CODE BANKING USING THE KEIL 8051 TOOLS

1. Relevant Devices

This application note applies to the following devices:

C8051F120, C8051F121, C8051F122, C8051F123, C8051F124, C8051F125, C8051F126, and C8051F127.

Introduction

The 8051 architecture supports a 64KB linear program memory space. Devices that have more than 64KB of program memory implement a code banking scheme to surmount this 64KB limit. This application note discusses software project management techniques and provides example applications that use code banking.

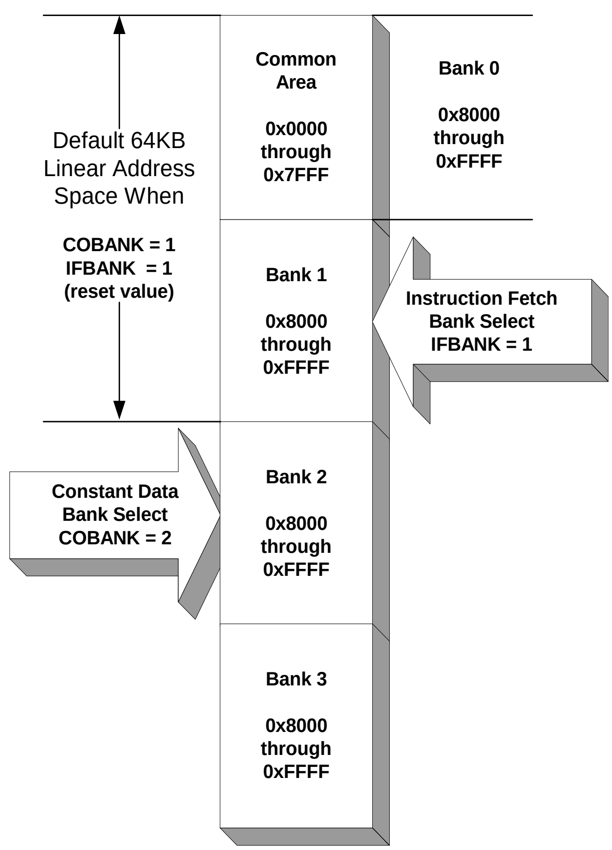
Key Points

- Projects requiring less than 64KB of FLASH can leave the PSBANK register at its default setting which provides a 64KB linear address space.
- Code banked projects with more than 64KB of program code typically contain multiple source files, one or more for each code bank.

Code Banking Overview

The C8051F12x family of devices has 128KB of on-chip FLASH, divided into 4 physical 32KB banks. This program memory space can be used to hold executable code or constant data. Figure 1 shows the code banking model implemented by these devices. Instruction fetch operations (normal code execution) are handled independently of constant data operations (MOVC instructions, and MOVX instructions when used for writing to FLASH). Each type of operation has its own bank select bits that may select any of the 4 banks as shown in Figure 1. All code bank switching is handled at the device level by writing to the PSBANK register. The COBANK and IFBANK bits in this register control switching for constant code accesses and instruction fetches, respectively. For more information on code bank switching, please refer to the C8051F12x datasheet.

Figure 1. C8051F12x Code Banking Model



For projects that require more than 64KB of code space or non-volatile data space, the user has the option of manually handling the bank switching in software or setting up a code- banked project. Both methods are discussed in this note.

User-Managed Bank Switching for Data Intensive Projects

User-managed bank switching is useful for projects that have less than 64KB of executable code but need to store large amounts of data in FLASH. In this situation, the Common area and Bank 1 are used for program memory while Bank 2 and Bank 3 are used for data storage. The project does not need to be set up for code banking.

The following data logging example shows how bank switching can be managed in application software.

Example 1: Data Logging Application

This application uses a 22.1184 MHz crystal oscillator to implement a software real-time clock (RTC). PCA0, configured to count Timer 0 overflows, generates an interrupt once every second. The interrupt handler records the current time and device temperature in a non-volatile log in FLASH.

The 112,640 byte log cycles through all 4 code banks recording time and temperature. Each data record has 6 fields as shown in Figure 2. The log is capable of storing 14080 records over a time period of 3.9 hours. Once the log is full, it continues logging at the beginning of log, erasing the FLASH page with the oldest data as it progresses.

Managing the Instruction Fetch Bank Select

Since this application uses less than 32KB of FLASH for program code, there will be no instruction fetches from the 0x8000 to 0xFFFF memory space. This makes the value of IFBANK a “don’t care”. However, if an application uses between 32KB and 64KB of FLASH for program code, IFBANK should be left at its reset value, targeting Bank 1.

Advancing Through the Code Banks

This application reserves the first 16KB of FLASH in the Common area for program code. The log starts at address 0x4000 in the Common area and ends at location 0xF7FF in Bank 3 as shown in

Figure 2. Log Record Structure

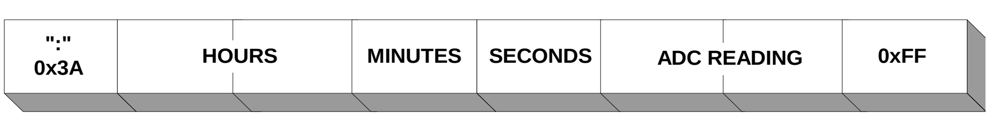


Figure 3.

After storing a record in the log, the FLASH write pointer is advanced to the next record and checked for code bank boundaries. There are three possible boundary conditions to consider when adjusting the FLASH write pointer. These cases are outlined in

Figure 3. FLASH Memory Map for Example 1

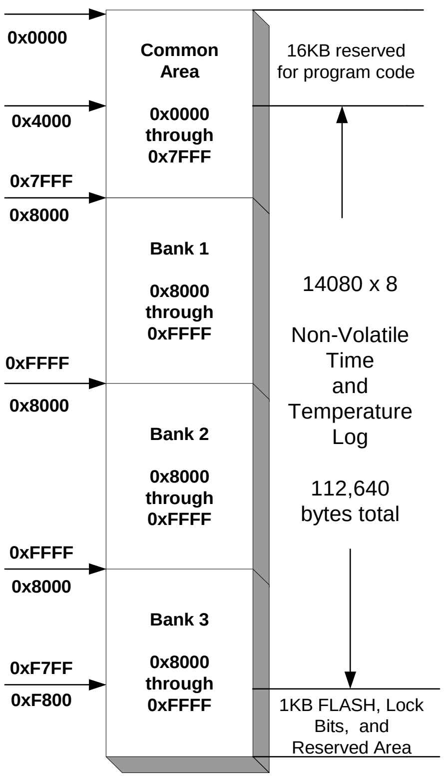


Table 1 .

Preserving the PSBANK Register in Functions and Interrupt Service Routines

A program must preserve and restore the value of the PSBANK register in every function and interrupt service routine that switches code banks.

Choosing Log Record Size

Example 1 only writes entire records to FLASH. If the record size is a power of 2 and the log starts at the beginning of a FLASH page, then all records will be contained within one of the code banks. If a record can cross a bank boundary, then bounds checking must be performed after every byte write.

Keeping Accurate Time

This application keeps track of time by implementing an interrupt driven real-time clock. With SYSCLK at 49.7664 MHZ, Timer 0 in mode 2 overflows exactly 4050 times every second when clocked by SYSCLK/48. PCA Module 0 is configured in “Software Timer Mode” to count Timer 0 overflows and generate an interrupt every second.

Table 1. FLASH Write Pointer Boundary Conditions

Condition	How to Detect	Typical Action
FLASH write pointer reaches the end of the Common area.	FLASH write pointer will point to location 0x8000.	No action is necessary if COBANK is always set to Bank 1 whenever the pointer is moved to the beginning of the log.
FLASH write pointer reaches the end of Bank 1 or Bank 2.	FLASH write pointer will point to location 0x0000.	FLASH write pointer should be set to 0x8000 and COBANK should be incremented.
FLASH write pointer reaches the end of the log.	FLASH write pointer will point to location 0xF800 and Bank 3 will be selected by COBANK.	FLASH write pointer should be reset to the first location in the log (0x4000) and COBANK should select Bank 1.

Project-Managed Bank Switching for Code-Intensive Applications

The Keil 8051 development tools support code banking by generating code that can automatically switch and preserve the PSBANK register. This allows the user to expand 64KB projects to 128KB without modifying existing modules. Using the code banking capability of the Silicon Labs IDE or the Keil µVision2 for projects containing more than 64KB of program code is recommended to take advantage of this feature.

To use the Keil 8051 tools for code banking, the project needs to be configured. The configuration required for code banking is supported in Version 1.7 and later of the Silicon Labs IDE and Version 2.00 and later of µVision2. Step-by-step instructions on how to configure a Silicon Labs IDE project and a µVision2 project for code banking are included in example 2.

Code banked projects typically contain multiple source files, one or more for each code bank. This greatly simplifies project configuration, as will be shown in example 2. In addition to source files, all projects configured for code banking must include a code banked version of STARTUP.A51 and L51_BANK.A51. These files can be found in the Silicon Labs IDE installation directory, 'Cygnal\Examples\C8051F12x\Asm'

Example 2: Project-Managed Code Banking

This example shows how to set up a code banked project using the Keil development tools. It uses Timer 3 and Timer 4 interrupts to blink the LED and output a 1 kHz sine wave on DAC1, respectively. The code that blinks the LED is located in Bank 3 and the code that outputs a sine wave is located in Bank 2. Since interrupts must be located in the Common area, both interrupts call a function in one of the banks to perform the desired task.

