AN112

UART In-Application Code Loading Examples

Relevant Devices

This application note applies to the following devices:

C8051F020, C8051F021, C8051F022, C8051F023, C8051F300, C8051F301, C8051F302, and C8051F303.

Introduction

A UART code loader provides in-system reprogrammability of program code space (FLASH) through the serial port. This application note gives an overview of in-application code loading on Silicon Labs devices and provides two complete examples. The examples included are a selective code loader and a firmware updater. This document also discusses design considerations related to in-application code loading.

Key Points

- FLASH memory locations must be erased before the new program code is written.
- An Intel Hexadecimal Object File ("HEX" file) is an ASCII file containing a complete or partial image of the programmable device's program code space (FLASH). An OMF-51 (binary linker output file) to Intel HEX converter is provided with the Silicon Labs IDE.
- A UART code loader can be controlled by a PC running a terminal program or any other embedded device that has a UART.

In-Application Code Loading Overview

To load code into a device through the UART, the device needs to run an application that manages the transfer of code from the host to its program memory. This application needs the ability to do the following tasks:

1. Configure the device for UART communication at a specified baud rate.
2. Erase program memory (FLASH) prior to receiving the download.
3. Download the new code and store it in program memory.
4. Execute the newly downloaded code.

Configuring the Device for UART communication

When using UART to communicate between two devices, both ends must be configured to run at the same baud rate, in 8-bit or 9-bit data mode, and with or without parity. The examples in this document use 8-bit data with no parity at a baud rate of 115200 bits per second. If a terminal program is used on the host, it should be configured as shown in the following table:

Table 1. Terminal Program Configuration

bits per second	115200
data bits	8
parity	none
stop bits	1
flow control	none

Erasing and Writing to FLASH

The program memory on all Silicon Labs 8051F devices is FLASH. In general, a code loader will need to erase one or more 512-byte FLASH pages before storing the new downloaded code. The method of erasing and writing to FLASH varies by device family. Refer to the FLASH Memory section of the device data sheet for details regarding the specific device family. Additionally, the Silicon Labs website contains application notes with code examples.

Downloading the New Code

Once the code loader has erased one or more FLASH pages, it will prompt the user to send the new code. There are many ways the host can encode the new code as long as the code loader can decode and interpret the information. A good format to use is the Intel Hexadecimal Object File format. An Intel HEX file is an ASCII file containing a complete or partial image of the programmable device's program code space (FLASH). This file is generated from the linker output file using the OH51 utility provided with the Silicon Labs IDE installation. The details of generating an Intel HEX file will be discussed later on in this document.

This example provides some error detection capability in that checksums are calculated on the received HEX records and compared with the record checksums. If an error is detected, the download operation is aborted.

Running the New Code

Once the new code is stored in FLASH, it can be called using a function pointer. Function pointers are implemented differently by different compilers. See the compiler documentation for specific information reguarding the compiler being used. A function pointer in the KEIL C51 compiler is a 3-byte generic pointer and is used as shown in Figure 1. The first byte of a generic pointer specifies the memory segment and the remaining two bytes specify the address. For example, a pointer to address 0x1000 in code space would be 0xFF1000. Consult the compiler documentation for additional information about function pointers.

Code Loader Considerations

Any code loading application will consist of at least two projects – one for the code loader and one for the code to be loaded. There is a certain level of difficulty when dealing with two separate projects that share the same resources. The considerations in this application note will attempt to address some of these difficulties and pitfalls, but be aware that it cannot cover them all. Make sure you are familiar with your compiler and linker documentation before starting any multi-project application. Pay special attention to the linker chapter regarding locating segments.

The main things to watch out for when using multiple projects is not to allow the data and code seg-

Figure 1. Using Function Pointers

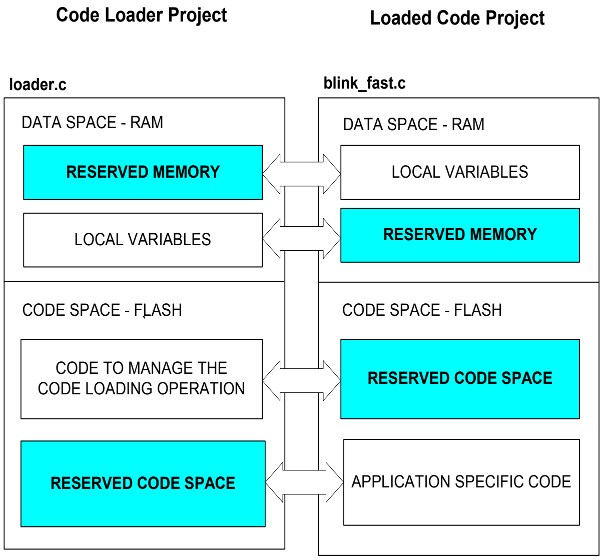
```
// declaring a function pointer
void (*f)();        // can point to a function that takes no arguments
void (*g)(int i);   // can point to a function that takes one argument

// assigning a function pointer
f = (void code *)0x1000;// f points to a function located at 0x1000 in code space
g = (void code *)0x1100;// g points to a function located at 0x1100 in code space

// calling a function using a function pointer
f();
g(5);
```

ments from the projects to overlap, as shown in Figure 2. The code segments should not share the same FLASH pages to allow downloading the second project without erasing the first. The data segments should not be allowed to overlap because code from either project can be executing at any given time. If both projects' variables were located at the same memory locations, they would corrupt each other's data. It is fairly simple to keep the code segments from overlapping; however, keeping the data segments from ovelapping is more challenging and can be harder to debug. Three methods for keeping segments from ovelapping are discussed below. An example using Method 3 is included in this document.

Method 1

The first method involves absolutely locating code segments with 'CODE' linker command line parameters. To keep the 'DATA' segments from overlapping, the function call trees are manually edited using the 'OVERLAY' linker command line parameter. This method is complex and should be reserved for large projects that need the extra memory capacity provided by overlaying.

Figure 2. Project Map



Method 2

The second method involves declaring and absolutely locating a set of global variables in both projects that is used only by the project that will be loaded. An easy way to accomplish this is by including a header file containg these declarations in both projects. Code segments are absolutely located using the linker's 'CODE' parameter. This method should be reserved for small projects where all variables declared in the loaded code are easy to keep track of.

Method 3

The third method of data management involves declaring all local variables as static. Once the projects for the loaded code are built, the MAP file is examined for the data segment size. Space for this segment is reserved in the loader project by declaring and absolutely locating an array of the same size as the segment. As an added precaution, the data segment may be absolutly located using the 'DATA' parameter to ensure that it will not move around. Code segments are also absolutely located using the linker's 'CODE' parameter. This is the preferred method if overlaying is not required and is used in the following example code.

In-application Code Loading Examples

The following examples show how in-application code loading can be used in various situations.

Selective Code Loader

This example contains three projects and uses the method 3 to manage memory. The main project named 'Loader' contains the code loader and is downloaded using the IDE. The other two projects, 'Blink_Fast' and 'Blink_Slow', contain functions that blink the green LED at different rates and are selectively downloaded using a terminal program.

The loader project takes the following items into account:

1. It sets aside a block of RAM at a specific address for use by global and static local variables in the loaded code.

2. It sets aside one or more pages of FLASH to store the loaded code. (These pages start at location 0x1000 in code space)

3. It predetermines the function locations and the number of functions defined in the loaded code.

The two projects that are selectively loaded take the following factors into account:

1. They only use RAM which has been set aside by the code loader project.

2. They absolutely locate all functions on one or more contiguous FLASH pages set aside by the loader project (at address 0x1000). This involves use of the '**CODE**' linker command line parameter as shown in Figure 3.

Figure 3. Locating Functions Using the linker 'CODE' directive.

```
The CODE directive is specified at
the linker command line. The command
line parameters are accessed from the
Silicon Labs IDE in the 'Project-
>Tool Chain Integration...' menu
under the 'linker' tab.

To locate a segment at 0x1000:
CODE(?PR?*?FILENAME(1000h))

To locate a function at 0x1050:
CODE(?PR?FUNCTIONAME?FILENAME(1050h))

NOTE: The 'CODE' directive takes
multiple parameters separated by a
comma.
```

As a word of caution, when locating functions or segments manually, one should always examine the MAP file (projectname.M51) for each project to make sure that the linker has done what was intended and that there are no overlapping sections.

Firmware Updating Example

The software in this example can load any independently developed project through the UART. The code for the loader is located at addresses higher than 0x1000 in FLASH. This allows the loaded project the first 4096 bytes of flash to work with. It is not neccessary to keep the data segments from overlapping in this project because only one project will be running at any given time. The following list shows the steps taken to update the firmware.

1. Initially, the 'updater' project is downloaded using the IDE.

2. Any other project may be downloaded into the target any number of times using the IDE or the 'updater' as long as it does not write over the 'updater' project.

3. The 'updater' can be called from the firmware using a function pointer. The 'updater' erases the first 8 pages of FLASH, receives the updated firmware through the UART, and resets the device, which executes the newly downloaded code.

