
Using the PIC[®] Devices' SSP and MSSP Modules for Slave I²C[™] Communication

*Author: Stephen Bowling and Naveen Raj
Microchip Technology Inc.*

INTRODUCTION

Many of the PIC[®] microcontroller devices have a Synchronous Serial Port (SSP) or Master Synchronous Serial Port (MSSP). These peripherals can be used to implement the SPI or I²C[™] communication protocols. The purpose of this application note is to provide the reader with a better understanding of the I²C protocol and to show how devices with these modules are used as a slave device on an I²C bus.

For more information on the I²C bus specification or the SSP and MSSP peripherals, you may refer to sources indicated in the "References" section.

THE I²C BUS SPECIFICATION

Although a complete discussion of the I²C bus specification is outside the scope of this application note, some of the basics will be covered here. The Inter-Integrated-Circuit, or I²C bus specification, was originally developed by Philips Inc. for the transfer of data between ICs at the PCB level. The physical interface for the bus consists of two open-collector lines; one for the clock (SCL) and one for data (SDA). The bus may have a one master/many slave configuration or may have multiple master devices. The master device is responsible for generating the clock source for the linked slave devices.

The I²C protocol supports either a 7-Bit Addressing mode, or a 10-Bit Addressing mode, permitting up to 128 or 1024 physical devices to be on the bus, respectively. In practice, the bus specification reserves certain addresses, so slightly fewer usable addresses are available. For example, the 7-Bit Addressing mode allows 112 usable addresses.

All data transfers on the bus are initiated by the master device, which always generates the clock signal on the bus. Data transfers are performed on the bus, eight bits at a time, MSb first. There is no limit to the amount of data that can be sent in one transfer.

The I²C protocol includes a handshaking mechanism. After each 8-bit transfer, a 9th clock pulse is sent by the master. At this time, the transmitting device on the bus releases the SDA line and the receiving device on the bus Acknowledges the data sent by the transmitting device. An ACK (SDA held low) is sent if the data was received successfully, or a NACK (SDA left high) is sent if it was not received successfully.

All changes on the SDA line must occur while the SCL line is low. This restriction allows two unique conditions to be detected on the bus; a Start sequence (**S**) and a Stop sequence (**P**). A Start sequence occurs when the master pulls the SDA line low, while the SCL line is high. The Start sequence tells all slaves on the bus that address bytes are about to be sent. The Stop sequence occurs when the SDA line goes high while the SCL line is high, and it terminates the transmission. Slave devices on the bus should reset their receive logic after the Stop sequence has been detected.

The I²C protocol also permits a Repeated Start condition (**Rs**), which allows the master device on the bus to perform a Start sequence, without a Stop sequence preceding it. The Repeated Start allows the master device to start a new data transfer without releasing control of the bus.

A typical I²C write transmission would proceed as shown in Figure 1. In this example, the master device will write two bytes to a slave device. The transmission is started when the master initiates a Start condition on the bus. Next, the master sends an address byte to the slave. The upper seven bits of the address byte contain the slave address. The LSb of the address byte specifies whether the I²C operation will be a read (LSb = 1) or a write (LSb = 0). On the ninth clock pulse, the master releases the SDA line so the slave can Acknowledge the reception. If the address byte was received by the slave and was the correct address, the slave responds with an ACK by holding the SDA line low. Assuming an ACK was received, the master sends out the data bytes. On the ninth clock pulse, after each data byte, the slave responds with an ACK. After the last data byte, the master initiates the Stop condition to free the bus.

A read operation is performed similar to the write operation and is shown in Figure 2. In this case, the R/W bit in the address byte is set to indicate a read operation. After the address byte is received, the slave device sends an ACK pulse and holds the SCL line low (clock stretching). By holding the SCL line, the slave can take as much time as needed to prepare the data to be sent back to the master. When the slave is ready, it releases SCL and the master device clocks the data from the slave buffer. On the ninth clock pulse, the slave latches the value of the ACK bit received from the master. If an ACK pulse was received, the slave must prepare the next byte of data to be transmitted. If a NACK was received, the data transmission is complete. In this case, the slave device should wait for the next Start condition.

For many I²C peripherals, such as nonvolatile EEPROM memory, an I²C write operation and a read operation are done in succession. For example, the write operation specifies the address to be read and the read operation gets the byte of data. Since the master device does not release the bus after the memory address is written to the device, a Repeated Start sequence is performed to read the contents of the memory address.

THE SSP MODULE

A block diagram of the SSP module for I²C Slave mode is shown in Figure 3. Key control and status bits required for I²C slave communication are provided in the following Special Function Registers:

- SSPSTAT
- SSPCON
- PIR1 (interrupt flag bits)
- PIE1 (interrupt enable bits)

Some of the bit functions in these registers vary, depending on whether the SSP module is used for I²C or SPI communications. The functionality of each for I²C mode is described here. For a complete description of each bit function, refer to the appropriate device data sheet.