This example contains three source files, one for each bank and the Common area, and two required project files, as listed in Table 2 .

Table 2. Files needed by Example 2

common.c
bank2.c
bank3.c
STARTUP.A51
L51_BANK.A51

Step by Step Instructions on Configuring a Code Banked Project Using the Silicon Labs IDE

The following steps show how to configure the code banked example project using the Silicon Labs IDE.

1. Start the Silicon Labs IDE and add the files listed in Table 2 to a new project. Additionally, an example project is included named 'example2.wsp'.

2. Open the *Tool Chain Integration* window from the *Project* menu and select the *Linker* tab as shown in Figure 4. Click on the *Customize* button. A window similar to the one shown in Figure 5 should appear on the screen. Select the processor you are using from the pull down menu. This enables code banking by adding the *BANKAREA* directive to the linker command line parameters. For more information on the *BANKAREA* directive, please refer to Chapter 9 of the Keil Assembler/Linker manual,

accessible in PDF format from the *Help* menu of the Silicon Labs IDE.

Figure 4. Tool Chain Integration Window

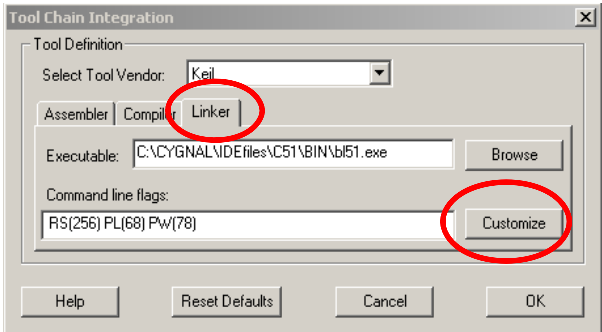
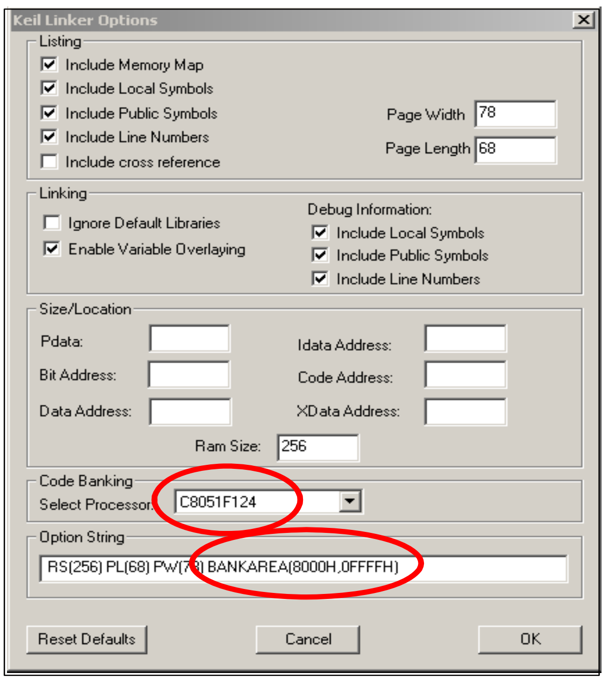


Figure 5. Linker Customization Window



3. Open the *Target Build Configuration* window from the *Project* menu and click the *Customize* button, as shown in Figure 6. The window shown in Figure 7 should appear on the screen.

Select the *Files to Link* tab.

Figure 6. Target Build Configuration Window

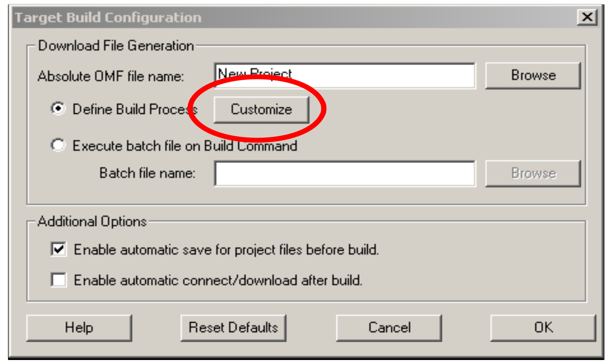
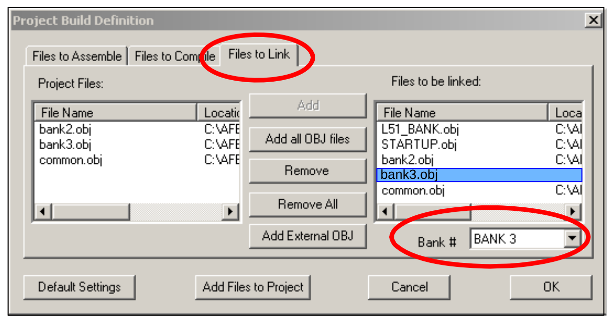


Figure 7. Project Build Definition Window



4. For each file in the *Files To Be Linked* list box, select the desired bank from the *Bank #* pull down menu. The 'L51_BANK.obj' and 'STARTUP.obj' files should be located in the Common area in all projects. The remaining project files should be located in their respective code banks, as shown in Table 3 .

Table 3. Code Bank Selection for Example 2

Filename	Code Bank
common.obj	Common area
bank2.obj	Bank 2
bank3.obj	Bank 3

Table 3. Code Bank Selection for Example 2

Filename	Code Bank
STARTUP.obj	Common area
L51_BANK.obj	Common area

5. Save the project by selecting *Save Project* from the *Project* menu.

6. Build the project by selecting *Build/Make Project* from the *Project* menu.

7. If the project has been configured correctly, an ‘example2.M51’ map file will be generated in the project folder. Inspect this file to verify that functions have been located in the proper bank. You should also notice that the sine table in ‘bank2.c’ has been located in the Common area because it is a constant code variable. Refer to the Keil linker manual for a description of the M51 file.

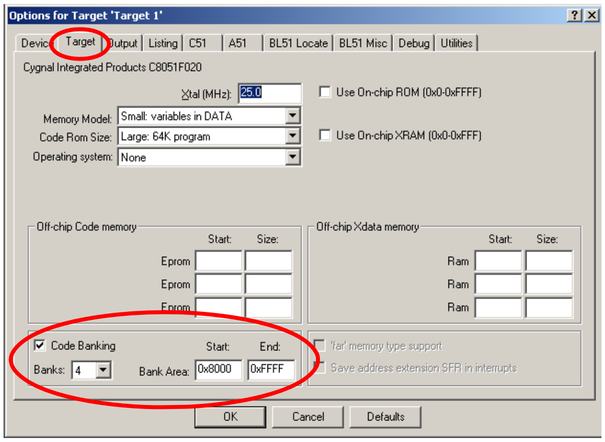
Step by Step Instructions on Configuring a Code Banked Project Using µVision2.

The following steps show how to configure the code banked example project using the Keil IDE, µVision2.

1. Start µVision2, create a new project and add the files listed in Table 2 to the project. Additionally, an example project is included named ‘example2.Uv2’.

2. Open the *Options for Target* window from the *Project* menu and select the *Target* tab, as shown in Figure 8. Click on the *Code Banking* check box at the bottom on the window. Next, select the number *4* from the *Banks:* pull down menu. Finally, edit the *Bank Area:* text boxes with a value of *0x8000* for the *Start:* value and *0xFFFF* for the *End:* value. When the selections match those shown in Figure 8, press the *OK* button to close the window.

Figure 8. Options for Target Window



3. Each file now needs to be assigned to a code bank or to the common area. Additionally, groups of files can be created and banking assignments made for the entire group. For each file/group in the *Project Workspace* window, right-click on the file-name/group-name and select *Options for File/Options for Group*, as shown in Figure 9. The window shown in Figure 10 should appear on the screen. Select the desired bank from the *Code Bank:* pull down menu. The ‘L51_BANK.obj’ and ‘STARTUP.obj’ files should be located in the Common area in all projects. The remaining project files should be located in their respective code banks, as shown in Table 4 .

4. Build the project by selecting *Build target* or *Rebuild all target files* from the *Project* menu.

5. If the project has been configured correctly, an ‘example2.M51’ map file will be generated in the project folder. Inspect this file to verify that functions have been located in the proper bank. You should also notice that the sine table in ‘bank2.c’ has been located in the Common area because it is a constant code variable. Refer to the Keil linker manual for a description of the M51 file.

Figure 9. Select “Options for File”

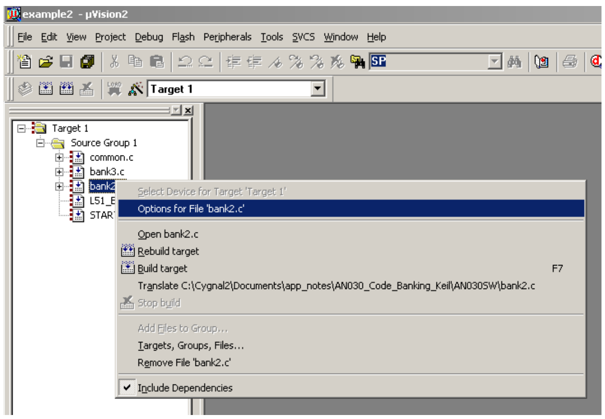


Figure 10. Options for File Window

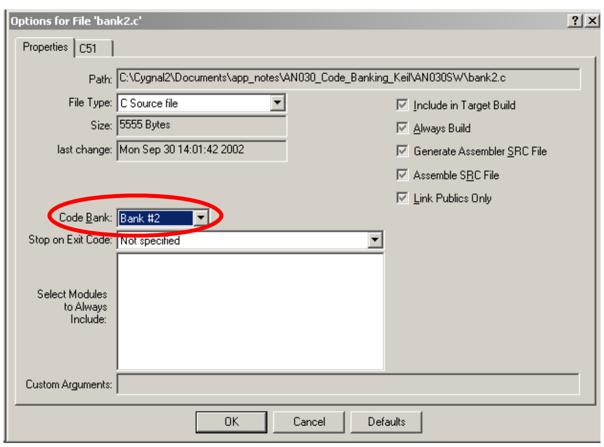


Table 4. Code Bank Selection for Example 2

Filename	Code Bank
common.obj	Common area
bank2.obj	Bank 2
bank3.obj	Bank 3
STARTUP.obj	Common area
L51_BANK.obj	Common area

Code Bank Assignment Considerations

Assigning files to code banks is a straightforward procedure. However, determining the best placement of functions in code banks is largely dependant on the nature of the project. This section outlines some guidelines to follow when assigning code banks.