Step-by-Step to Building and Running the Example Selective Code Loader

The following list will guide you through getting the example selective code loader up and running. There are two versions of the application, one for the C8051F02x and one for the C8051F30x. Instructions for the 'F30x are shown.

1. Start the Silicon Labs IDE and add ‘loader_F30x.c’ to a new project. Compile, link, and download this project to the target.

2. Open a new Silicon Labs IDE project and add ‘blink_fast_F30x.c’.

3. Now we need to locate the new project’s data segment at 0x08 in RAM. This is the location of the reserved buffer in the ‘loader’ project. If either project uses the ‘USING’ directive, change the 0x08 to an unused area of memory. We can locate the new project’s data segment by adding the following directive to the command line parameters found in the ‘**Project->Tool Chain Integration...**’ menu under the ‘**linker**’ tab.

   ```
   DATA(08h)
   ```

   We also need to locate all functions in the project at addresses higher than 0x1000 and locate the ‘blink_fast’ function at 0x1000. Add the following argument to the linker command line parameters.

   ```
         CODE(1000h,
   ?PR?BLINK_FAST?BLINK_FAST_F30x
   (1000h))
   ```

4. Compile and link the project. Examine the MAP file (blink_fast_F30x.M51) to ensure that the data segment does not exceed the number of bytes reserved by the ‘loader’ project.

5. Run the ‘OH51.EXE’ utility with the linker output file (BLINK_FAST_F30x) as its argument. The OH51 utility can be found in the ‘**C:\SILICONLABS\IDEfiles\C51\Bin**’ folder.

6. Repeat steps 2 through 5 for ‘blink_slow_F30x’

7. Start the terminal program and configure it as shown in the previous sections. Hit ‘go’ in the ‘loader_F30x’ project. Go through the series of commands to erase, load, and execute the ‘blink_fast’ function. When prompted to send a HEX file, use the ‘**send text file**’ command to send the appropriate ‘*.hex’ file.

Step-by-Step to Building and Running the Example Firmware Updater

1. Start the Silicon Labs IDE and add ‘updater_F30x.c’ to a new project.

2. Add the following to the command line parameters found in the ‘**Project->Tool Chain Integration...**’ menu under the ‘**linker**’ tab. This argument defines the location of the CODE segment and locates the main routine at 0x1000.

   ```
   CODE(1000h,   ?PR?MAIN?UPDATER_F30x
   (1000h))
   ```

3. Compile, link, and download this project to the target. Once the project is downloaded, disconnect the IDE.

4. Start a new instance of the Silicon Labs IDE and add the correct version of ‘blink_F30x.c’ to a new project. Compile, link, and download this project to the target. The green LED should be blinking.

5. Run the ‘OH51.EXE’ utility with the linker output file (BLINK_F30x) as its argument. The OH51 utility can be found in the ‘**C:\SILICONLABS\IDEfiles\C51\Bin**’ folder.

6. Start the terminal program and configure it as shown in the previous sections. Press the P0.3 switch for the ‘F30x. When prompted to send a HEX file, use the ‘**send text file**’ command to send ‘blink_F30x.hex’ or a different HEX file.

Example Software for the C8051F02x Family

Selective Code Loader//---

```
// loader_F02x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 28 JUN 02
//
// This program shows an example ‘selective code loader’ using the ‘F02x. It
// designates the FLASH page at 0x1000 for the code loaded through the UART.
//
// Control Function:
//
// The system is controlled via the hardware UART, operating at a baud rate
// determined by the constant <BAUDRATE>, using Timer1 overflows as the baud
// rate source.
//
// Received File Type:
//
// This example receives Intel HEX files which are OMF51 (linker output files)
// passed through the OH51 utility in the ‘CYGNAL\IDEfiles\C51\Bin’ folder.
//
// Note: Because this program writes to FLASH, the MONEN pin should be tied
//       high.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>                     // printf() and getchar()
#include <ctype.h>                     // tolower() and toint()

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F02x
//-----------------------------------------------------------------------------
sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR3RL   = 0x92;                 // Timer3 reload value
sfr16 TMR3     = 0x94;                 // Timer3 counter
sfr16 ADC0     = 0xbe;                 // ADC0 data
sfr16 ADC0GT   = 0xc4;                 // ADC0 greater than window
sfr16 ADC0LT   = 0xc6;                 // ADC0 less than window
sfr16 RCAP2    = 0xca;                 // Timer2 capture/reload
sfr16 T2       = 0xcc;                 // Timer2
sfr16 RCAP4    = 0xe4;                 // Timer4 capture/reload
sfr16 T4       = 0xf4;                 // Timer4
sfr16 DAC0     = 0xd2;                 // DAC0 data
sfr16 DAC1     = 0xd5;                 // DAC1 data

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

```
// Global CONSTANTS
//-----------------------------------------------------------------------------
#define TRUE         1
#define FALSE        0

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     115200            // Baud rate of UART in bps

sbit LED = P1^6;                       // LED=’1’ means ON
sbit SW2 = P3^7;                       // SW2=’0’ means switch pressed

//-----------------------------------------------------------------------------
// Reserved Memory Space
//-----------------------------------------------------------------------------

char reserved_memory_bank[2] _at_ 0x08;// This memory bank is used by the
                                       // functions that will be loaded
                                       // through the UART.
                                       // The memory bank location and size
                                       // are based on values from the M51 map
                                       // file generated when the loaded code
                                       // is linked.

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

void main (void);

// Support Subroutines
void print_menu(void);
void erase_flash_page(void);
void receive_code(void);
unsigned char hex2char();

// Initialization Subroutines
void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);

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

#define input_str_len 4                // buffer to hold characters entered
char input_str[input_str_len];         // at the command prompt

void (*f)();                           // function pointer declaration

bit code_erased = FALSE;               // flag used to indicate that the FLASH
                                       // erase operation is complete
bit f_valid = FALSE;                   // flag to indicate that the FLASH
                                       // programming operation is complete

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

void main (void)
```

```
{

   WDTCN = 0xde;                        // disable watchdog timer
   WDTCN = 0xad;

   PORT_Init ();                        // initialize crossbar and GPIO
   SYSCLK_Init ();                      // initialize oscillator
   UART0_Init ();                       // initialize UART0

   print_menu();                        // print the command menu

   while (1){

      printf(“\nEnter a command > “);
      gets(input_str, input_str_len);

      switch ( input_str[0] ){

         case ‘1’: erase_flash_page();
                   printf(“\nFlash page 0x1000 has been erased.\n”);
                   break;

         case ‘2’: printf(“\nReady to receive HEX file...\n”);
                   receive_code();
                   break;

         case ‘3’: if(f_valid){
                     f = (void code *) 0x1000;
                     f();
                     printf(“\nFinished\n”);
                   } else {
                     printf(“\n*** No function exists at 0x1000.\n”);
                   }
                   break;

         case ‘?’: print_menu();
                   break;

         default:  printf(“\n*** Unknown Command.\n”);
                   break;
      }

   } // end while

} // end main

//-----------------------------------------------------------------------------
// Support Subroutines
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// print_menu
//-----------------------------------------------------------------------------
//
// This routine uses prints the command menu to the UART.
//
void print_menu(void)
{

   printf(“\n\nC8051F02x Selective Code Loader Example\n”);
   printf(“------------------------------------------\n”);
   printf(“1. Erase the flash page at 0x1000\n”);
   printf(“2. Receive HEX file\n”);
   printf(“3. Execute the function at 0x1000\n”);
   printf(“?. Print Command List\n”);
}

//-----------------------------------------------------------------------------
// hex2char
//-----------------------------------------------------------------------------
//
// This routine converts a two byte ascii representation of a char to an
// 8-bit variable;
//
unsigned char hex2char()
{

   unsigned char retval;
   char byteH, byteL;

   // get a two-byte ASCII representation of a char from the UART
   byteH = _getkey();
   byteL = _getkey();

   // convert to a single 8 bit result
   retval = (char) toint(byteH) * 16;
   retval += (char) toint(byteL);
   return retval;
}

//-----------------------------------------------------------------------------
// erase_flash_page
//-----------------------------------------------------------------------------
//
// This routine erases the FLASH page located at 0x1000
//
void erase_flash_page(void)
{
   bit EA_state;
   char xdata* data pagePointer = 0x1000; // pointer to xdata space located
                                          // in data space

   EA_state = EA;                   // holds interrupt state

   EA = 0;                          // disable interrupts
   FLSCL |= 0x01;                   // enable FLASH write/erase
   PSCTL  = 0x03;                   // MOVX erases FLASH

   // Erase the FLASH page at 0x1000
   *pagePointer = 0;                // initiate the erase

   PSCTL = 0x00;                    // MOVX writes target XRAM
   FLSCL &= ~0x01;                  // disable FLASH write/erase

   EA =  EA_state;                  // restore interrupt state

   f_valid = FALSE;                 // indicate that code is no longer valid
   code_erased = TRUE;              // indicate that FLASH has been erased
}
```

