

```

/*-----+
|
|          DTMF Transmitter with Temperature sensor
|
| File Name : digitControl.c
| purpose :
|         This program provide the basic interface for the DTMF transmitter, the
|         program can send the phone number stored in Eeprom, and send the temperature
|         data in DTMF tone. of course, through RS232, Eeprom phone number can be
|         changed.
|-----*/

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

#include <reg51.h>
#include <stdio.h>
#include <ctype.h>           //use toint()
#include "MT8880C\configMT8880C.h"
#include "Eeprom\EepromConfig.h"
#include "DS1620\Ds1620.h"
#include "config.h"

/*----- F U N C T I O N   S E L E C T -----*/

// menu
char code menu[] = // menu of DTMF Transceiver
    "\n--Integrated DTMF Transceiver-1.3--"
    "\n1 Dial a phone digit by keying"
    "\n2 Dial the phone number stored in Eeprom"
    "\n3 transmit temperature data in DTMF tone"
    "\n4 Save a phone No. to Eeprom"
    "\n5 Load a phone No. in Eeprom"
    "\n6 Read current Temperature"
    ;

// meun function
void selectFunction (void)
{
    char cmd;           // selection
    char chDigit;       // tone digit

    scanf("%c", &cmd);
    switch(cmd)
    {
        case '1':
            turnOffTimer0();
            printf("\nsoftware reset :");
            MT8880C_Reset();           // software reset.
            printf("Dial a phone, press 'q' to exit");
            printf("\n?Digit : ");
            MT8880C_SetForTransmit_tone();           //setting before transmit a tone
            do
            {
                scanf("%c",&chDigit);
                MT8880C_TransDTMF_tone(chDigit);
            }
            while(chDigit!='q');
            printf("\n....ok!");
            turnOnTimer0();
            break;
        case '2':
            turnOffTimer0();
            dialPhoneInEeprom();
            turnOnTimer0();
            break;
        case '3':
            turnOffTimer0();
            dialPhoneAccordingTemperature();
            turnOnTimer0();
            break;
        case '4':
            turnOffTimer0();
            savePhoneNotoEeprom();
            turnOnTimer0();
            break;
        case '5':
            turnOffTimer0();
            loadPhoneNofromEeprom();
            turnOnTimer0();
            break;
        case '6':
            turnOffTimer0();
            ds1620_init(10,40);
            printf("\nCurrent temperature is: ");
            printf("%f",ds1620_ReadTemperature());
    }
}

```

```

        turnOnTimer0();
        break;
    case '?':    printf(menu);
}
}

/*_____ T I M E R 0   &   U A 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;
}

// initialize 16 bits timer0, with time interval equal 0.05s
void init_Timer0(void)
{
    TMOD |= 0x01; // set time0 as mode0
    TH0=(65536-46079)/256; // count 46080 machine cycle
    TL0=(65536-46079)%256; // 1 machine cycle = 12/11.0598M = 1.085us
    IE |=0x82; // 46080x1.085u=0.05s
    TR0=1;
}

// turn on timer0
void turnOnTimer0(void)
{
    TR0=1;
}

// turn off timer0
void turnOffTimer0(void)
{
    TR0=0;
}

// when timer0 time up, execute the code.
void timer0_ISR (void) interrupt 1
{
    TH0=(65536-46900)/256; // reset the timer 0.
    TL0=(65536-46900)%256; //fine tune. as crystal may not exactly 11.0598MHz
    P2 = 0xff;;

    /* send current temperature */
    if (P2 ==0xFE) //P2^0 = 0
    {
        turnOffTimer0();
        printf("\nSend data : ");
        dialPersonID();
        ds1620_init(10,40);
        dialPhoneAccordingTemperature();
        turnOnTimer0();
    }
    /* dial phone stored in Eeprom */
    if (P2 ==0xFD) //P2^1 = 0
    {
        turnOffTimer0();
        printf("\nSend phone Number stored in Eeprom : ");
        dialPhoneInEeprom();
        turnOnTimer0();
    }
}

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

/*
Main program of the DMFT Transmitter.
*/
void main(void)
{
    init_uart();    // utilize rs232 port
    printf(menu);   // print menu.
    init_Timer0();  // 0.05s time interval of time 0;
    MT8880C_Reset();
    while(1)

