Software SPI Examples for the C8051F30x Family

Introduction

This application note is a collection of routines that can be used to implement a master-mode SPI device in software. Eight different examples of a SPI master-mode transfer are provided. The examples contain two functions for each of the SPI clock phase and polarity options: an example written in 'C', and for increased speed, an assembly routine that can be called from 'C'. An example of how to call the routines from a 'C' program and an EEPROM interface example are also provided. SPI is a trademark of Motorola Inc.

The SPI functions described in this document are written to minimize the software overhead required for the SPI transaction. They are intended for use in a system where the C8051F30x device is the only SPI master on the bus.

Hardware Interface

These examples implement the SPI interface using three GPIO pins for transactions:

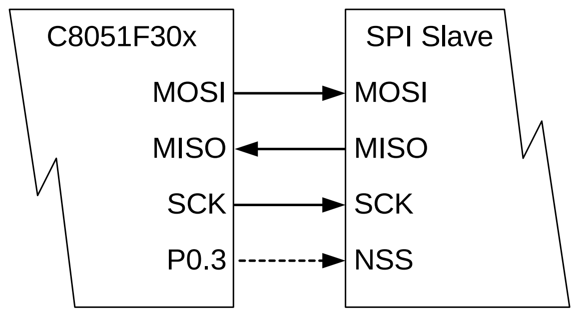
MOSI (Master Out / Slave In): This pin is used for serial data output from the C8051F30x, and should be configured as a digital push-pull output.

MISO (Master In / Slave Out): This pin is used for serial data input from the slave device, and should be configured as an open-drain digital pin.

SCK (Serial Clock): This pin is used as the serial data clock output from the C8051F30x device, and should be configured as a digital push-pull output.

Additionally, if a slave select signal is required for the slave device, a fourth GPIO pin is needed, and must be declared as a digital push-pull output for this purpose. All of the dedicated GPIO pins should be skipped by the crossbar. Figure 1 shows the connections between the SPI master (C8051F30x) and SPI slave devices.

Figure 1. Hardware Configuration

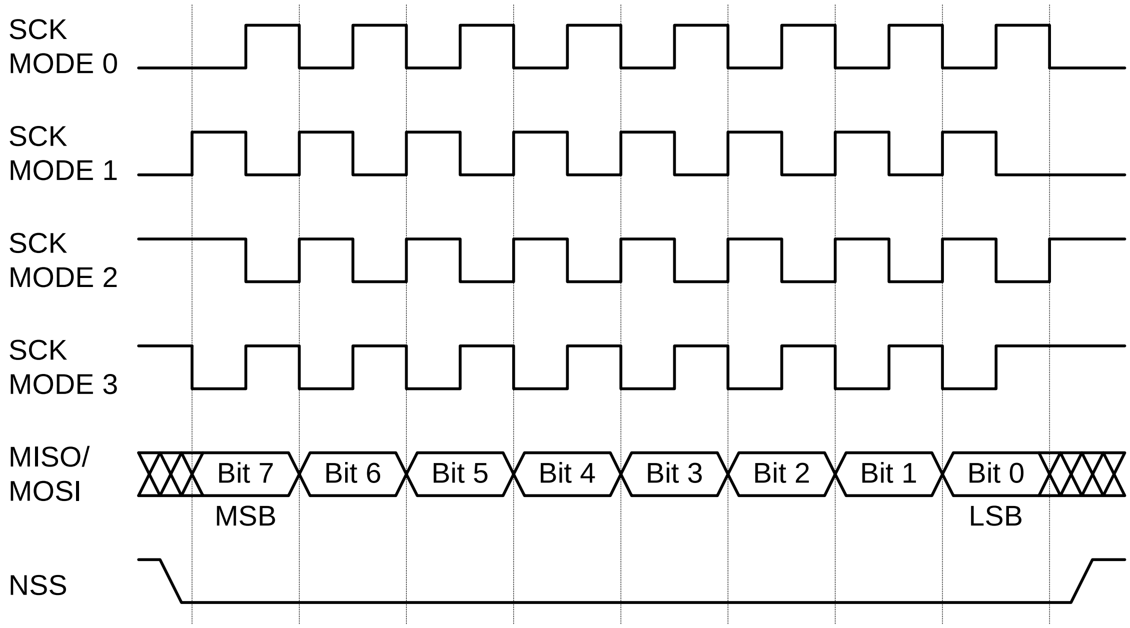


Function Descriptions

There are eight examples of a software SPI master contained in this applications note. Each of the four different SPI modes (Mode 0, Mode 1, Mode 2 and Mode 3) are given as examples in both 'C' and assembly. Table 1 lists the source files for each implementation. All of the routines can be called from 'C' using the same function prototype. Because of this, only one of the sample implementations should be included when building a project. The functions can be renamed if multiple SPI modes are needed in the same system.

The functions take a single character as an input parameter and return a single character. The input parameter is transmitted on the MOSI pin MSB-first. A byte is simultaneously received through the MISO pin MSB-first. The function returns the received data byte from MISO. SCK phase and

Figure 2. Serial Clock Phase / Polarity



polarity are determined by the SPI mode file that is included when building the project.

Table 1. SPI Function Source Files

Implementation	File Name
Mode 0 in ‘C’	SPI_MODE0.c
Mode 0 in Assembly	SPI_MODE0.asm
Mode 1 in ‘C’	SPI_MODE1.c
Mode 1 in Assembly	SPI_MODE1.asm
Mode 2 in ‘C’	SPI_MODE2.c
Mode 2 in Assembly	SPI_MODE2.asm
Mode 3 in ‘C’	SPI_MODE3.c
Mode 3 in Assembly	SPI_MODE3.asm

SPI Timing

It is important to ensure the timing specifications of the slave device are met when implementing a software SPI master. Because C8051F30x devices are capable of operating at high speeds, it is possible that the routines presented here may need to be modified to accommodate the timing restrictions of the SPI slave device. Each of the four SPI modes has a specific serial clock phase and polarity, as shown in Figure 2. In addition, the C and assembly routines have different timing specifications. Figure 3 shows the timing for Mode 0 and Mode 3. Figure 4 shows the timing for Mode 1 and Mode 2. Table 2 details the number of system clocks required for each timing parameter.

In a system that is running from a fast clock, any of the included routines may need to be slowed down or otherwise modified in order to meet the timing specifications for the SPI slave device.