The Common area is accessible by all code banks at all times. It is important to keep all code that must always be accessible in the Common area. For example, reset and interrupt vectors, interrupt service routines, code constants, bank switch code, and library functions should always be located in the Common area.

Assigning Code Banks for Maximum Performance

Code bank switching does not significantly affect the performance of most systems. However, to achieve maximum performance in time critical applications, programs should be structured so that frequent bank switching is not necessary. Bank switch code is not generated when the function being called resides in the Common area or in the same bank as its calling function. Placing frequently accessed functions or functions called from different banks in the Common area is essential to achieve maximum performance in time critical applications.

Code Constants

Code constants (strings, tables, etc.) are located in the Common area by default. This is the best location for code constants in most applications because they can be accessed using the MOVC instruction from any bank. If the Common area is not large enough to accommodate all code constants, they may be placed in one of the code banks but may only be accessed from code executing from the same bank in which they reside or the common area. They may not be accessed from code executing in another bank because the linker sets the constant code bank to the same bank as the instruction fetch bank. Constant data in a code bank may be accessed from the common area only if the bank in which they reside is the currently selected bank. Figure 11 shows how to invoke a linker-managed bank switch from C.

The *BANKx* linker control allows code constants to be located outside the Common area. For details on locating constant code variables in a code bank, please see the Keil Knowledge Base article at http://www.keil.com/support/docs/1615.htm.

Bank Switch Macros

The Keil 8051 tools support 3 modes of bank switching defined in ‘L51_BANK.A51’. Bank switching on the C8051F12x uses Mode 4 (bank switching via user-provided code). This mode allows the user to specify a macro for each code bank that selects it in the PSBANK register.

The PSBANK register contains two bank selects, COBANK for constant data and IFBANK for instruction fetches. Using the ‘L51_BANK.A51’ supplied by Silicon Labs, the COBANK and IFBANK always target the same code bank. This is why constant code tables must be located in the Common area or in the bank that accesses them.

The bank switch macros in ‘L51_BANK.A51’ may be changed to keep COBANK fixed regardless of the value of IFBANK. This would allow the user to dedicate one bank for constant data operations while using the other two banks for instruction fetches only. This dedicated bank would be available to code executing in any bank or the Common area.

The Common area may always be used for both instruction fetches and data storage regardless of the PSBANK register settings. For more information on bank switch macros, please see Chapter 9 of the Keil Assembler/Linker manual accessible in PDF format from the help menu of the Silicon Labs IDE.

Figure 11. Invoking a Linker-Managed Bank Switch From C

```
// declare the following function prototype
extern void switchbank (unsigned char bank_number);
    ...
switchbank(1);                // switch to code bank 1
```

