TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA ĐIỆN ĐIỆN TỬ

HCMUTE

BÁO CÁO THỰC TẬP

KIẾN TRÚC VÀ TỔ CHỨC MÁY TÍNH

MÃ MÔN HỌC & MÃ LỚP:

COOL325364_09

SINH VIÊN THỰC HIỆN:

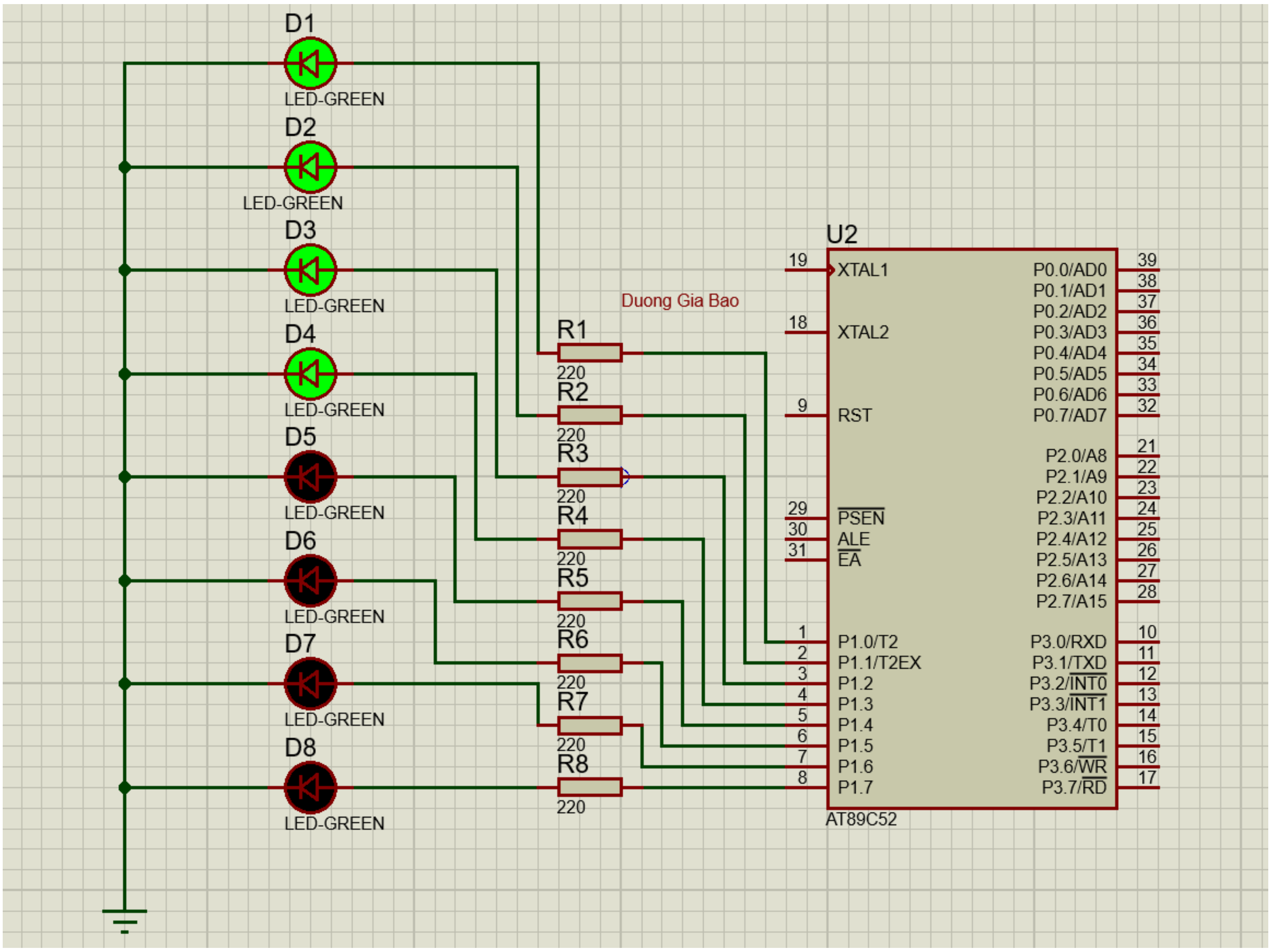
DƯƠNG GIA BẢO - 22119166

HỌC KỲ: 1 – NĂM HỌC: 2024 – 2025

Tp HCM, tháng 11 - 2024