FIGURE 1: TYPICAL I²C™ WRITE TRANSMISSION (7-BIT ADDRESS)

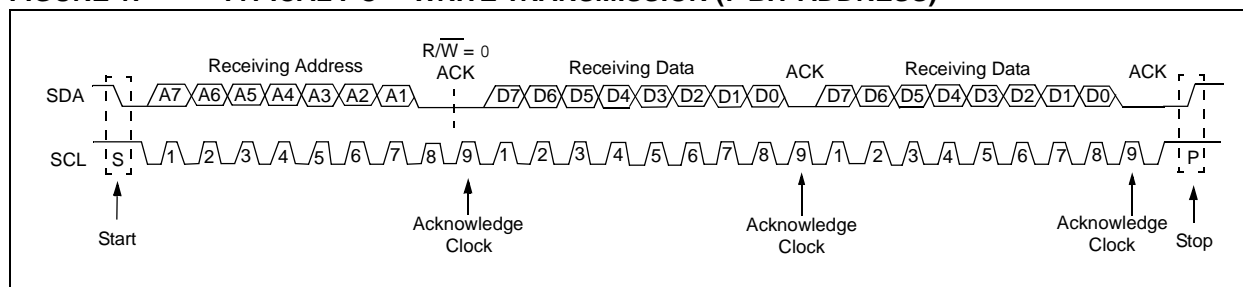


FIGURE 2: TYPICAL I²C™ READ TRANSMISSION (7-BIT ADDRESS)

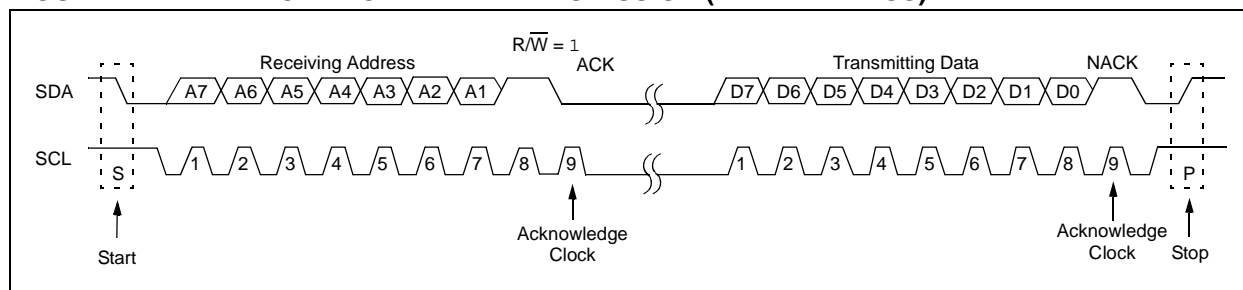
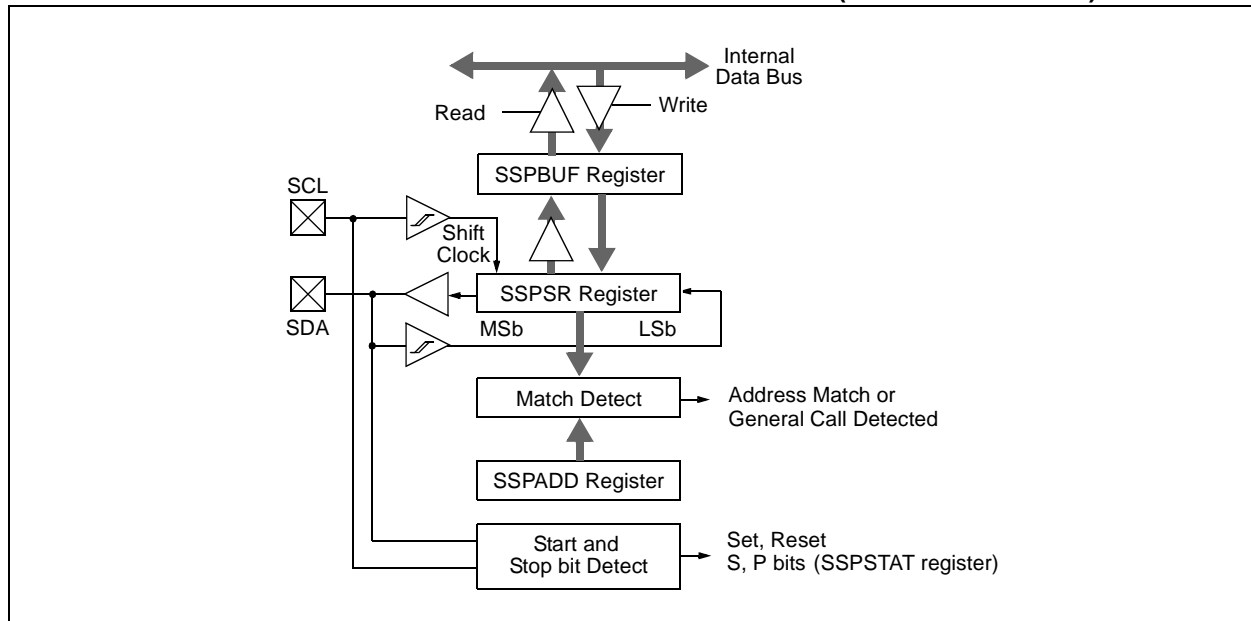


FIGURE 3: PIC® DEVICES' SSP MODULE BLOCK DIAGRAM (I²C™ SLAVE MODE)

SSP Bits that Indicate Module Status

BF (SSPSTAT<0>)

The BF (Buffer Full) bit tells the user whether a byte of data is currently in the SSP Buffer register, SSPBUF. This bit is cleared automatically when the SSPBUF register is read, or when a byte to be transmitted is completely shifted out of the register. The BF bit will become set under the following circumstances:

- When an address byte is received with the LSb cleared. This will be the first byte sent by the master device during an I²C write operation.
- Each time a data byte is received during an I²C write to the slave device.
- Each time a byte of data is written to SSPBUF to be transmitted to the master device. The BF bit will be cleared automatically when all bits have been shifted from SSPBUF to the master device.

There are certain cases where the BF flag will set when an address is received with the LSB set (read operation). Refer to **Appendix C: "Differences Between the I²C States in PIC16 and PIC18 Devices"**.