Software Examples

Example 1: User-Managed Code Banking (Data Logger with Real-Time Clock)

```
//-----------------------------------------------------------------------------
// Data_Logger_RTC.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 30 SEP 02
//
//
// This application uses a 22.1184 MHz crystal oscillator to implement a
// software real-time clock (RTC). PCA Module 0, configured to count Timer 0
// overflows in software timer mode, generates an interrupt every second.
// The interrupt handler records the current time and device temperature
// in a non-volatile log in FLASH.
//
// With SYSCLK at 49.7664 MHZ, Timer 0 in mode 2 overflows exactly 4050 times
// every second when clocked by SYSCLK/48. PCA0, clocked by Timer 0 overflows,
// is programmed to generate an interrupt every 4050 Timer 0 overflows,
// or once every second.
//
// The 112,640 byte log cycles through all 4 code banks recording time and
// temperature. Each data record is 8 bytes long. The log is capable of storing
// 14080 records over a time period of 3.9 hours. Once the log is full, it
// continues logging at the beginning of log, erasing the FLASH page with
// the oldest data as it progresses.
//
// When this code is built, the linker generates two multiple call to segments
// warnings. These warnings are generated because the FLASH support routines
// are called from the main routine and from interrupts. These warnings have
// been accounted for in the code by disabling interrupts before calling any
// FLASH support routines.
//
//
// Target: C8051F12x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>                  // SFR declarations
#include <stdio.h>                      // printf() and getchar()

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for 'F12x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                  // data pointer
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 RCAP3    = 0xca;                  // Timer3 capture/reload
sfr16 RCAP4    = 0xca;                  // Timer4 capture/reload
sfr16 TMR2     = 0xcc;                  // Timer2
sfr16 TMR3     = 0xcc;                  // Timer3
sfr16 TMR4     = 0xcc;                  // Timer4
sfr16 DAC0     = 0xd2;                  // DAC0 data
sfr16 DAC1     = 0xd2;                  // DAC1 data
sfr16 PCA0CP5  = 0xe1;                  // PCA0 Module 5 capture
sfr16 PCA0CP2  = 0xe9;                  // PCA0 Module 2 capture
sfr16 PCA0CP3  = 0xeb;                  // PCA0 Module 3 capture
sfr16 PCA0CP4  = 0xed;                  // PCA0 Module 4 capture
sfr16 PCA0     = 0xf9;                  // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                  // PCA0 Module 0 capture
sfr16 PCA0CP1  = 0xfd;                  // PCA0 Module 1 capture

//-----------------------------------------------------------------------------
// Type Definitions
//-----------------------------------------------------------------------------

typedef union UInt {                    // Byte addressable unsigned int
   unsigned int Int;
   unsigned char Char[2];
} UInt;

typedef union Long {                    // Byte addressable long
   long Long;
   unsigned int Int[2];
   unsigned char Char[4];
} Long;

typedef union ULong {                   // Byte addressable unsigned long
   unsigned long ULong;
   unsigned int Int[2];
   unsigned char Char[4];
} ULong;

typedef struct Record {                 // LOG record structure
   char start;
   unsigned int hours;
   unsigned char minutes;
   unsigned char seconds;
   unsigned int ADC_result;
   char end;
} Record;

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

#define EXTCLK       22118400           // External oscillator frequency in Hz
#define SYSCLK       49766400           // Output of PLL derived from
                                        // (EXTCLK*9/4)

#define BAUDRATE     115200             // Baud rate of UART in bps
                                        // Note: The minimum standard baud rate
                                        // supported by the UART0_Init routine
                                        // in this file is 19,200 bps when
```

```
                                       // SYSCLK = 49.76MHz.

#define SAMPLERATE   2000              // The ADC sampling rate in Hz

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

#define LOG_START 0x04000L             // Starting address of LOG
#define LOG_END   0x1F800L             // Last address in LOG + 1
#define RECORD_LEN 8                   // Record length in bytes
#define START_OF_RECORD ':'            // Start of Record symbol

#define FLASH_PAGESIZE 1024            // Number of bytes in each FLASH page

#define COBANK    0xF0                 // Bit mask for the high nibble of PSBANK

#define COBANK0   0x00                 // These macros define the bit mask values
#define COBANK1   0x10                 // for the PSBANK register used for
#define COBANK2   0x20                 // selecting COBANK. COBANK should always
#define COBANK3   0x30                 // be cleared then OR-Equaled (|=) with
                                       // the proper bit mask to avoid changing
                                       // the other bits in the PSBANK register


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

unsigned char SECONDS = 0;             // global RTC seconds counter
unsigned char MINUTES = 0;             // global RTC minutes counter
unsigned int  HOURS = 0;               // global RTC hours counter

unsigned int ADC_RESULT = 0;           // holds the oversampled and averaged
                                       // result from ADC0


bit LOG_FLAG = 0;                      // this flag is used to enable
                                       // and disable logging but does
                                       // not affect the real-time clock

bit LOG_ERASED = 0;                    // this flag indicates that the
                                       // LOG has been erased.
//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------

void main(void);
void RTC_update(void);
void print_menu(void);

// initialization routines
void SYSCLK_Init(void);
void PORT_Init(void);
void UART0_Init (void);
void ADC0_Init (void);
void Timer3_Init(int counts);
void RTC_Init (void);
void PCA0_ISR (void);
```

```
// FLASH support routines
void FLASH_PageErase (unsigned long addr);
void FLASH_Write (unsigned long dest, char* src, unsigned int numbytes);
void FLASH_ByteWrite (unsigned long dest, char dat);
void FLASH_Read ( char* dest, unsigned long src, unsigned int numbytes);
unsigned char FLASH_ByteRead (unsigned long addr);

// LOG support routines

void print_time(void);
void LOG_erase(void);
unsigned long find_current_record(void);
void LOG_print(char all_at_once);
void LOG_update(void);


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

void main (void)
{

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

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

   PORT_Init ();                          // initialize crossbar and GPIO
   SYSCLK_Init ();                        // initialize oscillator
   UART0_Init ();                         // initialize UART0
   ADC0_Init();                           // initialize ADC0
   RTC_Init ();                           // initializes Timer0 and the PCA
   Timer3_Init(SYSCLK/SAMPLERATE);        // initialize Timer3 to overflow
                                          // and generate interrupts at
                                          // <SAMPLERATE> Hz

                                          // to implement a real-time clock

   EA = 1;                                // enable global interrupts

   print_menu();                          // print the command menu


   while (1){

      SFRPAGE = UART0_PAGE;
      printf("\nEnter a command > ");
      gets(input_str, input_str_len);

      switch ( input_str[0] ){

         case '1': LOG_FLAG = 1;
                   SFRPAGE = UART0_PAGE;
                   printf("\nLogging has now started.\n");
                   break;

         case '2': LOG_FLAG = 0;
```

```
                   SFRPAGE = UART0_PAGE;
                   printf(“\nLogging has now stopped.\n”);
                   break;

         case ‘3’: LOG_FLAG = 0;
                   LOG_erase();
                   SFRPAGE = UART0_PAGE;
                   printf(“\nThe log has been erased and logging is stopped.\n”);
                   break;

         case ‘4’: LOG_print(FALSE);
                   print_menu();
                   break;

         case ‘5’: LOG_print(TRUE);
                   print_menu();
                   break;

         case ‘6’: print_time();
                   break;

         case ‘?’: print_menu();
                   break;

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

   } // end while

}


//-----------------------------------------------------------------------------
// RTC_update
//-----------------------------------------------------------------------------
//
//
void RTC_update(void)
{
   SECONDS++;
   if (SECONDS == 60) {
      SECONDS = 0;
      MINUTES++;
      if (MINUTES == 60) {
         MINUTES = 0;
         HOURS++;
      }
   }

}

//-----------------------------------------------------------------------------
// FLASH Support Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// FLASH_PageErase
//-----------------------------------------------------------------------------
```

```
//
// This function erases the FLASH page containing <addr>.
//
void FLASH_PageErase (unsigned long addr)
{
   char SFRPAGE_SAVE = SFRPAGE;        // Preserve current SFR page
   char PSBANK_SAVE = PSBANK;          // Preserve current code bank
   bit EA_SAVE = EA;                   // Preserve interrupt state

   ULong temp_addr;                    // Temporary ULong
   char xdata * idata pwrite;          // FLASH write/erase pointer

   temp_addr.ULong = addr;             // copy <addr> to a byte addressable
                                       // unsigned long

   // Extract address information from <addr>
   pwrite = (char xdata *) temp_addr.Int[1];

   // Extract code bank information from <addr>
   PSBANK &= ~COBANK;                  // Clear the COBANK bits


   if( temp_addr.Char[1] == 0x00){     // If the address is less than
                                       // 0x10000, the Common area and
      PSBANK |= COBANK1;               // Bank1 provide a 64KB linear
                                       // address space
   } else {                            // Else, Bank2 and Bank3 provide
                                       // a 64KB linear address space

      if (temp_addr.Char[2] & 0x80){   // If bit 15 of the address is
                                       // a ‘1’, then the operation should
         PSBANK |= COBANK3;            // target Bank3, else target Bank2

      } else {

         PSBANK |= COBANK2;
         temp_addr.Char[2] |= 0x80;
         pwrite = (char xdata *) temp_addr.Int[1];
      }
    }

   SFRPAGE = LEGACY_PAGE;

   EA = 0;                             // Disable interrupts
   FLSCL |= 0x01;                      // Enable FLASH writes/erases
   PSCTL = 0x03;                       // MOVX erases FLASH page

   *pwrite = 0;                        // Initiate FLASH page erase

   FLSCL &= ~0x01;                     // Disable FLASH writes/erases
   PSCTL = 0x00;                       // MOVX targets XRAM

   EA = EA_SAVE;                       // Restore interrupt state
   PSBANK = PSBANK_SAVE;               // Restore current code bank
   SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

//-----------------------------------------------------------------------------
// FLASH_Write
```

```
//-----------------------------------------------------------------------------
//
// This routine copies <numbytes> from <src> to the FLASH addressed by <dest>.
//
void FLASH_Write (unsigned long dest, char* src, unsigned int numbytes)
{

   unsigned int i;                     // Software Counter

   for (i = 0; i < numbytes; i++) {

      FLASH_ByteWrite( dest++, *src++);
   }
}

//-----------------------------------------------------------------------------
// FLASH_ByteWrite
//-----------------------------------------------------------------------------
//
// This routine writes <dat> to the FLASH byte addressed by <dest>.
//
void FLASH_ByteWrite (unsigned long dest, char dat)
{
   char SFRPAGE_SAVE = SFRPAGE;        // Preserve current SFR page
   char PSBANK_SAVE = PSBANK;          // Preserve current code bank
   bit EA_SAVE = EA;                   // Preserve interrupt state

   ULong temp_dest;                    // Temporary ULong

   char xdata * idata pwrite;          // FLASH write/erase pointer

   temp_dest.ULong = dest;             // copy <dest> to a byte
                                       // addressable unsigned long

   // Check if data byte being written is 0xFF
   // There is no need to write 0xFF to FLASH since erased
   // FLASH defaults to 0xFF.
   if(dat != 0xFF){

      // Extract address information from <dest>
      pwrite = (char xdata *) temp_dest.Int[1];

      // Extract code bank information from <addr>
      PSBANK &= ~COBANK;               // Clear the COBANK bits


      if( temp_dest.Char[1] == 0x00){  // If the address is less than
                                       // 0x10000, the Common area and
         PSBANK |= COBANK1;            // Bank1 provide a 64KB linear
                                       // address space
      } else {                         // Else, Bank2 and Bank3 provide
                                       // a 64KB linear address space

         if (temp_dest.Char[2] & 0x80){// If bit 15 of the address is
                                       // a ‘1’, then the operation should
            PSBANK |= COBANK3;         // target Bank3, else target Bank2

         } else {
```

```
            PSBANK |= COBANK2;
            temp_dest.Char[2] |= 0x80;
            pwrite = (char xdata *) temp_dest.Int[1];
         }
       }


      SFRPAGE = LEGACY_PAGE;

      EA = 0;                          // Disable interrupts
      FLSCL |= 0x01;                   // Enable FLASH writes/erases
      PSCTL = 0x01;                    // MOVX writes FLASH byte

      *pwrite = dat;                   // Write FLASH byte

      FLSCL &= ~0x01;                  // Disable FLASH writes/erases
      PSCTL = 0x00;                    // MOVX targets XRAM
   }

   EA = EA_SAVE;                       // Restore interrupt state
   PSBANK = PSBANK_SAVE;               // Restore current code bank
   SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

//-----------------------------------------------------------------------------
// FLASH_Read
//-----------------------------------------------------------------------------
//
// This routine copies <numbytes> from FLASH addressed by <src> to <dest>.
//
void FLASH_Read ( char* dest, unsigned long src, unsigned int numbytes)
{

   unsigned int i;                     // Software Counter

   for (i = 0; i < numbytes; i++) {

      *dest++ = FLASH_ByteRead(src++);
   }
}

//-----------------------------------------------------------------------------
// FLASH_ByteRead
//-----------------------------------------------------------------------------
//
// This routine returns to the value of the FLASH byte addressed by <addr>.
//
unsigned char FLASH_ByteRead (unsigned long addr)
{
   char SFRPAGE_SAVE = SFRPAGE;        // Preserve current SFR page
   char PSBANK_SAVE = PSBANK;          // Preserve current code bank

   ULong temp_addr;                    // Temporary ULong
   char temp_char;                     // Temporary char

   char code * idata pread;            // FLASH read pointer

   temp_addr.ULong = addr;             // copy <addr> to a byte addressable
                                       // unsigned long
```

```
   // Extract address information from <addr>
   pread = (char code *) temp_addr.Int[1];

   // Extract code bank information from <addr>
   PSBANK &= ~COBANK;                  // Clear the COBANK bits


   if( temp_addr.Char[1] == 0x00){     // If the address is less than
                                       // 0x10000, the Common area and
      PSBANK |= COBANK1;               // Bank1 provide a 64KB linear
                                       // address space
   } else {                            // Else, Bank2 and Bank3 provide
                                       // a 64KB linear address space

      if (temp_addr.Char[2] & 0x80){   // If bit 15 of the address is
                                       // a '1', then the operation should
         PSBANK |= COBANK3;            // target Bank3, else target Bank2

      } else {

         PSBANK |= COBANK2;
         temp_addr.Char[2] |= 0x80;
         pread = (char code *) temp_addr.Int[1];
      }
   }

   temp_char = *pread;                 // Read FLASH byte

   PSBANK = PSBANK_SAVE;               // Restore current code bank
   SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page

   return temp_char;
}

//-----------------------------------------------------------------------------
// Support Routines
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// print_menu
//-----------------------------------------------------------------------------
//
// This routine uses prints the command menu to the UART.
//
void print_menu(void)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

   SFRPAGE = UART0_PAGE;