MỤC LỤC

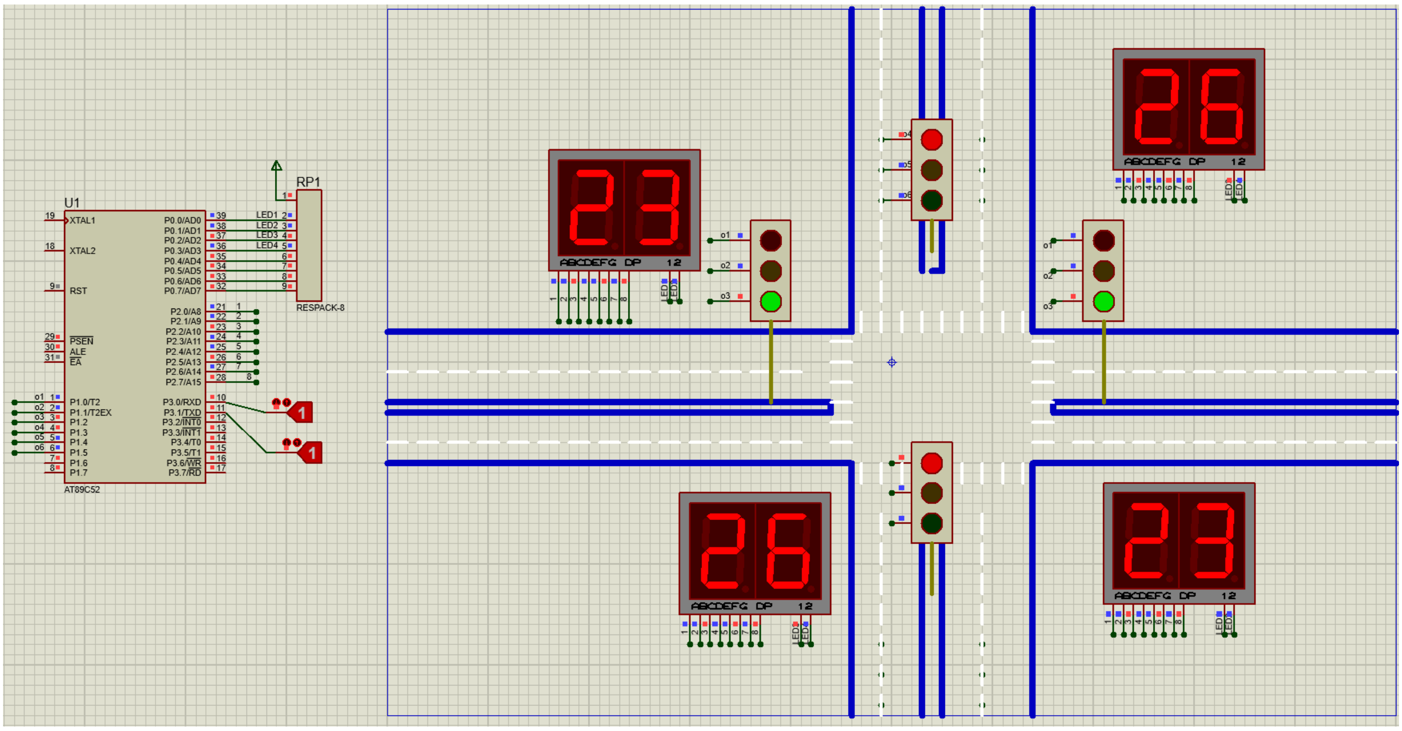
BÀI 1. ĐIỀU KHIỂN LED



Main.c

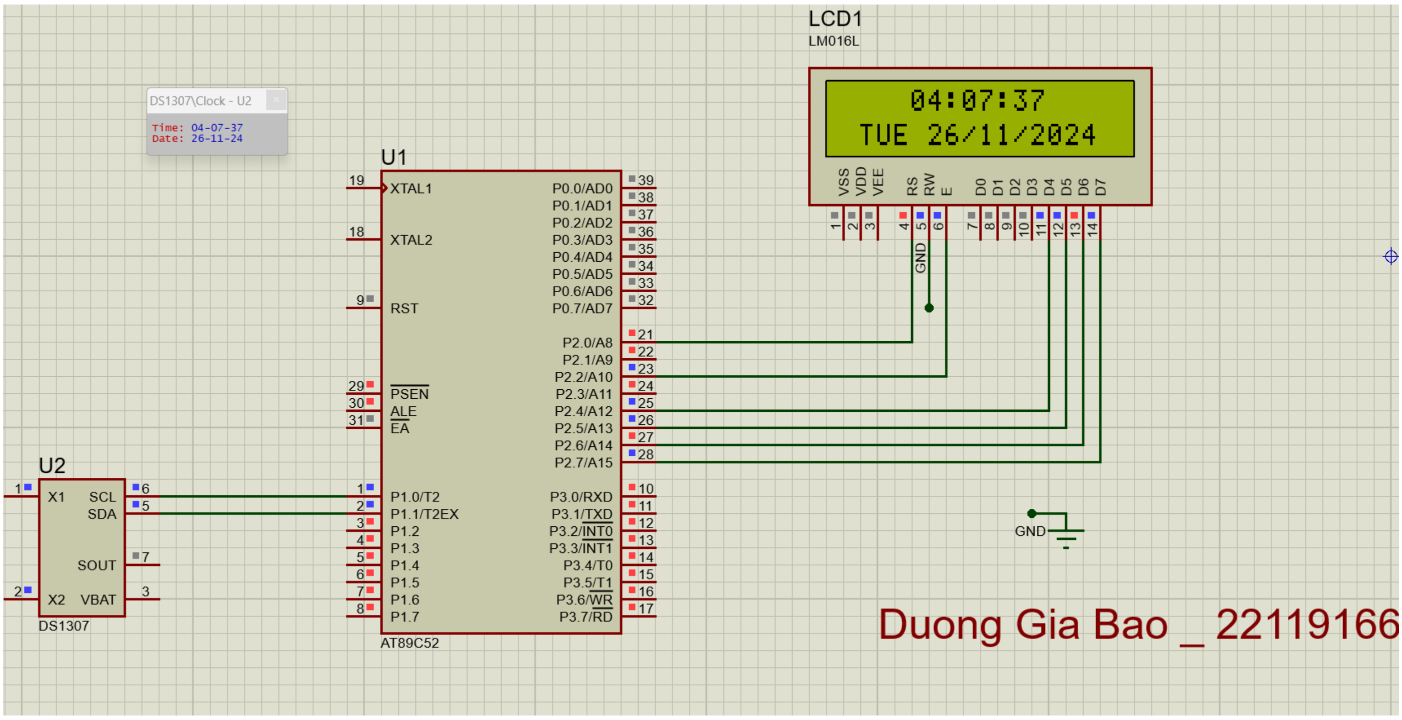
```
#include <REGX51.H>

int denbat[9] = {0X00, 0X01, 0X03, 0X07, 0X0F, 0X1F, 0X3F, 0X7F, 0XFF};
int dentat[9] =  {0XFF, 0XFE, 0XFC, 0XF8, 0XF0, 0XE0, 0XC0, 0X80, 0X00};

void delay(long time)
{
    time = time * 30;
    while(time--) {;}
}

int i;

void main()
{
    P1=0X00;
    while(1)
    {
        for(i=0; i<8; i++)
        {
            P1 = denbat[i];
            delay(300);
        }
        for(i=8; i>0; i--)
        {
            P1 = dentat[8-i];
            delay(200);
        }
    }
}
```