UA (SSPSTAT<1>)

The UA (Update Address) bit is used only in the 10-Bit Addressing modes. In the 10-Bit Addressing mode, an I²C slave address must be sent in two bytes. The upper half of the 10-bit address (1111 0 A9 A8 0) is first loaded into SSPADD for initial match detection. This particular address code is reserved in the I²C protocol for designating the upper half of a 10-bit address. When an address match occurs, the SSP module will

set the UA bit to indicate that the lower half of the address should be loaded into SSPADD for match detection.

R/W (SSPSTAT<2>)

The R/W (Read/Write) bit tells the user whether the master device is reading from, or writing to, the slave device. This bit reflects the state of the LSb in the address byte that is sent by the master. The state of the R/W bit is only valid for the duration of a particular I²C message and will be reset by a Stop condition, Start condition or a NACK from the master device.

S (SSPSTAT<3>)

The S (Start) bit is set if a Start condition occurred last on the bus. The state of this bit will be the inverse of the P (Stop) bit, except when the module is first initialized and both bits are cleared.

P (SSPSTAT<4>)

The P (Stop) bit is set if a Stop condition occurred last on the bus. The state of this bit will be the inverse of the S (Start) bit, except when the module is first initialized and both bits are cleared. The P bit can be used to determine when the bus is Idle.

D/A (SSPSTAT<5>)

The D/A (Data/Address) bit indicates whether the last byte of data received by the SSP module was a data byte or an address byte. For read operations, the last byte sent to the master device was a data byte when the D/A bit is set.

WCOL (SSPCON<7>)

The WCOL (Write Collision) bit indicates that SSPBUF was written while the previously written word is still transmitting. The previous contents of SSPBUF are not changed when the write collision occurs. The WCOL bit must be cleared in software.

SSPOV (SSPCON<6>)

The SSPOV (SSP Overflow) bit indicates that a new byte was received while SSPBUF was still holding the previous data. In this case, the SSP module will not generate an ACK pulse and SSPBUF will not be updated with the new data. Regardless of whether the data is to be used, the user must read SSPBUF whenever the BF bit becomes set, to avoid an SSP overflow condition. The user must read SSPBUF and clear the SSPOV bit to properly clear an overflow condition. If the user reads SSPBUF to clear the BF bit, but does not clear the SSPOV bit, the next byte of data received will be loaded into SSPBUF but the module will not generate an ACK pulse.

SSPIF (PIR1<3>)

The SSPIF (SSP Interrupt Flag) bit indicates that an I²C event has completed. The user must poll the status bits described here to determine what event occurred and the next action to be taken. The SSPIF bit must be cleared by the user.

SSP Bits for Module Control

SSPEN (SSPCON<5>)

The SSPEN (SSP Enable) bit enables the SSP module and configures the appropriate I/O pins as serial port pins.

CKE (SSPSTAT<6>)

The CKE (Clock Edge) bit has no function when the SSP module is configured for I²C mode and should be cleared.

SMP (SSPSTAT<7>)

The SMP (Sample Phase) bit has no function when the SSP module is configured for I²C mode and should be cleared.

CKP (SSPCON<4>)

The CKP (Clock Polarity) bit is used for clock stretching in the I²C protocol. When the CKP bit is cleared, the slave device holds the SCL pin low so that the master device on the bus is unable to send clock pulses. During clock stretching, the master device will attempt to send clock pulses until the clock line is released by the slave device.

Clock stretching is useful when the slave device can not process incoming bytes quickly enough, or when SSPBUF needs to be loaded with data to be transmitted to the master device. The SSP module performs

clock stretching automatically when data is read by the master device. The CKP bit will be cleared by the module after the address byte and each subsequent data byte is read. After SSPBUF is loaded, the CKP bit must be set in software to release the clock and allow the next byte to be transferred.

SSPM3:SSPM0 (SSPCON<3:0>)

The SSPM3:SSPM0 (SSP mode) bits are used to configure the SSP module for the SPI or I²C protocols. For specific values, refer to the appropriate device data sheet.

SSPIE (PIE1<3>)

The SSPIE (SSP Interrupt Enable) bit enables SSP interrupts. The appropriate global and peripheral interrupt enable bits must be set in conjunction with this bit to allow interrupts to occur.

Configuring the SSP for I²C Slave Mode

Before enabling the module, ensure that the pins used for SCL and SDA are configured as inputs by setting the appropriate TRIS bits. This allows the module to configure and drive the I/O pins as required by the I²C protocol.

The SSP module is configured and enabled using the SSPCON register. The SSP module can be configured for the following I²C Slave modes:

- I²C Slave mode, 7-bit address
- I²C Slave mode, 10-bit address
- I²C Slave mode, 7-bit address, Start and Stop interrupts enabled
- I²C Slave mode, 10-bit address, Start and Stop interrupts enabled

Of these four modes of operation, the first two are most commonly used in a slave device application. The second two modes provide interrupts when Start and Stop conditions are detected on the bus and are useful for detecting when the I²C bus is Idle. After the bus is detected Idle, the slave device could become a master device on the bus. Since there is no hardware support for master I²C communications in the SSP module, the master communication would need to be implemented in firmware.

SETTING THE SLAVE ADDRESS

The address of the slave node must be written to the SSPADD register (see Figure 3). For 7-Bit Addressing mode, bits<7:1> determine the slave address value. The LSB of the address byte is not used for address matching; this bit determines whether the transaction on the bus will be a read or write. Therefore, the value written to SSPADD will always have an even value (LSb = 0). Effectively, each slave node uses two addresses; one for write operations and another for read operations.