   printf("\n\nC8051F12x Data Logging Example\n");
   printf("---------------------------------------\n");
   printf("1. Start Logging\n");
   printf("2. Stop Logging\n");
   printf("3. Erase Log\n");
   printf("4. Print Log (one page at a time - Press CTRL+C to stop)\n");
   printf("5. Print Log (all at once - Press CTRL+C to stop)\n");
   printf("6. Print Elapsed Time Since Last Reset\n");
   printf("?. Print Command List\n");
```

```
   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

//-----------------------------------------------------------------------------
// print_time
//-----------------------------------------------------------------------------
//
// This routine uses prints the elapsed time since the last reset to the UART.
//
void print_time(void)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page
   bit EA_SAVE = EA;                // Preserve interrupt state

   SFRPAGE = UART0_PAGE;
   EA = 0;
   printf ("%05u:%02bu:%02bu\n", HOURS, MINUTES, SECONDS);
   EA = EA_SAVE;
   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

//-----------------------------------------------------------------------------
// find_current_record
//-----------------------------------------------------------------------------
//
//
unsigned long find_current_record(void)
{

   char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
   bit EA_SAVE = EA;                      // Preserve interrupt state

   unsigned long pRead = LOG_START;       // Pointer used to read from FLASH

   unsigned int i;                        // Software counter
   bit record_erased;                     // Temporary flag


   // Keep skipping records until an uninitialized record is found or
   // until the end of the log is reached
   while( pRead < LOG_END ){

      EA = 0;
      // Skip all records that have been initialized
      if(FLASH_ByteRead(pRead) == START_OF_RECORD ){

         // increment pRead to the next record
         pRead += RECORD_LEN;
         EA = EA_SAVE;
         continue;
      }

      // Verify that the Record is uninitialized, otherwise keep
      // searching for an uninitialized record
      record_erased = 1;
      for(i = 0; i < RECORD_LEN; i++){
         if( FLASH_ByteRead(pRead+i) != 0xFF ){
            record_erased = 0;
         }
```

```
      }
      if(!record_erased){
         // increment pRead to the next record
         pRead += RECORD_LEN;
         EA = EA_SAVE;
         continue;
      }

      EA = EA_SAVE;

      // When this code is reached, <pRead> should point to the beginning
      // of an uninitialized (erased) record;
      SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
      return pRead;

   }

   // This code is reached only when there are no uninitialized records
   // in the LOG. Erase the first FLASH page in the log and return
   // a pointer to the first record in the log.
   EA = 0;
   FLASH_PageErase(LOG_START);            // Erase the first page of the LOG
   EA = EA_SAVE;
   SFRPAGE = SFRPAGE_SAVE;                // Restore SFR page
   return LOG_START;
}

//-----------------------------------------------------------------------------
// LOG_erase
//-----------------------------------------------------------------------------
//
//
void LOG_erase(void)
{
   unsigned long pWrite = LOG_START;      // pointer used to write to FLASH
   bit EA_SAVE = EA;                     // save interrupt status

   // Keep erasing pages until <pWrite> reaches the end of the LOG.
   while( pWrite < LOG_END ){

      EA = 0;
      FLASH_PageErase(pWrite);
      EA = EA_SAVE;

      pWrite += FLASH_PAGESIZE;

   }

   LOG_ERASED = 1;                        // flag that LOG has been erased
}

//-----------------------------------------------------------------------------
// LOG_print
//-----------------------------------------------------------------------------
//
//
void LOG_print(char all_at_once)
{
   char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
```

```
bit EA_SAVE = EA;                          // save interrupt status

unsigned long pRead = LOG_START;           // Pointer used to read from FLASH

Record temp_rec;                           // Temporary record

// Keep printing records until the end of the log is reached
while( pRead < LOG_END ){

   // Copy a record from at <pRead> from the LOG into the local
   // Record structure <temp_rec>
   EA = 0;
   FLASH_Read( (char*) &temp_rec, pRead, RECORD_LEN);
   EA = EA_SAVE;

   // Validate Record
   if(temp_rec.start != ‘:’){
      SFRPAGE = SFRPAGE_SAVE;              // Restore SFR page
      return;
   }

   // Print the Record
   SFRPAGE = UART0_PAGE;

   RI0 = 0;                                // Clear UART Receive flag
                                           // to later check for the
                                           // user pressing CTRL+C

   EA = 0;                                 // disable interrupts

   // print the time and ADC reading
   printf (“%05u:%02bu:%02bu  ADC = 0x%04X\n”,  temp_rec.hours,
                                                temp_rec.minutes,
                                                temp_rec.seconds,
                                                temp_rec.ADC_result);



   EA = EA_SAVE;                            // restore interrupts
                                            // any pending interrupts will
                                            // be handled immediatly
  // check if we need to continue

  // if printing all data at once do not stop printing unless
  // the user presses CTRL+C, otherwise print 16 records and
  // then prompt user to press any key

  if(all_at_once){
     // Check if user has pressed CTRL+C
     if(RI0 && SBUF0 == 0x03){
        RI0 = 0;
        SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
        return;
     }

  // pause every 16 lines
  } else if( (pRead & ((RECORD_LEN*16)-1)) == 0  &&
             pRead > (LOG_START + RECORD_LEN)) {
```

```
      // wait for a key to be pressed then check if user has
      // pressed CTRL+C (0x03)
      printf("\npress any key to continue\n");
      if(_getkey() == 0x03) {
         SFRPAGE = SFRPAGE_SAVE;         // Restore SFR page
         return;
      }
   }


   // increment pRead to the next record
   pRead += RECORD_LEN;

   SFRPAGE = SFRPAGE_SAVE;               // Restore SFR page

 }

}


//-----------------------------------------------------------------------------
// LOG_update
//-----------------------------------------------------------------------------
//
//
void LOG_update(void)
{
   bit EA_SAVE = EA;                  // Preserve interrupt state
   Record temp_record;                // local LOG record structure

   static unsigned long pWrite = LOG_START;
                                      // pointer used to write to the LOG
   bit record_erased;                 // temporary flag
   unsigned int i;                    // temporary integer

   // record the time and ADC reading in the LOG if logging is enabled
   if(LOG_FLAG){

      if(LOG_ERASED){
         pWrite = LOG_START;
         LOG_ERASED = 0;

      } else {

         // find the current record if the record at pWrite is not erased
         record_erased = 1;
         for(i = 0; i < RECORD_LEN; i++){
            EA = 0;
            if( FLASH_ByteRead(pWrite+i) != 0xFF ){
               record_erased = 0;
            }
            EA = EA_SAVE;
         }
         if(!record_erased){
            pWrite = find_current_record();
         }


         // build the temporary record
```

```
         temp_record.start = START_OF_RECORD;
         temp_record.hours = HOURS;
         temp_record.minutes = MINUTES;
         temp_record.seconds = SECONDS;
         temp_record.ADC_result = ADC_RESULT;

         // write the temporary record to FLASH
         EA = 0;
         FLASH_Write( pWrite, (char*) &temp_record, RECORD_LEN);
         EA = EA_SAVE;

         // increment record pointer
         pWrite += RECORD_LEN;

         // if <pWrite> is past the end of the LOG, reset to the top
         if(pWrite >= LOG_END){
            pWrite = LOG_START;
         }
      } // end else
   } // end if(LOG_FLAG)

}




//-----------------------------------------------------------------------------
// Initialization Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use an external 22.1184 MHz
// crystal oscillator multiplied by a factor of 9/4 using the PLL as its
// clock source. The resulting frequency is 22.1184 MHz * 9/4 = 49.7664 MHz
//
void SYSCLK_Init (void)
{
   int i;                              // delay counter

   char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

   SFRPAGE = CONFIG_PAGE;              // set SFR page

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

   for (i=0; i < 256; i++) ;           // Wait for osc. to start up

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

   CLKSEL = 0x01;                      // Select the external osc. as
                                       // the SYSCLK source

   OSCICN = 0x00;                      // Disable the internal osc.

   //Turn on the PLL and increase the system clock by a factor of M/N = 9/4
```

```
   SFRPAGE = CONFIG_PAGE;

   PLL0CN  = 0x04;                  // Set PLL source as external osc.
   SFRPAGE = LEGACY_PAGE;
   FLSCL   = 0x10;                  // Set FLASH read time for 50MHz clk
                                    // or less
   SFRPAGE = CONFIG_PAGE;
   PLL0CN |= 0x01;                  // Enable Power to PLL
   PLL0DIV = 0x04;                  // Set Pre-divide value to N (N = 4)
   PLL0FLT = 0x01;                  // Set the PLL filter register for
                                    // a reference clock from 19 - 30 MHz
                                    // and an output clock from 45 - 80 MHz
   PLL0MUL = 0x09;                  // Multiply SYSCLK by M (M = 9)

   for (i=0; i < 256; i++) ;        // Wait at least 5us
   PLL0CN  |= 0x02;                 // Enable the PLL
   while(!(PLL0CN & 0x10));         // Wait until PLL frequency is locked
   CLKSEL  = 0x02;                  // Select PLL as SYSCLK source

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}


//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// This routine configures the Crossbar and GPIO ports.
//
void PORT_Init (void)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

   SFRPAGE = CONFIG_PAGE;           // set SFR page

   XBR0     = 0x04;                 // Enable UART0
   XBR1     = 0x00;
   XBR2     = 0x40;                 // Enable crossbar and weak pull-up


   P0MDOUT |= 0x01;                 // Set TX0 pin to push-pull
   P1MDOUT |= 0x40;                 // Set P1.6(LED) to push-pull

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1. In order to
// increase the clocking flexibility of Timer0, Timer1 is configured to count
// SYSCLKs.
//
// To use this routine SYSCLK/BAUDRATE/16 must be less than 256. For example,
// if SYSCLK = 50 MHz, the lowest standard baud rate supported by this
// routine is 19,200 bps.
//
void UART0_Init (void)
```

```
{
   char SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page

   SFRPAGE = UART0_PAGE;

   SCON0  = 0x50;                    // SCON0: mode 0, 8-bit UART, enable RX
   SSTA0  = 0x10;                    // Timer 1 generates UART0 baud rate and
                                     // UART0 baud rate divide by two disabled
   SFRPAGE = TIMER01_PAGE;
   TMOD   &= ~0xF0;
   TMOD   |=  0x20;                  // TMOD: timer 1, mode 2, 8-bit reload

   TH1 = -(SYSCLK/BAUDRATE/16);      // Set the Timer1 reload value
                                     // When using a low baud rate, this equation
                                     // should be checked to ensure that the
                                     // reload value will fit in 8-bits.

   CKCON |= 0x10;                    // T1M = 1; SCA1:0 = xx


   TL1 = TH1;                        // initialize Timer1
   TR1 = 1;                          // start Timer1

   SFRPAGE = UART0_PAGE;
   TI0 = 1;                          // Indicate TX0 ready

   SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
}

//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
//
// Configure ADC0 to start conversions on Timer3 Overflows and to
// use left-justified output mode.
//
void ADC0_Init (void)
{
   char SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page

   SFRPAGE = ADC0_PAGE;

   ADC0CN = 0x85;                    // ADC0 enabled; normal tracking
                                     // mode; ADC0 conversions are initiated
                                     // on Timer3 overflows; ADC0 data is
                                     // left-justified

   REF0CN = 0x07;                    // enable temp sensor, on-chip VREF,
                                     // and VREF output buffer
   AMX0SL = 0x0F;                    // Select TEMP sens as ADC mux output

   ADC0CF = ((SYSCLK/2500000) << 3);// ADC conversion clock = 2.5MHz

   ADC0CF |= 0x01;                   // PGA gain = 2

   EIE2 |= 0x02;                     // Enable ADC0 End-of-conversion
                                     // interrupts

   SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
```

```
}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
// This routine initializes Timer3 in auto-reload mode to overflow
// at intervals specified in <counts>.
//
void Timer3_Init (int counts)
{
   SFRPAGE = TMR3_PAGE;

   TMR3CN = 0;                       // STOP timer; set to auto-reload mode
   TMR3CF = 0x08;                    // Timer3 counts SYSCLKs
   RCAP3 = -counts;                  // set reload value
   TMR3 = RCAP3;
   TR3 = 1;                          // start Timer3

}

//-----------------------------------------------------------------------------
// RTC_Init
//-----------------------------------------------------------------------------
//
// This Routine initializes Timer0 and PCA0 to implement a real-time clock.
// Assuming <SYSCLK> is generated from a 22.1184 crystal oscillator, Timer0
// overflows exactly 1800 times per second when configured as an 8-bit timer
// that uses <SYSCLK>/48 as its timebase. PCA0 is configured to count
// Timer0 overflows and interrupt every 1800 Timer0 overflows, or every second.
// The PCA0 ISR updates a set of global RTC counters for seconds, minutes, hours,
// and days.
//
void RTC_Init(void)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

   SFRPAGE = TIMER01_PAGE;

   // configure Timer0 in Mode2: 8-bit Timer with Auto-Reload
   TMOD &= ~0x0F;                   // Clear Timer0 bits
   TMOD |= 0x02;                    // Mode2 Auto-Reload

   // configure Timer0 timebase to <SYSCLK>/48
   CKCON &= ~0x0F;                  // Clear bits
   CKCON |= 0x02;                   // Set Timer0 timebase

   // configure PCA0 to count Timer0 overflows
   PCA0MD = 0x04;

   // configure capture/compare module 0 to generate an interrupt when
   // the value of PCA0 reaches 4050 (0x0FD2)
   PCA0CP0 = 4050;                  // Set the value to match

   PCA0CPM0 &= ~0xFF;               // Clear bits
   PCA0CPM0 |= 0x49;                // Generate an interrupt when the
                                    // PCA0 value matches PCA0CP0

   EIE1 |= 0x08;                    // Enable PCA0 interrupts
```

```
   TR0 = 1;                         // Start Timer0
   PCA0CN |= 0x40;                  // Enable PCA0

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

//-----------------------------------------------------------------------------
// PCA0_ISR
//-----------------------------------------------------------------------------
//
//
void PCA0_ISR (void) interrupt 9
{

  if (CCF0) {
      CCF0 = 0;                         // clear Module0 capture/compare flag

      PCA0 = 0x00;                      // clear the PCA counter
      RTC_update();                     // update RTC variables
      LOG_update();                     // update LOG if logging is enabled

  } else

  if (CCF1) {
      CCF1 = 0;                         // clear Module1 capture/compare flag
  } else

  if (CCF2) {
      CCF2 = 0;                         // clear Module2 capture/compare flag
  } else

  if (CCF3) {
      CCF3 = 0;                         // clear Module3 capture/compare flag
  } else

  if (CCF4) {
      CCF4 = 0;                         // clear Module4 capture/compare flag
  } else

  if (CCF5) {
      CCF5 = 0;                         // clear Module5 capture/compare flag
  } else

  if (CF) {
      CF = 0;                           // clear PCA counter overflow flag
  }

}

//-----------------------------------------------------------------------------
// ADC0_ISR
//-----------------------------------------------------------------------------
//
// This ISR is called on the end of an ADC0 conversion.
//
void ADC0_ISR (void) interrupt 15
{
```

```
   static Long result = {0};             // byte addressable long variable
    int i;
    bit EA_SAVE = EA;
   //accumulate 256 temperature samples
   result.Long += ADC0;
   i++;

   if( i == 256 ) {

      i = 0;

    // take the average (Divide by 256 = shift right by 8)
    // Do this operation “result.Long >>= 8;” (170 SYSCLK cycles) using
      // three MOV instructions (9 SYSCLK cycles)
      // Assume Most Significant Byte only contains sign information

      result.Char[3] = result.Char[2];
      result.Char[2] = result.Char[1];
      result.Char[1] = result.Char[0];

      // update global <ADC_RESULT>
      ADC_RESULT = result.Int[1];
   }

}
```

