SILICON LABORATORIES

AN122

ANNOTATED 'C' EXAMPLES FOR THE 'F02X FAMILY

Relevant Devices

This application note applies to the following devices:

C8051F020, C8051F021, C8051F022, and C8051F023.

Introduction

This note contains example code written in 'C' that can be used as a starting point for the development of applications based on the C8051F02x family of devices.

Index of Programs by Peripheral

The following short descriptions provide an index to the attached programs, organized by peripheral.

ADC0 Examples

The following are example programs which use ADC0.

"ADC0_Buf1.c"

This program shows an example of using ADC0 in interrupt mode using Timer3 overflows as a start-of-conversion source to sample AIN0 <NUM_SAMPLES> times, storing the results in XDATA space. Once <NUM_SAMPLES> have been collected, the samples are transmitted out UART0. Once the transmission has completed, another <NUM_SAMPLES> of data are collected and the process repeats.

"ADC0_Int1.c"

This program shows an example of using ADC0 in interrupt mode using Timer3 overflows as a start-of-conversion to measure the output of the on-chip temperature sensor. The temperature is calculated from the ADC0 result and is transmitted out UART0.

"ADC0_Int2m.c"

This program shows an example of using ADC0 in interrupt mode using Timer3 overflows as a start-of-conversion to measure the the voltages on AIN0 through AIN7 and the temperature sensor. The voltages are calculated from the resulting codes and are transmitted out UART0.

"ADC0_OSA1.c"

This program shows an example of using ADC0 in interrupt mode using Timer3 overflows as a start-of-conversion to measure the output of the on-chip temperature sensor. The ADC0 results are filtered by a simple integrate-and-dump process whose integrate/decimate ratio is given by the constant <INT_DEC>. The temperature is calculated from the ADC0 result and is transmitted out UART0.

"ADC0_Poll1.c"

This program demonstrates operation of ADC0 in polled mode. The ADC0 is configured to use writes to AD0BUSY as its start of conversion source and to measure the output of the on-chip temperature sensor. The temperature sensor output is converted to degrees Celsius and is transmitted out UART0.

DAC0 Examples

The following are example programs which use DAC0. These can easily be converted to use DAC1 if desired.

“DAC0_DTMF1.c”

Example source code which outputs DTMF tones on DAC0. DAC0's output is scheduled to update at a rate determined by the constant <SAMPLERATED>, managed and timed by Timer4.

Oscillator Examples

The following are example programs which configure the internal and external oscillators. They also show how to measure the internal and external oscillator frequency and implement a real time clock.

“OSC_Cry1.c”

This program shows an example of how to configure the External Oscillator to drive a 22.1184 MHz crystal and to select this external oscillator as the system clock source. Also enables the missing clock detector reset function. Assumes an 22.1184 MHz crystal is attached between XTAL1 and XTAL2.

“OSC_Int1.c”

This program shows an example of how to configure the internal oscillator to its maximum frequency (~16 MHz). Also enables the Missing Clock Detector reset function.

“INT_OSC_Measure1”

This program shows an example of how the external oscillator can be used to measure the internal oscillator frequency. In this example, the internal oscillator is set to its highest setting. The external oscillator is configured for a 22.1184 MHz crystal. The PCA counter is used as a generic 16-bit counter that uses EXTCLK / 8 as its time base. We configure Timer0 to count SYSCLKs, and count the number of INTCLKs in 1 second's worth of EXTCLK / 8

“EXT_OSC_Measure1”

This program shows an example of how the internal oscillator can be used to measure the external oscillator frequency. In this example, the internal oscillator is set to its highest setting. The external oscillator is configured for a high frequency crystal. The PCA counter is used as a generic 16-bit counter that uses EXTCLK / 8 as its time base. We configured Timer0 to count SYSCLKs, and count the number of EXTCLK/8 ticks in 16 million SYSCLKs (or the number of SYSCLKs in 1 second) to obtain the external oscillator frequency.

“OSC_RTC_Cal1”

This program shows an example of how an external crystal oscillator can be used to measure the internal oscillator frequency to a sufficient degree to enable UART operation (better than +/- 2.5%). In this example, a 32.768kHz watch crystal (with associated loading capacitors) is connected between XTAL1 and XTAL2. The PCA counter is used as a generic 16-bit counter that uses EXTCLK / 8 as its time base. We preload it to generate an overflow in 8 counts. Timer0 is configured as a 16-bit counter that is set to count SYSCLKs. The internal oscillator is configured to its highest setting, and provides the system clock source.

A set of real time clock (RTC) values for seconds, minutes, hours, and days are maintained by the interrupt handler for Timer 3, which is configured to use EXTCLK / 8 as its time base and to reload every 4096 counts. This generates an interrupt once every second.

Timer Examples

“Timer0_Poll1.c”

This program shows an example of using Timer0 in polled mode to implement a delay counter with a resolution of 1 ms.

External Memory Interface (EMIF) Examples

“EMIF_1.c”

This program configures the external memory interface to read and write to an external SRAM mapped to the upper port pins (P4-7).

UART Examples

The following examples show how to use UART0 and UART1 in polled mode and in interrupt mode.

“UART0_Stdio1”

This program configures UART0 to operate in polled mode, suitable for use with the <stdio> functions printf() and scanf(), to which examples areprovided. Assumes an 22.1184 MHz crystal is attached between XTAL1 and XTAL2. The system clock frequency is stored in a global constant SYSCLK. The target UART baud rate is stored in a global constant BAUDRATE.

“UART0 Autobaud1”

This program shows an example of how the PCA can be used to enable accurate UART auto-baud detection when running from the on-chip internal oscillator. This algorithm assumes a 0x55 character (ASCII "U") is sent from the remote transmitter. Baud rates between 4800 to 19.2kbps can be reliably synchronized. UART0 is then configured to operate in polled mode, suitable for use with the <stdio> functions printf() and scanf().

“UART0_Int1”

This program configures UART0 to operate in interrupt mode, showing an example of a string transmitter and a string receiver. These strings are assumed to be NULL-terminated. Assumes an 22.1184 MHz crystal is attached between XTAL1 and XTAL2. The system clock frequency is stored in a global constant SYSCLK. The target UART baud rate is stored in a global constant BAUDRATE.

“UART1_Int1”

This program is the same as “UART0_Int1” except it uses UART1.

FLASH Examples

“FLASH_Scratch”

This program illustrates how to erase, write, and read FLASH memory from application code written in 'C' and exercises the upper 128-byte FLASH sector.

PCA Examples

“Freq_Gen1”

This program uses the PCA in Frequency Output mode to generate a square wave on P0.0.

SPI Examples

“SPI_EE_Pol1”

This program shows an example of how to interface to a SPI EEPROM using the SPI0 interface in polled-mode. The SPI EEPROM used here is a Microchip 25LC320 (4k bytes). Assumes a 22.1184MHz crystal is attached between XTAL1 and XTAL2.

“SPI_EE_Int1”

This program is the same as SPI_EE_Pol1 execpt it uses the SPI0 interface in interrupt mode.

Example Code

"ADC0_Buf1.c"

```
//-----------------------------------------------------------------------------
// ADC0_Buf1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 27 AUG 01
//
// This program shows an example of using ADC0 in interrupt mode using Timer3
// overflows as a start-of-conversion to sample AIN0 <NUM_SAMPLES> times,
// storing the results in XDATA space.  Once <NUM_SAMPLES> have been
// collected, the samples are transmitted out UART0.  Once the transmission
// has completed, another <NUM_SAMPLES> of data are collected and the process
// repeats.
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and  XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.  The
// ADC0 sampling rate is stored in a global constant SAMPLERATE0.  The number
// of samples collected during each batch is stored in <NUM_SAMPLES>.  The
// maximum value of <NUM_SAMPLES> is 2048 on a C8051F02x device with 4096
// bytes of XRAM (assuming no external RAM is connected to the External
// Memory Interface).
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     115200            // Baud rate of UART in bps
#define SAMPLERATE0  50000             // ADC0 Sample frequency in Hz
#define NUM_SAMPLES  2048              // number of ADC0 samples to take in
                                       // sequence
#define TRUE 1
#define FALSE 0

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void ADC0_Init (void);
void Timer3_Init (int counts);
void ADC0_ISR (void);

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

xdata unsigned samples[NUM_SAMPLES];   // array to store ADC0 results
bit ADC0_DONE;                         // TRUE when NUM_SAMPLES have been
                                       // collected

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

void main (void) {
   int i;                              // loop counter

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

   SYSCLK_Init ();                     // initialize oscillator
   PORT_Init ();                       // initialize crossbar and GPIO
   UART0_Init ();                      // initialize UART0
   Timer3_Init (SYSCLK/SAMPLERATE0);   // initialize Timer3 to overflow at
                                       // desired ADC0 sample rate

   ADC0_Init ();                       // init ADC

   EA = 1;                             // Enable global interrupts

   while (1) {
      // collect samples...
      ADC0_DONE = FALSE;
      LED = 1;                         // turn LED on during sample process
      EIE2 |= 0x02;                    // enable ADC0 interrupts
```

```
      while (ADC0_DONE == FALSE);       // wait for samples to be taken

      // upload samples to UART0
      LED = 0;                          // turn LED off during upload process
      for (i = 0; i < NUM_SAMPLES; i++) {
         printf ("%u\n", samples[i]);
      }
      printf ("\n");
   }
}

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

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

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

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

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

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

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

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
```

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

//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
//
// Configure ADC0 to use Timer3 overflows as conversion source, to
// generate an interrupt on conversion complete, and to use left-justified
// output mode.  Enables ADC end of conversion interrupt. Enables ADC0, but
// leaves ADC0 end-of-conversion interrupts disabled.
//
void ADC0_Init (void)
{
   ADC0CN = 0x05;                     // ADC0 disabled; normal tracking
                                      // mode; ADC0 conversions are initiated
                                      // on overflow of Timer3; ADC0 data is
                                      // left-justified
   REF0CN = 0x07;                     // enable temp sensor, on-chip VREF,
                                      // and VREF output buffer
   AMX0SL = 0x00;                     // Select AIN0 as ADC mux output
   ADC0CF = (SYSCLK/2500000) << 3;    // ADC conversion clock = 2.5MHz
   ADC0CF &= ~0x07;                   // PGA gain = 1
   EIE2 &= ~0x02;                     // disable ADC0 interrupts

   AD0EN = 1;                         // enable ADC0
}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
//
// Configure Timer3 to auto-reload at interval specified by <counts> (no
// interrupt generated) using SYSCLK as its time base.
//
void Timer3_Init (int counts)
{
   TMR3CN = 0x02;                     // Stop Timer3; Clear TF3;
                                      // use SYSCLK as timebase
   TMR3RL  = -counts;                 // Init reload values
   TMR3    = 0xffff;                  // set to reload immediately
   EIE2   &= ~0x01;                   // disable Timer3 interrupts
   TMR3CN |= 0x04;                    // start Timer3
}

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

//-----------------------------------------------------------------------------
// ADC0_ISR
```

```
//-----------------------------------------------------------------------------
//
// ADC0 end-of-conversion ISR
// Here we take the ADC0 sample and store it in the global array <samples[]>
// and update the local sample counter <num_samples>.  When <num_samples> ==
// <NUM_SAMPLES>, we disable ADC0 end-of-conversion interrupts and post
// ADC0_DONE = 1.
//
void ADC0_ISR (void) interrupt 15 using 3
{
   static unsigned num_samples = 0;    // ADC0 sample counter

   AD0INT = 0;                         // clear ADC0 conversion complete
                                       // indicator

   samples[num_samples] = ADC0;        // read and store ADC0 value

   num_samples++;                      // update sample counter

   if (num_samples == NUM_SAMPLES) {
      num_samples = 0;                 // reset sample counter
      EIE2 &= ~0x02;                   // disable ADC0 interrupts
      ADC0_DONE = 1;                   // set DONE indicator
   }
}
```

“ADC0_Int1.c”

```
//-----------------------------------------------------------------------------
// ADC0_int1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 18 AUG 01
//
// This program shows an example of using ADC0 in interrupt mode using Timer3
// overflows as a start-of-conversion to measure the output of the on-chip
// temperature sensor.  The temperature is calculated from the ADC0 result
// and is transmitted out UART0.
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and  XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.  The
// ADC0 sampling rate is stored in a global constant SAMPLERATE0.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     9600              // Baud rate of UART in bps
#define SAMPLERATE0  50000             // ADC0 Sample frequency in Hz

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void ADC0_Init (void);
void Timer3_Init (int counts);
void ADC0_ISR (void);

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

long result;                          // ADC0 decimated value

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

void main (void) {
   long temperature;                  // temperature in hundredths of a
                                      // degree C
   int temp_int, temp_frac;           // integer and fractional portions of
                                      // temperature

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

   SYSCLK_Init ();                    // initialize oscillator
   PORT_Init ();                      // initialize crossbar and GPIO
   UART0_Init ();                     // initialize UART0
   Timer3_Init (SYSCLK/SAMPLERATE0);  // initialize Timer3 to overflow at
                                      // sample rate

   ADC0_Init ();                      // init ADC

   AD0EN = 1;                         // enable ADC

   EA = 1;                            // Enable global interrupts

   while (1) {
      EA = 0;                         // disable interrupts
      temperature = result;           // get ADC value from global variable
      EA = 1;                         // re-enable interrupts

      // calculate temperature in hundredths of a degree
      temperature = temperature - 41380;
      temperature = (temperature * 100L) / 156;
      temp_int = temperature / 100;
      temp_frac = temperature - (temp_int * 100);
      printf ("Temperature is %+02d.%02d\n", temp_int, temp_frac);
   }
}

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

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

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

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

   for (i=0; i < 256; i++) ;           // XTLVLD blanking interval (>1ms)