BÀI 2. ĐÈN GIAO THÔNG _ TRÊN MÔ PHỎNG



```
#include <AT89X52.h>

sbit SW_chedo = P3 ^ 2; /*chan nhan tin hieu chuyen che do*/
sbit SW = P3 ^ 3; /*chan nhan tin hieu chuyen huong*/

#define Display P0 /*xuat du lieu led 7 doan*/
#define chonLED P2 /*P2.4,P2.3,P2.2 dieukhien 8 led 7 doan*/

int chedo = 1;
int chuyenhuong = 1;
int chuyenVang = 0;

/*du lien hien thi so*/
unsigned char code Code7segCatot[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d,
                                      0x7d, 0x07, 0x7f, 0x6f, 0x77};
/*du lieu trang thai den*/
unsigned char code StatusLED[] = {0x81, 0x82, 0x24, 0x44};

/*du lieu chan chon hien thi led*/
unsigned char code BNsochay[] = {0x00, 0x10, 0x04, 0x14},
              DTsochay[] = {0x10, 0x00, 0x14, 0x04};

void delay_ms(unsigned int time) {
  while (time--) {
    TMOD = 0x01;
    TH1 = 0xE8;
    TL1 = 0x90;
    TR1 = 1;
    while (!TF1)
      ;
    TF1 = 0;
  }
}

void HienThiDen(char Status) {
  int i;
  P3_6 = 0; /*clock canh len*/
  P3_5 = 0;/*chan chot du lieu*/
  for (i = 0; i < 8; i++) {
    P3_4 = Status >> 7;
    Status <<= 1;
    delay_ms(1);
    P3_6 = 1;
    delay_ms(1);
    P3_6 = 0;
  }
  P3_5 = 1;
}

void HienThiSoChay(char *Huong, int demA, int demB) {
  int tanso;
  int dem;
  for (dem = demA; dem >= demB; dem--) {
    if (chedo == 1) {
      return;
    }
    for (tanso = 0; tanso < 46; tanso++) {
      chonLED = Huong[0];
      Display = Code7segCatot[dem % 10];
      delay_ms(5);
      chonLED = Huong[1];
      Display = Code7segCatot[(dem - 3) % 10];
      delay_ms(5);
      chonLED = Huong[2];
      Display = Code7segCatot[dem / 10];
      delay_ms(5);
      chonLED = Huong[3];
      Display = Code7segCatot[(dem - 3) / 10];
      delay_ms(5);
      /* Clear LED Matrix */
      Display = 0x00;
      delay_ms(5);
    }
  }
}

void HienThiSoVang(char *Huong, int demA, int demB) {
  int tanso;
  int dem;
  for (dem = demA; dem >= demB; dem--) {
    if (chedo == 1) {
      return;
    }
    for (tanso = 0; tanso < 46; tanso++) {
      chonLED = Huong[0];
      Display = Code7segCatot[dem % 10];
      delay_ms(5);
      chonLED = Huong[1];
      Display = Code7segCatot[dem % 10];
      delay_ms(5);
      chonLED = Huong[2];
      Display = Code7segCatot[dem / 10];
      delay_ms(5);
      chonLED = Huong[3];
      Display = Code7segCatot[dem / 10];
      delay_ms(5);
      /* Clear LED Matrix */
      Display = 0x00;
      delay_ms(5);
    }
  }
}

void Traffic_Auto(void) {
  HienThiDen(StatusLED[0]);
  HienThiSoChay(BNsochay, 10, 3);
  HienThiDen(StatusLED[1]);
  HienThiSoVang(BNsochay, 2, 0);
  HienThiDen(StatusLED[2]);
  HienThiSoChay(DTsochay, 10, 3);
  HienThiDen(StatusLED[3]);
  HienThiSoVang(DTsochay, 2, 0);
}

void Interrupt_Timer0(void) {
  TMOD = 0x01;
  TH0 = 0xFC;
  TL0 = 0x18;
  ET0 = 1;
  EA = 1;
  TR0 = 1;
}

void ISR_TIMER(void) interrupt 1 {
  if (ET0 == 1) {
    if (SW_chedo == 0) {
      delay_ms(100);
      if (SW_chedo == 1) {
        chedo++;
      }
      if (chedo > 1)
        chedo = 0;
    }
    TH0 = 0xFC;
    TL0 = 0x18;
  }
}

void main(void) {
  Interrupt_Timer0();
  while (1) {
    if (chedo) {
      P0 = 0x00;
      if (SW == 0) {
        delay_ms(100);
        if (SW == 1) {
          chuyenhuong++;
        }
        if (chuyenhuong > 1)
          chuyenhuong = 0;
      }
      if (chuyenhuong) {
        if(chuyenVang) {
          HienThiDen(StatusLED[3]);
          delay_ms(2000);
        }
        HienThiDen(StatusLED[0]);
        chuyenVang = 0;
      } else {
        if(!chuyenVang) {
          HienThiDen(StatusLED[1]);
          delay_ms(2000);
        }
        HienThiDen(StatusLED[2]);
        chuyenVang = 1;
      }
    } else {
      Traffic_Auto();
    }
  }
}
```