```
//-----------------------------------------------------------------------------
// receive_code
//-----------------------------------------------------------------------------
//
// This routine receives HEX records through the UART and writes the
// function located at 0x1000.
//
// Hex Record Format:
//
// +--------+--------+------+-------+--------+------(n bytes)------+----------+
// | RECORD | RECLEN |    OFFSET    | RECORD |                      | CHECKSUM |
// |  MARK  |  (n)   |   (2 BYTES)  |  TYPE  |        DATA          |          |
// |  ‘:’   |        |              |        |                      |          |
// +--------+--------+------+-------+--------+------(n bytes)------+----------+
//
void receive_code(void)
{

   char xdata* data pwrite;               // pointer used for writing FLASH
   char code* data pread;                 // pointer used for reading FLASH
   unsigned int  len;                     // holds the HEX record length field
   char record_type;                      // holds the HEX record type field
   unsigned int offset;                   // holds the HEX record offset field
                                          // this is the starting address of
                                          // the code image contained in the
                                          // record

   char checksum;                         // holds the HEX record checksum field
   char flash_checksum;                   // holds the checksum calculated after
                                          // the FLASH has been programmed
   bit EA_state;                          // temporary holder used to restore
                                          // interrupts to their previous state

   char c;                                // temporary char
   int i;                                 // temporary int

   // make sure the FLASH page has been erased
   if(!code_erased){
      printf(“\n*** At least one FLASH page must be erased prior”);
      printf(“ to this operation.\n”);
      return;
   }

   // wait for the user to send HEX file

   do{

      while( c = _getkey() != ‘:’ );      // ignore all characters until
                                          // reaching the record mark field

      // get the record length
      len = hex2char();

      // get the starting address (offset field in HEX record)
      offset = hex2char();
      offset <<= 8;
      offset |= hex2char();
      // get the record type
      record_type = hex2char();
      if( record_type != 0 && record_type != 1 ){
         printf(“\n*** Cannot decode HEX file.\n”);
         return;
      }

      EA_state = EA;                      // save the interrupt enable bit state

      EA = 0;                             // disable interrupts (precautionary)
      FLSCL |= 0x01;                      // enable FLASH write/erase
      PSCTL  = 0x01;                      // MOVX writes FLASH

      pwrite = (char xdata*) offset;      // initialize the write pointer

      code_erased = FALSE;                // clear the code_erased flag

      // write the record into FLASH
      for( i = 0; i < len; i++){
         *pwrite = hex2char();            // write one byte to FLASH
         pwrite++;                        // increment FLASH write pointer

      }

      PSCTL = 0x00;                       // MOVX writes target XRAM
      FLSCL &= ~0x01;                     // disable FLASH write/erase
      EA = EA_state;                      // restore interrupts to previous state

      // verify the checksum
      pread =  (char code*) offset;       // initialize the read pointer
      checksum = hex2char();              // get the HEX record checksum field
      flash_checksum = 0;                 // set the flash_checksum to zero

      // add the data field stored in FLASH to the checksum
      for( i = 0; i < len; i++){
         flash_checksum += *pread++;
      }

      // add the remaining fields
      flash_checksum += len;
      flash_checksum += (char) (offset >> 8);
      flash_checksum += (char) (offset & 0x00FF);
      flash_checksum += record_type;
      flash_checksum += checksum;

      // verify the checksum (the flash_checksum should equal zero)
      if(flash_checksum != 0){
         printf(“*** Checksum failed, try again.”);
         return;
      }

   } while(record_type != 1);

   f_valid = TRUE;                     // flag that the “f()” function is valid

   _getkey();                          // remove carriage return from input
                                       // stream

   printf(“\nReceived OK\n”);
```

```
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use an 22.1184MHz crystal
// as its clock source.
//
void SYSCLK_Init (void)
{
   int i;                              // delay counter

   OSCXCN = 0x67;                      // start external oscillator with
                                       // 22.1184MHz crystal

   for (i=0; i < 256; i++) ;           // wait for osc to start

   while (!(OSCXCN & 0x80)) ;          // Wait for crystal osc. to settle

   OSCICN = 0x88;                      // select external oscillator as SYSCLK
                                       // source and enable missing clock
                                       // detector
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR0    = 0x04;                     // Enable UART0
   XBR1    = 0x00;
   XBR2    = 0x40;                     // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x01;                    // enable TX0 as a push-pull output
   P1MDOUT |= 0x40;                    // enable P1.6 (LED) as push-pull output
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0   = 0x50;                     // SCON0: mode 1, 8-bit UART, enable RX
   TMOD    = 0x20;                     // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);     // set Timer1 reload value for baudrate
   TR1    = 1;                         // start Timer1
   CKCON |= 0x10;                      // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                      // SMOD00 = 1
   TI0    = 1;                         // Indicate TX0 ready
}
```

```
//-----------------------------------------------------------------------------
// blink_fast_F02x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 21 JUN 02
//
// This program shows an example function that can be used with the
// ‘selective code loader example’ for the ‘F02x family.
//
//
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F02x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR3RL   = 0x92;                 // Timer3 reload value
sfr16 TMR3     = 0x94;                 // Timer3 counter
sfr16 ADC0     = 0xbe;                 // ADC0 data
sfr16 ADC0GT   = 0xc4;                 // ADC0 greater than window
sfr16 ADC0LT   = 0xc6;                 // ADC0 less than window
sfr16 RCAP2    = 0xca;                 // Timer2 capture/reload
sfr16 T2       = 0xcc;                 // Timer2
sfr16 RCAP4    = 0xe4;                 // Timer4 capture/reload
sfr16 T4       = 0xf4;                 // Timer4
sfr16 DAC0     = 0xd2;                 // DAC0 data
sfr16 DAC1     = 0xd5;                 // DAC1 data

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------
#define TRUE         1
#define FALSE        0

#define SYSCLK       22118400          // SYSCLK frequency in Hz

sbit LED = P1^6;                       // LED=’1’ means ON
sbit SW2 = P3^7;                       // SW1=’0’ means switch pressed


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

// Subroutines that will be loaded at address 0x1000
void blink_fast();
void wait_ms(int ms);
void Timer2_Init (int counts);

//-----------------------------------------------------------------------------
// blink_fast
//-----------------------------------------------------------------------------
//
// This routine uses blinks the LED twice every second for five seconds.
//
void blink_fast(void)
{
   static int i;

   Timer2_Init (SYSCLK/12/1000);       // initialize Timer2 to overflow
                                       // every millisecond
   for( i = 0; i < 10; i++){
      LED = 0;                         // turn LED off
      wait_ms(150);                    // execute delay loop
      LED = 1;                         // turn LED on
      wait_ms(150);                    // execute delay loop
   }
}

//-----------------------------------------------------------------------------
// wait_ms
//-----------------------------------------------------------------------------
//
// This routine uses Timer 2 to insert a delay of <ms> milliseconds.
// Timer 2 overflows once every millisecond
//
void wait_ms(int ms)
{
   TF2 = 0;                            // clear Timer 2 overflow flag
   TR2 = 1;                            // turn Timer 2 On


   while (ms != 0){
      if(TF2){
         TF2 = 0;
         ms--;
      }
   }

   TR2 = 0;                            // turn Timer 2 Off

}

//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// This routine initializes Timer2 to 16 bit auto reload mode
//
void Timer2_Init (int counts)
{

   CKCON &= ~0x20;                     // Timer 2 counts SYSCLK/12
   RCAP2 = -(counts);                  // set the reload value
   T2 = RCAP2;                         // init Timer2
   ET2 = 0;                            // disable Timer2 interrupts
   TR2 = 0;                            // Timer 2 OFF
}
```

```
//-----------------------------------------------------------------------------
// blink_slow_F02x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 21 JUN 02
//
// This program shows an example function that can be used with the
// ‘selective code loader example’ for the ‘F02x family.
//
//
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F02x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR3RL   = 0x92;                 // Timer3 reload value
sfr16 TMR3     = 0x94;                 // Timer3 counter
sfr16 ADC0     = 0xbe;                 // ADC0 data
sfr16 ADC0GT   = 0xc4;                 // ADC0 greater than window
sfr16 ADC0LT   = 0xc6;                 // ADC0 less than window
sfr16 RCAP2    = 0xca;                 // Timer2 capture/reload
sfr16 T2       = 0xcc;                 // Timer2
sfr16 RCAP4    = 0xe4;                 // Timer4 capture/reload
sfr16 T4       = 0xf4;                 // Timer4
sfr16 DAC0     = 0xd2;                 // DAC0 data
sfr16 DAC1     = 0xd5;                 // DAC1 data

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------
#define TRUE         1
#define FALSE        0

#define SYSCLK       22118400          // SYSCLK frequency in Hz

sbit LED = P1^6;                       // LED=’1’ means ON
sbit SW2 = P3^7;                       // SW1=’0’ means switch pressed

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

// Subroutines that will be loaded at address 0x1000
void blink_slow();
void wait_ms(int ms);
void Timer2_Init (int counts);
```