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

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

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

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

//-----------------------------------------------------------------------------
// ADC0_Init
```

```
//-----------------------------------------------------------------------------
//
// Configure ADC0 to use Timer3 overflows as conversion source, to
// generate an interrupt on conversion complete, and to use left-justified
// output mode.  Enables ADC end of conversion interrupt. Leaves ADC disabled.
//
void ADC0_Init (void)
{
   ADC0CN = 0x05;                      // ADC0 disabled; normal tracking
                                       // mode; ADC0 conversions are initiated
                                       // on overflow of Timer3; 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 ADC interrupts
}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
//
// Configure Timer3 to auto-reload at interval specified by <counts> (no
// interrupt generated) using SYSCLK as its time base.
//
void Timer3_Init (int counts)
{
   TMR3CN = 0x02;                      // Stop Timer3; Clear TF3;
                                       // use SYSCLK as timebase
   TMR3RL  = -counts;                  // Init reload values
   TMR3    = 0xffff;                   // set to reload immediately
   EIE2   &= ~0x01;                    // disable Timer3 interrupts
   TMR3CN |= 0x04;                     // start Timer3
}

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

//-----------------------------------------------------------------------------
// ADC0_ISR
//-----------------------------------------------------------------------------
//
// ADC0 end-of-conversion ISR
// Here we take the ADC0 sample and store it in the global variable <result>.
//
void ADC0_ISR (void) interrupt 15
{
   AD0INT = 0;                         // clear ADC conversion complete
                                       // indicator

   result = ADC0;                      // read ADC value
}
```

"ADC0_Int2m.c"

```
//-----------------------------------------------------------------------------
// ADC0_int2m.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 25 AUG 01
//
// This program shows an example of using ADC0 in interrupt mode using Timer3
// overflows as a start-of-conversion to measure the the voltages on AIN0
// through AIN7 and the temperature sensor.  The voltages are calculated from
// the resulting codes and are transmitted out UART0.
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and  XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.  The
// ADC0 sampling rate is stored in a global constant SAMPLERATE0.  The voltage
// reference value is stored in a constant VREF0, and is used to convert the
// resulting codes from the ADC0 measurements into a voltage.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     9600              // Baud rate of UART in bps
#define SAMPLERATE0  50000             // ADC0 Sample frequency in Hz
#define VREF0        2430              // VREF voltage in millivolts
```

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void ADC0_Init (void);
void Timer3_Init (int counts);
void ADC0_ISR (void);

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

long result[9];                        // AIN0-7 and temp sensor output
                                       // results

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

void main (void) {
   long voltage;                       // voltage in millivolts
   int i;                              // loop counter
                                       // voltage

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

   SYSCLK_Init ();                     // initialize oscillator
   PORT_Init ();                       // initialize crossbar and GPIO
   UART0_Init ();                      // initialize UART0
   Timer3_Init (SYSCLK/SAMPLERATE0);   // initialize Timer3 to overflow at
                                       // sample rate

   ADC0_Init ();                       // init ADC

   AD0EN = 1;                          // enable ADC

   EA = 1;                             // Enable global interrupts

   while (1) {
      for (i = 0; i < 9; i++) {
         EA = 0;                       // disable interrupts
         voltage = result[i];          // get ADC value from global variable
         EA = 1;                       // re-enable interrupts

         // calculate voltage in millivolts
         voltage = voltage * VREF0;
         voltage = voltage >> 16;
         printf ("Channel '%d' voltage is %ldmV\n", i, voltage);
      }
   }
}
```

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

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

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

   for (i=0; i < 256; i++) ;           // XTLVLD blanking interval (>1ms)

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

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

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

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

```
//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
//
// Configure ADC0 to use Timer3 overflows as conversion source, to
// generate an interrupt on conversion complete, and to use left-justified
// output mode.  Enables ADC end of conversion interrupt. Leaves ADC disabled.
// Note: here we also enable low-power tracking mode to ensure that minimum
// tracking times are met when ADC0 channels are changed.
//
void ADC0_Init (void)
{
   ADC0CN = 0x45;                      // ADC0 disabled; low-power tracking
                                       // mode; ADC0 conversions are initiated
                                       // on overflow of Timer3; ADC0 data is
                                       // left-justified
   REF0CN = 0x07;                      // enable temp sensor, on-chip VREF,
                                       // and VREF output buffer
   AMX0SL = 0x00;                      // Select AIN0 as ADC mux output
   ADC0CF = (SYSCLK/2500000) << 3;     // ADC conversion clock = 2.5MHz
   ADC0CF &= ~0x07;                    // PGA gain = 1

   EIE2 |= 0x02;                       // enable ADC interrupts
}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
//
// Configure Timer3 to auto-reload at interval specified by <counts> (no
// interrupt generated) using SYSCLK as its time base.
//
void Timer3_Init (int counts)
{
   TMR3CN = 0x02;                      // Stop Timer3; Clear TF3;
                                       // use SYSCLK as timebase
   TMR3RL  = -counts;                  // Init reload values
   TMR3    = 0xffff;                   // set to reload immediately
   EIE2   &= ~0x01;                    // disable Timer3 interrupts
   TMR3CN |= 0x04;                     // start Timer3
}

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

//-----------------------------------------------------------------------------
// ADC0_ISR
//-----------------------------------------------------------------------------
//
// ADC0 end-of-conversion ISR
// Here we take the ADC0 sample and store it in the global array <result>.
// We also select the next channel to convert.
//
void ADC0_ISR (void) interrupt 15
{
   static unsigned char channel = 0;   // ADC mux channel (0-8)
```

```
   AD0INT = 0;                            // clear ADC conversion complete
                                          // indicator

   result[channel] = ADC0;                // read ADC value

   channel++;                             // change channel
   if (channel == 9) {
      channel = 0;
   }

   AMX0SL = channel;                      // set mux to next channel
}
```

"ADC0_OSA1.c"

```
//-----------------------------------------------------------------------------
// ADC0_OSA1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 18 AUG 01
//
// This program shows an example of using ADC0 in interrupt mode using Timer3
// overflows as a start-of-conversion to measure the output of the on-chip
// temperature sensor.  The ADC0 results are filtered by a simple integrate-
// and-dump process whose integrate/decimate ratio is given by the constant
// INT_DEC.  The temperature is calculated from the ADC0 result
// and is transmitted out UART0.
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and  XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.  The
// ADC0 sampling rate is stored in a global constant SAMPLERATE0.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     9600              // Baud rate of UART in bps
#define SAMPLERATE0  50000             // ADC0 Sample frequency in Hz
#define INT_DEC      256               // integrate and decimate ratio
```

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void ADC0_Init (void);
void Timer3_Init (int counts);
void ADC0_ISR (void);

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

long result;                        // ADC0 decimated value

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

void main (void) {
   long temperature;                // temperature in hundredths of a
                                    // degree C
   int temp_int, temp_frac;         // integer and fractional portions of
                                    // temperature

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

   SYSCLK_Init ();                  // initialize oscillator
   PORT_Init ();                    // initialize crossbar and GPIO
   UART0_Init ();                   // initialize UART0
   Timer3_Init (SYSCLK/SAMPLERATE0);   // initialize Timer3 to overflow at
                                    // sample rate

   ADC0_Init ();                    // init ADC

   AD0EN = 1;                       // enable ADC

   EA = 1;                          // Enable global interrupts

   while (1) {
      EA = 0;                       // disable interrupts
      temperature = result;
      EA = 1;                       // re-enable interrupts

      // calculate temperature in hundredths of a degree
      temperature = temperature - 41380;
      temperature = (temperature * 100L) / 156;
      temp_int = temperature / 100;
      temp_frac = temperature - (temp_int * 100);
      printf ("Temperature is %+02d.%02d\n", temp_int, temp_frac);

      LED = ~SW1;                   // LED reflects state of switch
   }
```

```
}

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

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

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

   for (i=0; i < 256; i++) ;           // XTLVLD blanking interval (>1ms)

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

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

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

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

```
}

//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
//
// Configure ADC0 to use Timer3 overflows as conversion source, to
// generate an interrupt on conversion complete, and to use left-justified
// output mode.  Enables ADC end of conversion interrupt. Leaves ADC disabled.
//
void ADC0_Init (void)
{
   ADC0CN = 0x05;                      // ADC0 disabled; normal tracking
                                       // mode; ADC0 conversions are initiated
                                       // on overflow of Timer3; 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 ADC interrupts
}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
//
// Configure Timer3 to auto-reload at interval specified by <counts> (no
// interrupt generated) using SYSCLK as its time base.
//
void Timer3_Init (int counts)
{
   TMR3CN = 0x02;                      // Stop Timer3; Clear TF3;
                                       // use SYSCLK as timebase
   TMR3RL  = -counts;                  // Init reload values
   TMR3    = 0xffff;                   // set to reload immediately
   EIE2   &= ~0x01;                    // disable Timer3 interrupts
   TMR3CN |= 0x04;                     // start Timer3
}

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

//-----------------------------------------------------------------------------
// ADC0_ISR
//-----------------------------------------------------------------------------
//
// ADC0 end-of-conversion ISR
// Here we take the ADC0 sample, add it to a running total <accumulator>, and
// decrement our local decimation counter <int_dec>.  When <int_dec> reaches
// zero, we post the decimated result in the global variable <result>.
//
void ADC0_ISR (void) interrupt 15
{
   static unsigned int_dec=INT_DEC;    // integrate/decimate counter
                                       // we post a new result when
```

```
                                      // int_dec = 0
   static long accumulator=0L;        // here's where we integrate the
                                      // ADC samples

   AD0INT = 0;                        // clear ADC conversion complete
                                      // indicator

   accumulator += ADC0;               // read ADC value and add to running
                                      // total
   int_dec--;                         // update decimation counter

   if (int_dec == 0) {                // if zero, then post result
      int_dec = INT_DEC;              // reset counter
      result = accumulator >> 8;
      accumulator = 0L;               // reset accumulator
   }
}
```

“ADC0_Poll1.c”

```
//-----------------------------------------------------------------------------
// ADC0_Poll1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 18 AUG 01
//
// This program demonstrates operation of ADC0 in polled mode.  The ADC0 is
// configured to use writes to AD0BUSY as its start of conversion source and
// to measure the output of the on-chip temperature sensor.  The temperature
// sensor output is converted to degrees Celsius and is transmitted out UART0.
// Assumes an 22.1184MHz crystal is attached between XTAL1 and XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     9600              // Baud rate of UART in bps

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

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void ADC0_Init (void);

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

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

void main (void) {
   long temperature;                   // temperature in hundredths of a
                                       // degree C
   int temp_int, temp_frac;            // integer and fractional portions of
                                       // temperature

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

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

   ADC0_Init ();                       // init and enable ADC

   // *** check difference in tracking modes ***

   while (1) {
      AD0INT = 0;                      // clear conversion complete indicator
      AD0BUSY = 1;                     // initiate conversion
      while (AD0INT == 0);             // wait for conversion complete
      temperature = ADC0;              // read ADC0 data

      // calculate temperature in hundredths of a degree
      temperature = temperature - 41380;
      temperature = (temperature * 100L) / 156;
      temp_int = temperature / 100;
      temp_frac = temperature - (temp_int * 100);
      printf (“Temperature is %+02d.%02d\n”, temp_int, temp_frac);
   }
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use an 22.1184MHz crystal
// as its clock source.
//
void SYSCLK_Init (void)
```

```
{
   int i;                                  // delay counter

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

   for (i=0; i < 256; i++) ;               // XTLVLD blanking interval (>1ms)

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

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

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

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

//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
//
// Configure ADC0 to use AD0BUSY as conversion source, to use left-justified
// output mode, to use normal tracking mode, and to measure the output of
// the on-chip temperature sensor.  Disables ADC0 end of conversion interrupt
// and ADC0 window compare interrupt.
//
void ADC0_Init (void)
{
   ADC0CN = 0x81;                          // ADC0 enabled; normal tracking
```

```
                                            // mode; ADC0 conversions are initiated
                                            // on write to AD0BUSY; 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;                           // disable ADC0 EOC interrupt
   EIE1 &= ~0x04;                           // disable ADC0 window compare interrupt
}
```