BÀI 3. LỊCH VẠN NIÊN _ TRÊN MÔ PHỎNG



Main.c

```
#include"main.h"
#include"..\lib\Soft_I2c.h"
#include"..\lib\Lcd4.h"
#include"..\lib\Rtc_Ds1307.h"
#include"..\lib\LunarCalendar.h"
#include"Port.h"

unsigned char * code Days[] = {"SUN","MON","TUE","WED","THU","FRI","SAT"};

void main()
{
    unsigned char GIO, PHUT, GIAY, Mode, Day, Date, Month, Year, old_GIAY;
    unsigned char SolarDate, SolarMonth;
    char SolarYear;

    Soft_I2c_Init();
    Ds1307_Init();
    Lcd_Init();

    while(1)
    {
        Ds1307_Read_Time(&GIO, &PHUT, &GIAY, &Mode);
        if(old_GIAY != GIAY)
        {
            old_GIAY = GIAY;

            Lcd_Chr(1,5,GIO/10+0x30);
            Lcd_Chr_Cp(GIO%10+0x30);
            Lcd_Chr_Cp(':');
            Lcd_Chr_Cp(PHUT/10+0x30);
            Lcd_Chr_Cp(PHUT%10+0x30);
            Lcd_Chr_Cp(':');
            Lcd_Chr_Cp(GIAY/10+0x30);
            Lcd_Chr_Cp(GIAY%10+0x30);

            Ds1307_Read_Date(&Day, &Date, &Month, &Year);
            if(BTN == 1)
            {
                Lcd_Out(2,1,"                ");
                Lcd_Out(2,2,Days[Day-1]);
                Lcd_Chr_Cp(' ');
                Lcd_Chr_Cp(Date/10+0x30);
                Lcd_Chr_Cp(Date%10+0x30);
                Lcd_Chr_Cp('/');
                Lcd_Chr_Cp(Month/10+0x30);
                Lcd_Chr_Cp(Month%10+0x30);
                Lcd_Out_Cp("/20");
                Lcd_Chr_Cp(Year/10+0x30);
                Lcd_Chr_Cp(Year%10+0x30);
            }
            else
            {
                Solar2Lunar(Date, Month, Year, &SolarDate, &SolarMonth, &
SolarYear);
                Lcd_Out(2,1,"LUNAR:");
                Lcd_Chr_Cp(SolarDate/10+0x30);
                Lcd_Chr_Cp(SolarDate%10+0x30);
                Lcd_Chr_Cp('/');
                Lcd_Chr_Cp(SolarMonth/10+0x30);
                Lcd_Chr_Cp(SolarMonth%10+0x30);
                Lcd_Chr_Cp('/');
                Lcd_Chr_Cp((Year+2000)/1000+0x30);
                Lcd_Chr_Cp((Year+2000)/100%10+0x30);
                Lcd_Chr_Cp((Year+2000)/10%10+0x30);
                Lcd_Chr_Cp((Year+2000)%10+0x30);
            }
        }
    }
}
```

