```
//-----------------------------------------------------------------------------
// IR.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
//
// Interprets IR data from a Radio Shack Cat #15-1989 IR control and outputs
// result on LCD. Note, this routine is time based, so if a different clock
// source is used, the timing characteristics must be adjusted accordingly.
// Also, if a different transmitter is used the bit pattern must be changed
// as well.
//

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

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

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

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR3RL   = 0x92;                 // Timer3 reload value
sfr16 TMR3     = 0x94;                 // Timer3 counter


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

#define BAUDRATE     9600              // Baud rate of UART in bps
#define SYSCLK       18432000          // SYSCLK frequency in Hz

//-----------------------------------------------------------------------------
// Infrared Characteristics
//-----------------------------------------------------------------------------

#define QSIZE 33                        // 32 bit IR  & start of message bit
#define start_bit_min 108               // min length of start bit
#define start_bit_max 125               // max length of start bit
#define logic_high_min 19               // min length of a logic high
#define logic_high_max 20               // max length of logic high
#define logic_low_min 9                 // min length of logic low
#define logic_low_max 10                // max length of logic high

sbit IR = P0^2;                         // IR reciever -> ext interrupt



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

void SYSCLK_Init (void);
void PORT_Init (void);
void INT0_ISR (void);
void UART0_Init (void);
void Timer3_Init (int counts);
int enqueue(struct queue* qA, char ch);
int dequeue(struct queue* qA, char *ch);
```

```c
int full_q(struct queue* qA);
void init_q(struct queue* qA);
int empty_q(struct queue* qA);
char* translate (int);

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

struct queue
{
    int cnt;
    int front;
    int rear;
    char que[QSIZE];
};

struct queue timeq;
char* out_str = 0;                      // output character


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

void main (void)
{
    int sum;                            // sum value used in translation
    int i;                              // for loop counter
    char a;                             // dumby variable

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

    SYSCLK_Init ();                     // initialize oscillator
    PORT_Init ();                       // initialize crossbar and GPIO
    UART0_Init ();                      // initialize UART0
    Timer3_Init (SYSCLK/8);             // initialize Timer3 to overflow at
    init_q (&timeq);                    // initialize timeq

    EA = 1;                             // Enable global interrupts
    putchar (254);                      // send LCD command
    putchar (0x01);                     // clear LCD
    while (1)
    {

        sum =0;
        if (full_q (&timeq))
            if (dequeue (&timeq,&a))
                if (a==2)
                {
                    for (i=0;i<16;i++)              // discard first 16 values
                        dequeue(&timeq,&a);
                    while (dequeue (&timeq,&a))     // translate last 16 bits
                    {
                        sum = sum << 1;
                        if ((int) a==1) sum|= 0x01; // sum = last 16 bits
                    }
                    out_str = translate (sum);
                    if (out_str !=0)
                    {
                        putchar (254);              // LCD command
                        putchar (0x01);             // clear LCD
                        printf (" You pressed:\r   %s" ,out_str);
                    }

                }   // end if
    }   // end while

}

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

void PORT_Init (void)
{
                            // 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 - /IR Data (Open-Drain Output/Input)

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

                            // 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)
{
   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
```

```
}

//-----------------------------------------------------------------------------
// 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 = 0x01;                      // Stop Timer3; Clear TF3;
                                       // use extern_clock/8 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
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// INT0_ISR
//-----------------------------------------------------------------------------
//
// INT0 External Interrupt ISR
//
void INT0_ISR (void) interrupt 0
{
    int TMR;
    IE0 = 0;                        // Clear INT0 interrupt flag


    TMR = TMR3H;                    // set to timer 3 high nibble

        if ((TMR >= start_bit_min) && (TMR <= start_bit_max))
            enqueue (&timeq,2);
        else
        if ((TMR >= logic_low_min) && (TMR <=logic_low_max))
            enqueue (&timeq,0);
        else
        if ((TMR >= logic_high_min) && (TMR <=logic_high_max))
            enqueue (&timeq,1);

    TMR3=0;                         // reset timer
}   // end INT0_ISR

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

//-----------------------------------------------------------------------------
// Translate - IR Translation table
//-----------------------------------------------------------------------------
// This function translates the bit stream into the button
// label for the Radio Shack Cat #15-1989 IR control
// for example 4335 = 1000011101111
//
char* translate (int sum)
{
            switch (sum)
            {
                case 4335 : {out_str = "Power"; break;}
                case 16575 : {out_str = "Vol +"; break;}
                case -16321 : {out_str = "Vol -"; break;}
                case -28561 : {out_str = "Mute"; break;}
                case 255 : {out_str = "Ch +"; break;}
                case -32641 : {out_str = "Ch -"; break;}
                case 22695 : {out_str = "Last"; break;}
                case -30601 : {out_str = "1"; break;}
                case 18615 : {out_str = "2"; break;}
                case -14281 : {out_str = "3"; break;}
                case 10455 : {out_str = "4"; break;}
                case -22441 : {out_str = "5"; break;}
                case 26775 : {out_str = "6"; break;}
                case -6121 : {out_str = "7"; break;}
                case 6375 : {out_str = "8"; break;}
                case -26521 : {out_str = "9"; break;}
                case 2295 : {out_str = "0"; break;}
                default : out_str = 0;
            }
        return out_str;
}

//-----------------------------------------------------------------------------
// Queue Functions
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Init_q
//-----------------------------------------------------------------------------
void init_q(struct queue* qA)
{
    qA->cnt = 0;                    // set initial values
    qA->front = 0;
    qA->rear = 0;

}

//-----------------------------------------------------------------------------
// Enqueue
//-----------------------------------------------------------------------------
int enqueue(struct queue* qA, char ch)
{
    int index;

    if(qA->cnt >= QSIZE) return 0;  // enqueue fails
    index = qA->rear;               // move rear index
    qA->que[index] = ch;            // copy character
    index++;
    if(index >= QSIZE) index = 0;
    qA->rear = index;               // update rear index
    (qA->cnt)++;                    // increase count
    return 1;                       // report sucess
}

//-----------------------------------------------------------------------------
// Dequeue
//-----------------------------------------------------------------------------
int dequeue(struct queue* qA, char *ch)
{
    int index;

    if(qA->cnt <= 0) return 0;      // dequeue fails
    index = qA->front;
    *ch = qA->que[index];           // remove front item
    index++;                        // get front index
    if(index >= QSIZE) index = 0;
    qA->front = index;              // update front
    --(qA->cnt);                    // reduce count
    return 1;                       // return sucess
}
```

```
//-----------------------------------------------------------------------------
// Full_q
//-----------------------------------------------------------------------------
int full_q(struct queue* qA)
{
    return (qA->cnt >= QSIZE);
}
```