"DAC0_DTMF1.c"

```
//-----------------------------------------------------------------------------
// DAC0_DTMF1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 27 AUG 01
//
// Target: C8051F02x
// Tool chain: KEIL C51
//
// Description:
// Example source code which outputs DTMF tones on DAC0.  DAC0's output is
// scheduled to update at a rate determined by the constant SAMPLERATED,
// managed and timed by Timer4.
//
// Implements a 256-entry full-cycle sine table of 8-bit precision.
//
// The output frequency is proportional a 16-bit phase adder.
// At each DAC update cycle, the phase adder value is added to a running
// phase accumulator.<phase_accumulator>, the upper bits of which are used
// to access the sine lookup table.
//

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

#include <c8051f020.h>              // SFR declarations

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

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

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------
#define SYSCLK 22118400             // SYSCLK frequency in Hz

#define SAMPLERATED 100000L         // update rate of DAC in Hz

#define phase_precision 65536       // range of phase accumulator

// DTMF phase adder values based on SAMPLERATED and <phase_precision>
```

```
#define LOW697697 * phase_precision / SAMPLERATED
#define LOW770 770 * phase_precision / SAMPLERATED
#define LOW852 852 * phase_precision / SAMPLERATED
#define LOW941 941 * phase_precision / SAMPLERATED

#define HI1209 1209 * phase_precision / SAMPLERATED
#define HI1336 1336 * phase_precision / SAMPLERATED
#define HI1477 1477 * phase_precision / SAMPLERATED
#define HI1633 1633 * phase_precision / SAMPLERATED

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

void main (void);
void SYSCLK_Init (void);
void PORT_Init (void);

void Timer4_Init (int counts);
void Timer4_ISR (void);

//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------
unsigned phase_add1;                 // holds low-tone phase adder
unsigned phase_add2;                 // holds low-tone phase adder

bit tone1_en;                        // enable = 1 for tone 1
bit tone2_en;                        // enable = 1 for tone 2

char code SINE_TABLE[256] = {
   0x00, 0x03, 0x06, 0x09, 0x0c, 0x0f, 0x12, 0x15,
   0x18, 0x1c, 0x1f, 0x22, 0x25, 0x28, 0x2b, 0x2e,
   0x30, 0x33, 0x36, 0x39, 0x3c, 0x3f, 0x41, 0x44,
   0x47, 0x49, 0x4c, 0x4e, 0x51, 0x53, 0x55, 0x58,
   0x5a, 0x5c, 0x5e, 0x60, 0x62, 0x64, 0x66, 0x68,
   0x6a, 0x6c, 0x6d, 0x6f, 0x70, 0x72, 0x73, 0x75,
   0x76, 0x77, 0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7c,
   0x7d, 0x7e, 0x7e, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f,
   0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7e, 0x7e,
   0x7d, 0x7c, 0x7c, 0x7b, 0x7a, 0x79, 0x78, 0x77,
   0x76, 0x75, 0x73, 0x72, 0x70, 0x6f, 0x6d, 0x6c,
   0x6a, 0x68, 0x66, 0x64, 0x62, 0x60, 0x5e, 0x5c,
   0x5a, 0x58, 0x55, 0x53, 0x51, 0x4e, 0x4c, 0x49,
   0x47, 0x44, 0x41, 0x3f, 0x3c, 0x39, 0x36, 0x33,
   0x30, 0x2e, 0x2b, 0x28, 0x25, 0x22, 0x1f, 0x1c,
   0x18, 0x15, 0x12, 0x0f, 0x0c, 0x09, 0x06, 0x03,
   0x00, 0xfd, 0xfa, 0xf7, 0xf4, 0xf1, 0xee, 0xeb,
   0xe8, 0xe4, 0xe1, 0xde, 0xdb, 0xd8, 0xd5, 0xd2,
   0xd0, 0xcd, 0xca, 0xc7, 0xc4, 0xc1, 0xbf, 0xbc,
   0xb9, 0xb7, 0xb4, 0xb2, 0xaf, 0xad, 0xab, 0xa8,
   0xa6, 0xa4, 0xa2, 0xa0, 0x9e, 0x9c, 0x9a, 0x98,
   0x96, 0x94, 0x93, 0x91, 0x90, 0x8e, 0x8d, 0x8b,
   0x8a, 0x89, 0x88, 0x87, 0x86, 0x85, 0x84, 0x84,
   0x83, 0x82, 0x82, 0x81, 0x81, 0x81, 0x81, 0x81,
   0x80, 0x81, 0x81, 0x81, 0x81, 0x81, 0x82, 0x82,
   0x83, 0x84, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
   0x8a, 0x8b, 0x8d, 0x8e, 0x90, 0x91, 0x93, 0x94,
```

```
   0x96, 0x98, 0x9a, 0x9c, 0x9e, 0xa0, 0xa2, 0xa4,
   0xa6, 0xa8, 0xab, 0xad, 0xaf, 0xb2, 0xb4, 0xb7,
   0xb9, 0xbc, 0xbf, 0xc1, 0xc4, 0xc7, 0xca, 0xcd,
   0xd0, 0xd2, 0xd5, 0xd8, 0xdb, 0xde, 0xe1, 0xe4,
   0xe8, 0xeb, 0xee, 0xf1, 0xf4, 0xf7, 0xfa, 0xfd
};


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

void main (void) {

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

   SYSCLK_Init ();

   PORT_Init ();

   REF0CN = 0x03;                   // enable internal VREF generator

   DAC0CN = 0x97;                   // enable DAC0 in left-justified mode
                                    // using Timer4 as update scheduler
   Timer4_Init(SYSCLK/SAMPLERATED); // initialize T4 to generate DAC0
                                    // schedule

   tone1_en = 1;                    // enable low group tones
   tone2_en = 1;                    // enable high group tones

   phase_add1 = LOW697;
   phase_add2 = HI1633;

   EA = 1;                          // Enable global interrupts

   while (1);                       // spin forever
}

//-----------------------------------------------------------------------------
// Init Routines
//-----------------------------------------------------------------------------

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

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

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

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

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

}

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

//-----------------------------------------------------------------------------
// Timer4_Init
//-----------------------------------------------------------------------------
// This routine initializes Timer4 in auto-reload mode to generate interrupts
// at intervals specified in <counts>.
//
void Timer4_Init (int counts)
{
   T4CON = 0;                        // STOP timer; set to auto-reload mode
   CKCON |= 0x40;                    // T4M = '1'; Timer4 counts SYSCLKs
   RCAP4 = -counts;                  // set reload value
   T4 = RCAP4;
   EIE2 |= 0x04;                     // enable Timer4 interrupts
   T4CON |= 0x04;                    // start Timer4
}

//-----------------------------------------------------------------------------
// Interrupt Handlers
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Timer4_ISR
//-----------------------------------------------------------------------------
//
// 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 DAC0H during this ISR call is
// actually transferred to DAC0 at the next Timer4 overflow.
//
void Timer4_ISR (void) interrupt 16 using 3
{
   static unsigned phase_acc1 = 0;  // holds low-tone phase accumulator
   static unsigned phase_acc2 = 0;  // holds high-tone phase accumulator

   char temp1;                       // temp values for table results
   char temp2;
   char code *table_ptr;
```

```
   T4CON &= ~0x80;                    // clear T4 overflow flag

   table_ptr = SINE_TABLE;

   if ((tone1_en) && (tone2_en)) {
      phase_acc1 += phase_add1;       // update phase acc1 (low tone)
      temp1 = *(table_ptr + (phase_acc1 >> 8));

      phase_acc2 += phase_add2;       // update phase acc2 (high tone)
      // read the table value
      temp2 = *(table_ptr + (phase_acc2 >> 8));

      // now update the DAC value.  Note: the XOR with 0x80 translates
      // the bipolar table look-up into a unipolar quantity.
      DAC0H = 0x80 ^ ((temp1 >> 1) + (temp2 >> 1));
   } else if (tone1_en) {
      phase_acc1 += phase_add1;       // update phase acc1 (low tone)
      // read the table value
      temp1 = *(table_ptr + (phase_acc1 >> 8));

      // now update the DAC value.  Note: the XOR with 0x80 translates
      // the bipolar table look-up into a unipolar quantity.
      DAC0H = 0x80 ^ temp1;
   } else if (tone2_en) {
      phase_acc2 += phase_add2;       // update phase acc2 (high tone)
      // read the table value
      temp2 = *(table_ptr + (phase_acc2 >> 8));

      // now update the DAC value.  Note: the XOR with 0x80 translates
      // the bipolar table look-up into a unipolar quantity.
      DAC0H = 0x80 ^ temp2;
   }
}
```

“OSC_Cry1.c”

```
//-----------------------------------------------------------------------------
// OSC_CRY1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 25 AUG 01
//
// This program configures shows an example of how to configure the external
// oscillator to drive a 22.1184MHz crystal and to select this external
// oscillator as the system clock source.  Also enables the Missing Clock
// Detector reset function.  Assumes an 22.1184MHz crystal is attached
// between XTAL1 and XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz

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

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

void SYSCLK_Init (void);
```

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

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

void main (void) {

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

   SYSCLK_Init ();                        // initialize oscillator

   while (1);
}

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

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

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

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

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

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

“OSC_Int1.c”

```
//-----------------------------------------------------------------------------
// OSC_INT1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 25 AUG 01
//
// This program shows an example of how to configure the internal
// oscillator to its maximum frequency (~16MHz).  Also enables the Missing
// Clock Detector reset function.
//
// The system clock frequency is stored in a global constant SYSCLK.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations

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

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

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

#define SYSCLK       16000000          // SYSCLK frequency in Hz

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

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

void SYSCLK_Init (void);

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

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

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

void main (void) {

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

   SYSCLK_Init ();                      // initialize oscillator

   while (1);
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the internal oscillator to its maximum setting and
// selects the internal oscillator as the system clock source.  Also enables
// the missing clock detector reset function.
//
void SYSCLK_Init (void)
{
   OSCICN = 0x87;                       // configure internal oscillator to
                                        // highest frequency setting; select
                                        // internal oscillator as SYSCLK source
                                        // enable missing clock detector reset
}
```

"INT_OSC_Measure1"

```
//-----------------------------------------------------------------------------
// INT_OSC_Measure1.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 02 APR 02
//
// This program shows an example of how the external oscillator
// can be used to measure the internal oscillator frequency.
//
// In this example, the internal oscillator is set to its highest setting.
// The external oscillator is configured to its desired mode (external
// 22.1184MHz crystal).  The PCA counter is used as a generic 16-bit counter
// that uses EXTCLK / 8 as its time base.
//
// We configure Timer0 to count SYSCLKs, and count the number of INTCLKs
// in 1 second's worth of EXTCLK/8
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                  // SFR declarations
#include <stdio.h>
#include <math.h>

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

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

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

#define EXTCLK       22118400           // EXTCLK frequency in Hz
#define BAUDRATE     9600               // Baud rate of UART in bps

sbit LED = P1^6;                        // LED = 1 means ON
```

```
//-----------------------------------------------------------------------------
// Structures, Unions, Enumerations, and Type definitions
//-----------------------------------------------------------------------------

typedef union ULong {
   long Long;
   unsigned int UInt[2];
   unsigned char Char[4];
} ULong;

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void INTCLK_Measure (void);

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

long INTCLK;                        // holds INTCLK frequency in Hz
long SYSCLK;                        // holds SYSCLK frequency in Hz

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

void main (void) {

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

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

   EA = 1;                          // enable global interrupts

   INTCLK_Measure ();               // measure internal oscillator and
                                    // update INTCLK variable
   SYSCLK = INTCLK;

   UART0_Init ();                   // initialize UART0


   while (1) {
      INTCLK_Measure ();            // measure internal oscillator and
                                    // update INTCLK variable
      printf ("INTCLK = %ld Hz\n", INTCLK);
   }
}

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

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

```
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clocks
//
void SYSCLK_Init (void)
{
   int i;                              // delay counter

   OSCXCN = 0x67;                      // start external oscillator

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

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

   OSCICN = 0x07;                      // configure internal oscillator at max
                                       // frequency;
                                       // set INTCLK as SYSCLK source
}

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

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0  = 0x50;                      // SCON0: mode 1, 8-bit UART, enable RX
   TMOD   = 0x20;                      // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);     // set Timer1 reload value for baudrate
   TR1    = 1;                         // start Timer1
   CKCON |= 0x10;                      // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                      // SMOD00 = 1
   ES0    = 0;                         // disable UART0 interrupts
   TI0    = 1;                         // indicate ready for TX
}

//-----------------------------------------------------------------------------
// INTCLK_Measure
//-----------------------------------------------------------------------------
//
// This routine uses the external oscillator to measure the frequency of the
// internal oscillator.  Assumes that the external oscillator has been
// started and settled.  Also assumes that the internal oscillator operating
```

```
// at its highest frequency is selected as the system clock source.
//
// The measurement algorithm is as follows:
// 1. PCA set to '-1'; Timer0 set to 0x0000; PCA stopped; Timer0 stopped
// 2. PCA0 started (CR = 1)
// 3. On CF, Timer0 is started (TF0 = 1); CF is cleared, PCA0 remains running
// 4. When T0_high:Timer0 reach 0x00000000, stop PCA0
// 5. EXTCLK frequency = PCA0_high:PCA0
// Upon completion, the global variable EXTCLK contains the internal
// oscillator frequency in Hz.
//
// I believe the worst-case measurement error is around 20 system clocks.
// 20 / 16e6 = 1.3 ppm (far less than typical crystal ppm's of 20)
//
void INTCLK_Measure (void)
{
   unsigned int T0_high;                 // overflow bytes of Timer0
   unsigned int PCA0_high;               // overflow bytes of PCA0
   ULong temp;                           // byte-addressable Long

   // PCA0 counts up to EXTCLK/8; PCA0_high is the upper 2 bytes of this number
   temp.Long = -(EXTCLK >> 3);
   PCA0_high = temp.UInt[0];
   PCA0CN = 0x00;                        // Stop counter; clear all flags
   PCA0MD = 0x0b;                        // PCA counts in IDLE mode;
                                         // EXTCLK / 8 is time base;
                                         // overflow interrupt is enabled
   PCA0L = 0xFF;                         // set time base to '-1'
   PCA0H = 0xFF;

   // Timer0 counts from zero to INTCLK

   // init Timer0
   CKCON |= 0x08;                        // Timer0 counts SYSCLKs
   TCON &= ~0x30;                        // Stop timer; clear TF0
   TMOD &= ~0x0f;                        // Timer0 in 16-bit counter mode
   TMOD |=  0x01;
   T0_high = 0x0000;                     // init Timer0
   TL0 = 0x00;
   TH0 = 0x00;

   // start PCA0
   CR = 1;
   while (CF == 0);                      // wait for edge
   TR0 = 1;                              // Start Timer0
   CF = 0;                               // clear PCA overflow
   PCA0L = temp.Char[3];
   PCA0H = temp.Char[2];

   while (1) {                           // wait for 1 second
      if (CF) {                          // handle PCA0 overflow
         PCA0_high++;
         if (PCA0_high == 0x0000) {      // check for completion
            TR0 = 0;                     // Stop Timer0
            break;
         }
         CF = 0;
      }
      if (TF0) {                         // handle T0 overflow
```

```
        T0_high++;
        TF0 = 0;
      }
   }
   CR = 0;                                // Stop PCA0

