AN131

PORTING CONSIDERATIONS FROM 'F02X TO 'F12X

Relevant Devices

This application note applies to the following devices:

C8051F020, C8051F021, C8051F022, C8051F023, C8051F120, C8051F121, C8051F122, C8051F123, C8051F124, C8051F125, C8051F126, and C8051F127.

Introduction

The C8051F12x family has 128K of FLASH, 8.25K of RAM, and is capable of operating at speeds up to 100MHz. This family is pin compatible with the C8051F02x series, but due to added flexibility and functionality, is not code compatible.

This application note discusses differences between the C8051F12x series and the C8051F02x series. The main topics include clocking, SFR paging, code banking, and caching. Example initialization routines for the C8051F12x series and a checklist to use when porting a project from a C8051F02x to a C8051F12x device are included at the end of this note.

Key Points

- The C8051F12x series is pin compatible with the C8051F02x series but is not code compatible.
- Most of the new features in the C8051F12x series, such as the instruction cache and code banking registers, may be left at their default settings.
- The 'F12x devices implement 'SFR Paging'. To correctly read or write to an SFR register, the SFRPAGE register **must** be set to the correct SFR page.

Clocking

The main differences in clocking between the C8051F02x series and the C8051F12x series include an on-chip calibrated 24.5 MHz internal oscillator and a phase-locked loop (PLL). When porting code, be aware that the OSCICN register definition has changed and a new register, CLKSEL, has been added to accommodate the increased clocking flexibility.

The 24.5 MHz Internal Oscillator

The C8051F12x series has a calibrated 24.5 MHz (+/- 2%) internal oscillator, instead of the 16 MHz (+/- 20%) internal oscillator on the C8051F02x. On reset, the system starts operating at a frequency of approximately 3 MHz instead of 2 MHz.

Using the PLL to achieve operating frequencies up to 100 MHz

Operating 'F12x devices at frequencies greater than 30 MHz is accomplished by using the PLL to multiply a lower frequency oscillator source.

The input frequency range for the PLL is 5 to 30 MHz and can be derived from the internal or external oscillator. Given a stable input signal, the PLL output can have a wide range of frequencies based on the values of PLL0MUL and PLL0DIV. Keep in mind that the input clock signal is divided by PLL0DIV before it is fed to the phase detector. The phase detector input must be between 5 and 30 MHz. The maximum output frequency of the PLL is limited by the maximum operating frequency of the device.

Example code showing how to initialize the PLL is included at the end of this note. Please refer to the Oscillators section of the C8051F12x datasheet for step-by-step instructions for initializing the PLL.

SFR Paging

The C8051F12x series implements a ‘paged’ SFR scheme which greatly expands the number of available SFR addresses. This SFR address expansion provides support for more peripherals and gives the programmer added flexibility. For example, Port 4 through Port 7 now occupy bit-addressable SFR locations.

Reading and Writing to SFR Registers

To correctly read or write to an SFR, the SFRPAGE register must be set to the correct SFR page. The SFRPAGE register is accessible from all SFR pages. When changing SFR pages, it is recommended to use the named constants in Figure 1 instead of assigning the actual SFR page number. These are defined in the supplied ‘C8051F120.h’ and ‘C8051F120.inc’ files. This enhances code readability and reduces the porting effort to future platforms.

Figure 1. SFR Page Names

SFR Page Names	SFR Page Number
LEGACY_PAGE TIMER01_PAGE UART0_PAGE SPI0_PAGE EMI0_PAGE ADC0_PAGE SMB0_PAGE TMR2_PAGE DAC0_PAGE PCA0_PAGE	0x00

Figure 1. SFR Page Names

SFR Page Names	SFR Page Number
CPT0_PAGE UART1_PAGE TMR3_PAGE DAC1_PAGE	0x01
CPT1_PAGE ADC2_PAGE TMR4_PAGE	0x02
CONFIG_PAGE PLL0_PAGE	0x0F

SFR Paging and Interrupts

By default, SFR page switching is handled automatically by hardware when an interrupt occurs. Upon entry into an interrupt service routine (ISR), the SFRPAGE register will automatically switch to the SFR page containing the flag bit that caused the interrupt. Upon exit of the ISR, the SFR page is automatically restored to the SFR page in use prior to the interrupt.

For more information on SFR paging, please see the CIP-51 section of the C8051F12x datasheet.

Code Banking

The C8051F12x series supports code banking for projects requiring greater than 64KB of FLASH. All code bank switching is handled by writing to the PSBANK register. Projects smaller than 64KB can leave the PSBANK register at its default setting which provides a 64KB linear address space.

When code banking is used, the common area (FLASH addresses between 0x0000 to 0x7FFF) is always available regardless of the PSBANK register. The address space from 0x8000 to 0xFFFF can be mapped to one of 4 physical 32KB banks of FLASH, depending on the value of PSBANK. Please see the FLASH and CIP-51 sections of the C8051F12x datasheet for more detailed information on the code banking architecture.