Handling SSP Events in Software

Using the SSP module for slave I²C communication is, in general, a sequential process that requires the firmware to perform some action after each I²C event. The SSPIF bit indicates an I²C event on the bus has completed. The SSPIF bit may be polled in software or can be configured as an interrupt source. Each time the SSPIF bit is set, the I²C event must be identified by testing various bits in the SSPSTAT register.

For the purposes of explanation, it is helpful to identify all the possible states and discuss each one individually. There are a total of five valid states for the SSP module after an I²C event; these are described below.

The SSP module does not buffer events, so the cause of each I²C event must be determined as each new SSPIF interrupt occurs. As each event causes an interrupt, the code examines the various important I²C bits in the SSPSTAT register to determine what has just happened on the I²C bus, and determine which state the module is in. The code examples in **Appendix A: “Example Slave I²C Source Code”** and **Appendix B: “Example Slave I²C Source Code (Modified for Newer PIC18 Devices)”** show how this is done.

STATE 1: MASTER WRITE, LAST BYTE WAS AN ADDRESS

The master device on the bus has begun a new write operation by initiating a Start or Restart condition on the bus, then sending the slave I²C address byte. The LSb of the address byte is '0' to indicate that the master wishes to write data to the slave. The bits in the SSPSTAT register will have the following values:

- $S = 1$ (Start condition occurred last)
- $R/\overline{W} = 0$ (Master writing data to the slave)
- $D/\overline{A} = 0$ (Last byte was an address)
- $BF = 1$ (The buffer is full)

At this time, the SSP buffer is full and holds the previously sent address byte. The SSPBUF register must be read at this time to clear the BF bit, even if the address byte is to be discarded. If the SSPBUF is not read, the subsequent byte sent by the master will cause an SSP overflow to occur and the SSP module will NACK the byte.

STATE 2: MASTER WRITE, LAST BYTE WAS DATA

After the address byte is sent for an I²C write operation (State 1), the master may write one or more data bytes to the slave device. If SSPBUF was not full prior to the write, the slave node SSP module will generate an ACK pulse on the 9th clock edge. Otherwise, the SSPOV bit will be set and the SSP module will NACK the byte. The bits in the SSPSTAT register will have the following values after the master writes a byte of data to the slave:

- $S = 1$ (Start condition occurred last)
- $R/\overline{W} = 0$ (Master writing data to the slave)
- $D/\overline{A} = 1$ (Last byte was a data byte)
- $BF = 1$ (The buffer is full)

STATE 3: MASTER READ, LAST BYTE WAS AN ADDRESS

The master device on the bus has begun a new read operation by initiating a Start or a Restart condition on the bus, then sending the slave I²C address byte. The LSb of the address byte is '1' to indicate that the master wishes to read data from the slave. The bits in the SSPSTAT register will have the following values:

- $S = 1$ (Start condition occurred last)
- $R/\overline{W} = 1$ (Master reading data from the slave)
- $D/\overline{A} = 0$ (Last byte was an address)

At this time, the SSP buffer is ready to be loaded with data to be sent to the master. The CKP bit is also cleared to hold the SCL line low. The slave data is sent to the master by loading SSPBUF and then setting the CKP bit to release the SCL line.

STATE 4: MASTER READ, LAST BYTE WAS DATA

State 4 occurs each time the master has previously read a byte of data from the slave and wishes to read another data byte. The bits in the SSPSTAT register will have the following values:

- $S = 1$ (Start condition occurred last)
- $R/\overline{W} = 1$ (Master reading data from the slave)
- $D/\overline{A} = 1$ (Last byte sent was a data byte)
- $BF = 0$ (The buffer is empty)

At this time, the SSP buffer is ready to be loaded with data to be sent to the master. The CKP bit is also cleared to hold the SCL line low. The slave data is sent to the master by loading SSPBUF and then setting the CKP bit to release the SCL line.

STATE 5: MASTER NACK

State 5 occurs when the master has sent a NACK in response to data that has been received from the slave device. This action indicates that the master does not wish to read further bytes from the slave. The NACK signals the end of the I²C message and has the effect of resetting the slave I²C logic. The bits in the SSPSTAT register will have the following values:

- $S = 1$ (Start condition occurred last)
- $D/\overline{A} = 1$ (Last byte sent was a data byte)
- $BF = 0$ (The buffer is empty)
- $CKP = 1$ (Clock is released)

The NACK event is identified because the CKP bit remains set. Specifically, the status bits indicate that a data byte has been received from the master and the buffer is empty.

SSP Error Handling

Each time SSPBUF is read in the slave firmware, the user should check the SSPOV bit to ensure that no reception overflows have occurred. If an overflow occurred, the SSPOV bit must be cleared in software and SSPBUF must be read for further byte receptions to take place.

The action that is performed after a SSP overflow will depend on the application. The slave logic will NACK the master device when an overflow occurs. In a typical application, the master may try to resend the data until an ACK from the slave is detected.

After writing data to SSPBUF, the user should check the WCOL bit to ensure that a write collision did not occur. In practice, there will be no write collisions if the application firmware only writes to SSPBUF during states when the BF bit is cleared and the slave device is transmitting data to the master.

SOURCE CODE EXAMPLE

The current revision of this document includes two separate source code listings to implement the basic I²C slave functions described previously. The source code provided in **Appendix A: “Example Slave I²C Source Code”** is written in Microchip assembly language and will operate on any device in the PIC16 family of devices that has a SSP or MSSP module. The code in **Appendix B: “Example Slave I²C Source Code (Modified for Newer PIC18 Devices)”** is also written in assembly, and is designed to run on newer PIC18 family devices with the updated I²C state machine. **Appendix C: “Differences Between the I²C States in PIC16 and PIC18 Devices”** provides more information on identifying devices with the newer state machine.