   // read 32-bit Timer0 value

   temp.UInt[0] = T0_high;
   temp.Char[2] = TH0;
   temp.Char[3] = TL0;

   INTCLK = temp.Long;                    // INTCLK = Timer0
}
```

“EXT_OSC_Measure1”

```
//-----------------------------------------------------------------------------
// EXT_OSC_Measure1.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 01 MAR 02
//
// This program shows an example of how the internal oscillator
// can be used to measure the external oscillator frequency.
//
// In this example, the internal oscillator is set to its highest setting.
// The external oscillator is configured to its desired mode.  The PCA counter
// is used as a generic 16-bit counter that uses EXTCLK / 8 as its time base.
//
// We configured Timer0 to count SYSCLKs, and count the number of EXTCLK/8
// ticks in 16 million SYSCLKs(or the number of SYSCLKs in 1 second) to obtain
// the external oscillator frequency.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>
#include <math.h>

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

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

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

#define SYSCLK       13750000          // SYSCLK frequency in Hz
#define BAUDRATE     9600              // Baud rate of UART in bps

sbit LED = P1^6;                       // LED = 1 means ON
```

```
//-----------------------------------------------------------------------------
// Structures, Unions, Enumerations, and Type definitions
//-----------------------------------------------------------------------------

typedef union ULong {
   long Long;
   unsigned int UInt[2];
   unsigned char Char[4];
} ULong;

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void EXTCLK_Measure (void);


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

long EXTCLK;                           // holds EXTCLK frequency in Hz
unsigned char SECONDS;                 // seconds counter
unsigned char MINUTES;                 // minutes counter
unsigned char HOURS;                   // hours counter
unsigned int DAYS;                     // days counter

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

void main (void) {

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

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

   EA = 1;                             // enable global interrupts


   UART0_Init ();                      // initialize UART0


   while (1) {
      EXTCLK_Measure ();               // measure external oscillator and
                                       // update EXTCLK variable
      printf ("\nEXTCLK = %ld Hz\n\n", EXTCLK);
   }
}

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

```
//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clocks
//
void SYSCLK_Init (void)
{
   int i;                              // delay counter

   OSCXCN = 0x67;                      // start external oscillator

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

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

      if(i == 0){                      // A low frequency oscillator will
         OSCXCN = 0x62;                // not start up if XFCN is set too
      }                                // high. This statement lowers XFCN
      i--;                             // if the external oscillator does
                                       // not start up within a reasonable
                                       // amount of time.
   }
   OSCICN = 0x07;                      // select internal oscillator at max
                                       // frequency as the system clock source
}

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

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0  = 0x50;                      // SCON0: mode 1, 8-bit UART, enable RX
   TMOD   = 0x20;                      // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);     // set Timer1 reload value for baudrate
   TR1    = 1;                         // start Timer1
   CKCON |= 0x10;                      // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                      // SMOD00 = 1
   ES0    = 0;                         // disable UART0 interrupts
   TI0    = 1;                         // indicate ready for TX
}
```

```
//-----------------------------------------------------------------------------
// EXTCLK_Measure
//-----------------------------------------------------------------------------
//
// This routine uses the internal oscillator to measure the frequency of the
// external oscillator.  Assumes that the external oscillator has been
// started and settled.  Also assumes that the internal oscillator operating
// at its highest frequency is selected as the system clock source.
//
// The measurement algorithm is as follows:
// 1. PCA set to '-1'; Timer0 set to 0x0000; PCA stopped; Timer0 stopped
// 2. PCA0 started (CR = 1)
// 3. On CF, Timer0 is started (TF0 = 1); CF is cleared, PCA0 remains running
// 4. When T0_high:Timer0 reach 0x00000000, stop PCA0
// 5. EXTCLK frequency = PCA0_high:PCA0
// Upon completion, the global variable EXTCLK contains the internal
// oscillator frequency in Hz.
//
// I believe the worst-case measurement error is around 20 system clocks.
// 20 / 16e6 = 1.3 ppm (far less than typical crystal ppm's of 20)
//
void EXTCLK_Measure (void)
{
   unsigned int T0_high;                 // overflow bytes of Timer0
   unsigned int PCA0_high;               // overflow bytes of PCA0
   ULong temp;                           // byte-addressable Long

   // Timer0 counts up to SYSCLK; T0_high is the upper 2 bytes of this number
   temp.Long = -SYSCLK;
   T0_high = temp.UInt[0];
   TH0 = temp.Char[2];
   TL0 = temp.Char[3];

   // PCA0 counts from zero to EXTCLK / 8
   PCA0CN = 0x00;                        // Stop counter; clear all flags
   PCA0MD = 0x0b;                        // PCA counts in IDLE mode;
                                         // EXTCLK / 8 is time base;
                                         // overflow interrupt is enabled
   PCA0L = 0xFF;                         // set time base to '-1'
   PCA0H = 0xFF;
   PCA0_high = 0x0000;

   // init Timer0
   CKCON |= 0x08;                        // Timer0 counts SYSCLKs
   TCON &= ~0x30;                        // Stop timer; clear TF0
   TMOD &= ~0x0f;                        // Timer0 in 16-bit counter mode
   TMOD |=  0x01;

   // start PCA0
   CR = 1;
   while (CF == 0);                      // wait for edge
   TR0 = 1;                              // Start Timer0
   CF = 0;                               // clear PCA overflow

   while (1) {                           // wait for 1 second
      if (TF0) {                         // handle T0 overflow
         T0_high++;
```

```
         if (T0_high == 0x0000) {
            CR = 0;                        // stop PCA
            break;                         // exit loop
         }
         TF0 = 0;
      }
      if (CF) {                            // handle PCA0 overflow
         PCA0_high++;
         CF = 0;
      }
   }
   TR0 = 0;                                // Stop Timer0

   // read PCA0 value

   //SYSCLK = (T0_high << 16) | (TH0 << 8) | TL0; // 0xa0 clock cycles
   // = 0x1a clock cycles using the optimization below
   temp.UInt[0] = PCA0_high;
   temp.Char[2] = PCA0H;
   temp.Char[3] = PCA0L;

   EXTCLK = temp.Long << 3;                // EXTCLK = 8 * EXTCLK / 8
}
```

“OSC_RTC_Cal1”

```
//-----------------------------------------------------------------------------
// OSC_RTC_Cal1.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 01 MAR 02
//
// This program shows an example of how an external crystal oscillator
// can be used to measure the internal oscillator frequency to a sufficient
// degree to enable UART operation (better than +/- 2.5%).
//
// In this example, a 32.768kHz watch crystal (with associated loading
// capacitors) is connected between XTAL1 and XTAL2.  The PCA counter is
// used as a generic 16-bit counter that uses EXTCLK / 8 as its time base.
// We preload it to generate an overflow in 8 counts.  Timer0 is configured
// as a 16-bit counter that is set to count SYSCLKs.  The internal oscillator
// is configured to its highest setting, and provides the system clock source.
//
// The number of 16 MHz clock cycles that occur in 4096 cycles of
// EXTCLK / 8 when EXTCLK = 32.768kHz is  16,000,000.  It is calculated as
// follows: 16MHz / 32.768kHz * 8 * 8 * 512 = 31,250 * 512 = 16,000,000.
//
// The measurement algorithm used in SYSCLK_Measure () counts the total number
// of system clocks in 4096 periods of EXTCLK / 8 (1 full second) and maintains
// overflow counters using Timer0. The result is given in MHz.
//
// The system clock frequency is then stored in a global variable SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.
//
// A set of RTC values for seconds, minutes, hours, and days are also
// maintained by the interrupt handler for Timer 3, which is configured
// to use EXTCLK / 8 as its time base and to reload every 4096 counts.
// This generates an interrupt once every second.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>
#include <math.h>

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

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR3RL   = 0x92;                 // Timer3 reload value
sfr16 TMR3     = 0x94;                 // Timer3 counter
sfr16 ADC0     = 0xbe;                 // ADC0 data
sfr16 ADC0GT   = 0xc4;                 // ADC0 greater than window
sfr16 ADC0LT   = 0xc6;                 // ADC0 less than window
```

```
sfr16 RCAP2    = 0xca;                  // Timer2 capture/reload
sfr16 T2       = 0xcc;                  // Timer2
sfr16 RCAP4    = 0xe4;                  // Timer4 capture/reload
sfr16 T4       = 0xf4;                  // Timer4
sfr16 DAC0     = 0xd2;                  // DAC0 data
sfr16 DAC1     = 0xd5;                  // DAC1 data

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

//#define SYSCLK       22118400           // SYSCLK frequency in Hz
#define BAUDRATE     19200              // Baud rate of UART in bps

sbit LED = P1^6;                        // LED = 1 means ON

//-----------------------------------------------------------------------------
// Structures, Unions, Enumerations, and Type definitions
//-----------------------------------------------------------------------------

typedef union ULong {
   long Long;
   unsigned int UInt[2];
   unsigned char Char[4];
} ULong;

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void Timer3_Init (void);
void SYSCLK_Measure (void);

void Timer3_ISR (void);

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

long SYSCLK;                            // holds SYSCLK frequency in Hz
unsigned char SECONDS;                  // seconds counter
unsigned char MINUTES;                  // minutes counter
unsigned char HOURS;                    // hours counter
unsigned int DAYS;                      // days counter

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

void main (void) {

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

   SYSCLK_Init ();                      // initialize oscillator
   PORT_Init ();                        // initialize crossbar and GPIO
```

```
   Timer3_Init ();                    // initialize Timer 3 for RTC mode

   EA = 1;                            // enable global interrupts

   SYSCLK_Measure ();                // measure internal oscillator and
                                      // update SYSCLK variable

   UART0_Init ();                     // initialize UART0

   printf (“\nSYSCLK = %ld Hz\n\n”, SYSCLK);

   while (1) {

      OSCICN = 0x07;                  // switch to fast internal osc

      LED = 1;                        // turn LED on

      printf (“%u Days, %02u:%02u:%02u\n”, DAYS, (unsigned) HOURS,
              (unsigned) MINUTES, (unsigned) SECONDS);

      while (TI0 == 0);               // wait for end of transmission

      OSCICN = 0x08;                  // switch to EXTOSC

      LED = 0;                        // turn LED off

      PCON |= 0x01;                   // go into IDLE mode (CPU will be
                                      // awakened by the next interrupt,
                                      // and will re-print the RTC
                                      // values).
   }
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use a low-frequency crystal
// as its clock source.
//
void SYSCLK_Init (void)
{
   int i;                             // delay counter
   int current, last;                 // used in osc. stabilization check
   int tolerance_count;

   OSCXCN = 0x60;                     // start external oscillator with
                                      // low-frequency crystal

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

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

   OSCICN = 0x07;                     // select internal oscillator at its
```

```
                                             // fastest setting as the system
                                             // clock source

   // low-frequency crystal stabilization wait routine
   // Here we measure the number of system clocks in 8 EXTCLK / 8 periods.
   // We compare successive measurements.  When we obtain 1000 measurements
   // in a row that are all within 20 system clocks of each other, the
   // routine will exit.  This condition will only occur once the crystal
   // oscillator has fully stabilized at its resonant frequency.
   //
   // Note that this can take several seconds.

   // init PCA0
   PCA0CN = 0x00;                        // Stop counter; clear all flags
   PCA0MD = 0x0b;                        // PCA counts in IDLE mode;
                                         // EXTCLK / 8 is time base;
                                         // overflow interrupt is enabled


   // init Timer0
   TCON &= ~0x30;                        // Stop timer; clear TF0
   TMOD &= ~0x0f;                        // Timer0 in 16-bit counter mode
   TMOD |=  0x01;
   CKCON |= 0x08;                        // Timer0 counts SYSCLKs

   tolerance_count = 1000;               // wait for 1000 cycles in a row
                                         // to lie within 20 clocks of each
                                         // other
   current = 0;
   do {
      PCA0CN = 0x00;
      PCA0L = 0xFF;                      // set PCA time base to '-1'
      PCA0H = 0xFF;
      TCON &= ~0x30;
      TH0 = 0x00;                        // init T0 time base
      TL0 = 0x00;

      // start PCA0
      CR = 1;
      while (CF == 0);                   // wait for edge
      TR0 = 1;                           // Start Timer0
      CF = 0;                            // clear PCA overflow
      PCA0L = -8;                        // set PCA to overflow in 8 cycles
      PCA0H = (-8) >> 8;
      while (CF == 0);
      TR0 = 0;
      last = current;
      current = (TH0 << 8) | TL0;
      if (abs (current - last) > 20) {
         tolerance_count = 1000;         // falls outside bounds; reset
                                         // counter
      } else {
         tolerance_count--;              // in-bounds; update counter
      }
   } while (tolerance_count != 0);
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
```

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

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0  = 0x50;                      // SCON0: mode 1, 8-bit UART, enable RX
   TMOD   = 0x20;                      // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);     // set Timer1 reload value for baudrate
   TR1    = 1;                         // start Timer1
   CKCON |= 0x10;                      // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                      // SMOD00 = 1
   ES0    = 0;                         // disable UART0 interrupts
   TI0    = 1;                         // indicate ready for TX
}

//-----------------------------------------------------------------------------
// Timer3_Init
//-----------------------------------------------------------------------------
//
// Configure Timer3 for 16-bit auto-reload mode using EXTCLK / 8 as its time
// base and to reload every 4096 counts.  This will generate one interrupt
// every second.
//
void Timer3_Init (void)
{
   TMR3CN = 0x01;                      // Stop Timer3; Clear TF3; Timer3
                                       // counts SYSCLK / 8
   TMR3RL = -4096;                     // reload every 4096 counts
   TMR3 = TMR3RL;                      // init T3
   EIE2 |= 0x01;                       // Enable Timer3 interrupts
   TMR3CN |= 0x04;                     // Start Timer3
}

