```
//-----------------------------------------------------------------------------
// Keypad_DTMF.c
//-----------------------------------------------------------------------------
// Author: Baylor Electromechanical Systems
//
// Operates on an external 18.432MHz oscillator.
//
// This program interfaces Cygnal's C8051F02x with a 4x4, 16 pad keypad.  The
// program was designed for the Grayhill 96BB2-056-F.  With keypad DIP switch
// (SW5) toggled to the on positions, pins 1-8 on the keypad are
// connected to P2.0 - P2.7.
//
// 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.
//
//
// Target: Cygnal Educational Development Board / C8051F020
// 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 BAUDRATE     9600              // Baud rate of UART in bps
#define SYSCLK       18432000          // SYSCLK frequency in Hz

#define SAMPLE_RATE  50000             // Sample 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 LOW697  697 * 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 UART0_Init (void);
int button_dn(void);
unsigned int scankey (void);
void phase_select (void);
void delay_ms(int ms);

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

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

int rownum,colnum;

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

// Lookup table for converting keycode to ASCII
unsigned int keytab[4][4] ={{'1','2','3','A'},
                            {'4','5','6','B'},
                            {'7','8','9','C'},
                            {'*','0','#','D'}};


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) {

   unsigned int rd1;


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

   SYSCLK_Init ();
   PORT_Init ();
   UART0_Init ();                   // initialize UART0

   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

   EA = 1;                          // Enable global interrupts

    while (1)
    {
        if(button_dn())                     // check for key press
        {
            delay_ms(5);                    // delay for debouncing
            rd1 = scankey();                // read keypad
            if(rd1 != 0)
            {
                putchar (254);              // LCD command
                putchar (0x01);             // clear LCD
                printf (" You pressed:\r   ");
                putchar(rd1);
            }
            phase_select ();                // assign 2 tones for col and row

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

            while(button_dn());             // check for key release

            tone1_en = 0;                   // disable low group tones
            tone2_en = 0;                   // disable high group tones

        }
        delay_ms(5);
    }
}

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

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

```
// 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++) ;        // 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

   // PORT   3 CONFIGURATION
    P2MDOUT = 0xF0;                 // P2 u.n. push pull, lower-nibble input
    P2 = 0x0F;                      // upper nibble hi-imp, allowing input read

}

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


//-------------------------------------------------------------------------
// 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;
   }
}

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

```
//-----------------------------------------------------------------------------
// button_dn
//-----------------------------------------------------------------------------
//
// Function: test keypad for the presence of a key press.
// Return:   1 if keypress; 0 otherwise.

int button_dn()
{
    int tmp;
    tmp = (P2 & 0x0F)^0x0F;             // read P2.3->P2.0 and XOR output

    if(tmp)                             // if button is depressed, tmp != 0
        return 1;
    else
        return 0;
}

//-----------------------------------------------------------------------------
// scankey
//-----------------------------------------------------------------------------
//
// Function: read keypad and convert keypress into equiv. ASCII code.
// Return: ASCII equivalent of pressed key's label.

unsigned int scankey()
{
    int row = 0;
    int col = 0;

    P2 = 0x0F;                          // set data register
    P2MDOUT = 0xF0;                     // drive P2.3->P2.0 as output
    delay_ms(10);                       // let drive signals settle

    row = (P2 & 0x0F)^0x0F;             // read P2.3->P2.0 and XOR output

    delay_ms(2);

    if(row == 0)
        return 0;                       // no closure detected

    P2 = 0xF0;                          // set data register
    P2MDOUT = 0x0F;                     // drive P2.7->P2.4 as output
    delay_ms(2);                        // let drive signals settle

    col = (P2 & 0xF0)^0xF0;             // P2.7->P2.4 and XOR output
    col = col >> 4;                     // move hi nibble to lo nibble

    if(col == 0)
        return 0;                       // no closure detected

    P2 = 0x0F;                          // set data register
    P2MDOUT = 0xF0;                     // drive P2.3->P2.0 as output
    delay_ms(2);                        // let drive signals settle

    switch(row)                         // convert 1-of-4 to binary
        {
          case 1:   rownum = 0; break;
          case 2:   rownum = 1; break;
          case 4:   rownum = 2; break;
          case 8:   rownum = 3; break;
          default:  return 0;
        }

    switch(col)                         // convert 1-of-4 to binary
        {
          case 1:   colnum = 0; break;
```

```
          case 2:   colnum = 1; break;
          case 4:   colnum = 2; break;
          case 8:   colnum = 3; break;
          default:  return 0;
        }

    return keytab[rownum][colnum];      // return the ASCII value
}

//-----------------------------------------------------------------------------
// delay_ms
//-----------------------------------------------------------------------------
//
// an approximate x ms delay

void delay_ms(int ms)
{
    int y;
    int z;
    for (y=1; y<=250; y++) for (z=1; z<= ms; z++);
}

//-----------------------------------------------------------------------------
// phase_select
//-----------------------------------------------------------------------------
//


void phase_select ()
{
    switch(rownum)                          // convert 1-of-4 to binary
        {
          case 0:   phase_add1 = LOW697; break;
          case 1:   phase_add1 = LOW770; break;
          case 2:   phase_add1 = LOW852; break;
          case 3:   phase_add1 = LOW941; break;
          default:  break;
        }

    switch(colnum)                          // convert 1-of-4 to binary
        {
          case 0:   phase_add2 = HI1209; break;
          case 1:   phase_add2 = HI1336; break;
          case 2:   phase_add2 = HI1477; break;
          case 3:   phase_add2 = HI1633; break;
          default:  break;
        }
}
```