```

```
{  
    printf("\n[Enter selection : ]");  
    selectFunction();  
}  
}
```

```

#include <reg51.h>
#include <stdio.h>
#include <ctype.h>           //use toint()
#include "MT8880C\configMT8880C.h"
#include "Eeprom\EepromConfig.h"
#include "rprintf\rprintf.h"
#include "DS1620\Ds1620.h"

/*
    save the digit to Eeprom, until receive a enter key. sAddr stored the total number of
    digit of the phone number.
*/
void savePhoneNotoEeprom(void)
{
    char p,sAddr=0x40,addr;
    int totalDigit=0;
    addr=sAddr;

    printf("\nSave phone no. to Eerpom");
    printf("\nNo? : ");
    do
    {
        scanf("%c",&p);           // get phone No.
        saveChar(addr+1,p);       // save phone No.
        totalDigit++;
        addr++;
    }while(p!='\n');
    totalDigit--;
    saveChar(sAddr,totalDigit);   // used to stored the total number of digit.
}

/*
    load and print the phone number stored in Eeprom. The start address stored the total
    number of digit. That used to determine how long the phone digit is valid.
*/
void loadPhoneNofromEeprom(void)
{
    char addr=0x40;
    int i=0,totalDigit;

    totalDigit=loadChar(addr);    // starting address stored the total number
                                // of phone digit.
    printf("\nPhone No stored in Eeprom : ");
    do
    {
        printf("%c",loadChar(addr+1));
        i++;
        addr++;
    }while(i<totalDigit);
}

/*
    dial the phone number that stored in Eeprom. when load a digit, it will transmit a tone
    and print it out immediately.
*/
void dialPhoneInEeprom(void)
{
    char addr=0x40;
    char chDigit;
    int i=0,j=0,totalDigit;

    printf("\nTransmit the phone\n");
    /* setting before DTMF tone Transmit */
    MT8880C_Reset();           // software reset.
    MT8880C_SetForTransmit_tone(); //setting before transmit a tone

    /* Load & dial */
    totalDigit=loadChar(addr);  // starting address stored the total number
                                // of phone digit.
    do
    {
        // load the phone no to Eeprom
        {
            chDigit=loadChar(addr+1); // load digit
            MT8880C_TransDTMF_tone(chDigit); // send tone.
            for(j=0;j<30000;j++);
            printf("%c",chDigit);      // digit print out
            i++;
            addr++;
        }
    }while(i<totalDigit);
}