Figure 3. Timing For Mode 0 and Mode 3

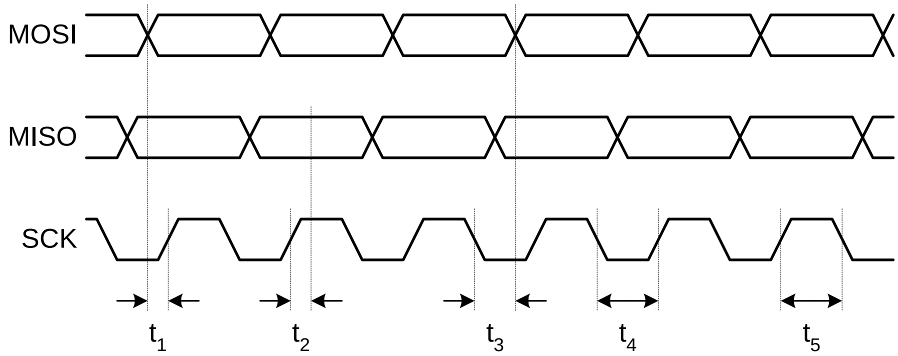


Figure 4. Timing For Mode 1 and Mode 2

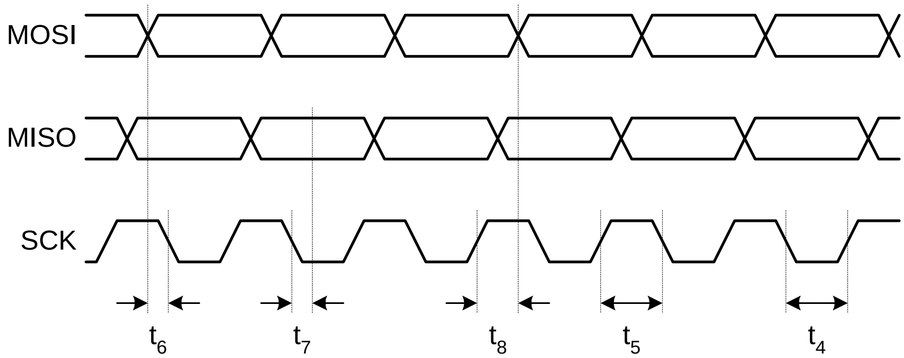


Table 2. SPI Timing Parameters

(Refer to Figure 3 and Figure 4 for Timing Diagram and Label Specifications)

Parameter	Description	SPI Mode	C Timing (SYSCLKs)	Assembly Timing (SYSCLKs)
t_1	MOSI Valid to SCK High (MOSI setup)	Mode 0 Mode 3	6 6	2 2
t_2	SCK High to MISO Latched	Mode 0 Mode 3	2 2	2 3
t_3	SCK Low to MOSI Change (MOSI hold)	Mode 0 Mode 3	7 4	5 2
t_4	SCK Low Time	Mode 0 Mode 1 Mode 2 Mode 3	13 11 8 10	7 7 5 5
t_5	SCK High Time	Mode 0 Mode 1 Mode 2 Mode 3	8 10 13 11	5 5 7 7

Table 2. SPI Timing Parameters

(Refer to Figure 3 and Figure 4 for Timing Diagram and Label Specifications)

Parameter	Description	SPI Mode	C Timing (SYSCLKs)	Assembly Timing (SYSCLKs)
t_6	MOSI Valid to SCK Low (MOSI setup)	Mode 1 Mode 2	6 6	2 2
t_7	SCK Low to MISO Latched	Mode 1 Mode 2	2 2	3 2
t_8	SCK High to MOSI Change (MOSI hold)	Mode 1 Mode 2	4 7	2 5
-	Function Call to Return From Function	All Modes	182	113

Using the Functions

To use one of the example SPI functions in a 'C' program, the file containing the function must first be assembled or compiled. The resulting object file then can be added to the build list for the project and linked to the host (calling) software.

The host software must correctly configure the GPIO pins of the C8051F30x device to the desired functionality. See "Hardware Interface" on page 1. A function prototype for the SPI_Transfer( ) function also needs to be declared within all files that call the routine. The 'C' prototype for all of the example functions is:

```
extern char SPI_Transfer(char);
```

The "extern" modifier tells the linker that the function itself will be defined in a separate object file.

The functions can be called with the line:

```
in_spi = SPI_Transfer(out_spi);
```

where *in_spi* and *out_spi* are variables of type *char* that are used for the incoming and outgoing SPI bytes, respectively.

Sample Usage Code

Two full 'C' programs are included that demonstrate the usage of the SPI routines. The first example program, "SPI_F300_Test.c", demonstrates the method used to call the SPI routine. The second example, "SPI_EE_F30x.c", implements a serial EEPROM interface using the Mode 0 or Mode 3 SPI routines.

SPI_F300_Test.c

In the file "SPI_F300_Test.c", a *for* loop is established which counts up from 0 to 255 repeatedly. The *for* loop variable *test_counter* is used as the outgoing SPI byte, while the *SPI_return* variable is the incoming SPI byte. The NSS signal is pulled low immediately before the function call to select the slave device, and is pulled high after the function call to de-select it. After the SPI data transfer, both the outgoing and incoming SPI bytes are transmitted to the UART, where the progress can be viewed from a PC terminal program.

To test the sample code, the files "SPI_F300_Test.c", "SPI_defs.h", and one of the example function files should be placed in a single directory. "SPI_defs.h" contains the *sbit* declarations for the four pins used in the SPI implementation. The file containing SPI_Transfer( ) and the "SPI_F300_Test.c" file should be compiled or assembled separately and included in the build. Once the code has been programmed into the FLASH on a C8051F30x device, the MISO and MOSI pins of the device can be tied together to verify that what is being shifted out is also being shifted back in. With a PC terminal (configured for 115,200 Baud, 8 Data Bits, No Parity, 1 Stop Bit, No Flow Control) connected through an RS-232 level translator to the UART, the terminal program should display that the numbers being sent out on MOSI are the same as what is being received on MISO. A section of the resulting output from this configuration looks like:

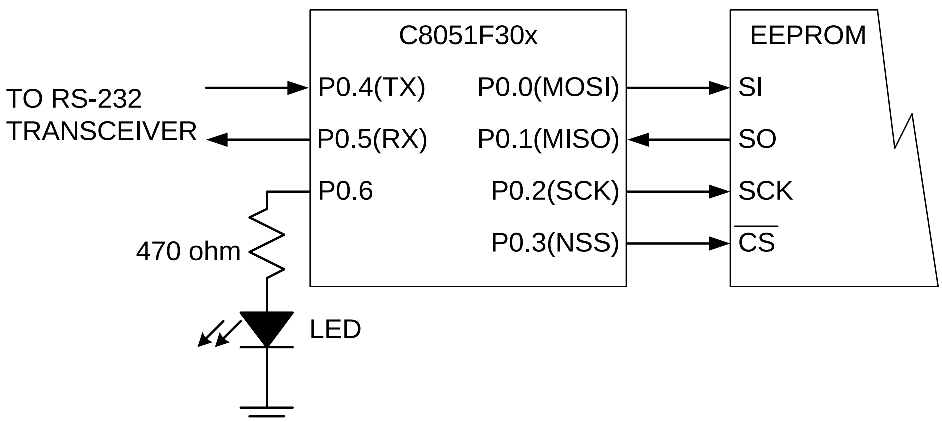
```
SPI Out = 0xFC, SPI In = 0xFC

SPI Out = 0xFD, SPI In = 0xFD

SPI Out = 0xFE, SPI In = 0xFE

SPI Out = 0xFF, SPI In = 0xFF

SPI Out = 0x00, SPI In = 0x00

SPI Out = 0x01, SPI In = 0x01

SPI Out = 0x02, SPI In = 0x02

SPI Out = 0x03, SPI In = 0x03

SPI Out = 0x04, SPI In = 0x04

SPI Out = 0x05, SPI In = 0x05
```

Tying the MOSI and MISO pins together does not completely test the functionality of the SPI implementation. However, it does verify that MOSI and MISO are being handled properly by the routine.

SPI_EE_F30x.c

"SPI_EE_F30x.c" uses a SPI routine to write to and read from a SPI EEPROM (Microchip 25LC320). To be compatible with the EEPROM's SPI interface, one of the Mode 0 or Mode 3 SPI functions must be used. The example code fills the EEPROM with two different patterns, and then verifies that the patterns have been written into the device by reading the EEPROM's contents back and checking them against the original pattern. During EEPROM writes, the LED (connected to P0.6) is lit, and during the EEPROM reads, the LED is unlit. If a read error occurs, the program will halt. If no errors occur, the LED will blink to signify that the test was successful. The progress of the program can also be monitored using a PC terminal (configured for 115,200 Baud, 8 Data Bits, No Parity, 1 Stop Bit, No Flow Control) connected through an RS-232 level translator to the UART. Figure 5 shows the connections between the C8051F30x and the EEPROM for this example.

