

```

/*-----+
AT89C51 as a decoder of RF modules AM-HRR3-433, AM Supter Regenerative Receivers
Name : reception.c
Purpose:
    For many remote applications, encoder/decoder pair IC always used for simpler design.
    This program provide the function same as RF decoder IC. Saving the cost for buying
    the encoder and decoder pair IC. Also, the transmission capacity, data rate should be
    higher, as we direct access the RF modules pair.
    Get the rf series byte data through the air and then export it to P1.
    The program is use the polling technique. Data protocol is asynchronous mode, that is
    1 start bit + 8 bits byte data + 2 stop bit, total 11 bits.
written by: Dillian Wong 27/5/04
-----*/

/*----- I N C L U D E S -----*/
#include <reg51.h>
#include <stdio.h>
#include "LCDDriver\LCDConfig.h"
#include "rPrintf\rprintf.h"

/*----- H A R D W A R E   P I N   C O N N E C T I O N -----*/
// define hardware pin connection
sbit DIN = P3^3;

/*----- P R O T O C O L -----*/
char getrcvdata(void);
void delay(void);
void delay2(void);

/*----- C O M P O R T -----*/
// com port with 9600 baud with crystal 11.0592MHz.
void init_uart(void){    SCON=0x50; TMOD|=0x20; TH1=253; TR1=1; TI=1; }

/*----- M A I N   P R O G R A M -----*/

void main(void)
{
    char rcvdata;
    init_uart();                // 9600 baud @ 11.0592MHz
    turnOnLCD();
    turnOnRS232();
    printf("#Rx:\n");

    while(1)
    {
        rcvdata=getrcvdata();
        // printf("(%c)",rcvdata);
        printf("(0x");
        printHex8(rcvdata);
        // printf(")");
        printf("  ");
        P1=rcvdata;
    }
}

/*----- A S Y C H R O N O U S   R E C E P T I O N -----*/
/*
using RF modules is different from direct line connection. The idle state in many design is
logic high. However, because RF receiver modules cannot stay in logic high in a long time.
Therefore, the idle states in RF is always logic low. In order to stick on the standard
asynchronous transmission mode, i remain the idle states in logic high. To achieve this,
before the data transmit, the transmitter will send a logic high first, to create a virtual
idle state. when DOUT falling, that means it is the start bit. and followed by 8 bits data and
2 stop bits.
*/
char getrcvdata(void)
{
    bit a,b,c,d,e,f,g,h,i,j,k,l;                //bit data
    // int p;                //checking use!
    char rcvdata=0x00;

    /* START receptions*/
    while(1)                //loop until a valid rcvdata
    {
        while(!DIN)                //rising edge, in idle
        {                        //trigger + pretend idle states
            delay2();

```

```

        if(DIN) {a=1;}else {a=0;}          /*start bit*/
        if(DIN)                             //high means a valid start bit
        {
            delay();
            if(DIN) {b=1;}else {b=0;} delay();      //8 bits byte data
            if(DIN) {c=1;}else {c=0;} delay();
            if(DIN) {d=1;}else {d=0;} delay();
            if(DIN) {e=1;}else {e=0;} delay();
            if(DIN) {f=1;}else {f=0;} delay();
            if(DIN) {g=1;}else {g=0;} delay();
            if(DIN) {h=1;}else {h=0;} delay();
            if(DIN) {i=1;}else {i=0;} delay();
            if(DIN) {j=1;}else {j=0;} delay();      //2 stop bits
            if(DIN) {k=1;}else {k=0;} delay();
            if(DIN) {l=1;}else {l=0;} delay();      //virtual idle states, after the DOUT
                                                    //will return to 0

//checking use!
/*      p=a;printf("%d",p);
        p=b;printf("%d",p);p=c;printf("%d",p);p=d;printf("%d",p);p=e;printf("%d",p);
        p=f;printf("%d",p);p=g;printf("%d",p);p=h;printf("%d",p);p=i;printf("%d",p);
        p=j;printf("%d",p);p=k;printf("%d",p);p=l;printf("%d",p);
*/

        /*extract the byte data */
        if (b) { rcvdata |= 0x01; }          //(LSB)
        if (c) { rcvdata |= 0x02; }
        if (d) { rcvdata |= 0x04; }
        if (e) { rcvdata |= 0x08; }
        if (f) { rcvdata |= 0x10; }
        if (g) { rcvdata |= 0x20; }
        if (h) { rcvdata |= 0x40; }
        if (i) { rcvdata |= 0x80; }          //(MSB)
        while(DIN);                          //wait until return idle state

        //1 high start bits , 2 low stop bits,
        //followed by a high bits means return to idle state
        if(a==1 && j==0 && k==0 && l==1)
            return rcvdata;                  //return a valid data
        else
            rcvdata=0x00;                    //if not valid, clear the buffer
    }
} //end of while(!DIN)
}

/*_____ DELAY USE TIMER0 _____*/
/*
    delay about 4.34ms
*/
void delay(void)
{
    unsigned int count=1000;
    TR0=0;                                  //turn off timer0, before setting
    TMOD |= 0x01;                          //set time0 as mode0
    TH0=(65536-count)/256;                  //count 2304 machine cycle
    TL0=(65536-count)%256;                  //1 machine cycle = 12/11.0598M = 1.085us
    IE |=0x82;                             //2304x1.085u=2.5ms
    TR0=1;                                  //turn on timer0;
    while(TR0);                             //wait until timer0 is off.