For larger projects, the user has the option of manually handling the bank switching in software or setting up a code banked project and allowing the linker to manage the bank switching. The advantages and disadvantages of both methods are discussed below.

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.

Bank selection for constant data (accessed via MOVC and MOVX instructions) is handled independently of bank selection for instruction fetches (normal code execution). The IFBANK bits, which control the instruction fetch operations, should be left at their reset values, targeting Bank 1. The COBANK bits, which control constant operations, should be set to select the desired bank before reading, writing, or erasing FLASH. If an interrupt changes the COBANK bits, it should restore them prior to ISR exit. The PSBANK register is not restored by hardware and should be managed by software.

Project-Managed Bank Switching

Allowing the linker to manage code banking is a must for projects that have more than 64KB of executable program code. It allows functions in one bank to call functions located in another bank without the programmer having to worry about bank switching. There is a restriction, however. Constant code variables and tables must be located in the common area or in the bank containing the function which accesses them. For more information on this topic and for step-by-step instructions on how to set up a code-banked project, please refer to AN130 on the Silicon Labs website.

Caching

The ‘F12x family of devices possess a branch target buffer and a pre-fetch engine which provide optimal performance for a broad range of circumstances. In most applications, the cache control registers should be left in their reset states. Please refer to the C8051F12x datasheet for more information on the cache controller.

Interrupt Vector Table

The interrupt vector table in the ‘F12x is different from the ‘F02x interrupt vector table. External Interrupt 6, External Interrupt 7, and the External Crystal OSC Ready interrupts have been removed. The ADC2 Window Comparator interrupt has been added as interrupt 17 (0x008B) and the ADC1 End of Conversion interrupt has been renamed to ADC2 End of Conversion and moved to interrupt 18

(0x0093). The interrupt vector changes are outlined in the tables below.

Figure 2. Interrupts Added to the ‘F12x

Interrupt Source	Interrupt Vector	Priority
ADC2 Window Comparator	0x008B	17

Figure 3. Interrupts No Longer Present in the ‘F12x

Interrupt Source	Interrupt Vector	Priority
External Interrupt 6	0x0093	18
External Interrupt 7	0x009B	19
External Crystal OSC Ready	0x00AB	21

Figure 4. Interrupts That Have Changed Locations in the ‘F12x

Interrupt Source	Interrupt Vector	Priority
ADC1 End of Conversion is now ADC2 End of Conversion	0x008B 0x0093	17 18

Device Comparison and Porting Checklist

When porting a project from a C8051F02x device to a C8051F12x device, some code modifications are required and others are made to fully utilize the enhanced flexibility and performance of the C8051F12x devices. Both types of modifications are discussed below, sorted by peripheral.

Analog-to-Digital Converter (ADC)

Both the 12-bit and 10-bit versions of ADC0 are identical to ADC0 on the ‘F02x devices. ADC1 (8-bit 500 ksps) has been renamed to ADC2 on the ‘F12x. ADC2 now supports differential mode in addition to single-ended mode and has its own Programmable Window Comparator. Also, the CNVSTR signal has been renamed to CNVSTR0. Please refer to the C8051F12x datasheet for ADC2 configuration information. Note that SFRPAGE should be set to ADC0_PAGE or ADC2_PAGE when reading or writing the corresponding ADC registers.

Digital-to-Analog Converter (DAC)

Both DAC0 and DAC1 on the ‘F12x are identical to DAC0 and DAC1 on the ‘F02x. However, be sure to set the SFRPAGE to DAC0_PAGE or DAC1_PAGE before reading or writing to any DAC registers.

Voltage Reference (VREF)

VREF1 on the ‘F02x has been renamed to VREF2 on the ‘F12x. Pin locations are unchanged. Note that SFRPAGE should be set to LEGACY_PAGE before any reads or writes to REF0CN.

Comparators

The comparators on the ‘F12x have been enhanced over the equivalent ‘F02x comparators with a new speed/power selection capability. As a result, two new registers, CPT0MD and CPT1MD, have been added to accommodate this new feature. These registers give the user the option of putting comparator 0 and comparator 1 in a low power mode. The reset value for these registers leaves the speed and power consumption of ‘F12x compara-

tors equivalent to the speed and power consumption of the ‘F02x comparators. Note that SFRPAGE should be set to CPT0_PAGE or CPT1_PAGE when reading or writing comparator registers.

Reset Sources

Forcing a power-on reset via software in the ‘F12x is accomplished by writing a ‘1’ to PINRSF (bit 0 in the RSTSRC register) instead of PORSF (bit 1 in the RSTSRC register) as in the ‘F02x. PORSF in the ‘F12x has been changed from read/write to read only.