Example 2: Project-Managed Code Banking

```
//-----------------------------------------------------------------------------
// common.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 18 SEP 02
//
// This example shows how to set up a code banking project using the Cygnal
// IDE and the KEIL development tools. It uses Timer3 and Timer4 interrupts
// to blink the LED and output a 1 kHz sine wave on DAC1, respectively. The
// code that blinks the LED is located in Bank 3 and the code that outputs a
// sine wave based on a 256 entry sine table is located in Bank 2. Since
// interrupts must be located in the Common area, both interrupts call
// a function in one of the banks to perform the desired task.
//
// The project should be configured for code banking as shown in AN030 before
// this project is built.
//
// This program uses the the 24.5 MHz internal oscillator multiplied by two
// for an effective SYSCLK of 49 MHz. This program also initializes UART1
// at <BAUDRATE> bits per second.
//
//
// Target: C8051F12x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>                    // SFR declarations
#include <stdio.h>                        // printf() and getchar()

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F12x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                    // data pointer
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 RCAP3    = 0xca;                    // Timer3 capture/reload
sfr16 RCAP4    = 0xca;                    // Timer4 capture/reload
sfr16 TMR2     = 0xcc;                    // Timer2
sfr16 TMR3     = 0xcc;                    // Timer3
sfr16 TMR4     = 0xcc;                    // Timer4
sfr16 DAC0     = 0xd2;                    // DAC0 data
sfr16 DAC1     = 0xd2;                    // DAC1 data
sfr16 PCA0CP5  = 0xe1;                    // PCA0 Module 5 capture
sfr16 PCA0CP2  = 0xe9;                    // PCA0 Module 2 capture
sfr16 PCA0CP3  = 0xeb;                    // PCA0 Module 3 capture
sfr16 PCA0CP4  = 0xed;                    // PCA0 Module 4 capture
sfr16 PCA0     = 0xf9;                    // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                    // PCA0 Module 0 capture
sfr16 PCA0CP1  = 0xfd;                    // PCA0 Module 1 capture
```

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

#define INTCLK       24500000          // Internal oscillator frequency in Hz
#define SYSCLK       49000000          // Output of PLL derived from (INTCLK*2)
#define BAUDRATE     115200            // Baud rate of UART in bps

#define SAMPLE_RATE_DAC 100000L        // DAC sampling rate in Hz
#define PHASE_PRECISION 65536          // range of phase accumulator

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

#define FREQUENCY 1000                 // frequency of output waveform in Hz

                                       // <phase_add> is the change in phase
                                       // between DAC1 samples; It is used in
                                       // the set_DAC1 routine in bank2
unsigned int phase_add = FREQUENCY * PHASE_PRECISION / SAMPLE_RATE_DAC;


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

// Common area functions
void main(void);
void SYSCLK_Init(void);
void PORT_Init(void);
void UART1_Init (void);
void DAC1_Init (void);
void Timer3_Init(int counts);
void Timer4_Init(int counts);

// code bank 2 functions
extern void set_DAC1(void);

// code bank 3 functions
extern void toggle_LED(void);
//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------

void main (void)
{

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

   PORT_Init ();                       // initialize crossbar and GPIO
   SYSCLK_Init ();                     // initialize oscillator
   UART1_Init ();                      // initialize UART1
   DAC1_Init ();                       // initialize DAC1