Soft_I2C.c

```
#include"main.h"
#include"port.h"
#include"Soft_I2c.h"
#include"intrins.h"

#ifdef  USE_I2CDELAY
    #define I2CDELAY()  {_nop_();_nop_();_nop_();_nop_();_nop_();}
```

```
#else
    #define I2CDELAY()
#endif

bit Soft_I2c_Get_Ack();
void Soft_I2c_Ack();
void Soft_I2c_Nak();

void Soft_I2c_Init()
{
    SOFT_I2C_SCL=1;
    SOFT_I2C_SDA=1;
}

void Soft_I2c_Start()
{
    SOFT_I2C_SCL = 1;
    I2CDELAY();
    SOFT_I2C_SDA = 0;
    I2CDELAY();
    SOFT_I2C_SCL = 0;
}

bit Soft_I2c_Get_Ack()
{
    bit result;
    SOFT_I2C_SDA = 1;
    I2CDELAY();
    SOFT_I2C_SCL = 1;
    I2CDELAY();
    result = SOFT_I2C_SDA;
    SOFT_I2C_SCL = 0;
    return result;
}

bit Soft_I2c_Write(unsigned char dat)
{
    unsigned char i;
    for(i=0;i<8;i++)
    {
        SOFT_I2C_SDA = (bit)(dat&0x80);
        SOFT_I2C_SCL = 1;
        I2CDELAY();
        SOFT_I2C_SCL = 0;
        dat<<=1;
    }
    return(Soft_I2c_Get_Ack());
}

void Soft_I2c_Ack()
```

```
{
    SOFT_I2C_SDA = 0;
    I2CDELAY();
    SOFT_I2C_SCL = 1;
    I2CDELAY();
    SOFT_I2C_SCL = 0;
}

void Soft_I2c_Nak()
{
    SOFT_I2C_SDA = 1;
    I2CDELAY();
    SOFT_I2C_SCL = 1;
    I2CDELAY();
    SOFT_I2C_SCL = 0;
}

unsigned char Soft_I2c_Read(bit ack)
{
    unsigned char i, dat=0;
    for(i=0;i<8;i++)
    {
        SOFT_I2C_SDA = 1;
        I2CDELAY();
        SOFT_I2C_SCL = 1;
        I2CDELAY();
        dat <<= 1;
        if(SOFT_I2C_SDA)
        {
            dat |= 0x01;
        }
        SOFT_I2C_SCL = 0;
    }
    if(ack)
    {
        Soft_I2c_Ack();
    }
    else
    {
        Soft_I2c_Nak();
    }
    return dat;
}

void Soft_I2c_Stop()
{
    SOFT_I2C_SDA = 0;
    I2CDELAY();
    SOFT_I2C_SCL = 1;
    I2CDELAY();
    SOFT_I2C_SDA = 1;
}
```