```
//------------------------------------------------------------------------------
// blink_slow
//------------------------------------------------------------------------------
//
// This routine uses blinks the LED once every second for five seconds.
//
void blink_slow(void)
{
   static int i;

   Timer2_Init (SYSCLK/12/1000);        // initialize Timer2 to overflow
                                        // every millisecond
   for( i = 0; i < 10; i++){
      LED = 0;                          // turn LED off
      wait_ms(500);                     // execute delay loop
      LED = 1;                          // turn LED on
      wait_ms(500);                     // execute delay loop
   }
}

//------------------------------------------------------------------------------
// wait_ms
//------------------------------------------------------------------------------
//
// This routine uses Timer 2 to insert a delay of <ms> milliseconds.
// Timer 2 overflows once every millisecond
//
void wait_ms(int ms)
{
   TF2 = 0;                             // clear Timer 2 overflow flag
   TR2 = 1;                             // turn Timer 2 On


   while (ms != 0){
      if(TF2){
         TF2 = 0;
         ms--;
      }
   }

   TR2 = 0;                             // turn Timer 2 Off

}

//------------------------------------------------------------------------------
// Timer2_Init
//------------------------------------------------------------------------------
//
// This routine initializes Timer2 to 16 bit auto reload mode
//
void Timer2_Init (int counts)
{

   CKCON &= ~0x20;                      // Timer 2 counts SYSCLK/12
   RCAP2 = -(counts);                   // set the reload value
   T2 = RCAP2;                          // init Timer2
   ET2 = 0;                             // disable Timer2 interrupts
   TR2 = 0;                             // Timer 2 OFF
}
```

Example Firmware Updater

```
//-----------------------------------------------------------------------------
// updater_F02x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 28 JUN 02
//
// This program shows an example Firmware Updater using the ‘F02x. It resides
// in FLASH at addresses above 0x1000 and is accessed through a function
// pointer casted as (void code*) 0x1000.
//
// Once the firmware update has taken place, the a software reset is issued
// and the updated firmware takes control of the system.
//
// Control Function:
//
// The system is controlled via the hardware UART, operating at a baud rate
// determined by the constant <BAUDRATE>, using Timer1 overflows as the baud
// rate source.
//
// Note: Because this program writes to FLASH, the MONEN pin should be tied
//       high.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>                     // printf() and getchar()
#include <stdlib.h>
#include <ctype.h>                     // tolower() and toint()

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F02x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR3RL   = 0x92;                 // Timer3 reload value
sfr16 TMR3     = 0x94;                 // Timer3 counter
sfr16 ADC0     = 0xbe;                 // ADC0 data
sfr16 ADC0GT   = 0xc4;                 // ADC0 greater than window
sfr16 ADC0LT   = 0xc6;                 // ADC0 less than window
sfr16 RCAP2    = 0xca;                 // Timer2 capture/reload
sfr16 T2       = 0xcc;                 // Timer2
sfr16 RCAP4    = 0xe4;                 // Timer4 capture/reload
sfr16 T4       = 0xf4;                 // Timer4
sfr16 DAC0     = 0xd2;                 // DAC0 data
sfr16 DAC1     = 0xd5;                 // DAC1 data