Figure 5. EEPROM Connection



Software Examples

```
//-----------------------------------------------------------------------------
// SPI_defs.h
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BD
// DATE: 7 DEC 01
//
// This file defines the pins used for the SPI device.
// The SPI device is mapped to pins P0.0 - P0.3, but can be modified to map to
// any of the available GPIO pins on the device.
//


#ifndef SPI_DEFS

#define  SPI_DEFS

sbit MOSI = P0^0;                       // Master Out / Slave In (output)
sbit MISO = P0^1;                       // Master In / Slave Out (input)
sbit SCK = P0^2;                        // Serial Clock (output)
sbit NSS = P0^3;                        // Slave Select (output to chip select)

#endif
```

```
//-----------------------------------------------------------------------------
// SPI_MODE0.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BD
// DATE: 14 DEC 01
//
// This file contains a ‘C’ Implementation of a Mode 0 Master SPI device.
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//
//

#include <c8051f300.h>                  // SFR declarations
#include “SPI_defs.h”                   // SPI port definitions

//-----------------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------------
//
// Simultaneously transmits and receives one byte <SPI_byte> using
// the SPI protocol.  SCK is idle-low, and bits are latched on SCK rising.
//
// Timing for this routine is as follows:
//
// Parameter                            Clock Cycles
// MOSI valid to SCK rising edge        6
// SCK rising to MISO latched           2
// SCK falling to MOSI valid            7
// SCK high time                        8
// SCK low time                         13

char SPI_Transfer (char SPI_byte)
{
   unsigned char SPI_count;             // counter for SPI transaction

   for (SPI_count = 8; SPI_count > 0; SPI_count--) // single byte SPI loop
   {
      MOSI = SPI_byte & 0x80;           // put current outgoing bit on MOSI
      SPI_byte = SPI_byte << 1;         // shift next bit into MSB

      SCK = 0x01;                       // set SCK high

      SPI_byte |= MISO;                 // capture current bit on MISO

      SCK = 0x00;                       // set SCK low
   }

   return (SPI_byte);

} // END SPI_Transfer
```

```
;-----------------------------------------------------------------------------
;  Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
;  All rights reserved.
;
;  FILE NAME   :  SPI_MODE0.ASM
;  DATE        :  14 DEC 01
;  TARGET MCU  :  C8051F30x
;  DESCRIPTION :  This is a function which implements a master SPI port on
;                 the C8051F30x series of devices.  The function can
;                 be called from a C program with the following function
;                 prototype:
;
;                 extern char SPI_Transfer (char);
;
;  NOTES :  Timing is as follows (Mode 0 SPI):
;           Parameter                     SYSCLKs
;           MOSI valid to SCK rising      2
;           SCK rising to MISO Latch      2
;           SCK falling to MOSI valid     5
;           SCK high time                 5
;           SCK low time                  7
;
;-----------------------------------------------------------------------------

NAME  SPI_MODE0

?PR?_SPI_Transfer?SPI_MODE0   SEGMENT CODE
   PUBLIC   _SPI_Transfer

$include (c8051f300.inc)            ; Include regsiter definition file.
$include (SPI_defs.h)               ; Include sbit definitions for SPI

   RSEG  ?PR?_SPI_Transfer?SPI_MODE0
_SPI_Transfer:
   USING 0

            MOV   A, R7             ; Store passed variable in A

            MOV   R7, #08H          ; Load R7 to count bits
            RLC   A                 ; Rotate MSB into Carry Bit
SPI_Loop:   MOV   MOSI, C           ; Move bit out to MOSI
            SETB  SCK               ; Clock High
            MOV   C, MISO           ; Move MISO into Carry Bit
            RLC   A                 ; Rotate Carry Bit into A
            CLR   SCK               ; Clock Low
            DJNZ  R7, SPI_Loop      ; Loop for another bit until finished

            MOV   R7, A             ; Store return value in R7

?C0001:
            RET                     ; Return from routine


END                                 ; END OF FILE

//-----------------------------------------------------------------------------
// SPI_MODE1.c
//-----------------------------------------------------------------------------
```

```
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BD
// DATE: 14 DEC 01
//
// This file contains a 'C' Implementation of a Mode 1 Master SPI device.
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//
//

#include <c8051f300.h>                // SFR declarations
#include "SPI_defs.h"                 // SPI port definitions

//-----------------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------------
//
// Simultaneously transmits and receives one byte <SPI_byte> using
// the SPI protocol.  SCK is idle-low, and bits are latched on SCK falling.
//
// Timing for this routine is as follows:
//
// Parameter                          Clock Cycles
// SCK rising edge to MOSI valid      4
// MOSI valid to SCK falling edge     6
// SCK falling to MISO latch          2
// SCK high time                      10
// SCK low time                       11

char SPI_Transfer (char SPI_byte)
{
   unsigned char SPI_count;           // counter for SPI transaction

   for (SPI_count = 8; SPI_count > 0; SPI_count--) // single byte SPI loop
   {
      SCK = 0x01;                     // set SCK high

      MOSI = SPI_byte & 0x80;         // put current outgoing bit on MOSI
      SPI_byte = SPI_byte << 1;       // shift next bit into MSB

      SCK = 0x00;                     // set SCK low

      SPI_byte |= MISO;               // capture current bit on MISO

   }

   return (SPI_byte);

} // END SPI_Transfer
```

```
;-----------------------------------------------------------------------------
;  Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
;  All rights reserved.
;
;  FILE NAME   :  SPI_MODE1.ASM
;  DATE        :  14 DEC 01
;  TARGET MCU  :  C8051F30x
;  DESCRIPTION :  This is a function which implements a master SPI device on
;                 the C8051F30x series of devices.  The function can
;                 be called from a C program with the following function
;                 prototype:
;
;                 extern char SPI_Transfer (char);
;
;  NOTES :  Timing is as follows (Mode 1 SPI):
;           Parameter                     SYSCLKs
;           SCK rising to MOSI valid      2
;           MOSI valid to SCK falling     2
;           SCK falling to MISO Latch     3
;           SCK high time                 5
;           SCK low time                  7
;
;-----------------------------------------------------------------------------

NAME  SPI_MODE1

?PR?_SPI_Transfer?SPI_MODE1   SEGMENT CODE