The code examples are simple applications that receive characters transmitted by a master device and store them in a data buffer. At the beginning of each new write operation by the master, the buffer contents are cleared when the master sends the address of the slave to do the write operation. When the master device begins a new read, the characters in the buffer will be returned. With minor modifications, the source code provided can be adapted to most applications that require I²C communications.

Each of the five I²C states discussed in this document are identified by XORing the bits in the SSPSTAT register with predetermined mask values. Once the state has been identified, the appropriate action is taken. All undefined states are handled by branching execution to a software trap.

I²C ACRONYMS

ACK: Acknowledge
BRG: Baud Rate Generator
BSSP: Basic Synchronous Serial Port
F/W: Firmware
I²C: Inter-Integrated Circuit
ISR: Interrupt Service Routine
MCU: Microcontroller Unit
MSSP: Master Synchronous Serial Port
NACK: Not Acknowledge
SDA: Serial Data Line
SCL: Serial Clock Line
SSP: Synchronous Serial Port

REFERENCES

The I²C™ Bus Specification, Philips Semiconductor, Version 2.1, 2000,
<http://www-us.semiconductors.com/i2c/>

PIC® Mid-Range MCU Family Reference Manual, Microchip Technology Inc., Document Number DS33023

AN735, “Using the PICmicro® MSSP Module for Master I²C™ Communications”, Microchip Technology Inc., Document Number DS00735A

AN578, “Use of the SSP Module in the I²C™ Multi-Master Environment”, Microchip Technology Inc., Document Number DS00578B

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX A: EXAMPLE SLAVE I²C SOURCE CODE

```

;-----
; File: an734.asm
;
; Written By: Stephen Bowling, Microchip Technology
;
; Version: 1.00
;
; Assembled using Microchip Assembler
;
; Functionality:
;
; This code implements the basic functions for an I2C slave device
; using the SSP module. All I2C functions are handled in an ISR.
; Bytes written to the slave are stored in a buffer. After a number
; of bytes have been written, the master device can then read the
; bytes back from the buffer.
;
; Variables and Constants used in the program:
;
; The start address for the receive buffer is stored in the variable
; 'RXBuffer'. The length of the buffer is denoted by the constant
; value 'RX_BUF_LEN'. The current buffer index is stored in the
; variable 'Index'.
;
;-----
;
; The following files should be included in the MPLAB project:
;
; an734.asm-- Main source code file
;
; 16f877a.lkr-- Linker script file
; (change this file for the device you are using)
;
;-----
;-----
; Include Files
;-----

#include <p16f877a.inc> ; Change to device that you are using.

```

AN734

```
;-----
;Constant Definitions
;-----

#define NODE_ADDR    0x22        ; I2C address of this node
                                   ; Change this value to address that
                                   ; you wish to use.

;-----
; Buffer Length Definition
;-----

#define RX_BUF_LEN   32          ; Length of receive buffer

;-----
; Variable declarations
;-----
    udata

WREGsave    res    1
STATUSsave  res    1
FSRsave     res    1
PCLATHsave  res    1

Index       res    1            ; Index to receive buffer
Temp        res    1            ;
RXBuffer     res    RX_BUF_LEN  ; Holds rec'd bytes from master device.

;-----
; Vectors
;-----

STARTUP code
    nop
    goto    Startup            ;
    nop                    ; 0x0002
    nop                    ; 0x0003
    goto    ISR                ; 0x0004

PROG code

;-----
; Macros
;-----

memset macro  Buf_addr,Value,Length

        movlw Length          ; This macro loads a range of data memory
        movwf Temp            ; with a specified value. The starting
        movlw Buf_addr        ; address and number of bytes are also
        movwf FSR              ; specified.
SetNext  movlw Value
        movwf INDF
        incf  FSR,F
        decfsz Temp,F
        goto  SetNext
    endm

LFSR  macro  Address,Offset    ; This macro loads the correct value
```



```

        movlw  Address      ; into the FSR given an initial data
        movwf  FSR          ; memory address and offset value.
        movf   Offset,W
        addwf  FSR,F
    endm

;-----
; Main Code
;-----

Startup
    bcf      STATUS,RP1
    bsf      STATUS,RP0
    call     Setup
    banksel  WREGsave
Main      clrwdt             ; Clear the watchdog timer.
    goto    Main           ; Loop forever.

;-----
; Interrupt Code
;-----

ISR
    movwf    WREGsave      ; Save WREG
    movf     STATUS,W      ; Get STATUS register
    banksel  STATUSsave    ; Switch banks, if needed.
    movwf    STATUSsave    ; Save the STATUS register
    movf     PCLATH,W;
    movwf    PCLATHsave    ; Save PCLATH
    movf     FSR,W         ;
    movwf    FSRsave       ; Save FSR

    banksel  PIR1
    btfss    PIR1,SSPIF    ; Is this a SSP interrupt?
    goto     $             ; No, just trap here.
    bcf      PIR1,SSPIF
    call     SSP_Handler   ; Yes, service SSP interrupt.

    banksel  FSRsave
    movf     FSRsave,W     ;
    movwf    FSR           ; Restore FSR
    movf     PCLATHsave,W  ;
    movwf    PCLATH        ; Restore PCLATH
    movf     STATUSsave,W  ;
    movwf    STATUS        ; Restore STATUS
    swapf    WREGsave,F    ;
    swapf    WREGsave,W    ; Restore WREG
    retfie                       ; Return from interrupt.