```

```
/*
    send person ID code for identification
*/
void dialPersonID(void)
{
    char chDigit;
    int j;

    /* setting before DTMF tone Transmit */
    MT8880C_Reset();                // software reset.
    MT8880C_SetForTransmit_tone(); // setting before transmit a tone

    chDigit = '#';                  // indicate the following is data
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = 'A';
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = 'B';
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = '1';
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = '2';
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = '3';
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = '4';
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = '5';
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);

    chDigit = '*';                  // indicate the end of is data
    MT8880C_TransDTMF_tone(chDigit);
    for(j=0;j<30000;j++);
    printf("%c",chDigit);
}

/*
    send the tone according the measuring temperature. This function will get the temperature form
    sensor first then convert it to tone digit for transmittion.
*/
void dialPhoneAccordingTemperature(void)
{
    float flt;                      // floating temperature reading
    int num;                        // float--> integer
    unsigned char a,b,c,d;          // for the segment of floating
    char chDigit;
    int j;

    /* setting before DTMF tone Transmit */
    MT8880C_Reset();                // software reset.
    MT8880C_SetForTransmit_tone(); // setting before transmit a tone

    ds1620_init(10,40);             // initialize DS1620
    flt = ds1620_ReadTemperature(); // get temperature reading.

    num=flt*10;                     // convert floating digit to integer, as only one floating point
```

```
/* This program code is direct copy from printInt. Change the putChar to chDigit. Then */
/* use MT8880C.c function to send dtmf tone */
a=num/0x03E8; // digit 1(MSB)
b=(num-a*0x03E8)/0x64; // digit 2
c=(num-a*0x03E8-b*0x64)/0x0A; // digit 3
d=num-a*0x03E8-b*0x64-c*0x0A; // digit 4(LSB)

printf("\n");

chDigit = '#'; // indicate the following is temperature
MT8880C_TransDTMF_tone(chDigit); // data.
for(j=0;j<30000;j++);
    printf("%c",chDigit);

if(a!=0x00){ // no. print out
    chDigit = hex2ascii(a);
    MT8880C_TransDTMF_tone(chDigit); // send tone.
    for(j=0;j<30000;j++);
    printf("%c",chDigit);
}
if((a+b)!=0x00){
    chDigit = hex2ascii(b);
    MT8880C_TransDTMF_tone(chDigit); // send tone.
    for(j=0;j<30000;j++);
    printf("%c",chDigit);
}
if((a+b+c)!=0x00){
    chDigit = hex2ascii(c);
    MT8880C_TransDTMF_tone(chDigit); // send tone.
    for(j=0;j<30000;j++);
    printf("%c",chDigit);
}
if((a+b+c+d)!=0x00){
    chDigit = hex2ascii(d);
    MT8880C_TransDTMF_tone(chDigit); // send tone.
    for(j=0;j<30000;j++);
    printf("%c",chDigit);
}

chDigit = '*'; // indicate the end of is temperature
MT8880C_TransDTMF_tone(chDigit); // data.
for(j=0;j<30000;j++);
    printf("%c",chDigit);
}
```

```
void turnOnTimer0(void);
void turnOffTimer0(void);

// PhoneNo.c
void savePhoneNoToEeprom(void);
void loadPhoneNoFromEeprom(void);
void dialPhoneInEeprom(void);
void dialPersonID(void);
void dialPhoneAccordingTemperature(void);
```

```

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

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

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

sbit CS = P1^5; // control
sbit RS = P1^4;
sbit RW = P1^7;
sbit b3 = P1^3; // data
sbit b2 = P1^2;
sbit b1 = P1^1;
sbit b0 = P1^0;

/* set or clr of important PIN */
#define setCS CS=1;while (CS!=1); // chip select, clr for select
#define clrCS CS=0;
#define setRS RS=1;while (RS!=1); // register select, set for select
#define clrRS RS=0;
#define setRW RW=1;while (RW!=1); // read write, set for read, clr for write
#define clrRW RW=0;

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

void MT8880C_Reset(void);
void MT8880C_SetForTransmit_tone(void);
void MT8880C_TransDTMF_tone(char ch);
int getToneDigit(char ch);

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

/*
A software reset must be included at the beginning of all programs to initialize the
control registers after power up. The initialization procedure should be implemented
100ms after power up.
*/
void MT8880C_Reset(void)
{
    int i;

    setCS;          //1) read status register
    setRS;setRW;
    clrCS;
    for(i=0;i<3;i++);
    setCS;

    setCS;          //2) write to control register
    setRS;clrRW;b3=0;b2=0;b1=0;b0=0;
    clrCS;
    for(i=0;i<3;i++);
    setCS;

    setCS;          //3) write to control register
    setRS;clrRW;b3=0;b2=0;b1=0;b0=0;
    clrCS;
    for(i=0;i<3;i++);
    setCS;

    setCS;          //4) write to control register
    setRS;clrRW;b3=1;b2=0;b1=0;b0=0;
    clrCS;
    for(i=0;i<3;i++);
    setCS;

    setCS;          //5) write to control register
    setRS;clrRW;b3=0;b2=0;b1=0;b0=0;
    clrCS;
    for(i=0;i<3;i++);
    setCS;

    setCS;          //6) read status register
    setRS;setRW;
    clrCS;
    for(i=0;i<3;i++);
    setCS;
}
/*