}

/*
    delay about 2.17ms
*/
void delay2(void)
{
    unsigned int count=500;
    TR0=0;                                  //turn off timer0, before setting
    TMOD |= 0x01;                          //set time0 as mode0
    TH0=(65536-count)/256;                  //count 2304 machine cycle
    TL0=(65536-count)%256;                  //1 machine cycle = 12/11.0598M = 1.085us
    IE |=0x82;                             //2304x1.085u=2.5ms
    TR0=1;                                  //turn on timer0;
    while(TR0);                             //wait until timer0 is off.
}

void timer0_ISR (void) interrupt 1{TR0=0;}

```

```

/*-----+
|
|                      LCD 16x2 Driver (HD44780)
|
| Name: Driver_Lcd16x2.c
| Purpose:
|   This is the function code used to modify the standard c function "printf" by
|   replacing the function "putchar" to suit the LCD driver HD44780. By adding this
|   file, LCD panel with a chip HD44780 can print out a string using the standard c
|   function "printf". \n, \r, and roll down will be included.
|
| By Dillian Wong, last modify data 19/9/03
|-----*/

/*----- I N C L U D E S -----*/

#include <reg51.h>
#include <stdio.h>
#include "LCDConfig.h"
unsigned char flag=0;

/*----- M A C R O S -----*/

#define DISP_BUSY    0x80
#define DISP_FUNC    0x38
#define DISP_ENTRY   0x06
#define DISP_CNTL    0x08
#define DISP_ON       0x04
#define DISP_CURSOR  0x02
#define DISP_CLEAR    0x01
#define DISP_HOME     0x02
#define DISP_POS      0x80
#define DISP_LINE2    0x40

/*----- P R O T O C O L -----*/

void disp_cmd(unsigned char);
void disp_init(void);
unsigned char disp_read(void);
void disp_write(unsigned char);
unsigned char disp_read_char(void);
void disp_copyLine2toLine1(void);
void disp_clrLine2(void);
char LCD_putchar (char c);
void LCD_Locate(int p);
void LCD_Clr(int s, int e);

/*----- B A S I C -----*/

/*
   writes a given command to the LCD panel and waits to assure that the command
   was completed by the panel.
*/
void disp_cmd(unsigned char cmd) {
    DISPDATA=cmd;
    REGSEL=0;
    RDWR=0;
    ENABLE=1;
    ENABLE=0;
    while (disp_read() & DISP_BUSY);           // wait for the Lcd
}

/*
   sends the correct data sequence to the display to initialize it for use.
*/
void disp_init(void) {
    while (disp_read() & DISP_BUSY);
    disp_cmd(DISP_FUNC);                       // set the display for an 8
                                                // bit bus, 2 display lines,
                                                // and a 5x7 dot font
    disp_cmd(DISP_ENTRY);                      // set the character entry
                                                // mode to increment display
                                                // address for each
                                                // character, but not to scroll
    disp_cmd(DISP_CNTL | DISP_ON);             // turn the display on, cursor off
    disp_cmd(DISP_CLEAR);                     // clear the display
}

/*

```

```
    reads from the LCD panel status register.
*/
unsigned char disp_read(void) {
    unsigned char value;
    DISPDATA=0xFF;
    REGSEL=0;
    RDWR=1;
    ENABLE=1;
    value=DISPDATA;
    ENABLE=0;
    return(value);
}

/*
    writes a data byte to the LCD panel.