   PUBLIC   _SPI_Transfer

$include (c8051f300.inc)            ; Include regsiter definition file.
$include (SPI_defs.h)               ; Include sbit definitions for SPI

   RSEG  ?PR?_SPI_Transfer?SPI_MODE1
_SPI_Transfer:
   USING 0

            MOV   A, R7             ; Store passed variable in A

            MOV   R7, #08H          ; Load R7 to count bits
SPI_Loop:   SETB  SCK               ; Clock High
            RLC   A                 ; Rotate MSB into Carry Bit
            MOV   MOSI, C           ; Move bit out to MOSI
            CLR   SCK               ; Clock Low
            MOV   C, MISO           ; Move MISO into Carry Bit
            DJNZ  R7, SPI_Loop      ; Loop for another bit until done

            RLC   A                 ; Rotate Carry Bit into A
            MOV   R7, A             ; Store return value in R7

?C0001:
            RET                     ; Return from routine


END                                 ; END OF FILE

//-----------------------------------------------------------------------------
// SPI_MODE2.c
//-----------------------------------------------------------------------------
```

```
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BD
// DATE: 14 DEC 01
//
// This file contains a ‘C’ Implementation of a Mode 2 Master SPI device.
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//
//

#include <c8051f300.h>                  // SFR declarations
#include “SPI_defs.h”                   // SPI port definitions

//-----------------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------------
//
// Simultaneously transmits and receives one byte <SPI_byte> using
// the SPI protocol.  SCK is idle-high, and bits are latched on SCK falling.
//
// Timing for this routine is as follows:
//
// Parameter                            Clock Cycles
// MOSI valid to SCK falling edge       6
// SCK falling to MISO latched          2
// SCK rising to MOSI valid             7
// SCK low time                         8
// SCK high time                        13

char SPI_Transfer (char SPI_byte)
{
   unsigned char SPI_count;             // counter for SPI transaction

   for (SPI_count = 8; SPI_count > 0; SPI_count--) // single byte SPI loop
   {
      MOSI = SPI_byte & 0x80;           // put current outgoing bit on MOSI
      SPI_byte = SPI_byte << 1;         // shift next bit into MSB

      SCK = 0x00;                       // set SCK low

      SPI_byte |= MISO;                 // capture current bit on MISO

      SCK = 0x01;                       // set SCK high

   }

   return (SPI_byte);

} // END SPI_Transfer
```

```
;-----------------------------------------------------------------------------
;  Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
;  All rights reserved.
;
;  FILE NAME   :  SPI_MODE2.ASM
;  DATE        :  14 DEC 01
;  TARGET MCU  :  C8051F30x
;  DESCRIPTION :  This is a function which implements a master SPI device on
;                 the C8051F30x series of devices.  The function can
;                 be called from a C program with the following function
;                 prototype:
;
;                 extern char SPI_Transfer (char);
;
;  NOTES :  Timing is as follows (Mode 2 SPI):
;           Parameter                     SYSCLKs
;           MOSI valid to SCK falling     2
;           SCK falling to MISO Latch     2
;           SCK rising to MOSI valid      5
;           SCK low time                  5
;           SCK high time                 7
;
;-----------------------------------------------------------------------------

NAME  SPI_MODE2

?PR?_SPI_Transfer?SPI_MODE2   SEGMENT CODE
   PUBLIC   _SPI_Transfer

$include (c8051f300.inc)            ; Include regsiter definition file.
$include (SPI_defs.h)               ; Include sbit definitions for SPI

   RSEG  ?PR?_SPI_Transfer?SPI_MODE2
_SPI_Transfer:
   USING 0

            MOV   A, R7             ; Store passed variable in A

            MOV   R7, #08H          ; Load R7 to count bits
            RLC   A                 ; Rotate MSB into Carry Bit
SPI_Loop:   MOV   MOSI, C           ; Move bit out to MOSI
            CLR   SCK               ; Clock Low
            MOV   C, MISO           ; Move MISO into Carry Bit
            RLC   A                 ; Rotate Carry Bit into A
            SETB  SCK               ; Clock High
            DJNZ  R7, SPI_Loop      ; Loop for another bit until finished

            MOV   R7, A             ; Store return value in R7

?C0001:
            RET                     ; Return from routine


END                                 ; END OF FILE

//-----------------------------------------------------------------------------
// SPI_MODE3.c
//-----------------------------------------------------------------------------
```

```
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BD
// DATE: 14 DEC 01
//
// This file contains a 'C' Implementation of a Mode 3 Master SPI device.
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//
//

#include <c8051f300.h>                  // SFR declarations
#include "SPI_defs.h"                   // SPI port definitions

//-----------------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------------
//
// Simultaneously transmits and receives one byte <SPI_byte> using
// the SPI protocol.  SCK is idle-high, and bits are latched on SCK rising.
//
// Timing for this routine is as follows:
//
// Parameter                            Clock Cycles
// SCK falling edge to MOSI valid       4
// MOSI valid to SCK rising edge       6
// SCK rising to MISO latch              2
// SCK low time                         10
// SCK high time                        11

char SPI_Transfer (char SPI_byte)
{
   unsigned char SPI_count;             // counter for SPI transaction

   for (SPI_count = 8; SPI_count > 0; SPI_count--) // single byte SPI loop
   {
      SCK = 0x00;                       // set SCK low

      MOSI = SPI_byte & 0x80;           // put current outgoing bit on MOSI
      SPI_byte = SPI_byte << 1;         // shift next bit into MSB

      SCK = 0x01;                       // set SCK high

      SPI_byte |= MISO;                 // capture current bit on MISO

   }

   return (SPI_byte);

} // END SPI_Transfer
```

```
;-----------------------------------------------------------------------------
; Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
; All rights reserved.
;
; FILE NAME   : SPI_MODE3.ASM
; DATE        : 14 DEC 01
; TARGET MCU  : C8051F30x
; DESCRIPTION : This is a function which implements a master SPI device on
;               the C8051F30x series of devices.  The function can
;               be called from a C program with the following function
;               prototype:
;
;               extern char SPI_Transfer (char);
;
; NOTES :  Timing is as follows (Mode 3 SPI):
;          Parameter                     SYSCLKs
;          SCK falling to MOSI valid     2
;          MOSI valid to SCK rising      2