```



```

    setting before transmit a tone.
*/
void MT8880C_SetForTransmit_tone(void)
{
    int m;

    /*tone out, DTMF, IRQ, select control regiser B*/
    setCS;          //write to control register A
    setRS; clrRW; b3=1; b2=1; b1=0; b0=1;
    clrCS;          //toggle chip select to fatch the input data
    for(m=0;m<3;m++);
    setCS;

    /*burst mode*/
    setCS;          //write to control register B
    setRS; clrRW; b3=0; b2=0; b1=0; b0=0;
    clrCS;          //toggle chip select to fatch the input data
    for(m=0;m<3;m++);
    setCS;
}

/*
    Transmit a DTMF tone, ch is the tone digit (0-9,*,#,A-D)
*/
void MT8880C_TransDTMF_tone(char ch)
{
    int m;
    int i=0,j=0,k=0,l=0,digit;

    digit=getToneDigit(ch); //get the interger that represent the tone

    i=(digit&0x0008)/8;      //calculate the b3,b2,b1,b0
    j=(digit&0x0004)/4;
    k=(digit&0x0002)/2;
    l=(digit&0x0001)/1;

    // printf("\ndigit is : %d",digit);    //check the data bits b3-b0
    // printf("\ni      is : %d",i);
    // printf("\nj      is : %d",j);
    // printf("\nk      is : %d",k);
    // printf("\nl      is : %d",l);

    setCS;          //write to transmit data register
    clrRS; clrRW; b3=i; b2=j; b1=k; b0=l;
    clrCS;          //toggle chip select to fatch the input data
    for(m=0;m<3;m++);
    setCS;
}

/*
    return the interger that represent the digal number or character.
    1-9,0,*,#,A-D.
*/
int getToneDigit(char ch)
{
    int digit;

    switch(ch)
    {
        case '1' :
            digit=1;
            break;
        case '2':
            digit=2;
            break;
        case '3':
            digit=3;
            break;
        case '4':
            digit=4;
            break;
        case '5':
            digit=5;
            break;
        case '6':
            digit=6;
            break;
        case '7':
            digit=7;

```

```
        break;
    case '8':
        digit=8;
        break;
    case '9':
        digit=9;
        break;
    case '0':
        digit=10;
        break;
    case '*':
        digit=11;
        break;
    case '#':
        digit=12;
        break;
    case 'A':
        digit=13;
        break;
    case 'B':
        digit=14;
        break;
    case 'C':
        digit=15;
        break;
    case 'D':
        digit=16;
        break;
    default:
        digit=0;
        break;
    }
    return digit;
}
```

```
/*
  Name :   configMT8880C.h

  Header file for MT8880C.c, notice that the header file name sometime cannot same
  as the source file name.
*/

/*_____ P R O T O C O L _____*/
void MT8880C_Reset(void);
void MT8880C_SetForTransmit_tone(void);
void MT8880C_TransDTMF_tone(char ch);
int getToneDigit(char ch);
```

```

/*-----
Eeprom Accesser (AT24C16A)

Name   : EepromAccess.c
Purpose:
    Provide a interface for easier access the operation of the eeprom AT24C16A. Via these
    function, user can easily access the memory storage device. save/load a character; save/
    load a integer from Eeprom, etc
-----*/

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

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

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

void saveChar(unsigned char addr,unsigned char ch);
unsigned char loadChar(unsigned char addr);
void saveInt(unsigned char addr, int intData);
int loadInt(unsigned char addr);

/*_____FUNCTION USED FOR EASILY ASSESS THE MEMORY STORAGE AND LOADING FORM I2C DEVICE_____*/

/*
    save a char to the Eeprom in address specified by variable addr.
*/
void saveChar(unsigned char addr,unsigned char ch)
{
    write_Byte(addr, ch);
}

/*
    load a char from address indicated by variable addr.
*/
unsigned char loadChar(unsigned char addr)
{
    return read_Byte(addr);
}

/*
    save a integer to Eeprom, use 2 bytes of data from address indicated by variable
    addr. the intData will be divided into High byte and Low byte, then stored into 2
    bytes.
*/
void saveInt(unsigned char addr, int intData)
{
    unsigned char H,L,sign=0x00;
    if(intData<0){
        intData=-intData;
        sign = 0x01;
    }
    H=((intData&0xff00)/0x00ff);
    L=((intData&0x00ff));
    write_Byte(addr,sign);
    write_Byte(addr+1,L);
    write_Byte(addr+2,H);
}

/*
    load a integer from addr. Get 2 bytes of data then combine it to form a integer.
*/
int loadInt(unsigned char addr)
{
    unsigned int intData=0x0000;
    unsigned char H,L,T,sign;

    sign=read_Byte(addr);
    L=read_Byte(addr+1);
    H=read_Byte(addr+2);

    intData = (H*256) + L;

    /*
        // initialize avoid destruction
        // must unsinged for char & int, as H, L must a
        // positive value for intData. (extra byte is used
        // to indicate the sign of intData).
        // extract data

        // combine H byte & L byte
    */
}