*/
void disp_write(unsigned char value) {
    DISPDATA=value;
    REGSEL=1;
    RDWR=0;
    ENABLE=1;
    ENABLE=0;
}

/*
    read data from CF or DD Ram
*/
unsigned char disp_read_char(void) {
    unsigned char value;
    DISPDATA=0xFF;
    REGSEL=1;
    RDWR=1;
    ENABLE=1;
    value=DISPDATA;
    ENABLE=0;
    return(value);
}

/*
    reads data from DD Ram in line2, and write char of these datas in line1
*/
void disp_copyLine2toLine1(void){
    unsigned int i;
    unsigned char k[16];
    disp_cmd(DISP_POS | DISP_LINE2);
    for(i=0;i<16;i++){
        k[i]=disp_read_char();
        while (disp_read() & DISP_BUSY);
    }
    disp_cmd(DISP_HOME);
    for(i=0;i<16;i++){
        disp_write(k[i]);
        while (disp_read() & DISP_BUSY);
    }
}

/*
    clear the display of Line2
*/
void disp_clrLine2(void){
    unsigned int i;
    disp_cmd(DISP_POS | DISP_LINE2);
    for(i=0;i<16;i++){
        disp_write(' ');
        while (disp_read() & DISP_BUSY);
    }
}

/* _____ A D V A N C E _____ */

/*
    clear the DD ram data of LCD from position s to position e
*/
void LCD_Clr(int s, int e)
{
    int i;
    LCD_Locate(s);
```

```

    for(i=0;i<=(e-s);i++){
        disp_write(' ');
        while (disp_read() & DISP_BUSY);
    }
}

/*
    locate the position of LCD
*/
void LCD_Locate(int p)
{
    /*
    HD44760 16x2 character LCD display
    the following .no is variable flag refer to
    +-----+
    | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 |
    | 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 |
    +-----+

    HD44760 16x2 character LCD display
    the following DDRam Address
    +-----+
    | 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F |
    | 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4A 0x4B 0x4C 0x4D 0x4E 0x4F |
    +-----+
    */

    if(p<=15){
        disp_cmd(DISP_POS | p);
        flag = p;
    }
    else if(p>15 && p<=31){
        disp_cmd(DISP_POS | (p+0x30));
        flag=p;
    }
    else if(p>31){
        disp_cmd(DISP_POS | 0x00);
        flag = 0;
    }
}

/*****
Function:    putchar
Description: This routine replaces the standard putchar
             function. Its job is to redirect output to the LCD panel.
Parameters:  c - char. This is h\next character to write to the
             display.
Returns:     The character just written.
*****/
char LCD_putchar(char c) {
    /*
    HD44760 16x2 character LCD display
    the following .no is variable flag refer to
    +-----+
    | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 |
    | 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 |
    +-----+
    */

    if (flag <=15){
        // standard format for the nextline
        if(c=='\n'){
            disp_clrLine2();
            disp_cmd(DISP_POS | DISP_LINE2);
            flag=16;
            return(c);
        }
        // standard format for return the line
        if(c=='\r'){
            disp_cmd(DISP_HOME);
            flag=1;
            return(c);
        }
    }

    if(flag==16){
        disp_cmd(DISP_POS | DISP_LINE2);
    }
}

```

```
    }

    if(flag>=16){
        // standard format for the nextline
        if(c=='\n'){
            disp_copyLine2toLine1();
            disp_clrLine2();
            disp_cmd(DISPOS | DISP_LINE2);
            flag=16;
            return(c);
        }
        // standard format for return the line
        if(c=='\r'){
            disp_cmd(DISPOS | DISP_LINE2);
            flag=16;
            return(c);
        }
    }

    if (flag>31){
        disp_copyLine2toLine1();
        disp_clrLine2();
        disp_cmd(DISPOS | DISP_LINE2);
        flag=16;
    }

    disp_write(c);                                // write the character to
                                                    // the display
    while (disp_read() & DISP_BUSY);              // wait for the display
    flag++;
    return(c);
}
```

```
/*
    Name : LcdAccess.c
    purpose:

*/

/*_____ I N C L U D E S _____*/
#include <reg51.h>
#include <stdio.h>
#include "LCDConfig.h"

/*_____ M A C R O S _____*/
#define true    1
#define false   0

/*_____ P R O T O C O L _____*/
char standard_putchar (char c);
char LCD_putchar(char c);
void disp_init(void);

/*_____ V A R I A B L I E S _____*/
bit isRS232 = true;
bit isLCD = false;

/*_____ F U N C T I O N _____*/

/*
    replace the standard putchar by adding a extra instruction.
*/
char putchar(char c)
{
    char ch;
    if(isRS232)
        ch = standard_putchar(c);
    if(isLCD)
        ch = LCD_putchar(c);
    return c;
}

/*
    flag used to indicate the status of display. Whether the lcd, com port should display the data
    out or not.