//-----------------------------------------------------------------------------
// SYSCLK_Measure
//-----------------------------------------------------------------------------
//
// This routine uses the external oscillator to measure the frequency of the
// internal oscillator.  Assumes that the external oscillator has been
// started and settled.  Also assumes that the internal oscillator operating
// at its highest frequency is selected as the system clock source.
//
// The measurement algorithm is as follows:
// 1. PCA set to '-1'; Timer0 set to 0x0000; PCA stopped; Timer0 stopped
// 2. PCA started (CR = 1)
```

```
// 3. On CF, Timer0 is started (TR0 = 1); CF is cleared, PCA remains running
// 4. PCA set to '-4096' (note, we have about 4000 system clocks to
//    perform this operation before actually missing a count)
// 5. On CF, Timer0 is stopped (TR0 = 0);
// 6. Timer0 contains the number of 16 MHz SYSCLKs in 4096 periods of EXTCLK/8
//
// Upon completion, the global variable SYSCLK contains the internal
// oscillator frequency in Hz.
//
// I believe the worst-case measurement error is around 20 system clocks.
// 20 / 16e6 = 1.3 ppm (far less than typical crystal ppm's of 20)
//
void SYSCLK_Measure (void)
{
   unsigned int T0_high;                  // keeps track of Timer0 overflows
   ULong temp;                            // byte-addressable Long

   // init PCA0
   PCA0CN = 0x00;                         // Stop counter; clear all flags
   PCA0MD = 0x0b;                         // PCA counts in IDLE mode;
                                          // EXTCLK / 8 is time base;
                                          // overflow interrupt is enabled
   PCA0L = 0xFF;                          // set time base to '-1'
   PCA0H = 0xFF;

   // init Timer0
   T0_high = 0;                           // clear overflow counter
   CKCON |= 0x08;                         // Timer0 counts SYSCLKs
   TCON &= ~0x30;                         // Stop timer; clear TF0
   TMOD &= ~0x0f;                         // Timer0 in 16-bit counter mode
   TMOD |=  0x01;
   TH0 = 0x00;                            // init time base
   TL0 = 0x00;

   // start PCA0
   CR = 1;
   while (CF == 0);                       // wait for edge
   TR0 = 1;                               // Start Timer0
   CF = 0;                                // clear PCA overflow
   PCA0L = -4096;                         // set PCA to overflow in 4096
                                          // cycles (1 second)
   PCA0H = (-4096) >> 8;

   while (CF == 0) {                      // wait for 1 second
      if (TF0) {                          // handle T0 overflow
         T0_high++;
         TF0 = 0;
      }
   }
   TR0 = 0;                               // Stop Timer0

   // read Timer0 value

   //SYSCLK = (T0_high << 16) | (TH0 << 8) | TL0; // 0xa0 clock cycles
   // = 0x1a clock cycles using the optimization below
   temp.UInt[0] = T0_high;
   temp.Char[2] = TH0;
   temp.Char[3] = TL0;
```

```
   SYSCLK = temp.Long;
}

//-----------------------------------------------------------------------------
// Timer3_ISR
//-----------------------------------------------------------------------------
//
// This ISR is called on overflow of Timer3, which occurs once every second.
// Here we update a set of global RTC counters for seconds, minutes, hours,
// and days.
//
void Timer3_ISR (void) interrupt 14 using 3
{
   TMR3CN &= ~0x80;                       // clear Timer 3 overflow flag

   SECONDS++;
   if (SECONDS == 60) {
      SECONDS = 0;
      MINUTES++;
      if (MINUTES == 60) {
         MINUTES = 0;
         HOURS++;
         if (HOURS == 24) {
            HOURS = 0;
            DAYS++;
         }
      }
   }
}
```

“Timer0_Poll1.c”

```
//-----------------------------------------------------------------------------
// Timer0_Poll1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 27 AUG 01
//
// This program shows an example of using Timer0 in polled mode to implement
// a delay counter with a resolution of 1 ms.
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and  XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
```

```
void Timer0_Delay (int ms);

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

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

void main (void) {

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

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

   while (1) {
      Timer0_Delay (100);               // delay for 100ms
      LED = ~LED;                       // change state of LED
   }
}

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

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

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

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

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

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

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
```

```
   XBR0    = 0x00;
   XBR1    = 0x00;
   XBR2    = 0x40;                        // Enable crossbar and weak pull-ups
   P1MDOUT |= 0x40;                       // enable P1.6 (LED) as push-pull output
}

//-----------------------------------------------------------------------------
// Timer0_Delay
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <ms> milliseconds before returning.
//
void Timer0_Delay (int ms)
{
   int i;                                 // millisecond counter

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

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

“EMIF_1”

```
//-----------------------------------------------------------------------------
// EMIF_1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 11 DEC 01
//
// This program configures the external memory interface to read and write
// to an external SRAM mapped to the upper port pins.  Assumes an external
// 22.1184MHz crystal is attached between XTAL1 and XTAL2.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

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

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

#define RAM_BANK     0x20;             // bank select bit is P4^5
#define RAM_CS       0x10;             // chip select bit is P4^4

sbit LED = P1^6;                       // LED = 1 means ON

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

void SYSCLK_Init (void);
```

```
void PORT_Init (void);
void UART0_Init (void);
void EMIF_Init (void);

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

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

void main (void) {

   unsigned char xdata *pchar;         // memory access pointer
   unsigned long i;

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

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

   P4 &= ~RAM_BANK;                    // select lower bank
   P4 &= ~RAM_CS;                      // assert RAM chip select

   // clear xdata space
   pchar = 0;
   for (i = 0; i < 65536; i++) {
      *pchar++ = 0;
      // print status to UART0
      if ((i % 16) == 0) {
         printf ("\nwriting 0x%04x: %02x ", (unsigned) i, (unsigned) 0);
      } else {
         printf ("%02x ", (unsigned) 0);
      }
   }

   // verify all are zero
   pchar = 0;
   for (i = 0; i < 65536; i++) {
      if (*pchar != 0) {
         printf ("Erase error!\n");
         while (1);
      }
      // print status to UART0
      if ((i % 16) == 0) {
         printf ("\nverifying 0x%04x: %02x ", (unsigned) i, (unsigned) *pchar);
      } else {
         printf ("%02x ", (unsigned) *pchar);
      }
      pchar++;
   }

   // write xdata space
   pchar = 0;
   for (i = 0; i < 65536; i++) {
```

```
      *pchar = ~i;
      // print status to UART0
      if ((i % 16) == 0) {
         printf ("\nwriting 0x%04x: %02x ", (unsigned) i, (unsigned) ((~i) & 0xff));
      } else {
         printf ("%02x ", (unsigned) ((~i) & 0xff));
      }
      pchar++;
   }

   // verify
   pchar = 0;
   for (i = 0; i < 65536; i++) {
      if (*pchar != ((~i) & 0xff)) {
         printf ("Verify error!\n");
         while (1);
      }
      // print status to UART0
      if ((i % 16) == 0) {
         printf ("\nverifying 0x%04x: %02x ", (unsigned) i, (unsigned) *pchar);
      } else {
         printf ("%02x ", (unsigned) *pchar);
      }
      pchar++;
   }

   while (1);
}

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

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

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


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

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

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

//-----------------------------------------------------------------------------
// PORT_Init
```

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

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

//-----------------------------------------------------------------------------
// EMIF_Init
//-----------------------------------------------------------------------------
//
// Configure the external memory interface to use upper port pins in
// non-multiplexed mode to a mixed on-chip/off-chip configuration without
// Bank Select.
//
void EMIF_Init (void)
{
   EMI0CF = 0x3c;                      // upper ports; non-muxed mode;
                                       // split mode w/o bank select
   EMI0TC = 0x00;                      // fastest timing (4-cycle MOVX)
   P74OUT |= 0xfe;                     // all EMIF pins configured as
                                       // push-pull
}
```

“UART0_Stdio1”

```
//-----------------------------------------------------------------------------
// UART0_Stdio1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 18 AUG 01
//
// This program configures UART0 to operate in polled mode, suitable for use
// with the <stdio> functions printf() and scanf(), to which examples are
// provided.  Assumes an 22.1184MHz crystal is attached between XTAL1 and
// XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

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

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

sbit LED = P1^6;                       // LED = 1 means ON

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

```
void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);

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

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

void main (void) {
   char input_char;

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

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

   // transmit example

   printf (“Howdy!\n”);

   // receive example: a ‘1’ turns LED on; a ‘0’ turns LED off.

   while (1) {
      input_char = getchar();
      printf (“ ‘%c’, 0x%02x\n”, (unsigned char) input_char, (unsigned) input_char);
      switch (input_char) {
         case ‘0’:
            LED = 0;
            break;
         case ‘1’:
            LED = 1;
            break;
         default:
            break;
      }
   }
}

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

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

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

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

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

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

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

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

"UART0_Autobaud1"

```
//-----------------------------------------------------------------------------
// UART0_Autobaud1.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 30 APR 02
//
// This program shows an example of how the PCA can be used to enable accurate
// UART auto-baud detection when running from the on-chip internal oscillator.
// This algorithm assumes a 0x55 character ( ASCII "U") is sent from the
// remote transmitter.  Baud rates between 4800 to 19.2kbps can be reliably
// synchronized.
//
// UART0 is then configured to operate in polled mode, suitable for use
// with the <stdio> functions printf() and scanf().
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

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

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

//-----------------------------------------------------------------------------
// Structures, Unions, Enumerations, and Type definitions
//-----------------------------------------------------------------------------

typedef union UInt {
   unsigned int Int;
   unsigned char UChar[2];
} UInt;

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

```
sbit LED = P1^6;                          // LED = 1 means ON

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);

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

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

void main (void) {

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

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

   // transmit example

   printf ("Howdy!\n");

   while (1);

}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the internal oscillator
// operating at its maximum frequency.
//
void SYSCLK_Init (void)
{
   OSCICN = 0x07;                         // internal osc max frequency
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
```

```
{
   XBR0    |= 0x04;                     // Enable UART0
   XBR2    |= 0x40;                     // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x01;                     // enable TX0 as a push-pull output
   P1MDOUT |= 0x40;                     // enable LED as push-pull output
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for auto-baud detect and 8-N-1.
//
// In this example, the CEX1 is directed to appear on P0.1 (which is where the
// UART0 RX pin will appear on 'F02x devices) and CEX0 is directed to appear
// on P0.0 (which is where the UART RX pin would appear).  CEX0 is not used.
//
// The detection algorithm assumes that the host transmitter will send a 0x55
// character, which is an ASCII captial "U".
//
//  ----+     +-----+     +-----+     +-----+     +-----+     +---------
//      |  S  | LSB |  1  |  2  |  3  |  4  |  5  |  6  | MSB |  P
//      +-----+     +-----+     +-----+     +-----+     +-----+
//
// The PCA time base is configured to count SYSCLKs.  SYSCLK is assumed to
// be operating from the internal oscillator at its max frequency, though this
// should not affect the operation of the algorithm (though a lower frequency
// will decrease the maximum UART baud rate that can be matched).
//
// A bit period at 9600bps is ~104us, or about 63ns.  Therefore there are about
// 1670 SYSCLKs in 1 bit period.
//
// The algorithm operates as follows:
//  1. The time at which the falling edge is recorded in <last_time>.
//  2. The times between successive rising and falling edges are recorded
//     in edge_array.
//  3. The average bit time is calculated from the information in edge_array.
//  4. Timer1 is configured in 8-bit auto-reload mode to reload 16 times faster
//     than the average bit time.
//  5. The PCA modules are disabled in the Crossbar and UART0 is enabled.
//
void UART0_Init (void)
{
   unsigned int edge_array[9];          // holds bit times of received
                                        // training character
   unsigned int last_time;              // PCA0 value when last edge occurred
   UInt temp;                           // byte-addressable unsigned int

   unsigned char i;                     // edge counter

   // Route CEX0 and CEX1 to P0.0 and P0.1 on Crossbar
   XBR2 = 0x00;                         // Disable Crossbar
   XBR0 = 0x10;                         // Enable CEX0 and CEX1 (P0.0, P0.1)
   XBR2 = 0x40;                         // Enable Crossbar and weak pull-ups

   // Configure PCA for edge-capture operation
   PCA0MD = 0x08;                       // Configure PCA time base to use SYSCLK
   PCA0CPM0 = 0x00;                     // Module 0 is not used
   PCA0CPM1 = 0x30;                     // Module 1 configured for edge capture
```

```
                                    // (no interrupt generated)
   PCA0CN = 0x40;                   // Start PCA counter; clear all flags

   while (!CCF1);                   // wait for edge
   CCF1 = 0;                        // clear edge flag

   temp.UChar[0] = PCA0CPH1;        // read edge time
   temp.UChar[1] = PCA0CPL1;
   last_time = temp.Int;

   for (i = 0; i < 8; i++) {

      while (!CCF1);                // wait for edge
      CCF1 = 0;                     // clear edge flag
      // read the edge
      temp.UChar[0] = PCA0CPH1;     // read edge time
      temp.UChar[1] = PCA0CPL1;

      // store the edge
      edge_array[i] = temp.Int - last_time;
      last_time = temp.Int;         // update last edge timer
   }

   // add 8 bit times in prep for averaging
   last_time = 0x0000;              // initialize holding variable
   for (i = 0; i < 8; i++) {
      last_time += edge_array[i];
   }

   last_time = last_time >> (3 + 4);   // divide by 8 for averaging
                                    // and by 16 for Timer reload rate

   // Disable CEX0 and CEX1 through Crossbar and enable UART0
   XBR2 = 0x00;                     // Disable Crossbar
   XBR0 = 0x04;                     // Enable UART0 in Crossbar
   XBR2 = 0x40;                     // Enable Crossbar and weak pull-ups

   // Configure UART0 and Timer1
   SCON0  = 0x50;                   // SCON0: mode 1, 8-bit UART, enable RX
   TMOD   = 0x20;                   // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -last_time;             // set Timer1 reload value for baudrate
   TR1    = 1;                      // start Timer1
   CKCON |= 0x10;                   // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                   // SMOD00 = 1
   TI0    = 1;                      // Indicate TX0 ready

   printf (“\nAuto-baud detection statistics:\n”);
   printf (“Edge Array:\n”);
   for (i = 0; i < 8; i++) {
      printf (“ bit width %d: %u SYSCLKs\n”, (int) i, edge_array[i]);
   }
   printf (“Timer 1 reload value: 0x%02x\n”, (unsigned) (TH1 & 0xFF));
}
```

