```
//-----------------------------------------------------------------------------
// Light_find.c
//-----------------------------------------------------------------------------
// Author: Baylor Electromechanical Systems
//
// Operates on an external 18.432MHz oscillator.
//
// Target: Cygnal Educational Development Board / C8051F020
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//
// This program dives a stepper motor and reads the voltage drop from across a
// photo resistor that is mounted on the shaft of the stepper motor.  The user
// initiates the initial motor rotation by pressing button P3.7.  At each
// step of the motor, the program reads 256 samples of the voltage into ADC0.0.
// It calculates the average by adding each sample to a running accumulator and
// shifting right by 8 bits (divide by 256).  The sampling technique of adding
// a set of values and decimating them (posting results every (n)th sample) is
// called 'integrate and dump.'  It is easy to implement and requires very few
// resources.
//
// It then compares the average voltage drop across the photo resistor to a
// previous value of the lowest voltage drop and saves that value if it is
// lower than the previous value.  A low voltage drop coresponds to a brighter
// light.  After a full rotation is achieved, the stepper motor reverses
// direction and returns to the position where the brightest light was measured.
// The program waits for the user to again press button P3.7 before it returns
// to the start position of the stepper motor.
//
// Assumes an 18.432MHz 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.
//
// For each power of 4 of <INT_DEC>, you gain 1 bit of effective resolution.
// For example, <INT_DEC> = 256 gain you 4 bits of resolution: 4^4 = 256.
//
// Also note that the ADC0 is configured for 'LEFT' justified mode.  In this
// mode, the MSB of the ADC word is located in the MSB position of the ADC0
// high byte.  Using the data in this way makes the magnitude of the resulting
// code independent of the number of bits in the ADC (12- and 10-bits behave
// the same).
//

//-----------------------------------------------------------------------------
// 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       18432000          // 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 = P1^7;                       // 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);
int stpdrvr(int cmd);
void delay_ms(int ms);

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

void main (void)
{
      long result,minvalue;              // AIN0 results & value at bright
light
      int steps;                               // steps from start postition
counter
      int minstep;                             // stores the location of
brightest light



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

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

```
      ADC0_Init ();                         // init ADC
      Timer3_Init (SYSCLK/SAMPLERATE0);   // initialize Timer3 to overflow at
                                                        // sample rate

      AD0EN = 1;                                        // enable ADC
      EA = 1;                                                 // Enable global
interrupts

      stpdrvr(0x00);                                    // reset device

      while(1)                                          // the BIG loop
      {
            while (!SW1)                                // begin search on button
press
            {
                  steps = 0;                            // initilize variables
                  minvalue = 65535;             // 65535 is above range (~no
light)
                  minstep = 0;



                  while(steps < 48)       // will step though motor 48 times 1
rev
                  {
                        static unsigned int_dec=INT_DEC;  // integrate/decimate
                                                                        // 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
                        while(int_dec > 0)
                        {
                              accumulator += ADC0;          // read ADC value
and add
                                                                        // to
running total
                              int_dec--;                            // update
decimation counter
                        }

                        int_dec = INT_DEC;                      // reset
counter
                        result = accumulator >> 8;
                        accumulator = 0L;                   // reset accumulator

                        if (result < minvalue)              // check if brighter
light
```

```
                                                                  // than
previous brightest
                        {
                              minvalue = result;                  // if
brightest so far,
                                                                  // save
value and position
                              minstep = steps;
                        }
//                      printf ("Position '%d' voltage is %ldmV\n", steps,
voltage);

                        delay_ms(150);
                        stpdrvr(0x01);               // step motor one step
clockwise
                        steps++;                     // increment step counter
                  }

                  delay_ms(3000);                    // pause to verify one
                                                           // revolution is
complete

                  while(steps >= minstep)       // step counter clockwise to
point
                                                           // of brightest
light
                  {
                        delay_ms(150);
                        stpdrvr(0x02);
                        steps--;
                  }

         stpdrvr(0);          // Set all stepper drivers to 0 to
                              // stop current consumption

                  while (SW1) {     }           // pause to verify if it is
brightest light
                                                     // before pressing button
again for reset

                  while(steps > 0)        // restart stepper at beginning
position
                  {
                        delay_ms(150);
                        stpdrvr(0x02);
                        steps--;
                  }

         stpdrvr(0);          // Set all stepper drivers to 0 to
            }                       // stop current consumption
      }
}

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

```
//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use an 18.432MHz crystal
// as its clock source.
//
void SYSCLK_Init (void)
{
   int i;                              // delay counter

   OSCXCN = 0x67;                      // start external oscillator with
                                       // 18.432MHz 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
   P2MDOUT = 0x0F;                                 // PORT 2 CONFIGURATION
}

//-----------------------------------------------------------------------------
// 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
}

//-------------------------------------------------------------------------
// Local Functions
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// stpdrvr
//-------------------------------------------------------------------------
//
//    Function: Implments the full step drive for a unipolar stepper motor
//    Parameters:  cmd = 0 (off); cmd = 1 (CW); cmd = 2 (CCW)
//    Return: bit pattern to send to P2


int stpdrvr(int cmd)
{
      int step;
```

```
      int fwave[5] = {0x00, 0x0A, 0x09, 0x05, 0x06};        // for full step
//    int fwave[9] = {0x00, 0x0A, 0x08, 0x09, 0x01, 0x05, 0x04, 0x06, 0x02};

      // for half step

      switch(cmd)
      {
            case 0: step = 0; break;
            case 1: step = step-1;                    // clockwise step
                        if(step < 1)
                              step = 4;
                        break;
            case 2: step = step+1;                    // counterclockwise step
                        if(step > 4)
                              step = 1;
                        break;
            default: step = 0;
      }
      P2 = fwave[step];

      return fwave[step];
}

//-------------------------------------------------------------------------
// delay_ms
//-------------------------------------------------------------------------
//
// an approximate x ms delay
void delay_ms(int ms)
{
      int y;
      int z;
      for (y=1; y<=63; y++) for (z=1; z<= ms; z++);
}
```