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

```
#define TRUE         1
#define FALSE        0

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     115200            // Baud rate of UART in bps

sbit LED = P1^6;                       // LED=’1’ means ON
sbit SW2 = P3^7;                       // SW2=’0’ means switch pressed


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

void main (void);

// Support Subroutines
void print_menu(void);
void erase_flash(void);
void receive_code(void);
unsigned char hex2char();

// Initialization Subroutines
void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);


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

void (*f)();                           // function pointer declaration

bit code_erased = FALSE;               // flag used to indicate that the FLASH
                                       // erase operation is complete
bit f_valid = FALSE;                   // flag to indicate that the FLASH
                                       // programming operation is complete

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

void main (void)
{

   char input;

   WDTCN = 0xde;                       // disable watchdog timer
   WDTCN = 0xad;

   EA = 0;                             // disable interrupts (this statement
                                       // is needed because the device is not
                                       // neccesarily in a reset state prior
                                       // to executing this code)

   PORT_Init ();                       // initialize crossbar and GPIO
   SYSCLK_Init ();                     // initialize oscillator
   UART0_Init ();                      // initialize UART0
   print_menu();                       // print the command menu

   while (1){

      printf(“Enter a command > “);
      input = getchar();

      switch ( input ){

         case ‘1’: erase_flash();
                   printf(“\n*** Flash pages erased.\n”);
                   receive_code();

         case ‘2’: printf(“\n** RESETTING **\n\n”);
                   RSTSRC = 0x10;      // reset the device

         case ‘?’: print_menu();
                   break;

         default:  print_menu();
                   printf(“\n*** Unknown Command\n”);
                   break;
      }

   } // end while

} // end main


//-----------------------------------------------------------------------------
// Support Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// print_menu
//-----------------------------------------------------------------------------
//
// This routine prints the command menu to the UART.
//
void print_menu(void)
{

   printf(“\n\nC8051F02x Firmware Updater\n”);
   printf(“---------------------------------\n”);
   printf(“1. Erase FLASH and Update Firmware\n”);
   printf(“2. Cancel Firmware Update\n”);
   printf(“?. Print Command List\n”);

}

//-----------------------------------------------------------------------------
// hex2char
//-----------------------------------------------------------------------------
//
// This routine converts a two byte ascii representation of a char to an
// 8-bit variable;
//
unsigned char hex2char()
```

```
{

   unsigned char retval;
   char byteH, byteL;

   // get a two-byte ASCII representation of a char from the UART
   byteH = _getkey();
   byteL = _getkey();

   // convert to a single 8 bit result
   retval = (char) toint(byteH) * 16;
   retval += (char) toint(byteL);
   return retval;
}

//-----------------------------------------------------------------------------
// erase_flash
//-----------------------------------------------------------------------------
//
// This routine erases the first 8 pages of FLASH (0x0000 to 0x0FFF).
//
void erase_flash(void)
{
   char xdata* data pagePointer = 0;// a pointer to xdata located in data space
                                    // points to the first FLASH page that
                                    // will be erased

   int i;                           // temporary int
   bit EA_state;                    // holds interrupt state

   printf("\n*** Erasing flash from 0x0000 to 0x0FFF");

   EA_state = EA;                   // save interrupt state

   EA = 0;                          // disable interrupts
   FLSCL |= 0x01;                   // enable FLASH write/erase
   PSCTL  = 0x03;                   // MOVX erases FLASH

   // Erase the first 8 FLASH pages
   for (i = 0; i < 8; i++){

      *pagePointer = 0;             // initiate the erase

      pagePointer += 512;           // advance to next FLASH page
   }

   PSCTL = 0x00;                    // MOVX writes target XRAM
   FLSCL &= ~0x01;                  // disable FLASH write/erase

   EA =  EA_state;                  // restore interrupt state

   f_valid = FALSE;                 // indicate that code is no longer valid
   code_erased = TRUE;              // indicate that FLASH has been erased
}

//-----------------------------------------------------------------------------
// receive_code
//-----------------------------------------------------------------------------
// This routine receives the new firmware through the UART in HEX record
// format.
//
// Hex Record Format:
//
// +--------+--------+------+-------+--------+------(n bytes)------+----------+
// | RECORD | RECLEN |    OFFSET    | RECORD |                      | CHECKSUM |
// |  MARK  |  (n)   |  (2 BYTES)  |  TYPE  |         DATA         |          |
// |  ‘:’   |        |              |        |                      |          |
// +--------+--------+------+-------+--------+------(n bytes)------+----------+
//
//
void receive_code(void)
{
   char xdata* data pwrite;               // pointer used for writing FLASH
   char code* data pread;                 // pointer used for reading FLASH
   unsigned char len;                     // holds the HEX record length field
   unsigned char record_type;             // holds the HEX record type field
   unsigned int offset;                   // holds the HEX record offset field
                                          // this is the starting address of
                                          // the code image contained in the
                                          // record

   char checksum;                         // holds the HEX record checksum field
   char flash_checksum;                   // holds the checksum calculated after
                                          // the FLASH has been programmed
   bit EA_state;                          // temporary holder used to restore
                                          // interrupts to their previous state

   char c;                                // temporary char
   int i;                                 // temporary int

   // make sure FLASH has been erased
   if(!code_erased){
      printf(“\n*** At least one FLASH page must be erased prior”);
      printf(“ to this operation.\n”);
      return;
   } else {

      printf(“\nReady to receive...\n”);
   }

   // wait for the user send HEX file

   do{

      while( c = _getkey() != ‘:’ );      // ignore all characters until
                                          // reaching the record mark field

      // get the record length
      len = hex2char();

      // get the starting address (offset field in HEX record)
      offset = hex2char();                // get the MSB
      offset <<= 8;
      offset |= hex2char();               // get the LSB

      // get the record type
      record_type = hex2char();
      if( record_type != 0 && record_type != 1 ){
         printf(“\n*** Cannot decode HEX file.\n”);
         return;
      }

      EA_state = EA;                      // save the interrupt enable bit state

      EA = 0;                             // disable interrupts (precautionary)
      FLSCL |= 0x01;                      // enable FLASH write/erase
      PSCTL  = 0x01;                      // MOVX writes FLASH

      pwrite = (char xdata*) offset;      // initialize the write pointer


      code_erased = FALSE;                // clear the code_erased flag

      // write the record into flash
      for( i = 0; i < len; i++){

         // check for valid pointer
         if(pwrite < 0x1000){
            *pwrite = hex2char();         // write one byte to FLASH
            pwrite++;                     // increment FLASH write pointer
         } else {
            printf(“\n\nExceeded Code Space.\n”);  // print error message
         }
      }

      PSCTL = 0x00;                       // MOVX writes target XRAM
      FLSCL &= ~0x01;                     // disable FLASH write/erase
      EA = EA_state;                      // restore interrupts to previous state

      // verify the checksum
      pread =  (char code*) offset;       // initialize the read pointer
      checksum = hex2char();              // get the HEX record checksum field
      flash_checksum = 0;                 // set the flash_checksum to zero

      // add the data field stored in FLASH to the checksum
      for( i = 0; i < len; i++)
      {
         flash_checksum += *pread++;
      }

      // add the remaining fields
      flash_checksum += len;
      flash_checksum += (char) (offset >> 8);
      flash_checksum += (char) (offset & 0x00FF);
      flash_checksum += record_type;
      flash_checksum += checksum;

      // verify the checksum (the flash_checksum should equal zero)
      if(flash_checksum != 0){
         printf(“*** Checksum failed, try again”);
         return;
      }

   } while(record_type != 1);

   f_valid = TRUE;                        // indicate that download is valid

   printf(“\n** Firmware Update Complete. **\n”);
```

```
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use an 22.1184MHz crystal
// as its clock source.
//
void SYSCLK_Init (void)
{
   int i;                              // delay counter

   OSCXCN = 0x67;                      // start external oscillator with
                                       // 22.1184MHz crystal

   for (i=0; i < 256; i++) ;           // wait for osc to start

   while (!(OSCXCN & 0x80)) ;          // Wait for crystal osc. to settle

   OSCICN = 0x88;                      // select external oscillator as SYSCLK
                                       // source and enable missing clock
                                       // detector
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR0    = 0x04;                     // Enable UART0
   XBR1    = 0x00;
   XBR2    = 0x40;                     // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x01;                    // enable TX0 as a push-pull output
   P1MDOUT |= 0x40;                    // enable P1.6 (LED) as push-pull output
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0   = 0x50;                     // SCON0: mode 1, 8-bit UART, enable RX
   TMOD    = 0x20;                     // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);     // set Timer1 reload value for baudrate
   TR1    = 1;                         // start Timer1
   CKCON |= 0x10;                      // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                      // SMOD00 = 1
   TI0    = 1;                         // Indicate TX0 ready
}
```

```
//------------------------------------------------------------------------------------
// blink_F02x.c
//------------------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW, FB
// DATE: 28 JUN 02
//
// This program flashes the green LED on the C8051F020 target board about five times
// a second using the interrupt handler for Timer3.
// Target: C8051F02x
//
// Tool chain: KEIL Eval ‘c’
//

//------------------------------------------------------------------------------------
// Includes
//------------------------------------------------------------------------------------
#include <c8051f020.h>                    // SFR declarations

//------------------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F02x
//------------------------------------------------------------------------------------

sfr16 DP       = 0x82;                    // data pointer
sfr16 TMR3RL   = 0x92;                    // Timer3 reload value
sfr16 TMR3     = 0x94;                    // Timer3 counter
sfr16 ADC0     = 0xbe;                    // ADC0 data
sfr16 ADC0GT   = 0xc4;                    // ADC0 greater than window
sfr16 ADC0LT   = 0xc6;                    // ADC0 less than window
sfr16 RCAP2    = 0xca;                    // Timer2 capture/reload
sfr16 T2       = 0xcc;                    // Timer2
sfr16 RCAP4    = 0xe4;                    // Timer4 capture/reload
sfr16 T4       = 0xf4;                    // Timer4
sfr16 DAC0     = 0xd2;                    // DAC0 data
sfr16 DAC1     = 0xd5;                    // DAC1 data

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

#define SYSCLK 2000000                    // approximate SYSCLK frequency in Hz

sbit  LED = P1^6;                         // green LED: ‘1’ = ON; ‘0’ = OFF
sbit  SW2 = P3^7;                         // SW2=’0’ means switch pressed

//------------------------------------------------------------------------------------
// Function PROTOTYPES
//------------------------------------------------------------------------------------
void PORT_Init (void);
void Timer3_Init (int counts);
void Timer3_ISR (void);

//------------------------------------------------------------------------------------
// MAIN Routine
//------------------------------------------------------------------------------------
void main (void) {

   void (*update_firmware)();             // function pointer to firmware updating
```

```
                                          // code that is located at 0x1000;


   // disable watchdog timer
   WDTCN = 0xde;
   WDTCN = 0xad;

   PORT_Init ();
   Timer3_Init (SYSCLK / 12 / 10);        // Init Timer3 to generate interrupts
                                          // at a 10Hz rate.

   EA = 1;                                // enable global interrupts

   update_firmware = (void code*)0x1000;  // assign the function pointer

   while (1) {                            // spin forever

      if (!SW2){                          // wait for switch before calling
                                          // the firmware update procedure
         update_firmware();

      }
   }
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR2    = 0x40;                     // Enable crossbar and weak pull-ups
   P1MDOUT |= 0x40;                    // enable P1.6 (LED) as push-pull output
}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
//
// Configure Timer3 to auto-reload and generate an interrupt at interval
// specified by <counts> using SYSCLK/12 as its time base.
//
void Timer3_Init (int counts)
{
   TMR3CN = 0x00;                      // Stop Timer3; Clear TF3;
                                       // use SYSCLK/12 as timebase
   TMR3RL  = -counts;                  // Init reload values
   TMR3    = 0xffff;                   // set to reload immediately
   EIE2   |= 0x01;                     // enable Timer3 interrupts
   TMR3CN |= 0x04;                     // start Timer3
}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

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

```
// Timer3_ISR
//------------------------------------------------------------------------------------
// This routine changes the state of the LED whenever Timer3 overflows.
//
void Timer3_ISR (void) interrupt 14
{
   TMR3CN &= ~(0x80);                     // clear TF3
   LED = ~LED;                            // change state of LED
}
```

Example Software For the C8051F30x Family

Selective Code Loader

```
//-----------------------------------------------------------------------------
// loader_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 28 JUN 02
//
// This program shows an example ‘selective code loader’ using the ‘F30x. It
// designates the flash page at 0x1000 for the code loaded through the UART.
//
// Control Function:
//
// The system is controlled via the hardware UART, operating at a baud rate
// determined by the constant <BAUDRATE>, using Timer1 overflows as the baud
// rate source.
//
// Received File Type:
//
// This example receives Intel HEX files which are OMF51 (linker output files)
// passed through the OH51 utility in the ‘CYGNAL\IDEfiles\C51\Bin’ folder.
//
// Note: Because this program writes to FLASH, the VDD monitor is enabled in
// in the initialization routine.
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f300.h>                 // SFR declarations
#include <stdio.h>                     // printf() and getchar()
#include <ctype.h>                     // tolower() and toint()

//-----------------------------------------------------------------------------
// 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 TRUE         1
```

```
#define FALSE        0

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

sbit LED = P0^2;                       // LED='1' means ON
sbit SW2 = P0^3;                       // SW2='0' means switch pressed
sbit TX0 = P0^4;                       // UART0 TX pin
sbit RX0 = P0^5;                       // UART0 RX pin


//-----------------------------------------------------------------------------
// Reserved Memory Space
//-----------------------------------------------------------------------------

char reserved_memory_bank[2] _at_ 0x08;// This memory bank is used by the
                                       // functions that will be loaded
                                       // through the UART
                                       // The memory bank location and size
                                       // are based on values from the M51 map
                                       // file generated when the loaded code
                                       // is linked.