The instruction prefetch engine must be enabled in order to disable the watchdog timer. The instruction prefetch engine is enabled by default upon reset.

Note that SFRPAGE should be set to LEGACY_PAGE before any reads or writes to RSTSRC.

Oscillators

Please see the “Clocking” section of this document.

FLASH Memory

Please see the “Code Banking” section of this document.

External Memory Interface (EMIF)

The external memory interface on the ‘F12x is identical to the one on the ‘F02x. However, because the ‘F12x devices can operate significantly faster than the ‘F02x devices, be sure to check the timing requirements for devices on the bus. Note that SFRPAGE should be set to LEGACY_PAGE before any reads or writes to EMIF registers.

Port Input/Output

Port 4 through Port 7 now occupy bit addressable SFR locations on the ‘F12x. Separate PnMDOUT registers have been added for each port and the P74OUT register has been removed. Note that SFRPAGE should be set to CONFIG_PAGE when reading or writing to Port 4 through Port 7 and the port input and output mode (PnMDIN and PnMDOUT) registers.

The CEX5 and CNVSTR2 signals have been added as Crossbar inputs. The CNVSTR signal has been renamed to CNVSTR0. If the application does not use any of the newly added or renamed signals, then the Crossbar configuration code will not need modification. Note that SFRPAGE should be set to CONFIG_PAGE before reading or writing to crossbar registers.

System Management Bus/I2C Bus (SMBUS0)

The formula for calculating SMB0CR has changed on the ‘F12x. Please refer to the System Management Bus section of the C8051F12x datasheet for more information about SMBUS0. Note that SFRPAGE should be set to SMB0_PAGE before any reads or writes to SMBUS0 registers.

Enhanced Serial Peripheral Interface (SPI0)

The ‘F12x series features an enhanced Serial Peripheral Interface. The enhanced SPI0 supports double buffered transmits and multi-byte transactions when in slave mode. Also, it can now operate in 3-wire or 4-wire mode making the NSS signal optional. The SPI0 configuration registers have changed on the ‘F12x. For more information on the enhanced SPI0, please refer to the C8051F12x datasheet. Note that SFRPAGE should be set to SPI0_PAGE before any reads or writes to SPI0 registers.

UART

For UART0 on the ‘F12x, timer selection for baud rate generation has been moved to the newly added SSTA0 register. UART0 now supports using

Timer 1, Timer 2, Timer 3, or Timer 4 as its baud rate source.

UART1 no longer requires an external crystal for baud rate generation when used with the calibrated 24.5 MHz internal oscillator. Due to these changes, the baud rate calculation equations have changed. UART1 supports using Timer 1 as its baud rate source. Hardware address decoding, synchronous mode, and fixed baud rate mode are not supported by UART1.

Note that SFRPAGE should be set to UART0_PAGE or UART1 PAGE before accessing any UART registers. Be aware that the timer registers may not appear on the same page as the UART registers. SFRPAGE should also be set to the proper UART page prior to calls to 'printf' or to other input/output stream functions to direct the operations to UART0 or UART1.

Timers

For Timer 0 and Timer 1, an additional prescaler has been added which allows them to be clocked from SYSCLK, SYSCLK divided by 4, SYSCLK divided by 12, or SYSCLK divided by 48. Timer 0 and Timer 1 SFRs are located on the "TIMER01_PAGE" SFR page.

Timer 2, Timer 3, and Timer 4 on the 'F12x are enhanced forms of the equivalent 'F02x timers. These new timers support output toggle mode, and down count capability. For more information on these timers, please refer to the C8051F12x datasheet. Note that SFRPAGE should be set to TMR2_PAGE, TMR3_PAGE, or TMR4_PAGE before any reads or writes to Timer 2, 3, or 4 registers, respectively.

Programmable Counter Array (PCA)

In the 'F12x, an additional capture/compare module was added for a total of 6 capture/compare modules. Note that SFRPAGE should be set to PCA0_PAGE before any reads or writes to PCA0 registers.

JTAG

The JTAG device ID has changed for the 'F12x series. See the JTAG section of the 'F12x datasheet for details on the JTAG interface.