```

```
    Bug elimination, reason still not know. Maybe due to overflow of calculation. When intData
    >=0x0100, it will less 256. eg. 0x3489-->0x3389, 0xedx66-->0xec66. 0x0045-->0x0045. Thus
    it is need to add 256 to intData, when intData>=0x0100.
*/
T=((intData&0xff00)/0x00ff);          // extract combined H byte
if(T<H)                                // when intData > =0x0100, add 256 to it.
    intData=(H*256) + L + 256;

if(sign==0x00)                        // 0x00 means positive
    return intData;
else                                   // 0x01 means negative
    return -intData;
}
```

```

/*-----*/
Name: advanced.c
Purpose:
provide the advanced function of the i2c eeprom. write byte, read byte, write page
and sequential read.(PS: function read_Byte is call by value, read_Page is call by
address)
/*-----*/

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

/*
write a byte to i2c devices.
*/
void write_Byte(unsigned char eepromAddr, wrData)
{
    sendStart();                // start
    sendSlaveAddress(SLAVE_ADDRESS); // 7 bit slave address
    sendBitIndicateWrite();      // a bit for write mode
    waitAck();
    sendByte(eepromAddr);        // eeprom address
    waitAck();
    sendByte(wrData);            // a data
    waitAck();
    sendStop();                 // byte written completed
}

/*
write a page to i2c devices, eepromAddr indicate the starting address for write
and eepromAddr + PAGE_SIZE is the ending address. the data for write is store in
array wrbuf[]. the array is pass by address that the caller determined.
*/
void write_Page(unsigned char eepromAddr, unsigned char wrbuf[PAGE_SIZE])
{
    unsigned int i;
    sendStart();                // start notation
    sendSlaveAddress(SLAVE_ADDRESS); // 7 bit slave address
    sendBitIndicateWrite();      // a bit for write mode
    waitAck();
    sendByte(eepromAddr);        // eeprom address
    waitAck();
    for(i=0;i<PAGE_SIZE;i++)    // send the data for write
    {
        sendByte(wrbuf[i]);      // some data
        waitAck();
    }
    sendStop();                 // page written completed
}

/*
read a byte in a particular address of eeprom memory, and return a readed value
to the caller.
*/
unsigned char read_Byte(unsigned char eepromAddr)
{
    unsigned char rcvdata=0x00;

    /* set the ptr of eeprom address */
    sendStart();                // set the eeprom location address
    sendSlaveAddress(SLAVE_ADDRESS); // 7 bit address for slave location
    sendBitIndicateWrite();      // bit indicated for write mode to
    waitAck();                  // set the memory address of eeprom
    sendByte(eepromAddr);
    waitAck();

    /* current address read */
    sendStart();
    sendSlaveAddress(SLAVE_ADDRESS);
    sendBitIndicateRead();
    waitAck();
    rcvdata=getByte();
    masterNoAck();
    sendStop();                 // read completed
    return rcvdata;
}