;          SCK rising to MISO Latch      3
;          SCK low time                  5
;          SCK high time                 7
;
;-----------------------------------------------------------------------------

NAME  SPI_MODE3

?PR?_SPI_Transfer?SPI_MODE3   SEGMENT CODE
   PUBLIC   _SPI_Transfer

$include (c8051f300.inc)            ; Include regsiter definition file.
$include (SPI_defs.h)               ; Include sbit definitions for SPI

   RSEG  ?PR?_SPI_Transfer?SPI_MODE3
_SPI_Transfer:
   USING 0

            MOV   A, R7             ; Store passed variable in A

            MOV   R7, #08H          ; Load R7 to count bits
SPI_Loop:   CLR   SCK               ; Clock Low
            RLC   A                 ; Rotate MSB into Carry Bit
            MOV   MOSI, C           ; Move bit out to MOSI
            SETB  SCK               ; Clock High
            MOV   C, MISO           ; Move MISO into Carry Bit
            DJNZ  R7, SPI_Loop      ; Loop for another bit until finished

            RLC   A                 ; Rotate Carry Bit into A
            MOV   R7, A             ; Store return value in R7

?C0001:
            RET                     ; Return from routine


END                                 ; END OF FILE

//-----------------------------------------------------------------------------
// SPI_F300_Test.c
//-----------------------------------------------------------------------------
```

```
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BD
// DATE: 14 DEC 01
//
// This program demonstrates how a collection of SPI master
// routines for the 8051F30x processors can be used in a C program.
//
// This program sets up the GPIO pins on the C8051F30x device for the correct
// functionality, then uses the SPI_Transfer function to send and receive
// information through the SPI pins. As information is sent, the progress of
// the program is sent out through the UART to be monitored on a connected
// terminal program.
//
// For this code to be functional, *one* of the following files should also be
// compiled or assembled, and the resulting object file must be linked to the
// object file produced from this file:
//
//    SPI_MODE0.c    Mode 0 SPI Master Implementation in C
//    SPI_MODE0.asm  Mode 0 SPI Master Implementation in Assembly
//    SPI_MODE1.c    Mode 1 SPI Master Implementation in C
//    SPI_MODE1.asm  Mode 1 SPI Master Implementation in Assembly
//    SPI_MODE2.c    Mode 2 SPI Master Implementation in C
//    SPI_MODE2.asm  Mode 2 SPI Master Implementation in Assembly
//    SPI_MODE3.c    Mode 3 SPI Master Implementation in C
//    SPI_MODE3.asm  Mode 3 SPI Master Implementation in Assembly
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------

#include <c8051f300.h>                 // SFR declarations
#include <stdio.h>                     // Standard I/O
#include “SPI_defs.h”                  // SPI port definitions

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F30x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR2RL   = 0xca;                 // Timer2 reload value
sfr16 TMR2     = 0xcc;                 // Timer2 counter
sfr16 PCA0CP1  = 0xe9;                 // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                 // PCA0 Module 2 Capture/Compare
sfr16 PCA0     = 0xf9;                 // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                 // PCA0 Module 0 Capture/Compare

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------

#define SYSCLK       24500000          // SYSCLK frequency in Hz
#define BAUDRATE     115200            // Baud rate of UART in bps

//-----------------------------------------------------------------------------
```

```
// Function PROTOTYPES
//-----------------------------------------------------------------------------

void PORT_Init (void);                  // Port I/O configuration
void SYSCLK_Init (void);                // SYSCLK Initialization
void UART0_Init (void);                 // UART0 Initialization

extern char SPI_Transfer (char);        // SPI Transfer routine

//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------

void main (void) {

   unsigned char test_counter, SPI_return;   // used to test SPI routine

   // Disable Watchdog timer
   PCA0MD &= ~0x40;                     // WDTE = 0 (clear watchdog timer
                                        // enable)

   SYSCLK_Init ();                      // initialize oscillator
   PORT_Init ();                        // initialize ports and GPIO
   UART0_Init ();                       // initialize UART0
   EA = 1;                              // enable global interrupts

   while (1)
   {
      for (test_counter = 0; test_counter <= 0xFF; test_counter++)
      {

         NSS = 0x00;                    // select SPI Slave device

         SPI_return = SPI_Transfer(test_counter); // send/receive SPI byte

         NSS = 0x01;                    // de-select SPI Slave device

         printf("\nSPI Out = 0x%02X, SPI In = 0x%02X", (unsigned)test_counter,
         (unsigned)SPI_return);
                                        // send SPI data out to UART
                                        // for verification purposes
      }
   }
}

//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports.
```

```
// P0.0 - MOSI (push-pull)
// P0.1 - MISO
// P0.2 - SCK (push-pull)
// P0.3 - NSS (push-pull)
// P0.4 - UART TX (push-pull)
// P0.5 - UART RX
// P0.6 -
// P0.7 -
//
void PORT_Init (void)
{
   XBR0    =  0x0F;                    // skip SPI pins in XBAR
   XBR1    =  0x03;                    // UART0 TX and RX pins enabled
   XBR2    =  0x40;                    // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x1D;                    // enable TX0, MOSI, SCK, and NSS as
                                       // push-pull outputs
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the internal 24.5 MHz clock
// as its clock source.
//
void SYSCLK_Init (void)
{

   OSCICN = 0x07;                      // select internal oscillator as SYSCLK
                                       // source
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <BAUDRATE> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0 = 0x10;                       // SCON0: 8-bit variable bit rate
                                       //        level of STOP bit is ignored
                                       //        RX enabled
                                       //        ninth bits are zeros
                                       //        clear RI0 and TI0 bits
   if (SYSCLK/BAUDRATE/2/256 < 1)
   {
      TH1 = -(SYSCLK/BAUDRATE/2);
      CKCON &= ~0x13;
      CKCON |= 0x10;                   // T1M = 1; SCA1:0 = xx
   }
   else if (SYSCLK/BAUDRATE/2/256 < 4)
   {
      TH1 = -(SYSCLK/BAUDRATE/2/4);
      CKCON &= ~0x13;
      CKCON |=  0x01;                  // T1M = 0; SCA1:0 = 01
   }
   else if (SYSCLK/BAUDRATE/2/256 < 12)