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

void main (void);

// Support Subroutines
void print_menu(void);
void erase_flash_page(void);
void receive_code(void);
unsigned char hex2char();

// Initialization Subroutines
void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);


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

#define input_str_len 4                // buffer to hold characters entered
char input_str[input_str_len];         // at the command prompt

void (*f)();                           // function pointer declaration

bit code_erased = FALSE;               // flag used to indicate that the FLASH
                                       // erase operation is complete
bit f_valid = FALSE;                   // flag to indicate that the FLASH
                                       // programming operation is complete

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

```
void main (void)
{

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

   PORT_Init ();                       // initialize crossbar and GPIO
   SYSCLK_Init ();                     // initialize oscillator
   UART0_Init ();                      // initialize UART0

   print_menu();                       // print the command menu

   while (1){

      printf(“\nEnter a command > “);
      gets(input_str, input_str_len);

      switch ( input_str[0] ){

         case ‘1’: erase_flash_page();
                   printf(“\nFlash page 0x1000 has been erased.\n”);
                   break;

         case ‘2’: printf(“\nReady to receive HEX file...\n”);
                   receive_code();
                   break;

         case ‘3’: if(f_valid){
                     f = (void code *) 0x1000;
                     f();
                     printf(“\nFinished\n”);
                   } else {
                     printf(“\n*** No function exists at 0x1000.\n”);
                   }
                   break;

         case ‘?’: print_menu();
                   break;

         default:  printf(“\n*** Unknown Command.\n”);
                   break;
      }

   } // end while

} // end main


//-----------------------------------------------------------------------------
// Support Subroutines
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// print_menu
//-----------------------------------------------------------------------------
//
// This routine uses prints the command menu to the UART.
//
void print_menu(void)
```

```
{

   printf(“\n\nC8051F30x Selective Code Loader Example\n”);
   printf(“------------------------------------------\n”);
   printf(“1. Erase the flash page at 0x1000\n”);
   printf(“2. Receive HEX file\n”);
   printf(“3. Execute the function at 0x1000\n”);
   printf(“?. Print Command List\n”);

}
//-----------------------------------------------------------------------------
// hex2char
//-----------------------------------------------------------------------------
//
// This routine converts a two byte ascii representation of a char to an
// 8-bit variable;
//
unsigned char hex2char()
{

   unsigned char retval;
   char byteH, byteL;

   // get a two-byte ASCII representation of a char from the UART
   byteH = _getkey();
   byteL = _getkey();

   // convert to a single 8 bit result
   retval = (char) toint(byteH) * 16;
   retval += (char) toint(byteL);
   return retval;
}

//-----------------------------------------------------------------------------
// erase_flash_page
//-----------------------------------------------------------------------------
//
// This routine erases the FLASH page located at 0x1000
//
void erase_flash_page(void)
{
   char xdata* data pagePointer = 0x1000; // pointer to xdata space located
                                          // in data space
   bit EA_state;                    // holds interrupt state

   PSCTL = 0x03;                    // MOVX erases FLASH

   FLKEY = 0xA5;                    // FLASH lock and key sequence 1
   FLKEY = 0xF1;                    // FLASH lock and key sequence 2

   // Erase the FLASH page at 0x1000
   *pagePointer = 0;                // initiate the erase

   PSCTL = 0;                       // MOVX writes target XRAM

   EA =  EA_state;                  // restore interrupt state

   f_valid = FALSE;                 // indicate that code is no longer valid
   code_erased = TRUE;              // indicate that FLASH has been erased
```

```
}

//-----------------------------------------------------------------------------
// receive_code
//-----------------------------------------------------------------------------
//
// This routine receives HEX records through the UART and writes the
// function located at 0x1000.
//
// Hex Record Format:
//
// +--------+--------+------+-------+--------+------(n bytes)------+----------+
// | RECORD | RECLEN |    OFFSET    | RECORD |                     | CHECKSUM |
// |  MARK  |  (n)   |   (2 BYTES)  |  TYPE  |        DATA         |          |
// |  ‘:’   |        |              |        |                     |          |
// +--------+--------+------+-------+--------+------(n bytes)------+----------+
//
void receive_code(void)
{
   char xdata* data pwrite;               // pointer used for writing FLASH
   char code* data pread;                 // pointer used for reading FLASH
   unsigned int  len;                     // holds the HEX record length field
   char record_type;                      // holds the HEX record type field
   unsigned int offset;                   // holds the HEX record offset field
                                          // this is the starting address of
                                          // the code image contained in the
                                          // record

   char checksum;                         // holds the HEX record checksum field
   char flash_checksum;                   // holds the checksum calculated after
                                          // the FLASH has been programmed
   bit EA_state;                          // temporary holder used to restore
                                          // interrupts to their previous state

   char c;                                // temporary char
   int i;                                 // temporary int


   // make sure the flash page has been erased
   if(!code_erased){
      printf(“\n*** At least one FLASH page must be erased prior to “);
      printf(“this operation.\n”);
      return;
   }

   // wait for the user to send HEX file

   do{

      while( c = _getkey() != ‘:’ );

      // get the length
      len = hex2char();

      // get the offset
      offset = hex2char();
      offset <<= 8;
      offset |= hex2char();
      // get the record type
      record_type = hex2char();
      if( record_type != 0 && record_type != 1 ){
         printf(“\n*** Cannot decode HEX file.\n”);
         return;
      }

      EA_state = EA;                      // save the interrupt enable bit state

      EA = 0;                             // disable interrupts (precautionary)
      PSCTL = 1;                          // MOVX writes to FLASH

      pwrite = (char xdata*) offset;      // initialize the write pointer

      code_erased = FALSE;                // clear the code_erased flag

      // write the record into flash
      for( i = 0; i < len; i++){
         FLKEY = 0xA5;                    // FLASH lock and key sequence 1
         FLKEY = 0xF1;                    // FLASH lock and key sequence 2
         *pwrite = hex2char();            // write one byte to FLASH
         pwrite++;                        // increment FLASH write pointer
      }

      PSCTL = 0;                          // MOVX writes target XRAM
      EA = EA_state;                      // restore interrupts to previous state

      // verify the checksum
      pread =  (char code*) offset;       // initialize the read pointer
      checksum = hex2char();              // get the HEX record checksum field
      flash_checksum = 0;                 // set the flash_checksum to zero

      // add the data field stored in FLASH to the checksum
      for( i = 0; i < len; i++)
      {
         flash_checksum += *pread++;
      }

      // add the remaining fields
      flash_checksum += len;
      flash_checksum += (char) (offset >> 8);
      flash_checksum += (char) (offset & 0x00FF);
      flash_checksum += record_type;
      flash_checksum += checksum;

      // verify the checksum (the flash_checksum should equal zero)
      if(flash_checksum != 0){
         printf(“*** Checksum failed, try again.”);
         return;
      }


   } while(record_type != 1);


   f_valid = TRUE;                  // flag that f() is valid
   _getkey();                       // clear carriage return
                                    // from the input stream

   printf(“\nReceived OK.\n”);
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the internal 24.5MHz
// oscillator as its clock source.  Enables missing clock detector reset. Also
// configures and enables the external crystal oscillator.
//
void SYSCLK_Init (void)
{

   OSCICN |= 0x03;                     // configure internal oscillator for
                                       // its maximum frequency
   RSTSRC = 0x06;                      // enable missing clock detector and
                                       // VDD monitor

}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports.
// P0.0 -
// P0.1 -
// P0.2 - LED (push-pull)
// P0.3 - SW2
// P0.4 - UART TX (push-pull)
// P0.5 - UART RX
// P0.6 -
// P0.7 - C2D
//
void PORT_Init (void)
{
   XBR0    = 0x04;                     // P0.2 skipped by the crossbar
   XBR1    = 0x03;                     // UART0 TX and RX pins enabled
   XBR2    = 0x40;                     // Enable crossbar and weak pull-ups
   P0MDIN &= ~0x00;                    // no analog inputs
   P0MDOUT |= 0x14;                    // enable TX0 and P0.2 as
                                       // push-pull output
}

//-----------------------------------------------------------------------------
// 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 |= 0x10;                   // T1M = 1; SCA1:0 = xx
   } else if (SYSCLK/BAUDRATE/2/256 < 4) {
      TH1 = -(SYSCLK/BAUDRATE/2/4);
      CKCON |=  0x01;                  // T1M = 0; SCA1:0 = 01
      CKCON &= ~0x12;
   } 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 |=  0x02;                  // T1M = 0; SCA1:0 = 10
      CKCON &= ~0x11;
   }

   TL1 = 0xff;                         // set Timer1 to overflow immediately
   TMOD &= ~0xf0;                      // TMOD: timer 1 in 8-bit autoreload
   TMOD |=  0x20;
   TR1 = 1;                            // START Timer1
   TI0 = 1;                            // Indicate TX0 ready
}
```

```
//-----------------------------------------------------------------------------
// blink_fast_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 28 JUN 02
//
// This program shows an example function that can be used with the
// ‘selective code loader example’ for the ‘F30x family.
//
//
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f300.h>                 // SFR declarations

//-----------------------------------------------------------------------------
// 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 TRUE         1
#define FALSE        0

#define SYSCLK       24500000          // SYSCLK frequency in Hz

sbit LED = P0^2;                       // LED=’1’ means ON
sbit SW2 = P0^3;                       // SW2=’0’ means switch pressed
sbit TX0 = P0^4;                       // UART0 TX pin
sbit RX0 = P0^5;                       // UART0 RX pin

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

// Subroutines that will be loaded at address 0x1000
void blink_fast();
void wait_ms(int ms);
void Timer2_Init (int counts);

//-----------------------------------------------------------------------------
// blink_fast
//-----------------------------------------------------------------------------
```

```
//
// This routine uses blinks the LED twice every second for five seconds.
//
void blink_fast(void)
{
   static int i;

   Timer2_Init(SYSCLK/12/1000);        // Initialize timer 2 to overflow every
                                       // millisecond
   for( i = 0; i < 10; i++){
      LED = 0;                         // turn LED off
      wait_ms(150);                    // execute delay loop
      LED = 1;                         // turn LED on
      wait_ms(150);                    // execute delay loop
   }
}

//-----------------------------------------------------------------------------
// wait_ms
//-----------------------------------------------------------------------------
//
// This routine uses Timer 2 to insert a delay of <ms> milliseconds.
// Timer 2 overflows once every millisecond
//
void wait_ms(int ms)
{
   TF2H = 0;                           // clear Timer 2 overflow flag
   TR2 = 1;                            // turn Timer 2 on

   while (ms != 0){
      if(TF2H){
         TF2H = 0;
         ms--;
      }
   }

   TR2 = 0;                            // turn Timer 2 Off

}

//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// This routine initializes Timer2 to 16 bit auto reload mode
//
void Timer2_Init (int counts)
{

   TMR2CN = 0x00;                      // Clear TF2H, TF2L; disable TF2L
                                       // interrupts; T2 in 16-bit mode;
                                       // Timer2 stopped; Timer2 prescaler
                                       // is set to EXTCLK/12
   CKCON &= ~0x60;                     // Timer 2 uses T2 prescaler as clock
                                       // source
   TMR2RL = -(counts);                 // set the reload value
   TMR2 = TMR2RL;                      // init Timer2
   ET2 = 0;                            // disable Timer2 interrupts

}
```

```
//-----------------------------------------------------------------------------
// blink_slow_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 28 JUN 02
//
// This program shows an example function that can be used with the
// ‘selective code loader example’ for the ‘F30x family.
//
//
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f300.h>                 // SFR declarations

//-----------------------------------------------------------------------------
// 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 TRUE         1
#define FALSE        0

#define SYSCLK       24500000          // SYSCLK frequency in Hz

sbit LED = P0^2;                       // LED=’1’ means ON
sbit SW2 = P0^3;                       // SW2=’0’ means switch pressed
sbit TX0 = P0^4;                       // UART0 TX pin
sbit RX0 = P0^5;                       // UART0 RX pin


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

// Subroutines that will be loaded at address 0x1000
void blink_slow();
void wait_ms(int ms);
void Timer2_Init (int counts);
```

```
//-----------------------------------------------------------------------------
// blink_slow
//-----------------------------------------------------------------------------
//
// This routine uses blinks the LED once every second for five seconds.
//
void blink_slow(void)
{
   static int i;

   Timer2_Init(SYSCLK/12/1000);        // Initialize timer 2 to overflow every
                                       // millisecond
   for( i = 0; i < 10; i++){
      LED = 0;                         // turn LED off
      wait_ms(500);                    // execute delay loop
      LED = 1;                         // turn LED on
      wait_ms(500);                    // execute delay loop
   }

}

//-----------------------------------------------------------------------------
// wait_ms
//-----------------------------------------------------------------------------
//
// This routine uses Timer 2 to insert a delay of <ms> milliseconds.
// Timer 2 overflows once every millisecond
//
void wait_ms(int ms)
{
   TF2H = 0;                           // clear Timer 2 overflow flag
   TR2 = 1;                            // turn Timer 2 on

   while (ms != 0){
      if(TF2H){
         TF2H = 0;
         ms--;
      }
   }

   TR2 = 0;                            // turn Timer 2 Off

}

//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// This routine initializes Timer2 to 16 bit auto reload mode
//
void Timer2_Init (int counts)
{

   TMR2CN = 0x00;                      // Clear TF2H, TF2L; disable TF2L
                                       // interrupts; T2 in 16-bit mode;
                                       // Timer2 stopped; Timer2 prescaler
                                       // is set to EXTCLK/12
   CKCON &= ~0x60;                     // Timer 2 uses T2 prescaler as clock
                                       // source
   TMR2RL = -(counts);                    // set the reload value
   TMR2 = TMR2RL;                         // init Timer2
   ET2 = 0;                               // disable Timer2 interrupts

}
```