“UART0_Int1”

```
//-----------------------------------------------------------------------------
// UART0_Int1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 28 AUG 01
//
// This program configures UART0 to operate in interrupt mode, showing an
// example of a string transmitter and a string receiver.  These strings are
// assumed to be NULL-terminated.
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>
#include <string.h>

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     115200            // Baud rate of UART in bps
#define RX_LENGTH    16                // length of UART RX buffer

sbit LED = P1^6;                       // LED = 1 means ON

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void UART0_ISR (void);

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

bit TX_Ready;                          // '1' means okay to TX
char *TX_ptr;                          // pointer to string to transmit

bit RX_Ready;                          // '1' means RX string received
char idata RX_Buf[RX_LENGTH];          // receive string storage buffer

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {
   int i;                              // loop counter
   char tx_buf[8];                     // transmit buffer

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

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

   EA = 1;                             // enable global interrupts

   // transmit example: here we print the numbers 0 through 999, one
   // per line to UART0.

   while (!TX_Ready);                  // wait for TX ready
   TX_Ready = 0;                       // claim transmitter
   TX_ptr = "Howdy!";                  // set buffer pointer
   TI0 = 1;                            // start transmit

   for (i = 0; i < 1000; i++) {
      while (!TX_Ready);               // wait for TX ready
      TX_Ready = 0;                    // claim transmitter
      sprintf (tx_buf, "%d\r\n", i);   // build a string
      TX_ptr = tx_buf;                 // set buffer pointer
      TI0 = 1;                         // start transmit
   }

   // receive example: here we input a line using the receive function
   // and print the line using the transmit function.

   while (1) {                         // echo messages
      while (RX_Ready == 0) ;          // wait for string
      while (!TX_Ready) ;              // wait for transmitter to be available
      TX_Ready = 0;                    // claim transmitter
      TX_ptr = RX_Buf;                 // set TX buffer pointer to point to
                                       // received message
```

```
      TI0 = 1;                          // start transmit
      while (!TX_Ready) ;               // wait for transmission to complete
      TX_Ready = 0;
      TX_ptr = "\r\n";                  // send CR+LF
      TI0 = 1;                          // start transmit
      while (!TX_Ready) ;               // wait for transmission to complete
      RX_Ready = 0;                     // free the receiver
   }
}

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

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

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

   for (i=0; i < 256; i++) ;            // wait for XTLVLD to stabilize

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

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

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

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
```

```
   SCON0  = 0x50;                     // SCON0: mode 1, 8-bit UART, enable RX
   TMOD   = 0x20;                     // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);    // set Timer1 reload value for baudrate
   TR1    = 1;                        // start Timer1
   CKCON |= 0x10;                     // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                     // SMOD00 = 1
   ES0    = 1;                        // enable UART0 interrupts

   TX_Ready = 1;                      // indicate TX ready for transmit
   RX_Ready = 0;                      // indicate RX string not ready
   TX_ptr = NULL;
}

//-----------------------------------------------------------------------------
// Interrupt Handlers
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// UART0_ISR
//-----------------------------------------------------------------------------
//
// Interrupt Service Routine for UART0:
// Transmit function is implemented as a NULL-terminated string transmitter
// that uses the global variable <TX_ptr> and the global semaphore <TX_Ready>.
// Example usage:
//    while (TX_Ready == 0);          // wait for transmitter to be available
//    TX_Ready = 0;                   // claim transmitter
//    TX_ptr = <pointer to string to transmit>;
//    TI0 = 1;                        // initiate transmit
//
// Receive function is implemented as a CR-terminated string receiver
// that uses the global buffer <RX_Buf> and global indicator <RX_Ready>.
// Once the message is received, <RX_Ready> is set to '1'.  Characters
// received while <RX_Ready> is '1' are ignored.
//
void UART0_ISR (void) interrupt 4 using 3
{
   static unsigned char RX_index = 0;  // receive buffer index
   unsigned char the_char;

   if (RI0 == 1) {                    // handle receive function
      RI0 = 0;                        // clear RX complete indicator
      if (RX_Ready != 1) {            // check to see if message pending
         the_char = SBUF0;
         if (the_char != '\r') {      // check for end of message
            // store the character
            RX_Buf[RX_index] = the_char;
            // increment buffer pointer and wrap if necessary
            if (RX_index < (RX_LENGTH - 2)) {
               RX_index++;
            } else {
               RX_index = 0;          // if length exceeded,
               RX_Ready = 1;          // post message complete, and
                                      // NULL-terminate string
               RX_Buf[RX_index-1] = '\0';
            }
         } else {
            RX_Buf[RX_index] = '\0';  // NULL-terminate message
            RX_Ready = 1;             // post message ready
```

```
            RX_index = 0;               // reset RX message index
         }
      } else {
         // ignore character -- previous message has not been processed
      }
   } else if (TI0 == 1) {               // handle transmit function
      TI0 = 0;                          // clear TX complete indicator
      the_char = *TX_ptr;               // read next character in string
      if (the_char != '\0') {
         SBUF0 = the_char;              // transmit it
         TX_ptr++;                      // get ready for next character
      } else {                          // character is NULL
         TX_Ready = 1;                  // indicate ready for next TX
      }
   }
}
```

"UART1_Int1"

```
//-----------------------------------------------------------------------------
// UART1_Int1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 30 OCT 01
//
// This program configures UART1 to operate in interrupt mode, showing an
// example of a string transmitter and a string receiver.  These strings are
// assumed to be NULL-terminated.
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and XTAL2.
//
// The system clock frequency is stored in a global constant SYSCLK.  The
// target UART baud rate is stored in a global constant BAUDRATE.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>
#include <string.h>

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define BAUDRATE     115200            // Baud rate of UART in bps
#define RX_LENGTH    16                // length of UART RX buffer

sbit LED = P1^6;                       // LED = 1 means ON

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART1_Init (void);
void UART1_ISR (void);

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

bit TX_Ready;                          // '1' means okay to TX
char *TX_ptr;                          // pointer to string to transmit

bit RX_Ready;                          // '1' means RX string received
char idata RX_Buf[RX_LENGTH];          // receive string storage buffer

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {
   int i;                              // loop counter
   char tx_buf[8];                     // transmit buffer

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

   SYSCLK_Init ();                     // initialize oscillator
   PORT_Init ();                       // initialize crossbar and GPIO
   UART1_Init ();                      // initialize UART0

   EA = 1;                             // enable global interrupts

   // transmit example: here we print the numbers 0 through 999, one
   // per line to UART0.

   while (!TX_Ready);                  // wait for TX ready
   TX_Ready = 0;                       // claim transmitter
   TX_ptr = "Howdy!";                  // set buffer pointer
   SCON1 |= 0x02;                      // TI1 = 1; start transmit

   for (i = 0; i < 1000; i++) {
      while (!TX_Ready);               // wait for TX ready
      TX_Ready = 0;                    // claim transmitter
      sprintf (tx_buf, "%d\r\n", i);   // build a string
      TX_ptr = tx_buf;                 // set buffer pointer
      SCON1 |= 0x02;                   // TI1 = 1; start transmit
   }

   // receive example: here we input a line using the receive function
   // and print the line using the transmit function.

   while (1) {                         // echo messages
      while (RX_Ready == 0) ;          // wait for string
      while (!TX_Ready) ;              // wait for transmitter to be available
      TX_Ready = 0;                    // claim transmitter
      TX_ptr = RX_Buf;                 // set TX buffer pointer to point to
                                       // received message
```

```
      SCON1 |= 0x02;                      // TI1 = 1; start transmit
      while (!TX_Ready) ;                 // wait for transmission to complete
      TX_Ready = 0;
      TX_ptr = "\r\n";                    // send CR+LF
      SCON1 |= 0x02;                      // TI1 = 1; start transmit
      while (!TX_Ready) ;                 // wait for transmission to complete
      RX_Ready = 0;                       // free the receiver
   }
}

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

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

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

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

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

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

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR2    |= 0x44;                       // Enable UART1, crossbar and weak
                                          // pull-ups
   P0MDOUT |= 0x01;                       // enable TX1 as a push-pull output
   P1MDOUT |= 0x40;                       // enable LED as push-pull output
}

//-----------------------------------------------------------------------------
// UART1_Init
//-----------------------------------------------------------------------------
//
// Configure UART1 using Timer1, for <baudrate> and 8-N-1.
//
void UART1_Init (void)
{
```

```
   SCON1  = 0x50;                     // SCON1: mode 1, 8-bit UART, enable RX
   TMOD   = 0x20;                     // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);    // set Timer1 reload value for baudrate
   TR1    = 1;                        // start Timer1
   CKCON |= 0x10;                     // Timer1 uses SYSCLK as time base
   PCON  |= 0x10;                     // SMOD1 = 1
   EIE2  |= 0x40;                     // enable UART1 interrupts

   TX_Ready = 1;                      // indicate TX ready for transmit
   RX_Ready = 0;                      // indicate RX string not ready
   TX_ptr = NULL;
}

/*
//-----------------------------------------------------------------------------
// UART1_Init
//-----------------------------------------------------------------------------
//
// Configure UART1 using Timer4, for <baudrate> and 8-N-1.
//
void UART1_Init (void)
{
   SCON1  = 0x50;                     // SCON1: mode 1, 8-bit UART, enable RX
   T4CON  = 0x30;                     // Stop Timer; clear int flags; enable
                                      // UART baudrate mode; enable 16-bit
                                      // auto-reload timer function; disable
                                      // external count and capture modes
   RCAP4  = -(SYSCLK/BAUDRATE/32);    // set Timer reload value for baudrate
   T4     = RCAP4;                    // initialize Timer value
   CKCON |= 0x40;                     // Timer4 uses SYSCLK as time base

   T4CON |= 0x04;                     // TR4 = 1; start Timer4
   PCON  |= 0x10;                     // SMOD1 = 1
   EIE2  |= 0x40;                     // enable UART1 interrupts

   TX_Ready = 1;                      // indicate TX ready for transmit
   RX_Ready = 0;                      // indicate RX string not ready
   TX_ptr = NULL;
}
*/

//-----------------------------------------------------------------------------
// Interrupt Handlers
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// UART1_ISR
//-----------------------------------------------------------------------------
//
// Interrupt Service Routine for UART1:
// Transmit function is implemented as a NULL-terminated string transmitter
// that uses the global variable <TX_ptr> and the global semaphore <TX_Ready>.
// Example usage:
//    while (TX_Ready == 0);          // wait for transmitter to be available
//    TX_Ready = 0;                   // claim transmitter
//    TX_ptr = <pointer to string to transmit>;
//    SCON1 |= 0x02;                  // TI1 = 1; initiate transmit
//
// Receive function is implemented as a CR-terminated string receiver
```

```
// that uses the global buffer <RX_Buf> and global indicator <RX_Ready>.
// Once the message is received, <RX_Ready> is set to '1'.  Characters
// received while <RX_Ready> is '1' are ignored.
//
void UART1_ISR (void) interrupt 20 using 3
{
   static unsigned char RX_index = 0;  // receive buffer index
   unsigned char the_char;

   if ((SCON1 & 0x01) == 0x01) {       // handle receive function
      SCON1 &= ~0x01;                  // RI1 = 0; clear RX complete
                                       // indicator
      if (RX_Ready != 1) {             // check to see if message pending
         the_char = SBUF1;
         if (the_char != '\r') {       // check for end of message
            // store the character
            RX_Buf[RX_index] = the_char;
            // increment buffer pointer and wrap if necessary
            if (RX_index < (RX_LENGTH - 2)) {
               RX_index++;
            } else {
               RX_index = 0;           // if length exceeded,
               RX_Ready = 1;           // post message complete, and
                                       // NULL-terminate string
               RX_Buf[RX_index-1] = '\0';
            }
         } else {
            RX_Buf[RX_index] = '\0';   // NULL-terminate message
            RX_Ready = 1;              // post message ready
            RX_index = 0;              // reset RX message index
         }
      } else {
         ; // ignore character -- previous message has not been processed
      }
   } else if ((SCON1 & 0x02) == 0x02) {// handle transmit function
      SCON1 &= ~0x02;                  // TI1 = 0; clear TX complete
                                       // indicator
      the_char = *TX_ptr;              // read next character in string
      if (the_char != '\0') {
         SBUF1 = the_char;             // transmit it
         TX_ptr++;                     // get ready for next character
      } else {                         // character is NULL
         TX_Ready = 1;                 // indicate ready for next TX
      }
   }
}
```