   Timer3_Init(SYSCLK/12/1000);        // initialize Timer3 to overflow
```

```
                                    // every millisecond

   Timer4_Init(SYSCLK/SAMPLE_RATE_DAC);// initialize Timer4 to overflow
                                    // <SAMPLE_RATE_DAC> times per
                                    // second

   EA = 1;                          // enable global interrupts

   while(1);

}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Timer3_ISR
//-----------------------------------------------------------------------------
// This routine changes the state of the LED whenever Timer3 overflows 250 times.
//
// NOTE: The SFRPAGE register will automatically be switched to the Timer 3 Page
// When an interrupt occurs.  SFRPAGE will return to its previous setting on exit
// from this routine.
//
void Timer3_ISR (void) interrupt 14
{
   static int i;                    // software interrupt counter

   TF3 = 0;                         // clear Timer3 overflow flag
   i++;                             // increment software counter

   // toggle the LED every 250ms
   if (i == 250) {
      toggle_LED();                 // toggle the green LED
      i = 0;                        // clear software counter
   }
}
//-----------------------------------------------------------------------------
// Timer4_ISR -- Wave Generator
//-----------------------------------------------------------------------------
//
// This ISR is called on Timer4 overflows.  Timer4 is set to auto-reload mode
// and is used to schedule the DAC output sample rate in this example.
// Note that the value that is written to DAC1 during this ISR call is
// actually transferred to DAC1 at the next Timer4 overflow.
//
void Timer4_ISR (void) interrupt 16
{
  TF4 = 0;                          // clear Timer4 overflow flag
  set_DAC1();
}


//-----------------------------------------------------------------------------
// Initialization Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// SYSCLK_Init
```

```
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the internal oscillator
// at 24.5 MHz multiplied by two using the PLL.
//
void SYSCLK_Init (void)
{
   int i;                             // software timer

   char SFRPAGE_SAVE = SFRPAGE;       // Save Current SFR page

   SFRPAGE = CONFIG_PAGE;             // set SFR page

   OSCICN = 0x83;                     // set internal oscillator to run
                                      // at its maximum frequency

   CLKSEL = 0x00;                     // Select the internal osc. as
                                      // the SYSCLK source

   //Turn on the PLL and increase the system clock by a factor of M/N = 2
   SFRPAGE = CONFIG_PAGE;

   PLL0CN  = 0x00;                    // Set internal osc. as PLL source
   SFRPAGE = LEGACY_PAGE;
   FLSCL   = 0x10;                    // Set FLASH read time for 50MHz clk
                                      // or less
   SFRPAGE = CONFIG_PAGE;
   PLL0CN |= 0x01;                    // Enable Power to PLL
   PLL0DIV = 0x01;                    // Set Pre-divide value to N (N = 1)
   PLL0FLT = 0x01;                    // Set the PLL filter register for
                                      // a reference clock from 19 - 30 MHz
                                      // and an output clock from 45 - 80 MHz
   PLL0MUL = 0x02;                    // Multiply SYSCLK by M (M = 2)

   for (i=0; i < 256; i++) ;          // Wait at least 5us
   PLL0CN  |= 0x02;                   // Enable the PLL
   while(!(PLL0CN & 0x10));           // Wait until PLL frequency is locked
   CLKSEL  = 0x02;                    // Select PLL as SYSCLK source

   SFRPAGE = SFRPAGE_SAVE;            // Restore SFR page
}


//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// This routine configures the crossbar and GPIO ports.
//
void PORT_Init (void)
{
   char SFRPAGE_SAVE = SFRPAGE;       // Save Current SFR page

   SFRPAGE = CONFIG_PAGE;             // set SFR page

   XBR0     = 0x00;
   XBR1     = 0x00;
   XBR2     = 0x44;                   // Enable crossbar and weak pull-up
                                      // Enable UART1
```

```
   P0MDOUT |= 0x01;                 // Set TX1 pin to push-pull
   P1MDOUT |= 0x40;                 // Set P1.6(LED) to push-pull

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

//-----------------------------------------------------------------------------
// UART1_Init
//-----------------------------------------------------------------------------
//
// Configure the UART1 using Timer1, for <baudrate> and 8-N-1.
//
void UART1_Init (void)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

   SFRPAGE = UART1_PAGE;
   SCON1   = 0x10;                  // SCON1: mode 0, 8-bit UART, enable RX

   SFRPAGE = TIMER01_PAGE;
   TMOD   &= ~0xF0;
   TMOD   |=  0x20;                 // TMOD: timer 1, mode 2, 8-bit reload


   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 &= ~0x13;               // Clear all T1 related bits
      CKCON |=  0x01;               // T1M = 0; SCA1:0 = 01
   } else if (SYSCLK/BAUDRATE/2/256 < 12) {
      TH1 = -(SYSCLK/BAUDRATE/2/12);
      CKCON &= ~0x13;               // T1M = 0; SCA1:0 = 00
   } else {
      TH1 = -(SYSCLK/BAUDRATE/2/48);
      CKCON &= ~0x13;               // Clear all T1 related bits
      CKCON |=  0x02;               // T1M = 0; SCA1:0 = 10
   }

   TL1 = TH1;                       // initialize Timer1
   TR1 = 1;                         // start Timer1

   SFRPAGE = UART1_PAGE;
   TI1 = 1;                         // Indicate TX1 ready

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page

}


//-----------------------------------------------------------------------------
// DAC1_Init
//-----------------------------------------------------------------------------
//
// Configure DAC1 to update on Timer4 overflows and enable the the VREF buffer.
//
//
```

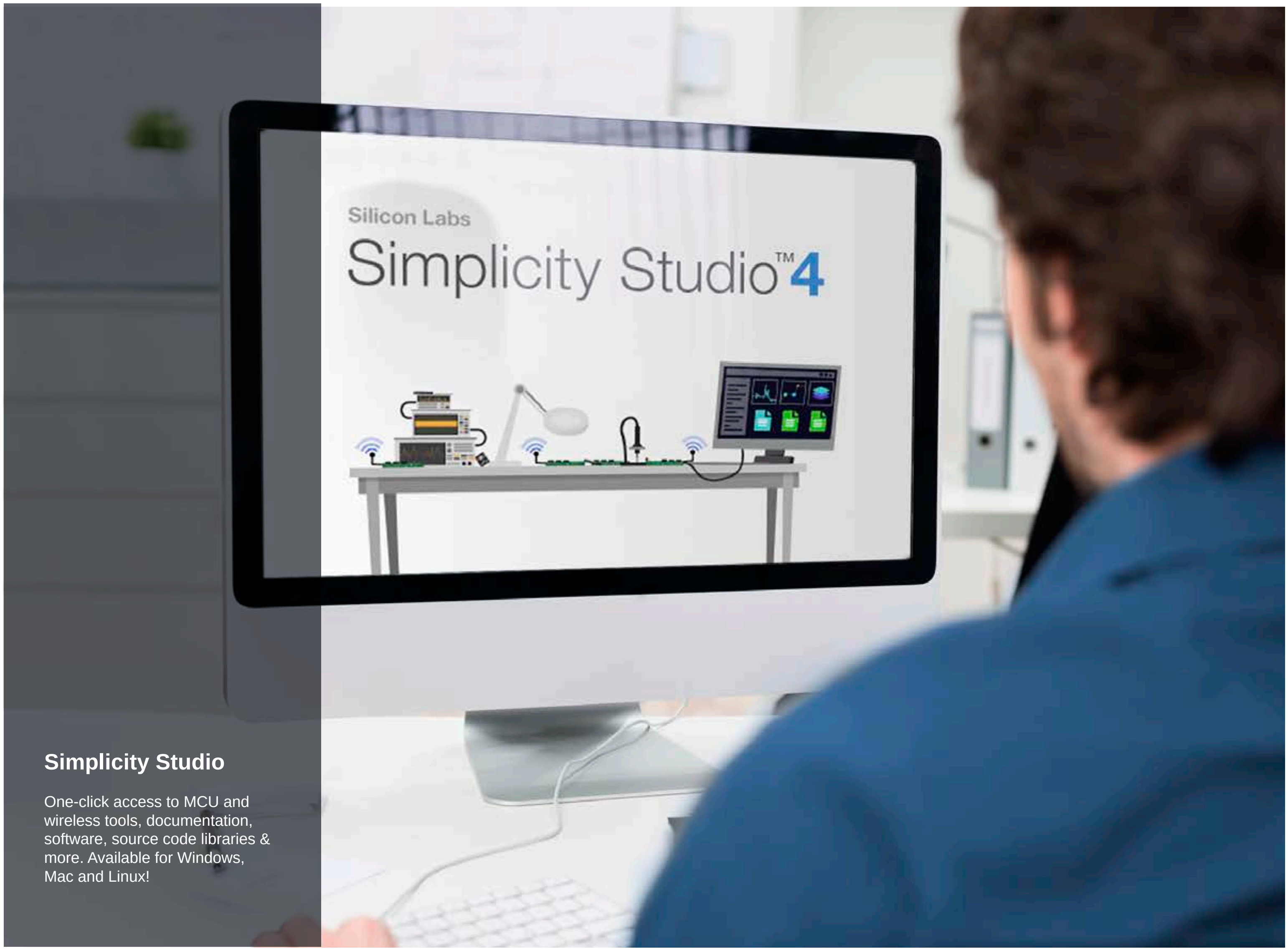
```
void DAC1_Init(void){