```

```
;-----
Setup
;
; Initializes program variables and peripheral registers.
;-----

        banksel    PCON
        bsf        PCON,NOT_POR
        bsf        PCON,NOT_BOR
        banksel    Index          ; Clear various program variables
        clrf       Index
        clrf       PORTB
        clrf       PIR1
        banksel    TRISB
        clrf       TRISB
        movlw      0x36           ; Setup SSP module for 7-bit
        banksel    SSPCON
        movwf      SSPCON        ; address, slave mode
        movlw      NODE_ADDR
        banksel    SSPADD
        movwf      SSPADD
        clrf       SSPSTAT
        banksel    PIE1          ; Enable interrupts
        bsf        PIE1,SSPIE
        bsf        INTCON,PEIE   ; Enable all peripheral interrupts
        bsf        INTCON,GIE    ; Enable global interrupts
        bcf        STATUS,RP0
        return

;-----
SSP_Handler
;-----
; The I2C code below checks for 5 states:
;-----
; State 1: I2C write operation, last byte was an address byte.
; SSPSTAT bits: S = 1, D_A = 0, R_W = 0, BF = 1
;
; State 2: I2C write operation, last byte was a data byte.
; SSPSTAT bits: S = 1, D_A = 1, R_W = 0, BF = 1
;
; State 3: I2C read operation, last byte was an address byte.
; SSPSTAT bits: S = 1, D_A = 0, R_W = 1 (see Appendix C for more information)
;
; State 4: I2C read operation, last byte was a data byte.
; SSPSTAT bits: S = 1, D_A = 1, R_W = 1, BF = 0
;
; State 5: Slave I2C logic reset by NACK from master.
; SSPSTAT bits: S = 1, D_A = 1, BF = 0, CKP = 1 (see Appendix C for more information)
;
; For convenience, WriteI2C and ReadI2C functions have been used.
;-----

        banksel    SSPSTAT
        movf       SSPSTAT,W      ; Get the value of SSPSTAT
        andlw      b' 00101101'  ; Mask out unimportant bits in SSPSTAT.
        banksel    Temp
        movwf      Temp           ; Put masked value in Temp
                                   ; for comparison checking.
```

```

State1:
    movlw    b'00001001'    ; Write operation, last byte was an
    xorwf    Temp,W          ; address, buffer is full.
    btfss    STATUS,Z        ; Are we in State1?
    goto     State2          ; No, check for next state.....
    memset    RXBuffer,0,RX_BUF_LEN ; Clear the receive buffer.
    clrf      Index          ; Clear the buffer index.
    bankssel SSPBUF          ; Do a dummy read of the SSPBUF.
    movf      SSPBUF,W
    return

State2:
    movlw    b'00101001'    ; Write operation, last byte was data,
    xorwf    Temp,W          ; buffer is full.
    btfss    STATUS,Z        ; Are we in State2?
    goto     State3          ; No, check for next state.....
    LFSR      RXBuffer,Index ; Point to the buffer.
    bankssel SSPBUF          ; Get the byte from the SSP.
    movf      SSPBUF,W
    movwf    INDF            ; Put it in the buffer.
    incf      Index,F        ; Increment the buffer pointer.
    movf      Index,W        ; Get the current buffer index.
    sublw    RX_BUF_LEN      ; Subtract the buffer length.
    btfsc     STATUS,Z        ; Has the index exceeded the buffer length?
    clrf      Index          ; Yes, clear the buffer index.
    return

State3:
    movf      Temp,W          ; Read operation, last byte was an address,
    andlw    b'00101100'     ; Mask BF bit in SSPSTAT
    xorlw    b'00001100'
    btfss    STATUS,Z        ; Are we in State3?
    goto     State4          ; No, check for next state.....
    clrf      Index          ; Clear the buffer index.
    LFSR      RXBuffer,Index ; Point to the buffer
    movf      INDF,W          ; Get the byte from buffer.
    call     WriteI2C         ; Write the byte to SSPBUF
    incf      Index,F        ; Increment the buffer index.
    return

State4:
    bankssel SSPCON          ; Read operation, last byte was data,
    btfsc     SSPCON, CKP     ; buffer is empty.
    goto     State5
    movlw    b'00101100'
    xorwf    Temp,W
    btfss    STATUS,Z        ; Are we in State4?
    goto     State5          ; No, check for next state....
    movf      Index,W        ; Get the current buffer index.
    sublw    RX_BUF_LEN      ; Subtract the buffer length.
    btfsc     STATUS,Z        ; Has the index exceeded the buffer length?
    clrf      Index          ; Yes, clear the buffer index.
    LFSR      RXBuffer,Index ; Point to the buffer
    movf      INDF,W          ; Get the byte
    call     WriteI2C         ; Write to SSPBUF
    incf      Index,F        ; Increment the buffer index.
    return

```

AN734

```
State5:
    movf    Temp,W        ; NACK received when sending data to the master
    andlw   b'00101000'   ; Mask RW bit in SSPSTAT
    xorlw   b'00101000'   ;
    btfss   STATUS,Z      ;
    goto    I2CErr        ;
    return                               ; If we aren't in State5, then something is
                                       ; wrong.

I2CErr    nop
    banksel PORTB         ; Something went wrong! Set LED
    bsf     PORTB,7       ; and loop forever. WDT will reset
    goto    $             ; device, if enabled.
    return

;-----
; WriteI2C
;-----

WriteI2C
    banksel SSPSTAT
    btfsc   SSPSTAT,BF    ; Is the buffer full?
    goto    WriteI2C      ; Yes, keep waiting.
    banksel SSPCON        ; No, continue.

DoI2CWrite
    bcf     SSPCON,WCOL    ; Clear the WCOL flag.
    movwf   SSPBUF        ; Write the byte in WREG
    btfsc   SSPCON,WCOL    ; Was there a write collision?
    goto    DoI2CWrite
    bsf     SSPCON,CKP     ; Release the clock.
    return
end
```