Example Firmware Updater

```
//-----------------------------------------------------------------------------
// updater_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 28 JUN 02
//
// This program shows an example Firmware Updater using the ‘F300. It resides
// in FLASH at addresses above 0x1000 and is accessed through a function
// pointer casted as (void code*) 0x1000.
//
// Once the firmware update has taken place, the a software reset is issued
// and the updated firmware takes control of the system.
//
// Control Function:
//
// The system is controlled via the hardware UART, operating at a baud rate
// determined by the constant <BAUDRATE>, using Timer1 overflows as the baud
// rate source.
//
// Note: Because this program writes to FLASH, the VDD monitor is enabled in
// in the initialization routine.
//
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f300.h>                 // SFR declarations
#include <stdio.h>                     // printf() and getchar()
#include <stdlib.h>
#include <ctype.h>                     // tolower() and toint()

//-----------------------------------------------------------------------------
// 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 TRUE         1
#define FALSE        0

#define SYSCLK       24500000          // SYSCLK frequency in Hz
```

```
#define BAUDRATE     115200            // Baud rate of UART in bps

sbit LED = P0^2;                       // LED=’1’ means ON
sbit SW2 = P0^3;                       // SW2=’0’ means switch pressed
sbit TX0 = P0^4;                       // UART0 TX pin
sbit RX0 = P0^5;                       // UART0 RX pin


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

void main (void);

// Support Subroutines
void print_menu(void);
void erase_flash(void);
void receive_code(void);
unsigned char hex2char();

// Initialization Subroutines
void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);


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

void (*f)();                           // function pointer declaration

bit code_erased = FALSE;               // flag used to indicate that the FLASH
                                       // erase operation is complete
bit f_valid = FALSE;                   // flag to indicate that the FLASH
                                       // programming operation is complete

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

void main (void)
{

   char input;


   EA = 0;                             // Disable interrupts (precautionary)

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

   PORT_Init ();                       // initialize crossbar and GPIO
   SYSCLK_Init ();                     // initialize oscillator
   UART0_Init ();                      // initialize UART0

   print_menu();                       // print the command menu

   while (1){
      printf(“Enter a command > “);
      input = getchar();

      switch ( input ){

         case ‘1’: erase_flash();
                   printf(“\n*** Flash pages erased\n”);
                   receive_code();
                   printf(“\n** Firmware Update Complete **\n”);

         case ‘2’: printf(“\n** RESETTING **\n\n”);
                   RSTSRC = 0x10;      // reset the device

         case ‘?’: print_menu();
                   break;

         default: print_menu();
                  printf(“\n*** Unknown Command\n”);
      }

   } // end while

} // end main


//-----------------------------------------------------------------------------
// Support Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// print_menu
//-----------------------------------------------------------------------------
//
// This routine prints the command menu to the UART.
//
void print_menu(void)
{

   printf(“\n\nC8051F30x Firmware Updater\n”);
   printf(“---------------------------------\n”);
   printf(“1. Erase FLASH and Update Firmware\n”);
   printf(“2. Cancel Firmware Update\n”);
   printf(“?. Print Command List\n”);

}

//-----------------------------------------------------------------------------
// hex2char
//-----------------------------------------------------------------------------
//
// This routine converts a two byte ascii representation of a char to an
// 8-bit variable;
//
unsigned char hex2char()
{

   unsigned char retval;
   char byteH, byteL;
   // get a two-byte ASCII representation of a char from the UART
   byteH = _getkey();
   byteL = _getkey();

   // convert to a single 8 bit result
   retval = (char) toint(byteH) * 16;
   retval += (char) toint(byteL);
   return retval;
}

//-----------------------------------------------------------------------------
// erase_flash
//-----------------------------------------------------------------------------
//
// This routine erases the first 8 pages of FLASH (0x0000 to 0x0FFF).
//
void erase_flash(void)
{
   char xdata* data pagePointer = 0;// a pointer to xdata located in data space
                                    // points to the first FLASH page that
                                    // will be erased

   int i;                           // temporary int
   bit EA_state;                    // holds interrupt state

   printf(“\n*** Erasing flash from 0x0000 to 0x0FFF”);

   EA_state = EA;                   // save interrupt state

   PSCTL = 3;                       // MOVX erases FLASH

   // Erase the first 8 FLASH pages
   for (i = 0; i < 8; i++){
      FLKEY = 0xA5;                 // FLASH lock and key sequence 1
      FLKEY = 0xF1;                 // FLASH lock and key sequence 2

      *pagePointer = 0;             // initiate the erase

      pagePointer += 512;
   }

   PSCTL = 0;                       // MOVX writes target XRAM

   EA =  EA_state;                  // restore interrupt state

   f_valid = FALSE;                 // indicate that code is no longer valid
   code_erased = TRUE;              // indicate that FLASH has been erased
}

//-----------------------------------------------------------------------------
// receive_code
//-----------------------------------------------------------------------------
// This routine receives the new firmware through the UART in HEX record
// format.
//
// Hex Record Format:
//
// +--------+--------+------+-------+-------+------(n bytes)------+----------+
// | RECORD | RECLEN |    OFFSET     | RECORD |                     | CHECKSUM |
// |  MARK  |  (n)   |   (2 BYTES)  |  TYPE  |        DATA         |          |
// |  ‘:’   |        |             |        |                     |          |
// +--------+--------+------+-------+--------+------(n bytes)------+----------+
//
//
void receive_code(void)
{
   char xdata* data pwrite;               // pointer used for writing FLASH
   char code* data pread;                 // pointer used for reading FLASH
   unsigned char len;                     // holds the HEX record length field
   unsigned char record_type;             // holds the HEX record type field
   unsigned int offset;                   // holds the HEX record offset field
                                          // this is the starting address of
                                          // the code image contained in the
                                          // record

   char checksum;                         // holds the HEX record checksum field
   char flash_checksum;                   // holds the checksum calculated after
                                          // the FLASH has been programmed
   bit EA_state;                          // temporary holder used to restore
                                          // interrupts to their previous state

   char c;                                // temporary char
   int i;                                 // temporary int

   // make sure FLASH has been erased
   if(!code_erased){
      printf(“\n*** At least one FLASH page must be erased prior to this operation\n”);
      return;
   } else {

      printf(“\nReady to receive...\n”);
   }

   // wait for the user send HEX file

   do{

      while( c = _getkey() != ‘:’ );      // ignore all characters until
                                          // reaching the record mark field

       // get the record length
      len = hex2char();

       // get the starting address (offset field in HEX record)
      offset = hex2char();                // get the MSB
      offset <<= 8;
      offset |= hex2char();               // get the LSB


      // get the record type
      record_type = hex2char();
      if( record_type != 0 && record_type != 1 ){
         printf(“\n*** Cannot decode HEX file.\n”);
         return;
      }
      EA_state = EA;                      // save the interrupt enable bit state
      EA = 0;                             // disable interrupts (precautionary)

      PSCTL = 1;                          // MOVX writes to FLASH

      pwrite = (char xdata*) offset;      // initialize the write pointer


      code_erased = FALSE;                // clear the code_erased flag

      // write the record into FLASH
      for( i = 0; i < len; i++){

         // check for valid pointer
         if(pwrite < 0x1000){
            FLKEY = 0xA5;                 // FLASH lock and key sequence 1
            FLKEY = 0xF1;                 // FLASH lock and key sequence 2
            *pwrite = hex2char();         // write one byte to FLASH
            pwrite++;                     // increment FLASH write pointer
         } else {
            printf(“\n\nExceeded Code Space.\n”);  // print error message
         }
     }

      PSCTL = 0;                          // MOVX writes target XRAM
      EA = EA_state;                      // restore interrupts to previous state

      // verify the checksum
      pread =  (char code*) offset;       // initialize the read pointer
      checksum = hex2char();              // get the HEX record checksum field
      flash_checksum = 0;                 // set the flash_checksum to zero

      // add the data field stored in FLASH to the checksum
      for( i = 0; i < len; i++)
      {
         flash_checksum += *pread++;
      }

      // add the remaining fields
      flash_checksum += len;
      flash_checksum += (char) (offset >> 8);
      flash_checksum += (char) (offset & 0x00FF);
      flash_checksum += record_type;
      flash_checksum += checksum;

      // verify the checksum (the flash_checksum should equal zero)
      if(flash_checksum != 0){
         printf(“*** checksum failed, try again”);
         return;
      }


   } while(record_type != 1);


   f_valid = TRUE;                        // indicate that download is valid

}
```