```

```

/*-----
Name: Eeprombasic.c
Purpose:
provide the basic level control for the combined advanced function. Eg, send a start
notation, stop notation, send a slave address, byte and acknowledgement.
-----*/

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

// a start notation for the i2c slave
void sendStart(void){
    int i;
    setSDA;
    setSCL;
    clrSDA;
    clrSCL;
    for(i=0;i<100;i++);          // delay for work
}

// a stop notation for the i2c slave
void sendStop(void){
    int i;
    clrSDA;
    setSCL;
    setSDA;
    clrSCL;
    for(i=0;i<100;i++);          // delay for work
}

// sends one byte of data to a i2c slave
void sendByte(unsigned char b){
    unsigned char mask;
    mask = 0x80;
    do{
        if ( b & mask ){          // send 8 bits
            setSDA;              // bit is high
        }
        else{
            clrSDA;              // bit is low
        }
        setSCL;                  // toggle a clock
        clrSCL;
        mask = mask/2;           // shift mask right
    }while (mask>0);
}

// gets one byte of data from i2c device
unsigned char getByte(void){
    int i;
    unsigned char rcvdata=0x00;
    unsigned char mask;          // variable for getting the reading data

    mask = 0x80;
    do{
        setSCL;                  // store 8 bit data
        for(i=0;i<50;i++);        // negative edge clock data out
        if (SDA==1)              // delay for work
            rcvdata |= mask;
        clrSCL;
        for(i=0;i<50;i++);        // delay for work
        mask = mask/2;
    }while (mask>0);
    return rcvdata;
}

// sends lower 7 bit of data to a i2c slave
void sendSlaveAddress(unsigned char b){
    unsigned char mask;
    mask = 0x80;
    b*=2;                        // shift b left
    do{
        if ( b & mask ){          // send 7 bits
            setSDA;              // bit is high
        }
        else{
            clrSDA;              // bit is low
        }
        setSCL;                  // toggle a clock
    }
}

```

```
        clrSCL;
        mask = mask/2;                // shift mask right
    }while (mask>1);
}

// wait until acknowledgment is received from slave
void waitAck(void){
    int i;
    for(i=0;i<50;i++);                // delay for work
    setSDA;                           // release SDA for acknowledge
    setSCL;                           // send clock for acknowledge
    while(SDA);
    clrSCL;
    for(i=0;i<50;i++);                // delay for work
}

// master acknowledge by sending a clr bit to slave
void masterAck(void){
    clrSDA;
    setSCL;
    clrSCL;
}

// master disacknowledge by sending a set bit to slave
void masterNoAck(void){
    setSDA;
    setSCL;
    clrSCL;
}

// a bit after the slave address, clr indicate i2c write
void sendBitIndicateWrite(void){
    clrSDA;
    setSCL;
    clrSCL;
}

// a bit after the slave address, set indicate i2c read
void sendBitIndicateRead(void){
    setSDA;
    setSCL;
    clrSCL;
}
```



```

/*-----
| Name: Eepromconfig.h
| Purpose:
| config the hardware setting, define the hardware control pin with a notation name
| select the require lib for the MPU, and define the protocol for the program access
|-----*/

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

sbit    SCL = P3^4;           // i2c clock pin
sbit    SDA = P3^3;           // i2c data pin

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

#define setSCL      SCL=1;while(SCL!=1);
#define clrSCL      SCL=0;
#define setSDA      SDA=1;
#define clrSDA      SDA=0;

#define PAGE_SIZE   7          // max no. of page size for read/ write
#define SLAVE_ADDRESS 0x50     // the address of the i2c slave

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

// EepromBasic.c
void sendStart(void);
void sendStop(void);
void sendByte(unsigned char);
unsigned char getByte(void);
void sendSlaveAddress(unsigned char b);
void waitAck(void);
void masterAck(void);
void masterNoAck(void);
void sendBitIndicateWrite(void);
void sendBitIndicateRead(void);

// EepromAdvanced.c
void write_Byte(unsigned char eepromAddr, eepromData);
void write_Page(unsigned char eepromAddr, unsigned char wrbuf[PAGE_SIZE]);
unsigned char read_Byte(unsigned char eepromAddr);

// EepromCheck.c
void eepromCheck(void);

// EepromAccess.c
void saveChar(unsigned char addr,unsigned char ch);
unsigned char loadChar(unsigned char addr);
void saveInt(unsigned char addr, int intData);
int loadInt(unsigned char addr);

