yes, read SSP1BUF
                    }
                    else
                    {
                        temp = SSP1BUF;                 // No, discard data
```

```
                    }
                }
                SSP1CON1bits.CKP = 1;                   // Release clock stretch
            }
        }
    }
    if(PIR3bits.BCL1IF == 1)
    {
        temp = SSP1BUF;                                 // Clear BF
        PIR3bits.BCL1IF = 0;                            // Clear BCLIF
        SSP1CON1bits.CKP = 1;                           // Release clock stretching
    }
    PIR3bits.SSP1IF = 0;                                // Clear SSP1IF
}
```

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 Slave 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: 978-1-5224-5878-4

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