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the internal 24.5MHz
// oscillator as its clock source.  Enables missing clock detector reset and
// VDD monitor.
//
void SYSCLK_Init (void)
{

   OSCICN |= 0x03;                     // configure internal oscillator for
                                       // its maximum frequency
   RSTSRC = 0x06;                      // enable missing clock detector and
                                       // VDD monitor

}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports.
// P0.0 -
// P0.1 -
// P0.2 - LED (push-pull)
// P0.3 - SW2
// P0.4 - UART TX (push-pull)
// P0.5 - UART RX
// P0.6 -
// P0.7 - C2D
//
void PORT_Init (void)
{
   XBR0    = 0x04;                     // P0.2 skipped by the crossbar
   XBR1    = 0x03;                     // UART0 TX and RX pins enabled
   XBR2    = 0x40;                     // Enable crossbar and weak pull-ups
   P0MDIN &= ~0x00;                    // no analog inputs
   P0MDOUT |= 0x14;                    // enable TX0 and P0.2 as
                                       // push-pull output
}

//-----------------------------------------------------------------------------
// 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 |= 0x10;                   // T1M = 1; SCA1:0 = xx
   } else if (SYSCLK/BAUDRATE/2/256 < 4) {
      TH1 = -(SYSCLK/BAUDRATE/2/4);
      CKCON |=  0x01;                  // T1M = 0; SCA1:0 = 01
      CKCON &= ~0x12;
   } 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 |=  0x02;                  // T1M = 0; SCA1:0 = 10
      CKCON &= ~0x11;
   }

   TL1 = 0xff;                         // set Timer1 to overflow immediately
   TMOD &= ~0xf0;                      // TMOD: timer 1 in 8-bit autoreload
   TMOD |=  0x20;
   TR1 = 1;                            // START Timer1
   TI0 = 1;                            // Indicate TX0 ready
}
```

```
//-----------------------------------------------------------------------------
// blink_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW, FB
// DATE: 28 JUN 02
//
// This program flashes the green LED on the C8051F30x target board about
// five times a second using the interrupt handler for Timer2.
//
// Target: C8051F30x
//
// Tool chain: KEIL Eval ‘c’
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f300.h>                    // SFR declarations

//-----------------------------------------------------------------------------
// 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 / 8         // SYSCLK frequency in Hz

sbit LED = P0^2;                          // LED=’1’ means ON
sbit SW2 = P0^3;                          // SW2=’0’ means switch pressed

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void SYSCLK_Init (void);
void PORT_Init (void);
void Timer2_Init (int counts);
void Timer2_ISR (void);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {

   void (*update_firmware)();             // function pointer to firmware
                                          // updating code that is located
                                          // at 0x1000;
   // disable watchdog timer
   PCA0MD &= ~0x40;                       // WDTE = 0 (clear watchdog timer
                                          // enable)

   SYSCLK_Init ();                        // Initialize system clock to
                                          // 24.5MHz internal oscillator

   PORT_Init ();                          // Initialize crossbar and GPIO
   Timer2_Init (SYSCLK / 12 / 10);        // Init Timer2 to generate
                                          // interrupts at a 10Hz rate.

   EA = 1;                                // enable global interrupts

   update_firmware = (void code*) 0x1000; // assign the function pointer

   while (1) {                            // spin forever

      if (!SW2){
         update_firmware();

      }
   }
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the internal 24.5MHz / 8
// oscillator as its clock source.  Also enables missing clock detector reset
// and the VDD Monitor.
//
// NOTE: This program must not disable the VDD monitor since it is enabled by
// Firmware Updater.  If this program disables the VDD monitor, there is
// potential for going into an infinite loop turning the VDD monitor on
// and off.
//
void SYSCLK_Init (void)
{
   OSCICN = 0x04;                         // configure internal oscillator for
                                          // its lowest frequency
   RSTSRC = 0x06;                         // enable missing clock detector
                                          // and VDD Monitor.
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports.
// P0.0 -
// P0.1 -
// P0.2 - LED (push-pull)
// P0.3 - SW2
// P0.4 -
// P0.5 -
// P0.6 -
// P0.7 - C2D
//
```

```
void PORT_Init (void)
{
   XBR0     = 0x04;                       // skip P0.2 (LED) in crossbar pin
                                          // assignments
   XBR1     = 0x00;                       // no digital peripherals selected
   XBR2     = 0x40;                       // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x04;                       // enable LED as a push-pull output
}

//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// Configure Timer2 to 16-bit auto-reload and generate an interrupt at
// interval specified by <counts> using SYSCLK/12 as its time base.
//
void Timer2_Init (int counts)
{
   TMR2CN  = 0x00;                        // Stop Timer2; Clear TF2;
                                          // use SYSCLK/12 as timebase
   CKCON  &= ~0x60;                       // Timer2 clocked based on T2XCLK;

   TMR2RL  = -counts;                     // Init reload values
   TMR2    = 0xffff;                      // set to reload immediately
   ET2     = 1;                           // enable Timer2 interrupts
   TR2     = 1;                           // start Timer2
}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Timer2_ISR
//-----------------------------------------------------------------------------
// This routine changes the state of the LED whenever Timer2 overflows.
//
void Timer2_ISR (void) interrupt 5
{
   TF2H = 0;                              // clear Timer2 interrupt flag
   LED = ~LED;                            // change state of LED
}
```

Notes:

Contact Information

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: productinfo@silabs.com
Internet: www.silabs.com

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.