“FLASH_Scratch”

```
//-----------------------------------------------------------------------------
// FLASH_Scratch1.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 24 JUN 02
//
// This program illustrates how to erase, write, and read FLASH memory from
// application code written in ‘C’.  This routine exercises the upper 128-
// byte FLASH sector.
//
// Note: debugging operations are not possible while SFLE = 1.
//
// Note: because this code contains routines which write to FLASH memory,
// the on-chip VDD monitor should be enabled by tying the MONEN pin high
// to VDD.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                  // SFR declarations
#include <stdio.h>

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

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

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

#define SYSCLK       22118400           // SYSCLK frequency in Hz

#define SCRATCH_ADDR 0x0000             // address of Scratchpad

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

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

void SYSCLK_Init (void);

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

char code my_array[] = “Monkeys”;

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

void main (void) {
   unsigned char xdata * data pwrite;  // FLASH write pointer
   unsigned char code * data pread;    // read pointer
   unsigned char test_array[16];
   unsigned char i;
   unsigned char temp;

   // Disable Watchdog timer
   WDTCN = 0xde;
   WDTCN = 0xad;

   SYSCLK_Init ();                     // initialize oscillator

   // erase Scratchpad
   FLSCL |= 0x01;                      // Enable FLASH writes/erases

   PSCTL |= 0x03;                      // set PSWE = PSEE = 1
   PSCTL |= 0x04;                      // set SFLE = 1 (enable
                                       // access to scratchpad)

   // initialize FLASH write pointer to SCRATCH ADDRESS
   pwrite = (unsigned char xdata *) SCRATCH_ADDR;

   *pwrite = 0x00;                     // erase SCRATCH PAGE

   PSCTL &= ~0x07;                     // set PSWE = PSEE = SLFE = 0

   // copy array into SCRATCH PAGE
   pread = my_array;                   // you can set a breakpoint at
                                       // this line to verify that the
                                       // Scratch Pad Memory has been
                                       // erased.

   PSCTL |= 0x01;                      // set PSWE = 1 so that MOVX
                                       // writes will target FLASH
                                       // memory
   for (i = 0; i < sizeof(my_array); i++) {
      PSCTL &= ~0x04;                  // clear SLFE to enable FLASH
                                       // reads from non-Scratch Pad memory
      temp = pread[i];                 // read the source character
      PSCTL |= 0x04;                   // set SFLE to enable FLASH writes
                                       // to Scratch Pad memory
      pwrite[i] = temp;                // write the byte
```

```
   }

   PSCTL &= ~0x07;                      // set PSWE = PSEE = SFLE = 0
   FLSCL &= ~0x01;                      // clear FLWE to disable FLASH
                                        // write/erases

   // copy first 16 bytes from Scratch Pad memory to a local RAM array
   PSCTL |= 0x04;                       // set SFLE = 1 to access Scratch
                                        // Pad memory
   pread = (unsigned char code *) SCRATCH_ADDR;

   for (i = 0; i < 16; i++) {
      test_array[i] = pread[i];
   }

   PSCTL &= ~0x04;                      // clear SFLE to disable access
                                        // to Scratch Pad memory

   while (1);                           // you can set a breakpoint at
                                        // this line to verify that the
                                        // string has been written to the
                                        // Scratch Pad memory
}

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

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

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

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

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

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

“Freq_Gen1”

```
//-----------------------------------------------------------------------------
// Freq_Gen1.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 09 APR 02
//
// This program uses the PCA in Frequency Output mode to generate a square
// wave on P0.0.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations

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

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

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

#define SYSCLK       22118400          // SYSCLK frequency in Hz
#define FREQ         172800            // Frequency to generate in Hz
#define BAUDRATE     9600              // Baud rate of UART in bps

sbit LED = P1^6;                       // LED = 1 means ON

//-----------------------------------------------------------------------------
// Structures, Unions, Enumerations, and Type definitions
//-----------------------------------------------------------------------------

typedef union ULong {
   long Long;
   unsigned int UInt[2];
   unsigned char Char[4];
} ULong;
```

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

void SYSCLK_Init (void);
void PORT_Init (void);
void PCA0_Init (void);

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

void main (void) {

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

   SYSCLK_Init ();                  // initialize oscillator
   PORT_Init ();                    // initialize crossbar and GPIO
   PCA0_Init ();                    // initialize PCA0
   while (1);
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clocks
//
void SYSCLK_Init (void)
{
   int i;                           // delay counter

   OSCXCN = 0x67;                   // start external oscillator

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

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

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

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR0    |= 0x08;                 // Enable CEX0 on P0.0
   XBR2    |= 0x40;                 // Enable crossbar and weak pull-ups
```

```
   P0MDOUT |= 0x01;                      // enable P0.0 as a push-pull output
}

//-----------------------------------------------------------------------------
// PCA0_Init
//-----------------------------------------------------------------------------
//
// Configure PCA0:  PCA uses SYSCLK as time base; overflow interrupt
// disabled.
// Module 0 configured in Frequency Output mode to generate a frequency equal
// to the constant FREQ.
//
void PCA0_Init (void)
{
   // configure PCA time base to use SYSCLK; overflow interrupt disabled
   PCA0CN = 0x00;                        // Stop counter; clear all flags
   PCA0MD = 0x08;                        // Time base uses SYSCLK

   // Configure Module 0 to Frequency Output mode to toggle at 2*FREQ
   PCA0CPM0 = 0x46;                      // Frequency Output mode
   PCA0CPH0 = SYSCLK/FREQ/2;             // Set frequency

   // Start PCA counter
   CR = 1;
}
```

“SPI_EE_Pol1”

```
//------------------------------------------------------------------------------
// SPI_EE_Poll1.c
//------------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 14 SEP 01
//
// This program shows an example of how to interface to a SPI EEPROM using
// the SPI0 interface in polled-mode. The SPI EEPROM used here is a Microchip
// 25LC320 (4k bytes).  The hardware connections are as follows:
//
// P0.0 - TX -- UART used for display/testing purposes
// P0.1 - RX
//
// P0.2 - SCK  (connected to SCK on EEPROM)
// P0.3 - MISO (connected to SI on EEPROM)
// P0.4 - MOSI (connected to SO on EEPROM)
// P0.5 - NSS  (unconnected, but pulled high by on-chip pull-up resistor)
//
// P1.7 - EE_CS (connected to /CS on EEPROM)
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and XTAL2.
//
// In this example, the attached SPI device is loaded with a test pattern.
// The EEPROM contents are then verified with the test pattern.  If the test
// pattern is verified with no errors, the LED blinks on operation complete.
// Otherwise, the LED stays off.  Progress can also be monitored by a terminal
// connected to UART0 operating at 115.2kbps.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                  // SFR declarations
#include <stdio.h>

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

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

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

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

sbit        LED = P1^6;               // LED='1' means ON
sbit        EE_CS = P1^7;             // EEPROM CS signal

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void SPI0_Init (void);
void Timer0_ms (unsigned ms);
void Timer0_us (unsigned us);

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


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

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

void main (void) {

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

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

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


   SPI0_Init ();                      // initialize SPI0

   // fill EEPROM with 0xFF's
   LED = 1;
```

```
for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++) {
   test_byte = 0xff;
   EE_Write (EE_Addr, test_byte);

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

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

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

// fill EEPROM memory with LSB of EEPROM address.
LED = 1;
for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++) {
   test_byte = EE_Addr & 0xff;
   EE_Write (EE_Addr, test_byte);

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

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

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

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

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

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

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

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

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

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

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR0   |= 0x06;                      // Enable SPI0 and UART0
   XBR1    = 0x00;
   XBR2    = 0x40;                      // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x15;                     // enable P0.0 (TX), P0.2 (SCK), and
                                        // P0.4 (MOSI) as push-pull outputs
   P1MDOUT |= 0xC0;                     // enable P1.6 (LED) and P1.7 (EE_CS)
                                        // as push-pull outputs
}

//-----------------------------------------------------------------------------
// SPI0_Init
//-----------------------------------------------------------------------------
//
// Configure SPI0 for 8-bit, 2MHz SCK, Master mode, polled operation, data
// sampled on 1st SCK rising edge.
//
void SPI0_Init (void)
```

```
{
   SPI0CFG = 0x07;                        // data sampled on 1st SCK rising edge
                                          // 8-bit data words

   SPI0CN = 0x03;                         // Master mode; SPI enabled; flags
                                          // cleared
   SPI0CKR = SYSCLK/2/2000000;            // SPI clock <= 2MHz (limited by
                                          // EEPROM spec.)
}

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


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

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

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

//-----------------------------------------------------------------------------
// Timer0_us
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <us> microseconds before returning.
```

```
//
void Timer0_us (unsigned us)
{
   unsigned i;                         // millisecond counter

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

   for (i = 0; i < us; i++) {          // count microseconds
      TR0 = 0;                         // STOP Timer0
      TH0 = (-SYSCLK/1000000) >> 8;    // set Timer0 to overflow in 1us
      TL0 = -SYSCLK/1000000;
      TR0 = 1;                         // START Timer0
      while (TF0 == 0);                // wait for overflow
      TF0 = 0;                         // clear overflow indicator
   }
}

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

   EE_CS = 0;                          // select EEPROM
   EE_CS = 0;                          // find out why Keil compiler is
                                       // optimizing one of these out

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

   // transmit READ opcode
   SPIF = 0;
   SPI0DAT = EE_READ;
   while (SPIF == 0);

   // transmit Address MSB-first
   SPIF = 0;                           // transmit MSB of address
   SPI0DAT = (Addr >> 8);
   while (SPIF == 0);

   SPIF = 0;                           // transmit LSB of address
   SPI0DAT = Addr;
   while (SPIF == 0);

   // initiate dummy transmit to read data
   SPIF = 0;
   SPI0DAT = 0;
   while (SPIF == 0);

   retval = SPI0DAT;                   // read data from SPI

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

```
   EE_CS = 1;                          // de-select EEPROM

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

   return retval;
}

//-----------------------------------------------------------------------------
// EE_Write
//-----------------------------------------------------------------------------
//
// This routine writes a single EEPROM byte <value> to address <Addr>.  Here
// we implement post-write polling, and return once the write operation has
// completed.  This prevents us from having to poll before an EEPROM Read
// or Write operation.
//
void EE_Write (unsigned Addr, unsigned char value)
{

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

   // transmit WREN (Write Enable) opcode
   SPIF = 0;
   SPI0DAT = EE_WREN;
   while (SPIF == 0);

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

   EE_CS = 1;                          // de-select EEPROM to set WREN latch

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

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

   // transmit WRITE opcode
   SPIF = 0;
   SPI0DAT = EE_WRITE;
   while (SPIF == 0);

   // transmit Address MSB-first
   SPIF = 0;                           // transmit MSB of address
   SPI0DAT = (Addr >> 8);
   while (SPIF == 0);

   SPIF = 0;                           // transmit LSB of address
   SPI0DAT = Addr;
   while (SPIF == 0);

   // transmit data
   SPIF = 0;
   SPI0DAT = value;
   while (SPIF == 0);

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

```
   EE_CS = 1;                             // deselect EEPROM (initiate EEPROM
                                          // write cycle)

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

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

      EE_CS = 0;                          // select EEPROM to begin polling

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

      SPIF = 0;
      SPI0DAT = EE_RDSR;                  // send Read Status register opcode
      while (SPIF == 0);

      SPIF = 0;
      SPI0DAT = 0;                        // dummy write to read status register
      while (SPIF == 0);

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

   } while (SPI0DAT & 0x01);              // poll until WIP (Write In
                                          // Progress) bit goes to '0'

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

“SPI_EE_Int1”

```
//-----------------------------------------------------------------------------
// SPI_EE_Int1.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// AUTH: BW
// DATE: 14 SEP 01
//
// This program shows an example of how to interface to an SPI EEPROM using
// the SPI0 interface in interrupt-mode. The SPI EEPROM used here is a
// Microchip 25LC320 (4k bytes).  The hardware connections are as follows:
//
// P0.0 - TX -- UART used for display/testing purposes
// P0.1 - RX
//
// P0.2 - SCK  (connected to SCK on EEPROM)
// P0.3 - MISO (connected to SI on EEPROM)
// P0.4 - MOSI (connected to SO on EEPROM)
// P0.5 - NSS  (unconnected, but pulled high by on-chip pull-up resistor)
//
// P1.7 - EE_CS (connected to /CS on EEPROM)
//
// Assumes an 22.1184MHz crystal is attached between XTAL1 and XTAL2.
//
// In this example, the attached SPI device is loaded with a test pattern.
// The EEPROM contents are then verified with the test pattern.  If the test
// pattern is verified with no errors, the LED blinks on operation complete.
// Otherwise, the LED stays off.  Progress can also be monitored by a terminal
// connected to UART0 operating at 115.2kbps.
//
// Target: C8051F02x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

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

#include <c8051f020.h>                 // SFR declarations
#include <stdio.h>

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F00x
//-----------------------------------------------------------------------------

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

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

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

#define     TRUE     1
#define     FALSE    0

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

sbit        LED = P1^6;                 // LED='1' means ON
sbit        EE_CS = P1^7;               // EEPROM CS signal

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

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

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void SPI0_Init (void);
void Timer0_Init (void);
void Timer0_ms (unsigned ms);

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

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

bit EE_Ready = FALSE;                   // semaphore for SPI0/EEPROM
bit EE_WR = FALSE;                      // TRUE = write; FALSE = read
unsigned EE_Addr = 0x0000;              // EEPROM address
unsigned char EE_Data = 0x00;           // EEPROM data

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

void main (void) {

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

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

   SYSCLK_Init ();                      // initialize oscillator
   PORT_Init ();                        // initialize crossbar and GPIO
   UART0_Init ();                       // initialize UART0
   Timer0_Init ();                      // initialize Timer0
```

```
SPI0_Init ();                       // initialize SPI0

EA = 1;                             // enable global interrupts

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

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

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

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

// fill EEPROM memory with LSB of EEPROM address.
LED = 1;
for (test_addr = 0; test_addr < EE_SIZE; test_addr++) {
   test_byte = test_addr & 0xff;
   EE_Write (test_addr, test_byte);