```

```

/*-----+
|                                     |
|               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);
    hex8H = hex2ascii((hex8&0xf0)/0x0f);
    putchar(hex8H);
    putchar(hex8L);
}

/*
   print 16 bit hex data
*/
void printHex16(int hex16)
{
    char hex16H,hex16L;
    hex16H=((hex16&0xff00)/0x00ff);
    hex16L=((hex16&0x00ff));
    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)
    {
        return 0x46;
    }
    if (bits_data >=0x00 & bits_data <=0x09)
    {
        bits_data +=0x30;
        return bits_data;
    }
    if (bits_data >=0x0a & bits_data <=0x0f);
    {
        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[]);void printBit(bit b);

unsigned char hex2ascii(unsigned char bits_data);
```

```

/*-----+
|                                     |
|           Dittal Thermometer and Thermostat (DS1620)           |
| Name: DS1620.c                                                 |
| Purpose:                                                         |
| By Dillian Wong, last modify data 4/10/03                     |
+-----*/

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

#include <reg51.h>
#include <stdio.h>
#include "rPrintf\rprintf.h"

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

#define READ_TEMPERATURE      0xAA      // reads last converted temperature value
#define START_CONVERT_T      0xEE      // initates temperature conversion
#define STOP_CONVERT_T       0x22      // halts temperature conversion
#define WRITE_TH              0x01      // write high temperature limit
#define WRITE_TL              0x02      // write low temperature limit
#define READ_TH               0xA1      // read high temperature limit
#define READ_TL               0xA2      // read low temperature limit
#define WRITE_CONFIG          0x0C      // write configuration data
#define READ_CONFIG           0xAC      // read configuration data
#define CONTINUOUS            0x02      // set for continuous conversion
#define ONESHOT                0x01      // set for one temperature conversion
#define TEMPSTEP              0.5       // digit step of the temperature
#define T00LOW                0x00
#define GOOD                  0x01
#define T00HIGH               0x02

/*----- H A R D W A R E -----*/

sbit DQ      = P3^5;
sbit CLK     = P3^6;
sbit RST     = P3^7;
/* set or clr of important PIN */
#define setDQ      DQ=1;while (DQ!=1);          // data in
#define clrDQ      DQ=0;
#define setCLK     CLK=1;while (CLK!=1);          // clock
#define clrCLK     CLK=0;
#define setRST     RST=1;while (RST!=1);          // chip reset
#define clrRST     RST=0;

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

void ds1620_init(float TH, float TL);
void ds1620_sendByte(unsigned char dat);
unsigned char ds1620_readByte(void);
float ds1620_toDegree(unsigned char dat);
float ds1620_ReadTemperature(void);
float ds1620_ReadTH(void);
float ds1620_ReadTL(void);

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

/*
    initialize the digital thermometer with the High temperature trigger and Low
    temperature trigger.
*/
void ds1620_init(float TH, float TL)
{
    int numTH,numTL;          // no. step
    unsigned char thH,thL,tlH,tlL;

    /* convert the float input to digit step */
    numTH=TH/TEMPSTEP;        // the no. step of TH
    numTL=TL/TEMPSTEP;        // the no. step of TL

    // high temperature limit
    thH=((numTH&0xff00)/0x00ff); // high byte of TH
    if(!(TH>0))
    {
        thH = 0x01;
    }
    thL=((numTH&0x00ff));      // low byte of TH