   {
```

```
      TH1 = -(SYSCLK/BAUDRATE/2/12);
      CKCON &= ~0x13;                    // T1M = 0; SCA1:0 = 00
   }
   else
   {
      TH1 = -(SYSCLK/BAUDRATE/2/48);
      CKCON &= ~0x13;
      CKCON |=  0x02;                    // T1M = 0; SCA1:0 = 10
   }

   TL1 = 0xff;                           // set Timer1 to overflow immediately
   TMOD |= 0x20;                         // TMOD: timer 1 in 8-bit autoreload
   TMOD &= ~0xD0;                        // mode
   TR1 = 1;                              // START Timer1
   TI0 = 1;                              // Indicate TX0 ready
}



//-----------------------------------------------------------------------------
// SPI_EE_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BD
// DATE: 14 DEC 01
//
// This program demonstrates how a collection of SPI master routines for the
// 8051F30x devices can be used in a C program.
//
// In this example, a Microchip 25LC320 4k X 8 Serial EEPROM is interfaced to a
// SPI master device implemented in the C8051F30x. The EEPROM is written with
// two test patterns: 1) all locations are 0xFF and 2) each location is written
// with the LSB of the corresponding address.
// The EEPROM contents are then verified with the test patterns.  If the test
// patterns are verified with no errors, the LED blinks on operation completion.
// Otherwise, the LED stays off.  Progress can also be monitored by a terminal
// connected to UART0 operating at 115.2kbps.
//
// For this code to be functional, *one* of the following files should also be
// compiled or assembled, and the resulting object file must be linked to the
// object file produced from this code:
//
//    SPI_MODE0.c    Mode 0 SPI Master Implementation in C
//    SPI_MODE0.asm  Mode 0 SPI Master Implementation in Assembly
//    SPI_MODE3.c    Mode 3 SPI Master Implementation in C
//    SPI_MODE3.asm  Mode 3 SPI Master Implementation in Assembly
//
//    This EEPROM’s serial port will only operate with a Mode 0 or Mode 3
//    SPI configuration.
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
```

```
#include <c8051f300.h>                  // SFR declarations
#include <stdio.h>                      // Standard I/O
#include “SPI_defs.h”                   // SPI port definitions

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F30x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                  // data pointer
sfr16 TMR2RL   = 0xca;                  // Timer2 reload value
sfr16 TMR2     = 0xcc;                  // Timer2 counter
sfr16 PCA0CP1  = 0xe9;                  // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                  // PCA0 Module 2 Capture/Compare
sfr16 PCA0     = 0xf9;                  // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                  // PCA0 Module 0 Capture/Compare

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------

#define  SYSCLK      24500000           // SYSCLK frequency in Hz
#define  BAUDRATE    115200             // Baud rate of UART in bps

#define  EE_SIZE     4096               // EEPROM size in bytes
#define  EE_READ     0x03               // EEPROM Read command
#define  EE_WRITE    0x02               // EEPROM Write command
#define  EE_WRDI     0x04               // EEPROM Write disable command
#define  EE_WREN     0x06               // EEPROM Write enable command
#define  EE_RDSR     0x05               // EEPROM Read status register
#define  EE_WRSR     0x01               // EEPROM Write status register

sbit     LED = P0^6;                    // LED Indicator

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------

void PORT_Init (void);                  // Port I/O configuration
void SYSCLK_Init (void);                // SYSCLK Initialization
void UART0_Init (void);                 // UART0 Initialization

extern char SPI_Transfer (char);        // SPI Transfer routine

void Timer0_ms (unsigned ms);
void Timer0_us (unsigned us);

unsigned char EE_Read (unsigned Addr);
void EE_Write (unsigned Addr, unsigned char value);


//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
```

```
void main (void) {

   unsigned EE_Addr;                    // address of EEPROM byte
   unsigned char test_byte;

   // Disable Watchdog timer
   PCA0MD &= ~0x40;                     // WDTE = 0 (clear watchdog timer
                                        // enable)

   SYSCLK_Init ();                      // initialize oscillator
   PORT_Init ();                        // initialize ports and GPIO
   UART0_Init ();                       // initialize UART0
   EA = 1;                              // enable global interrupts

   SCK = 0;

   // fill EEPROM with 0xFF’s
   LED = 1;
   for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
   {
      test_byte = 0xff;
      EE_Write (EE_Addr, test_byte);

      // print status to UART0
      if ((EE_Addr % 16) == 0)
      {
         printf (“\nwriting 0x%04x: %02x “, EE_Addr, (unsigned) test_byte);
      }
      else
      {
         printf (“%02x “, (unsigned) test_byte);
      }
   }

   // verify EEPROM with 0xFF’s
   LED = 0;
   for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
   {
      test_byte = EE_Read (EE_Addr);

      // print status to UART0
      if ((EE_Addr % 16) == 0)
      {
         printf (“\nverifying 0x%04x: %02x “, EE_Addr, (unsigned) test_byte);
      }
      else
      {
         printf (“%02x “, (unsigned) test_byte);
      }
      if (test_byte != 0xFF)
      {
         printf (“Error at %u\n”, EE_Addr);
         while (1);                     // stop here on error
      }
   }

   // fill EEPROM memory with LSB of EEPROM address.
   LED = 1;
   for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
```

```
   {
      test_byte = EE_Addr & 0xff;
      EE_Write (EE_Addr, test_byte);

      // print status to UART0
      if ((EE_Addr % 16) == 0)
      {
         printf ("\nwriting 0x%04x: %02x ", EE_Addr, (unsigned) test_byte);
      }
      else
      {
         printf ("%02x ", (unsigned) test_byte);
      }
   }

   // verify EEPROM memory with LSB of EEPROM address
   LED = 0;
   for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
   {
      test_byte = EE_Read (EE_Addr);

      // print status to UART0
      if ((EE_Addr % 16) == 0)
      {
         printf ("\nverifying 0x%04x: %02x ", EE_Addr, (unsigned) test_byte);
      }
      else
      {
         printf ("%02x ", (unsigned) test_byte);
      }
      if (test_byte != (EE_Addr & 0xFF))
      {
         printf ("Error at %u\n", EE_Addr);
         while (1);                    // stop here on error
      }
   }

   while (1)
   {                                   // Flash LED when done
      Timer0_ms (100);
      LED = ~LED;
   }
}

//-----------------------------------------------------------------------------
// Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports.
// P0.0 - MOSI (push-pull)
// P0.1 - MISO
```