APPENDIX B: EXAMPLE SLAVE I²C SOURCE CODE (MODIFIED FOR NEWER PIC18 DEVICES)

```

;-----
; File: an734_PIC18.asm
;
; The following files should be included in the MPLAB project:
;;
; an734_PIC18.asm-- Main source code file
;;
; 18F8722.lkr-- Linker script file
; (change this file for the device you are using)
;
;-----

#define    RX_BUF_LEN 32
ADDRESS    equ 0x22
udata      0x00
FSRsave     res 1
PCLATHsave  res 1
Index       res 1
Temp        res 1
RXBuffer    res RX_BUF_LEN
;-----
; Include Files
;-----
#include<p18F8722.inc>
CONFIG OSC = HS,FCMEN = OFF,IESO = OFF,PWRT = OFF,BOREN = OFF
CONFIG WDT = OFF
CONFIG STVREN = OFF, LVP = OFF,XINST = OFF,DEBUG = OFF
CONFIG CP0 = OFF,CP1 = OFF,CP2 = OFF,CP3 = OFF,CPB = OFF

memset macro Buf_addr,Value,Length
    movlw Length                ; This macro loads a range of data memory
    movwf Temp                  ; with a specified value. The starting
    movlw Buf_addr              ; address and number of bytes are also
    movwf FSR0L                 ; specified.
SetNext
    movlw Value
    movwf INDF0
    incf FSR0L,F
    decfsz Temp,F
    goto SetNext
endm

load macro Address,Offset      ; This macro loads the correct value
    movlw Address              ; into the FSR given an initial data
    movwf FSR0L                ; memory address and offset value.
    movf Offset,W
    addwf FSR0L,F
endm

```

AN734

```
PRG    CODE    0x00
      goto    Start

INT1    CODE    0x08
      goto    Int
INT2    CODE    0x18
      goto    Int

MAIN    CODE    0x30
;-----
; Main Code
;-----

Start
      clrfs   Index           ;res 1
      clrfs   Temp            ;res 1
      clrfs   RXBuffer        ;res RX_BUF_LEN
      call    Setup

Main
      goto    Main

Setup
      bsf     TRISC,3
      bsf     TRISC,4
      clrfs   FSR0L
      clrfs   FSR0H
      movlw   ADDRESS          ;Load Address , Slave node
      movwf   SSP1ADD
      movlw   0x36
      movwf   SSP1CON1
      clrfs   SSP1STAT
      clrfs   SSP1CON2
      bsf     SSP1CON2,SEN      ;Enable Clock Stretching for both transmit and slave
      bcf     PIR1,SSPIF       ;Clear MSSP interrupt flag
      bsf     PIE1,SSPIE       ;Enable MSSP interrupt enable bit
      movlw   0xC0              ;Enable global and peripheral Interrupt
      movwf   INTCON
      return

;-----
; Interrupt Code
;-----

Int
      movf    FSR0L,W           ;
      movwf   FSRsave          ; Save FSR

      btfss   PIR1,SSPIF       ; Is this a SSP interrupt?
      goto    $                ; No, just trap here.
      bcf     PIR1,SSPIF
      call    SSP_Handler      ; Yes, service SSP interrupt.

      movf    FSRsave,W        ;
      movwf   FSR0L           ; Restore FSR

      bsf     SSPCON1,CKP       ; Release clock( for transmit and receive)
      retfie  FAST              ; Return from interrupt
```

```

;-----
; State 1: I2C write operation, last byte was an address byte
; SSPSTAT bits: S = 1, D_A = 0, R_W = 0, BF = 1
;
; State 2: I2C write operation, last byte was a data byte
; SSPSTAT bits: S = 1, D_A = 1, R_W = 0, BF = 1
;
; State 3: I2C read operation, last byte was an address byte
; SSPSTAT bits: S = 1, D_A = 0, R_W = 1 (see Appendix C for more information)
;
; State 4: I2C read operation, last byte was a data byte
; SSPSTAT bits: S = 1, D_A = 1, R_W = 1, BF = 0
;
; State 5: Slave I2C logic reset by NACK from master
; SSPSTAT bits: S = 1, D_A = 1, BF = 0, CKP = 1 (see Appendix C for more information)
; For convenience, WriteI2C and ReadI2C functions have been used.
;-----
SSP_Handler
    movf    SSPSTAT,W                ; Get the value of SSPSTAT
    andlw   b'00101101'             ; Mask out unimportant bits in SSPSTAT.
    movwf   Temp                    ; for comparision checking.

State1:
    movlw   b'00001001'             ; Write operation, last byte was an
    xorwf   Temp,W                  ; address, buffer is full.
    btfss   STATUS,Z                ; Are we in State1?
    goto    State2                  ; No, check for next state.....
    memset  RXBuffer,0,RX_BUF_LEN   ; Clear the receive buffer.
    clrf    Index                   ; Clear the buffer index.
    movf    SSPBUF,W                ; Do a dummy read of the SSPBUF.
    return

State2:
    movlw   b'00101001'             ; Write operation, last byte was data,
    xorwf   Temp,W                  ; buffer is full.
    btfss   STATUS,Z                ; Are we in State2?
    goto    State3                  ; No, check for next state.....
    load    RXBuffer,Index           ; Point to the buffer.
    movf    SSPBUF,W                ; Get the byte from the SSP.
    movwf   INDF0                   ; Put it in the buffer.
    incf    Index,F                 ; Increment the buffer pointer.
    movf    Index,W                 ; Get the current buffer index.
    sublw   RX_BUF_LEN              ; Subtract the buffer length.
    btfsc   STATUS,Z                ; Has the index exceeded the buffer length?
    clrf    Index
    return

State3:
    movf    Temp,W                  ;
    andlw   b'00101100'             ; Mask BF bit in SSPSTAT
    xorlw   b'00001100'             ;
    btfss   STATUS,Z                ; Are we in State3?
    goto    State4                  ; No, check for next state.....
    movf    SSPBUF,W                ;
    clrf    Index                   ; Clear the buffer index.
    load    RXBuffer,Index           ; Point to the buffer
    movf    INDF0,W                 ; Get the byte from buffer.
    call    WriteI2C                ; Write the byte to SSPBUF
    incf    Index,F                 ; Increment the buffer index.
    return