```

```

    // low temperature limit
    tlH=((numTL&0xff00)/0x00ff);          // high byte of TH
    if(!(TL>0))
    {
        tlH = 0x01;
    }
    tLL=((numTL&0x00ff));                // low byte of TH

/*
printf("Digital Code is : ");
printHex8(thH);
printHex8(thL);
printHex8(tlH);
printHex8(tlL);
*/

/* start the configuration */
ds1620_sendByte(WRITE_CONFIG);
ds1620_sendByte(CONTINUOUS);
clrRST;
setRST;
ds1620_sendByte(WRITE_TH);              // set TH
ds1620_sendByte(thL);
ds1620_sendByte(thH);
clrRST;
setRST;
ds1620_sendByte(WRITE_TL);              // set TL
ds1620_sendByte(tlL);
ds1620_sendByte(tlH);
clrRST;
setRST;
ds1620_sendByte(START_CONVERT_T);        // issues start convert T command
}

/*
send 8 bit data to the digital thermometer.
*/
void ds1620_sendByte(unsigned char dat)
{
    unsigned char i,mask=1;
    setRST;
    for (i=0; i<8; i++)
    {
        clrCLK;
        DQ = (dat & mask);                // send bit to 1620
        setCLK;
        mask=(mask<<1);                  // setup to send next bit
    }
}

/*
get 8 bit data from the digital thermometer.
*/
unsigned char ds1620_readByte(void)
{
    unsigned char i,rcv=0,mask=1;
    for (i=0; i<8; i++)
    {
        clrCLK;
        if (DQ) rcv|=mask;                // read bit and or if = 1
        setCLK;
        mask=(mask<<1);                  // setup for next read and or
    }
    return rcv;
}

/*
the binary data represend a interger step. For 0x01, it may represent 5, for 0x03,
it may represnet 15. So. this function is for the conversation of the binary data
to the refered interger.
*/
float ds1620_toDegree(unsigned char dat)
{
    float fdeg=0;
    unsigned int i;

    // only for positive data, if dat is negative, the 2 complement will required.
    for(i=0;i<dat;i++)
    {

```

```
fdeg +=0.5;
}

return fdeg;
}

/*
issues Ds1620 measures the current temperature and then read data from the
thermometer,convert digit ouputs and return the decimal interger of temperature.
*/
float ds1620_ReadTemperature(void)
{
    unsigned char datH,datL;
    float deg;

    clrRST;
    setRST;
    ds1620_sendByte(READ_TEMPERATURE);           // read temperature T
    datL = ds1620_readByte();
    datH = ds1620_readByte();

    if(datH&0x01)
    {
        deg = -ds1620_toDegree(~datL+1 );        // if the temperature is negative
                                                // take complement
    }
    else
    {
        deg = ds1620_toDegree(datL);              // if the temperature is positive
    }

    return deg;                                  // return T in interger form
}

/*
reads stored value of high temperature limit from TH register.
*/
float ds1620_ReadTH()
{
    unsigned char datH,datL;
    float deg;

    clrRST;
    setRST;
    ds1620_sendByte(READ_TH);                    // read TH
    datL = ds1620_readByte();
    datH = ds1620_readByte();

    if(datH&0x01)
    {
        deg = -ds1620_toDegree(~datL+1 );        // if the temperature is negative
                                                // take complement
    }
    else
    {
        deg = ds1620_toDegree(datL);              // if the temperature is positive
    }

    return deg;                                  // return T in interger form
}

/*
reads stored value of low temperature limit from TL register.
*/
float ds1620_ReadTL()
{
    unsigned char datH,datL;
    float deg;

    clrRST;
    setRST;
    ds1620_sendByte(READ_TL);                    // read TH
    datL = ds1620_readByte();
    datH = ds1620_readByte();

    if(datH&0x01)
    {
        deg = -ds1620_toDegree(~datL+1 );        // if the temperature is negative
                                                // take complement
    }
    else
    {

```

```
    deg = ds1620_toDegree(datL);          // if the temperature is positive
}
return deg;                             // return T in interger form
}
```

/*

 Name : DS1620.h

 Header file for DS1620.c
*/

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

```
void ds1620_init(float TH, float TL);  
float ds1620_ReadTemperature(void);  
float ds1620_ReadTH(void);  
float ds1620_ReadTL(void);
```