Software Examples

Example 1

```
//-----------------------------------------------------------------------------
// F12x_INIT_1.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 19 SEP 02
//
// This file contains example initialization routines for the C8051F12x series
// of devices.
//
// This program uses the the 24.5 MHz internal oscillator multiplied by two
// for an effective SYSCLK of 49 MHz. This program also initializes and uses
// 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

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


//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void main(void);
void SYSCLK_Init(void);
void PORT_Init(void);
void UART1_Init (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


   SFRPAGE = UART1_PAGE;               // Direct printf output to UART1
   printf("Hello\n");                  // Print a string

   while(1);

}

//-----------------------------------------------------------------------------
// 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

}
```

Example 2

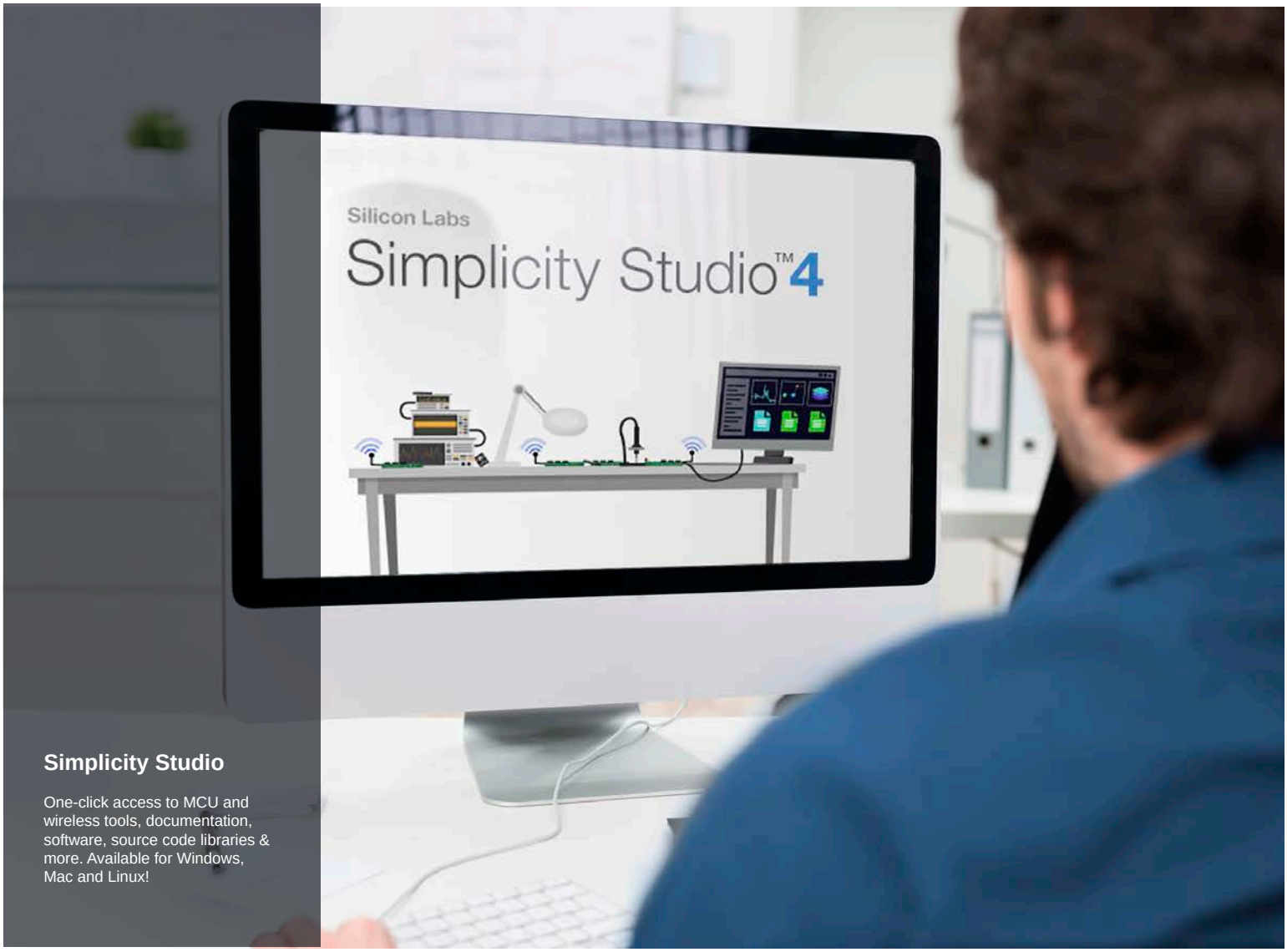
```
//-----------------------------------------------------------------------------
// F12x_INIT_2.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 19 SEP 02
//
// This file contains example initialization routines for the C8051F12x series
// of devices.
//
// This program uses a 22.1184 Mhz crystal oscillator multiplied by (9/4)
// for an effective SYSCLK of 49.7664 Mhz. This program also initializes and
// uses UART0 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 EXTCLK        22118400          // External oscillator frequency in Hz
#define SYSCLK        49760000          // 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.

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


//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void main(void);
void SYSCLK_Init(void);
void PORT_Init(void);
void UART0_Init (void);

//-----------------------------------------------------------------------------
// 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


   SFRPAGE = UART0_PAGE;                // Direct printf output to UART0
   printf("Hello\n");                   // Print a string

   while(1);

}

//-----------------------------------------------------------------------------
// 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
}
```



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com