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

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

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

```
         printf ("%02x ", (unsigned) test_byte);
      }
      if (test_byte != (test_addr & 0xFF)) {
         printf ("Error at %u\n", test_addr);
         while (1);                       // stop here on error
      }
   }

   ET0 = 0;                               // disable Timer0 interrupts

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

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

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

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

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

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

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

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR0   |= 0x06;                        // Enable SPI0 and UART0
   XBR1    = 0x00;
   XBR2    = 0x40;                        // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x15;                       // enable P0.0 (TX), P0.2 (SCK), and
                                          // P0.4 (MOSI) as push-pull outputs
   P1MDOUT |= 0xC0;                       // enable P1.6 (LED) and P1.7 (EE_CS)
                                          // as push-pull outputs
}
```

```
//-----------------------------------------------------------------------------
// SPI0_Init
//-----------------------------------------------------------------------------
//
// Configure SPI0 for 8-bit, 2MHz SCK, Master mode, interrupt operation, data
// sampled on 1st SCK rising edge.  SPI0 interrupts are enabled here
//
void SPI0_Init (void)
{
   SPI0CFG = 0x07;                     // data sampled on 1st SCK rising edge
                                       // 8-bit data words

   SPI0CN = 0x03;                      // Master mode; SPI enabled; flags
                                       // cleared
   SPI0CKR = SYSCLK/2/2000000;         // SPI clock <= 2MHz (limited by
                                       // EEPROM spec.)
   EE_Ready = TRUE;                    // post SPI0/EEPROM available
   EIE1 |= 0x01;                       // enable SPI0 interrupts
}

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

//-----------------------------------------------------------------------------
// Timer0_Init
//-----------------------------------------------------------------------------
//
// Configure Timer0 for 16-bit interrupt mode.
//
void Timer0_Init (void)
{
   TCON  &= ~0x30;                     // STOP Timer0 and clear overflow flag
   TMOD  &= ~0x0f;                     // configure Timer0 to 16-bit mode
   TMOD  |=  0x01;
   CKCON |=  0x08;                     // Timer0 counts SYSCLKs
}

//-----------------------------------------------------------------------------
// Timer0_ms
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <ms> milliseconds before returning.
//
```

```
void Timer0_ms (unsigned ms)
{
   unsigned i;                          // millisecond counter

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

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

//-----------------------------------------------------------------------------
// EE_Read
//-----------------------------------------------------------------------------
//
// This routine reads and returns a single EEPROM byte whose address is
// given in <Addr>.
//
unsigned char EE_Read (unsigned Addr)
{
   while (EE_Ready == FALSE);           // wait for EEPROM available

   EE_Ready = FALSE;                    // claim EEPROM

   EE_Addr = Addr;                      // initialize EEPROM address
   EE_WR = FALSE;                       // set up for READ operation

   SPIF = 1;                            // initiate EEPROM operation

   while (EE_Ready == FALSE);           // wait for operation complete

   return EE_Data;                      // return data
}

//-----------------------------------------------------------------------------
// EE_Write
//-----------------------------------------------------------------------------
//
// This routine writes a single EEPROM byte <value> to address <Addr>.  Here
// we implement pre-write polling.
//
void EE_Write (unsigned Addr, unsigned char value)
{
   while (EE_Ready == FALSE);           // wait for EEPROM available

   EE_Ready = FALSE;                    // claim EEPROM

   EE_Addr = Addr;                      // initialize EEPROM address
   EE_Data = value;                     // initialize EEPROM data
   EE_WR = TRUE;                        // set up for WRITE operation
```

```
   SPIF = 1;                           // initiate EEPROM operation
}

//-----------------------------------------------------------------------------
// Timer0_ISR
//-----------------------------------------------------------------------------
//
// Timer0 implements a delay which is used by the SPI0_ISR to manage setup
// and hold requirements on the EE_CS line.  This ISR initiates a SPI0
// interrupt when called, and stops Timer0.
//
void Timer0_ISR (void) interrupt 1 using 3
{
   TR0 = 0;                            // STOP Timer0
   SPIF = 1;                           // initiate SPI0 interrupt
}

//-----------------------------------------------------------------------------
// SPI0_ISR
//-----------------------------------------------------------------------------
//
// This ISR implements a state machine which handles byte-level read and
// write operations to an attached EEPROM.
//
void SPI0_ISR (void) interrupt 6 using 3
{
   enum SPI0_state { RESET, RD_S0, RD_S1, RD_S2, RD_S3, RD_S4, RD_S5, RD_S6,
                     RD_S7, WR_S0, WR_S1, WR_S2, WR_S3, WR_S4, WR_S5, WR_S6,
                     WR_S7, WR_S8, WR_S9, WR_S10, WR_S11, WR_S12, WR_S13,
                     WR_S14, WR_S15};

   static enum SPI0_state state = RESET;

   SPIF = 0;                           // clear SPI interrupt flag

   switch (state) {
      case RESET: // assert EE_CS; set Timer0 to cause SPI0 interrupt in
                  // 250ns (CS setup time); decode EE_WR to determine
                  // whether next state is RD_S0 (read) or WR_S0 (write).
         EE_CS = 0;                    // assert CS signal on EEPROM

         // set Timer0 to interrupt 250ns from now
         ET0 = 0;                      // disable Timer0 interrupts
         TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
         TH0 = (-SYSCLK/4000000) >> 8; // set Timer0 to overflow in 250ns
         TL0 = -SYSCLK/4000000;
         ET0 = 1;                      // enable Timer0 interrupts
         TR0 = 1;                      // START Timer0

         // decode EE_Write flag to determine whether operation is a read
         // or a write
         if (EE_WR == TRUE) {
            state = WR_S0;             // set up for a write
         } else {
            state = RD_S0;             // set up for a read
         }
         break;

      case RD_S0: // transmit READ op-code
```

```
         SPI0DAT = EE_READ;            // transmit READ opcode
         state = RD_S1;                // advance to next state
         break;

      case RD_S1: // transmit MSB of Address
         SPI0DAT = EE_Addr >> 8;       // transmit MSB of Address
         state = RD_S2;                // advance to next state
         break;

      case RD_S2: // transmit LSB of Address
         SPI0DAT = EE_Addr;            // transmit LSB of Address
         state = RD_S3;                // advance to next state
         break;

      case RD_S3: // transmit dummy read to get data from EEPROM
         SPI0DAT = 0;                  // transmit dummy read
         state = RD_S4;                // advance to next state
         break;

      case RD_S4: // wait 250ns (EEPROM CS hold time)

         // set Timer0 to interrupt 250ns from now
         ET0 = 0;                      // disable Timer0 interrupts
         TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
         TH0 = (-SYSCLK/4000000) >> 8; // set Timer0 to overflow in 250ns
         TL0 = -SYSCLK/4000000;
         ET0 = 1;                      // enable Timer0 interrupts
         TR0 = 1;                      // START Timer0
         state = RD_S5;                // advance to next state
         break;

      case RD_S5: // raise CS and wait 500ns (EEPROM CS disable time)
         EE_CS = 1;                    // de-assert EEPROM CS

         // set Timer0 to interrupt 500ns from now
         ET0 = 0;                      // disable Timer0 interrupts
         TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
         TH0 = (-SYSCLK/2000000) >> 8; // set Timer0 to overflow in 500ns
         TL0 = -SYSCLK/2000000;
         ET0 = 1;                      // enable Timer0 interrupts
         TR0 = 1;                      // START Timer0

         state = RD_S6;                // advance to next state
         break;

      case RD_S6: // read data from SPI0 and post EEPROM ready
         EE_Data = SPI0DAT;            // read EEPROM data from SPI0
         EE_Ready = TRUE;              // indicate EEPROM ready for
                                       // next operation
         state = RESET;                // reset state variable
         break;

      case WR_S0: // transmit WRITE ENABLE opcode
         SPI0DAT = EE_WREN;            // transmit WREN opcode
         state = WR_S1;                // advance to next state
         break;

      case WR_S1: // wait at least 250ns (CS hold time)
         // set Timer0 to interrupt 250ns from now
```

```
      ET0 = 0;                       // disable Timer0 interrupts
      TCON &= ~0x30;                 // STOP Timer0 and clear overflow flag
      TH0 = (-SYSCLK/4000000) >> 8; // set Timer0 to overflow in 250ns
      TL0 = -SYSCLK/4000000;
      ET0 = 1;                       // enable Timer0 interrupts
      TR0 = 1;                       // START Timer0

      state = WR_S2;                 // advance to next state
      break;

   case WR_S2: // raise CS and wait 500ns (CS disable time)
      EE_CS = 1;                     // deassert CS

      // set Timer0 to interrupt 500ns from now
      ET0 = 0;                       // disable Timer0 interrupts
      TCON &= ~0x30;                 // STOP Timer0 and clear overflow flag
      TH0 = (-SYSCLK/2000000) >> 8; // set Timer0 to overflow in 500ns
      TL0 = -SYSCLK/2000000;
      ET0 = 1;                       // enable Timer0 interrupts
      TR0 = 1;                       // START Timer0

      state = WR_S3;                 // advance to next state
      break;

   case WR_S3: // assert CS and wait 250ns (CS setup time)
      EE_CS = 0;                     // assert CS

      // set Timer0 to interrupt 250ns from now
      ET0 = 0;                       // disable Timer0 interrupts
      TCON &= ~0x30;                 // STOP Timer0 and clear overflow flag
      TH0 = (-SYSCLK/4000000) >> 8; // set Timer0 to overflow in 250ns
      TL0 = -SYSCLK/4000000;
      ET0 = 1;                       // enable Timer0 interrupts
      TR0 = 1;                       // START Timer0

      state = WR_S4;                 // advance to next state
      break;

   case WR_S4: // transmit WRITE opcode
      SPI0DAT = EE_WRITE;            // transmit WRITE opcode
      state = WR_S5;                 // advance to next state
      break;

   case WR_S5: // transmit MSB of Address
      SPI0DAT = EE_Addr >> 8;        // transmit MSB of Address
      state = WR_S6;                 // advance to next state
      break;

   case WR_S6: // transmit LSB of Address
      SPI0DAT = EE_Addr;             // transmit LSB of Address
      state = WR_S7;                 // advance to next state
      break;

   case WR_S7: // transmit DATA
      SPI0DAT = EE_Data;             // transmit DATA
      state = WR_S8;                 // advance to next state
      break;

   case WR_S8: // wait 250ns (CS hold time)
```

```
      // set Timer0 to interrupt 250ns from now
      ET0 = 0;                      // disable Timer0 interrupts
      TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
      TH0 = (-SYSCLK/4000000) >> 8; // set Timer0 to overflow in 250ns
      TL0 = -SYSCLK/4000000;
      ET0 = 1;                      // enable Timer0 interrupts
      TR0 = 1;                      // START Timer0

      state = WR_S9;                // advance to next state
      break;

   case WR_S9: // deassert CS and wait 500ns (CS disable time)
      EE_CS = 1;                    // deassert CS

      // set Timer0 to interrupt 500ns from now
      ET0 = 0;                      // disable Timer0 interrupts
      TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
      TH0 = (-SYSCLK/2000000) >> 8; // set Timer0 to overflow in 500ns
      TL0 = -SYSCLK/2000000;
      ET0 = 1;                      // enable Timer0 interrupts
      TR0 = 1;                      // START Timer0

      state = WR_S10;               // advance to next state
      break;

   case WR_S10: // assert CS and wait 250ns (begin polling RDSR)
      EE_CS = 0;                    // assert CS

      // set Timer0 to interrupt 250ns from now
      ET0 = 0;                      // disable Timer0 interrupts
      TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
      TH0 = (-SYSCLK/4000000) >> 8; // set Timer0 to overflow in 250ns
      TL0 = -SYSCLK/4000000;
      ET0 = 1;                      // enable Timer0 interrupts
      TR0 = 1;                      // START Timer0

      state = WR_S11;               // advance to next state
      break;

   case WR_S11: // transmit Read Status Register opcode
      SPI0DAT = EE_RDSR;            // transmit RDSR opcode
      state = WR_S12;               // advance to next state
      break;

   case WR_S12: // transmit dummy write to read Status Register
      SPI0DAT = 0;                  // dummy write (after this completes,
                                    // SPI0DAT will contain Read Status
                                    // Register contents, which are decoded
                                    // in WR_S15 below)
      state = WR_S13;               // advance to next state
      break;

   case WR_S13: // wait 250ns (CS hold time)
      // set Timer0 to interrupt 250ns from now
      ET0 = 0;                      // disable Timer0 interrupts
      TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
      TH0 = (-SYSCLK/4000000) >> 8; // set Timer0 to overflow in 250ns
      TL0 = -SYSCLK/4000000;
      ET0 = 1;                      // enable Timer0 interrupts
```

```
         TR0 = 1;                      // START Timer0

         state = WR_S14;               // advance to next state
         break;

      case WR_S14: // deassert CS and wait 500ns (CS disable time)
         EE_CS = 1;                    // deassert CS

         // set Timer0 to interrupt 500ns from now
         ET0 = 0;                      // disable Timer0 interrupts
         TCON &= ~0x30;                // STOP Timer0 and clear overflow flag
         TH0 = (-SYSCLK/2000000) >> 8; // set Timer0 to overflow in 500ns
         TL0 = -SYSCLK/2000000;
         ET0 = 1;                      // enable Timer0 interrupts
         TR0 = 1;                      // START Timer0

         state = WR_S15;               // advance to next state
         break;

      case WR_S15: // check WIP bit (LSB of RDSR): if '1', then poll again;
                   // otherwise, RESET and post Write Complete
         if (SPI0DAT & 0x01) {         // TRUE if write in progress
            state = WR_S10;            // poll RDSR again
            SPIF = 1;                  // initiate new polling operation
         } else {                      // we're done. clean up.
            EE_Ready = TRUE;           // indicate EEPROM available
            state = RESET;             // reset state variable
         }
         break;

      default:
         while (1);                    // error
   }
}
```

Contact Information

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

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