   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

   SFRPAGE = DAC1_PAGE;

   DAC1CN = 0x94;                   // Enable DAC1 in left-justified mode
                                    // managed by Timer4 overflows
   SFRPAGE = LEGACY_PAGE;

   REF0CN |= 0x03;                  // Enable the internal VREF (2.4v) and
                                    // the Bias Generator

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page

}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
//
// Configure Timer3 to auto-reload mode and to generate interrupts
// at intervals specified by <counts> using SYSCLK/12 as its time base.
//
//
void Timer3_Init (int counts)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

   SFRPAGE = TMR3_PAGE;

   TMR3CN = 0x00;                   // Stop Timer; Clear overflow flag;
                                    // Set to Auto-Reload Mode

   TMR3CF = 0x00;                   // Configure Timer to increment;
                                    // Timer counts SYSCLKs/12

   RCAP3 = -counts;                 // Set reload value
   TMR3 = RCAP3;                    // Initialize Timer to reload value

   EIE2   |= 0x01;                  // enable Timer3 interrupts
   TR3 = 1;                         // start Timer

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page

}

//-----------------------------------------------------------------------------
// Timer4_Init
//-----------------------------------------------------------------------------
// Configure Timer4 to auto-reload mode and to generate interrupts
// at intervals specified in <counts> using SYSCLK as its time base.
//
void Timer4_Init (int counts)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

   SFRPAGE = TMR4_PAGE;
```

```
   TMR4CN = 0x00;                     // Stop Timer4; Clear overflow flag (TF4);
                                      // Set to Auto-Reload Mode

   TMR4CF = 0x08;                     // Configure Timer4 to increment;
                                      // Timer4 counts SYSCLKs

   RCAP4 = -counts;                   // Set reload value
   TMR4 = RCAP4;                      // Initialzie Timer4 to reload value

   EIE2 |= 0x04;                      // enable Timer4 interrupts
   TR4 = 1;                           // start Timer4

   SFRPAGE = SFRPAGE_SAVE;            // Restore SFR page
}
```

```
//-----------------------------------------------------------------------------
// bank2.c
//-----------------------------------------------------------------------------
//
// AUTH: FB
// DATE: 18 SEP 02
//
// Target: C8051F12x
// Tool chain: KEIL C51
//
// This file contains routines used by the code banking example in AN030.
// All routines in this file are located in Code Bank 2.
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>                  // SFR declarations

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F12x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
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 RCAP3    = 0xca;                 // Timer3 capture/reload
sfr16 RCAP4    = 0xca;                 // Timer4 capture/reload
sfr16 TMR2     = 0xcc;                 // Timer2
sfr16 TMR3     = 0xcc;                 // Timer3
sfr16 TMR4     = 0xcc;                 // Timer4
sfr16 DAC0     = 0xd2;                 // DAC0 data
sfr16 DAC1     = 0xd2;                 // DAC1 data
sfr16 PCA0CP5  = 0xe1;                 // PCA0 Module 5 capture
sfr16 PCA0CP2  = 0xe9;                 // PCA0 Module 2 capture
sfr16 PCA0CP3  = 0xeb;                 // PCA0 Module 3 capture
sfr16 PCA0CP4  = 0xed;                 // PCA0 Module 4 capture
sfr16 PCA0     = 0xf9;                 // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                 // PCA0 Module 0 capture
sfr16 PCA0CP1  = 0xfd;                 // PCA0 Module 1 capture

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

extern int phase_add;

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

int code SINE_TABLE[256] = {

   0x0000, 0x0324, 0x0647, 0x096a, 0x0c8b, 0x0fab, 0x12c8, 0x15e2,
   0x18f8, 0x1c0b, 0x1f19, 0x2223, 0x2528, 0x2826, 0x2b1f, 0x2e11,
   0x30fb, 0x33de, 0x36ba, 0x398c, 0x3c56, 0x3f17, 0x41ce, 0x447a,
   0x471c, 0x49b4, 0x4c3f, 0x4ebf, 0x5133, 0x539b, 0x55f5, 0x5842,
   0x5a82, 0x5cb4, 0x5ed7, 0x60ec, 0x62f2, 0x64e8, 0x66cf, 0x68a6,
```

```
   0x6a6d, 0x6c24, 0x6dca, 0x6f5f, 0x70e2, 0x7255, 0x73b5, 0x7504,
   0x7641, 0x776c, 0x7884, 0x798a, 0x7a7d, 0x7b5d, 0x7c29, 0x7ce3,
   0x7d8a, 0x7e1d, 0x7e9d, 0x7f09, 0x7f62, 0x7fa7, 0x7fd8, 0x7ff6,
   0x7fff, 0x7ff6, 0x7fd8, 0x7fa7, 0x7f62, 0x7f09, 0x7e9d, 0x7e1d,
   0x7d8a, 0x7ce3, 0x7c29, 0x7b5d, 0x7a7d, 0x798a, 0x7884, 0x776c,
   0x7641, 0x7504, 0x73b5, 0x7255, 0x70e2, 0x6f5f, 0x6dca, 0x6c24,
   0x6a6d, 0x68a6, 0x66cf, 0x64e8, 0x62f2, 0x60ec, 0x5ed7, 0x5cb4,
   0x5a82, 0x5842, 0x55f5, 0x539b, 0x5133, 0x4ebf, 0x4c3f, 0x49b4,
   0x471c, 0x447a, 0x41ce, 0x3f17, 0x3c56, 0x398c, 0x36ba, 0x33de,
   0x30fb, 0x2e11, 0x2b1f, 0x2826, 0x2528, 0x2223, 0x1f19, 0x1c0b,
   0x18f8, 0x15e2, 0x12c8, 0x0fab, 0x0c8b, 0x096a, 0x0647, 0x0324,
   0x0000, 0xfcdc, 0xf9b9, 0xf696, 0xf375, 0xf055, 0xed38, 0xea1e,
   0xe708, 0xe3f5, 0xe0e7, 0xdddd, 0xdad8, 0xd7da, 0xd4e1, 0xd1ef,
   0xcf05, 0xcc22, 0xc946, 0xc674, 0xc3aa, 0xc0e9, 0xbe32, 0xbb86,
   0xb8e4, 0xb64c, 0xb3c1, 0xb141, 0xaecd, 0xac65, 0xaa0b, 0xa7be,
   0xa57e, 0xa34c, 0xa129, 0x9f14, 0x9d0e, 0x9b18, 0x9931, 0x975a,
   0x9593, 0x93dc, 0x9236, 0x90a1, 0x8f1e, 0x8dab, 0x8c4b, 0x8afc,
   0x89bf, 0x8894, 0x877c, 0x8676, 0x8583, 0x84a3, 0x83d7, 0x831d,
   0x8276, 0x81e3, 0x8163, 0x80f7, 0x809e, 0x8059, 0x8028, 0x800a,
   0x8000, 0x800a, 0x8028, 0x8059, 0x809e, 0x80f7, 0x8163, 0x81e3,
   0x8276, 0x831d, 0x83d7, 0x84a3, 0x8583, 0x8676, 0x877c, 0x8894,
   0x89bf, 0x8afc, 0x8c4b, 0x8dab, 0x8f1e, 0x90a1, 0x9236, 0x93dc,
   0x9593, 0x975a, 0x9931, 0x9b18, 0x9d0e, 0x9f14, 0xa129, 0xa34c,
   0xa57e, 0xa7be, 0xaa0b, 0xac65, 0xaecd, 0xb141, 0xb3c1, 0xb64c,
   0xb8e4, 0xbb86, 0xbe32, 0xc0e9, 0xc3aa, 0xc674, 0xc946, 0xcc22,
   0xcf05, 0xd1ef, 0xd4e1, 0xd7da, 0xdad8, 0xdddd, 0xe0e7, 0xe3f5,
   0xe708, 0xea1e, 0xed38, 0xf055, 0xf375, 0xf696, 0xf9b9, 0xfcdc,
};

//-----------------------------------------------------------------------------
// set_DAC1
//-----------------------------------------------------------------------------

void set_DAC1(void)
{
   char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

   static unsigned phase_acc = 0;      // holds phase accumulator

   int temp1;                          // temporary 16-bit variable

   // increment phase accumulator
   phase_acc += phase_add;

   // read the table value
   temp1 = SINE_TABLE[phase_acc >> 8];

   // Add a DC bias to change the the rails from a bipolar (-32768 to 32767)
   // to unipolar (0 to 65535)
   // Note: the XOR with 0x8000 translates the bipolar quantity into
   // a unipolar quantity.

   SFRPAGE = DAC1_PAGE;

   DAC1 = 0x8000 ^ temp1;              // write to DAC1

   SFRPAGE = SFRPAGE_SAVE;             // restore SFR page

}
```

```
//-----------------------------------------------------------------------------
// bank3.c
//-----------------------------------------------------------------------------
//
// AUTH: FB
// DATE: 18 SEP 02
//
// Target: C8051F12x
// Tool chain: KEIL C51
//
// This file contains routines used by the code banking example in AN030.
// All routines in this file are located in Code Bank 3.
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>              // SFR declarations


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

sbit LED = P1^6;


//-----------------------------------------------------------------------------
// toggle_led
//-----------------------------------------------------------------------------

void toggle_led(void)
{
   LED = ~LED;

}
```



IoT Portfolio	SW/HW	Quality	Support and Community
www.silabs.com/IoT	*www.silabs.com/simplicity*	*www.silabs.com/quality*	*community.silabs.com*

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

http://www.silabs.com