```c
//-----------------------------------------------------------------------------
// Magcard.c
//-----------------------------------------------------------------------------
// Author: Baylor Electromechanical Systems
//
// Operates on an external 18.432 MHz oscillator.
//
// Target: Cygnal Educational Development Board / C8051F020
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//
// This program receives the data stream from a magnetic card reader and
// outputs the data in ASCII format over a RS-232 protocol.  The code is
// tailored for a TTL card reader that reads only the second track on standard
// financial transaction cards.
//

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

sbit LED = P1^6;                       // LED='1' means ON
sbit STROBE = P0^2;
sbit DATA = P0^3;
sbit CARD = P0^4;

// 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'}};


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

void SYSCLK_Init (void);
void PORT_Init (void);
void INT0_ISR (void);
void UART0_Init (void);
```

```
int button_dn(void);
unsigned int scankey (void);
void delay_ms(int ms);


//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------
    bit inbit;
    bit ended = 0;
    bit started = 0;
    bit transmit_now = 0;
    int char_index = 0;
    int bit_index = 0;
    int num_chars = 0;
    int parity_errors = 0;
    bit parity = 1;

    char char_data;
    char track2[40];

    char bdata error_temp = 0;      // bit-addressable character location
    sbit bit0 = error_temp^0;       // parity bit (bit 4) of error_temp



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

void main (void)
{
    unsigned int rd1;
    bit scrolling=1;

    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


    while (1)
    {

        putchar (254);
        putchar (0x01);
        printf(" Swipe Card...\n");
        while (transmit_now==0)         // wait for transmit state to be set
        {   }

        parity_errors=0;
        for (char_index=0; char_index<num_chars; char_index++)
        {

            // ----- CHECK FOR PARITY ERRORS -----//
            error_temp = track2[char_index];    // copy each character to the
                                                // bit-addressable location
                                                // "error_temp"
            parity=1;
            for(bit_index=0; bit_index<5; bit_index++)
            {
                parity ^= bit0;                 // XOR "parity" with each bit
                error_temp = error_temp>>1;
            }
            if (parity) parity_errors++;        // if "parity" <> 0, then
```

```
                                                // parity error in character

            // ----- SEND CHARACTERS TO DISPLAY -----//
            char_data = track2[char_index];
            char_data &= 0x0F;                  // clear parity bit
            char_data += 0x30;                  // add ASCII offset

            if ((char_data!=';')&&(char_data!='?'))  // suppress start
                                                     // and end sentinels
                putchar(char_data);             // send character to display
        }


        while (CARD==0)             // wait for card load line to go high
        {   }

        transmit_now=0;
        started=0;
        ended=0;
        scrolling=1;

    while (scrolling)
    {
        if(button_dn())                     // check for key press
        {
            delay_ms(5);                    // delay for debouncing
            rd1 = scankey();                // read keypad
            if(rd1 != 0)
             {
                if (rd1=='*')
                 {
                   putchar (254);
                   putchar (28);
                 }
                if (rd1=='#')
                 {
                   putchar (254);
                   putchar (24);
                 }
                if (rd1=='0') scrolling=0;
             }
        }
        delay_ms(250);
    }

   }
}

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

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use an 18.432 MHz 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
//-----------------------------------------------------------------------------
void PORT_Init (void)
// Configure the Crossbar and GPIO ports
{
                            // DIGITAL CROSSBAR CONFIGURATION
    XBR0 = 0x04;            // XBAR0: Initial Reset Value
    XBR1 = 0x04;            // XBAR1: INT0 Input Enable
    XBR2 = 0x40;            // XBAR2: Enable weak pull-ups


    P0MDOUT = 0x01;         // PORT 0 CONFIGURATION
    // P0.0 = UART TX0        (Push-Pull Output)
    // P0.1 = UART RX0        (Open-Drain Output/Input)
    // P0.2 = /INT0 - /Strobe (Open-Drain Output/Input)
    // P0.3 = /Data           (Open-Drain Output/Input)
    // P0.4 = /Card Present   (Open-Drain Output/Input)
    // P0.5 = unassigned      (Open-Drain Output/Input)
    // P0.6 = unassigned      (Open-Drain Output/Input)
    // P0.7 = unassigned      (Open-Drain Output/Input)

                            // PORT 1 CONFIGURATION
    P1MDOUT = 0x40;         // P1.6 (LED) is push-pull output

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


                            // INTERRUPT CONFIGURATION
    IE = 0x01;              // Enable INT0 External Interrupt
    IT0 = 1;                // INT0 External Interrupt on falling edges
}

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

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

{
   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
}


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

//-----------------------------------------------------------------------------
// INT0_ISR
//-----------------------------------------------------------------------------
//
// INT0 External Interrupt ISR
```

```
//

void INT0_ISR (void) interrupt 0
  // INT0 External Interrupt Routine
  // When first nonzero bit on the data line is detected, this routine
  // enables the input state by setting the "started" flag.  Successive
  // bits are added to the track2 array in 5 bit characters.  After each
  // character is added, this routine compares the character to the end
  // sentinel [1111].  Once the end sentinel is detected, the routine
  // disables the input state by setting the "ended" flag, and begins the
  // transmission state by
{
    IE0 = 0;                            // Clear INT0 interrupt flag

    if (DATA)   inbit = 0;
    else        inbit = 1;              // invert logic on data line

    if ((inbit==1)&&(!started))         // first nonzero bit is detected
    {
        started=1;                      // initialize state variables
        ended=0;
        char_index=0;
        bit_index=0;

        putchar (254);
        putchar (0x02);

    }

    if ((started)&&(!ended))            // input state has been started
    {                                   // and has not yet been ended
        char_data = track2[char_index];
        char_data = char_data>>1;
        if (inbit)  char_data |= 0x10;  // add a '1' bit to current character
        else        char_data &= 0x0F;  // add a '0' bit to current character
        track2[char_index] = char_data;
        bit_index++;                    // increment the bit index

        if (bit_index==5)
        {
            char_index++;               // move to next chararacter in track2
            bit_index=0;                // reset the bit index
            char_data &= 0x1F;          // clear the 3 MSB
            ended = (char_data==0x1F);  // check for end sentinel [11111]

            if (ended)                  // end sentinel has been detected
            {
                num_chars=char_index;   // note the number of track2 characters
                transmit_now=1;         // enable the transmission state
            }

        }

    }

}   // end INT0_ISR

//-----------------------------------------------------------------------------
// 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(void)
{
    int row = 0;
    int col = 0;
    int k,j;

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

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

    return keytab[j][k];    // return the ASCII value at that row and column
```

```c
}

//-----------------------------------------------------------------------------
// delay_ms
//-----------------------------------------------------------------------------
//
// Function: approximate x ms delay
// Return: void

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