*/
void turnOnLCD(void) {    disp_init(); isLCD=true;    }    // indicate lcd on
void turnOffLCD(void) {    isLCD=false;    }    // indicate lcd off
void turnOnRS232(void) {    isRS232=true;    }    // indicate com port on
void turnOffRS232(void) {    isRS232=false;    }    // indicate com port off
bit isLCDOn(void) {    return isLCD;    }    // indicate where LCD is on or off
bit isRS232On(void) {    return isRS232;    }    // indicate where RS232 is on or off.
```

```
/*
    Name : standardPutChar.c
    purpose:
        standard putchar function of keil C
*/

/*_____ I N C L U D E S _____*/
#include <reg51.h>

/*_____ M A C R O S _____*/
#define XON  0x11
#define XOFF 0x13

/*_____ F U N C T I O N _____*/

/*
    standard putchar function of keil C
*/
char standard_putchar (char c) {
    if (c == '\n') {
        if (RI) {
            if (SBUF == XOFF) {
                do {
                    RI = 0;
                    while (!RI);
                } while (SBUF != XON);
                RI = 0;
            }
        }
        while (!TI);
        TI = 0;
        SBUF = 0x0d;
        /* output CR */
    }
    if (RI) {
        if (SBUF == XOFF) {
            do {
                RI = 0;
                while (!RI);
            } while (SBUF != XON);
            RI = 0;
        }
    }
    while (!TI);
    TI = 0;
    return (SBUF = c);
}
```



```
/* _____ H A R D W A R E _____ */

sbit ENABLE   =  P0^2;           // display enable output (EN)
sbit RDWR     =  P0^1;           // display access mode output (RW)
sbit REGSEL   =  P0^0;           // display register select output (DI)
sfr  DISPDATA =  0xA0;           // data bus to the display panel,P2

/* _____ P R O T O C O L _____ */

// LCDAccess.c
void turnOnLCD(void);             // turn on LCD, default is off
void turnOffLCD(void);           // turn off LCD
void turnOnRS232(void);          // turn on RS232 port
void turnOffRS232(void);         // turn on RS232 port
bit  isLCDOn(void);              // indicate where LCD is on or off
bit  isRS232On(void);            // indicate where RS232 is on of off.

// Driver_Lcd16x2.c
void LCD_Locate(int p);          // location the DD ram position of LCD
void LCD_Clr(int s, int e);      // clear the DD ram data from position s
                                  // to e.
```

```

/*-----+
|                                     |
|               Print Hex8 or Hex16 |
|                                     |
| Name: printHex.c                  |
|                                     |
| Purpose:                          |
|   print the hex8 or hex16 data the streaming output. |
|-----*/

/*_____ I N C L U D E S _____*/

#include <reg51.h>
#include <stdio.h>

/*_____ P R O T O C O L _____*/
void printHex8(unsigned char hexdata);
void printHex16(int hex16);
unsigned char hex2ascii(unsigned char bits_data);

/*_____ F U N C T I O N _____*/

/*
   print 8 bit hex data
*/
void printHex8(unsigned char hex8)
{
    char hex8H,hex8L;
    hex8L = hex2ascii(hex8&0x0f);           // convert low byte
    hex8H = hex2ascii((hex8&0xf0)/0x0f);     // convert high byte
    putchar(hex8H);                          // print out
    putchar(hex8L);
}

/*
   print 16 bit hex data
*/
void printHex16(int hex16)
{
    char hex16H,hex16L;
    hex16H=((hex16&0xff00)/0x00ff);          // high byte
    hex16L=((hex16&0x00ff));                // low byte
    printHex8(hex16H);
    printHex8(hex16L);
}

/*
   convert the hex data(4 bits) to ascii code(8 bits)
*/
unsigned char hex2ascii(unsigned char bits_data)
{
    if(bits_data >=0x0f)                    // all invalid data return 'F'
    {
        return 0x46;
    }
    if (bits_data >=0x00 & bits_data <=0x09) // return '0'-'9'
    {
        bits_data +=0x30;
        return bits_data;
    }
    if (bits_data >=0x0a & bits_data <=0x0f); // return 'A'-'F'
    {
        bits_data +=0x37;
        return bits_data;
    }
}

```

```
/*
  Name :  rprintf.h

  Header file for rprintf.c
*/

/* _____ P R O T O C O L _____ */

void printHex8(unsigned char hexdata);
void printHex16(int hex16);
void printInt(int num);
void printFloat(float flt);
void printStr(char str[]);
void rprintf(char type[],char content[]);
```