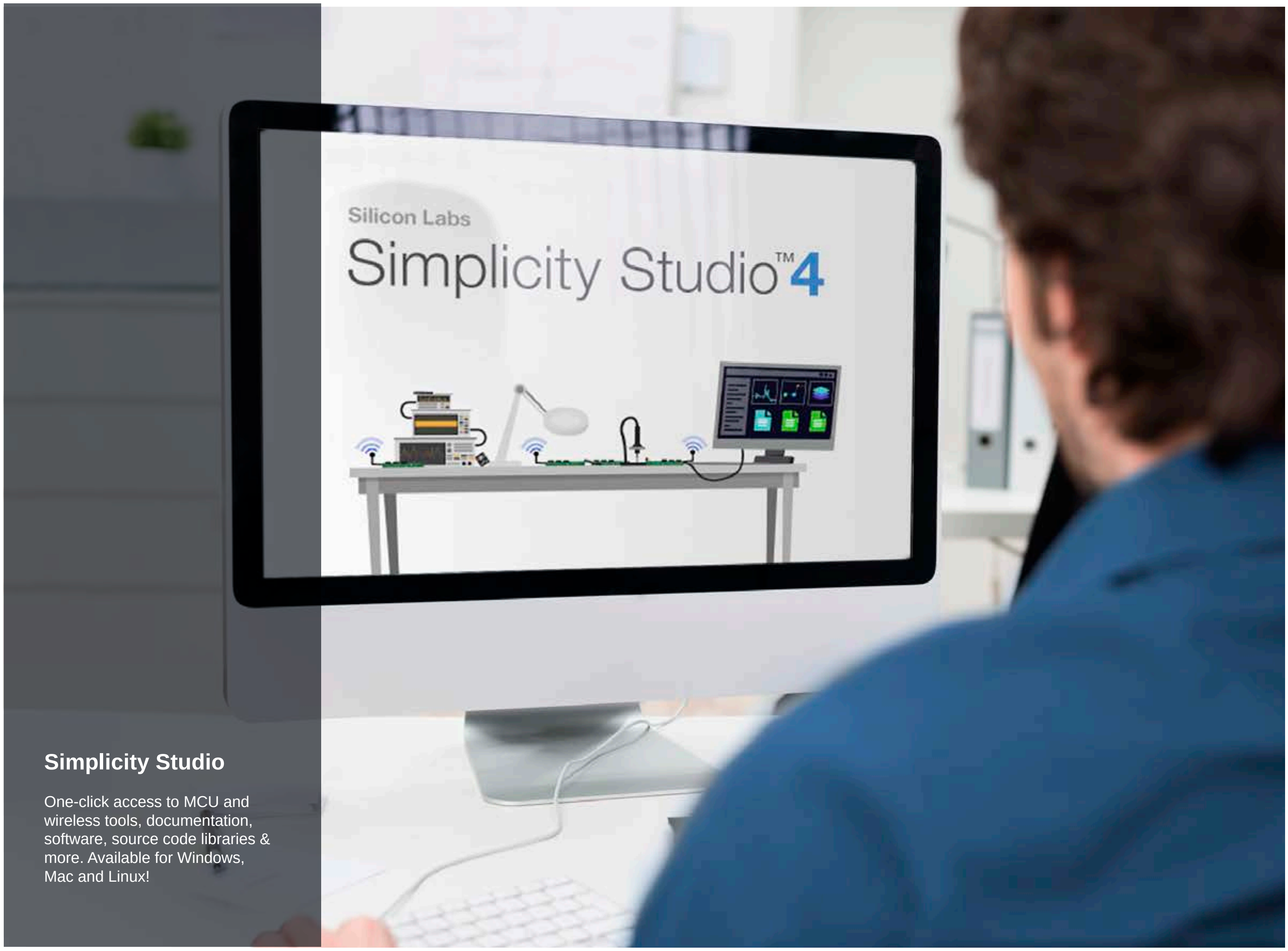
```
// P0.2 - SCK (push-pull)
// P0.3 - NSS (push-pull)
// P0.4 - UART TX (push-pull)
// P0.5 - UART RX
// P0.6 - LED
// P0.7 -
//
void PORT_Init (void)
{
   XBR0    =  0x0F;                    // skip SPI pins in XBAR
   XBR1    =  0x03;                    // UART0 TX and RX pins enabled
   XBR2    =  0x40;                    // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x5D;                    // enable TX0, MOSI, SCK, LED and NSS as
                                       // push-pull outputs
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the internal 24.5 MHz clock
// as its clock source.
//
void SYSCLK_Init (void)
{

   OSCICN = 0x07;                      // select internal oscillator as SYSCLK
                                       // source
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <BAUDRATE> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0 = 0x10;                       // SCON0: 8-bit variable bit rate
                                       //        level of STOP bit is ignored
                                       //        RX enabled
                                       //        ninth bits are zeros
                                       //        clear RI0 and TI0 bits
   if (SYSCLK/BAUDRATE/2/256 < 1)
   {
      TH1 = -(SYSCLK/BAUDRATE/2);
      CKCON &= ~0x13;
      CKCON |= 0x10;                   // T1M = 1; SCA1:0 = xx
   }
   else if (SYSCLK/BAUDRATE/2/256 < 4)
   {
      TH1 = -(SYSCLK/BAUDRATE/2/4);
      CKCON &= ~0x13;
      CKCON |=  0x01;                  // T1M = 0; SCA1:0 = 01
   }
   else if (SYSCLK/BAUDRATE/2/256 < 12)
   {
      TH1 = -(SYSCLK/BAUDRATE/2/12);
      CKCON &= ~0x13;                  // T1M = 0; SCA1:0 = 00
```

```
   }
   else
   {
      TH1 = -(SYSCLK/BAUDRATE/2/48);
      CKCON &= ~0x13;
      CKCON |=  0x02;                   // T1M = 0; SCA1:0 = 10
   }

   TL1 = 0xff;                          // set Timer1 to overflow immediately
   TMOD |= 0x20;                        // TMOD: timer 1 in 8-bit autoreload
   TMOD &= ~0xD0;                       // mode
   TR1 = 1;                             // START Timer1
   TI0 = 1;                             // Indicate TX0 ready
}

//-----------------------------------------------------------------------------
// Timer0_ms
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <ms> milliseconds before returning.
//
void Timer0_ms (unsigned ms)
{
   unsigned i;                          // millisecond counter

   TCON  &= ~0x30;                      // STOP Timer0 and clear overflow flag
   TMOD  &= ~0x0f;                      // configure Timer0 to 16-bit mode
   TMOD  |=  0x01;
   CKCON |=  0x08;                      // Timer0 counts SYSCLKs

   for (i = 0; i < ms; i++)             // count milliseconds
   {
      TR0 = 0;                          // STOP Timer0
      TH0 = (-SYSCLK/1000) >> 8;        // set Timer0 to overflow in 1ms
      TL0 = -SYSCLK/1000;
      TR0 = 1;                          // START Timer0
      while (TF0 == 0);                 // wait for overflow
      TF0 = 0;                          // clear overflow indicator
   }
}

//-----------------------------------------------------------------------------
// Timer0_us
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <us> microseconds before returning.
//
void Timer0_us (unsigned us)
{
   unsigned i;                          // millisecond counter

   TCON  &= ~0x30;                      // STOP Timer0 and clear overflow flag
   TMOD  &= ~0x0f;                      // configure Timer0 to 16-bit mode
   TMOD  |=  0x01;
   CKCON |=  0x08;                      // Timer0 counts SYSCLKs

   for (i = 0; i < us; i++) {           // count microseconds
      TR0 = 0;                          // STOP Timer0
      TH0 = (-SYSCLK/1000000) >> 8;     // set Timer0 to overflow in 1us
```

```
      TL0 = -SYSCLK/1000000;
      TR0 = 1;                         // START Timer0
      while (TF0 == 0);                // wait for overflow
      TF0 = 0;                         // clear overflow indicator
   }
}

//-----------------------------------------------------------------------------
// EE_Read
//-----------------------------------------------------------------------------
//
// This routine reads and returns a single EEPROM byte whose address is
// given in <Addr>.
//
unsigned char EE_Read (unsigned Addr)
{
   unsigned char retval;               // value to return

   NSS = 0;                            // select EEPROM

   Timer0_us (1);                      // wait at least 250ns (CS setup time)

   // transmit READ opcode
   retval = SPI_Transfer(EE_READ);

   // transmit Address MSB-first
   retval = SPI_Transfer((Addr & 0xFF00) >> 8);   // transmit MSB of address

   retval = SPI_Transfer((Addr & 0x00FF));        // transmit LSB of address

   // initiate dummy transmit to read data

   retval = SPI_Transfer(0x00);

   Timer0_us (1);                      // wait at least 250ns (CS hold time)

   NSS = 1;                            // de-select EEPROM

   Timer0_us (1);                      // wait at least 500ns (CS disable time)

   return retval;
}

//-----------------------------------------------------------------------------
// EE_Write
//-----------------------------------------------------------------------------
//
// This routine writes a single EEPROM byte <value> to address <Addr>.
//
void EE_Write (unsigned Addr, unsigned char value)
{
   unsigned char retval;               // return value from SPI

   NSS = 0;                            // select EEPROM
   Timer0_us (1);                      // wait at least 250ns (CS setup time)

   // transmit WREN (Write Enable) opcode
   retval = SPI_Transfer(EE_WREN);
```

```
   Timer0_us (1);                    // wait at least 250ns (CS hold time)

   NSS = 1;                          // de-select EEPROM to set WREN latch
   Timer0_us (1);                    // wait at least 500ns (CS disable
                                     // time)

   NSS = 0;                          // select EEPROM
   Timer0_us (1);                    // wait at least 250ns (CS setup time)

   // transmit WRITE opcode
   retval = SPI_Transfer(EE_WRITE);

   // transmit Address MSB-first
   retval = SPI_Transfer((Addr & 0xFF00) >> 8);   // transmit MSB of address

   retval = SPI_Transfer((Addr & 0x00FF));        // transmit LSB of address

   // transmit data
   retval = SPI_Transfer(value);

   Timer0_us (1);                    // wait at least 250ns (CS hold time)

   NSS = 1;                          // deselect EEPROM (initiate EEPROM
                                     // write cycle)

   // now poll Read Status Register (RDSR) for Write operation complete
   do {

      Timer0_us (1);                 // wait at least 500ns (CS disable
                                     // time)

      NSS = 0;                       // select EEPROM to begin polling

      Timer0_us (1);                 // wait at least 250ns (CS setup time)

      retval = SPI_Transfer(EE_RDSR);

      retval = SPI_Transfer(0x00);

      Timer0_us (1);                 // wait at least 250ns (CS hold
                                     // time)
      NSS = 1;                       // de-select EEPROM

   } while (retval & 0x01);          // poll until WIP (Write In
                                     // Progress) bit goes to ‘0’

   Timer0_us (1);                    // wait at least 500ns (CS disable
                                     // time)
}
```


SILICON LABS



IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com