```

State4

```
    btfsc  SSPCON1,CKP          ;
    goto   State5
    movlw  b'00101100'         ; buffer is empty.
    xorwf  Temp,W
    btfss  STATUS,Z             ; Are we in State4?
    goto   State5              ; No, check for next state....
    movf   Index,W             ; Get the current buffer index.
    sublw  RX_BUF_LEN          ; Subtract the buffer length.
    btfsc  STATUS,Z            ; Has the index exceeded the buffer length?
    clrf   Index               ; Yes, clear the buffer index.
    load   RXBuffer,Index      ; Point to the buffer
    movf   INDF0,W             ; Get the byte
    call   WriteI2C            ; Write to SSPBUF
    incf   Index,F             ; Increment the buffer index.
    return
```

State5

```
    movf   Temp,W              ;
    andlw  b'00101000'         ; Mask RW bit in SSPSTAT
    xorlw  b'00101000'
    btfss  STATUS,Z            ; Are we in State5?
    goto   I2Cerr              ; No, check for next state....
    return
```

I2Cerr

```
    nop                          ; Something went wrong! Set LED
    bsf    PORTB,7               ; and loop forever. WDT will reset
    goto   $                     ; device, if enabled.
```

; WriteI2C

WriteI2C

```
    btfsc  SSPSTAT,BF          ; Is the buffer full?
    goto   WriteI2C            ; Yes, keep waiting.
```

DoI2CWrite

```
    bcf    SSPCON1,WCOL        ; Clear the WCOL flag.
    movwf  SSPBUF              ; Write the byte in WREG
    btfsc  SSPCON1,WCOL        ; Was there a write collision?
    goto   DoI2CWrite
    return
end
```


APPENDIX C: DIFFERENCES BETWEEN THE I²C STATES IN PIC16 AND PIC18 DEVICES

This application note and its accompanying code (**Appendix A: “Example Slave I²C Source Code”**) were originally written to describe the implementation of I²C slave operations in PIC16 devices. This revision (August 2008) updates the description to make it compatible with PIC18 devices. The original document defined the five states of the I²C state machine, in terms of SSPSTAT status bits, as follows:

- State 1: (Write operation, last byte is an address byte)
 - $S = 1$
 - $D/\overline{A} = 0$
 - $R/\overline{W} = 0$
 - $BF = 1$
- State 2: (Write operation, last byte is a data byte)
 - $S = 1$
 - $D/\overline{A} = 1$
 - $R/\overline{W} = 0$
 - $BF = 1$
- State 3: (Read operation, last byte is an address byte)
 - $S = 1$
 - $D/\overline{A} = 0$
 - $R/\overline{W} = 1$
 - $BF = 0$
- State 4: (Read operation, last byte is a data byte)
 - $S = 1$
 - $D/\overline{A} = 1$
 - $R/\overline{W} = 1$
 - $BF = 0$
- State 5: (Logic reset by NACK from master)
 - $S = 1$
 - $D/\overline{A} = 1$
 - $R/\overline{W} = 0$
 - $BF = 0$

Older PIC18 devices, as defined in **Section C.1 “Older PIC18 Devices with the PIC16 State Machine”**, implement the I²C state machine with the same bit definitions as previously described.

Later PIC18 devices implement with these changes in States 3 and 5:

- State 3: In PIC16 and older PIC18 devices, the BF flag is not set. In newer PIC18 devices, the BF flag is set and needs to be read and cleared for State 3.
- State 5: In PIC16 and older PIC18 devices, the R/\overline{W} flag is expected to be cleared. In newer PIC18 devices, R/\overline{W} remains set. Instead of testing this bit, the state machine tests the CKP bit, expecting it to be set.

C.1 Older PIC18 Devices with the PIC16 State Machine

These PIC18 family devices use I²C state machines that behave the same as PIC16 devices:

- PIC18C452 Family (PIC18C242/252/442/452)
- PIC18C458 Family (PIC18C248/258/448/458)
- PIC18C601/801
- PIC18F4431 Family (PIC18F2231/2431/4231/4431)
- PIC18F8720 Family (PIC18F6520/6620/6720/8520/8620/8720)
- PIC18F1220/1320

Any PIC18 device not explicitly listed here uses the I²C state machine with the updated definitions of States 3 and 5.

NOTES:

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

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

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

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

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC³² logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

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

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta

Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

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

Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara

Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto

Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office

Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney

Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu

Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR

Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing

Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao

Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai

Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang

Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen

Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan

Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen

Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian

Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai

Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore

Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi

Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune

Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama

Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu

Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul

Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang

Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila

Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore

Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu

Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung

Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei

Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok

Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels

Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen

Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris

Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich

Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan

Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen

Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid

Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham

Tel: 44-118-921-5869
Fax: 44-118-921-5820