LCD4.h

```
#include"Main.h"
#include"Port.h"
#include"LCD4.h"
#include"Delay.h"
#include"String.h"

#ifdef  CHECKBUSY
    #message "Lcd - Use check busy method."
    #ifndef LC_RW
        #error "Lcd - Define LC_RW, please."
    #endif
#else
    #message "Lcd - Use delay method."
#endif

void Lcd_Write_High_Nibble(unsigned char);
void Lcd_Write_Low_Nibble(unsigned char );
void Lcd_Delay_us(unsigned char);
#ifdef  CHECKBUSY
void Lcd_Busy();
#endif

void Lcd_Write_High_Nibble(unsigned char b)
{
    LCD_D7 = b & 0x80;
    LCD_D6 = b & 0x40;
    LCD_D5 = b & 0x20;
    LCD_D4 = b & 0x10;
}

void Lcd_Write_Low_Nibble(unsigned char b)
{
    LCD_D7 = b & 0x08;
    LCD_D6 = b & 0x04;
    LCD_D5 = b & 0x02;
    LCD_D4 = b & 0x01;
}

void Lcd_Delay_us(unsigned char t)
{
    while(t--);
}

#ifdef  CHECKBUSY
void Lcd_Busy()
{
    bit busy_flag;
    LCD_D7 = 1;
    LCD_RS = 0;
    LCD_RW = 1;
    do{
        LCD_EN = 0;
        LCD_EN = 1;
        busy_flag = LCD_D7;
        LCD_EN = 0;
        LCD_EN = 1;
    }while(busy_flag);
    LCD_EN = 0;
}
#endif

void Lcd_Init()
{
    LCD_RS = 0;
    LCD_EN = 0;
#ifdef  LC_RW
    LCD_RW = 0;
#endif

    Delay_ms(20);

    Lcd_Write_Low_Nibble(0x03);
    LCD_EN = 1;
    LCD_EN = 0;
    Delay_ms(5);

    Lcd_Write_Low_Nibble(0x03);
    LCD_EN = 1;
    LCD_EN = 0;
    Lcd_Delay_us(100);

    Lcd_Write_Low_Nibble(0x03);
    LCD_EN = 1;
    LCD_EN = 0;
#ifdef  CHECKBUSY
    Lcd_Busy();
#else
    Delay_ms(1);
#endif

    Lcd_Write_Low_Nibble(0x02);
    LCD_EN = 1;
    LCD_EN = 0;
    Delay_ms(1);

    Lcd_Cmd(_LCD_4BIT_2LINE_5x7FONT);
    Lcd_Cmd(_LCD_TURN_ON);
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Cmd(_LCD_ENTRY_MODE);
}

void Lcd_Cmd(unsigned char cmd)
{
#ifdef  LC_RW
    LCD_RW = 0;
#endif
    LCD_RS = 0;
    Lcd_Write_High_Nibble(cmd);
    LCD_EN = 1;
    LCD_EN = 0;

    Lcd_Write_Low_Nibble(cmd);
    LCD_EN = 1;
    LCD_EN = 0;

#ifdef  CHECKBUSY
    Lcd_Busy();
#else
    switch(cmd)
    {
        case _LCD_CLEAR:
        case _LCD_RETURN_HOME:
            Delay_ms(2);
            break;
        default:
            Lcd_Delay_us(37);
            break;
    }
#endif
}

void Lcd_Chr_Cp(unsigned char achar)
{
#ifdef  LC_RW
    LCD_RW = 0;
#endif
    LCD_RS = 1;
    Lcd_Write_High_Nibble(achar);
    LCD_EN = 1;
    LCD_EN = 0;

    Lcd_Write_Low_Nibble(achar);
    LCD_EN = 1;
    LCD_EN = 0;

#ifdef  CHECKBUSY
    Lcd_Busy();
#else
```

```
    Lcd_Delay_us(37+4);
#endif
}

void Lcd_Chr(unsigned char row, unsigned char column,
    unsigned char out_char)
{
    unsigned char add;
    add = (row==1?0x80:0xC0);
    add += (column - 1);
    Lcd_Cmd(add);
    Lcd_Chr_Cp(out_char);
}

void Lcd_Out_Cp(unsigned char * str)
{
    unsigned char i = 0;
    while(str[i])
    {
        Lcd_Chr_Cp(str[i]);
        i++;
    }
}

void Lcd_Out(unsigned char row, unsigned char column,
    unsigned char* text)
{
    unsigned char add;
    add = (row==1?0x80:0xC0);
    add += (column - 1);
    Lcd_Cmd(add);
    Lcd_Out_Cp(text);
}

void Lcd_Custom_Chr(unsigned char location, unsigned char * lcd_char)
{
    unsigned char i;
    Lcd_Cmd(0x40+location*8);
    for (i = 0; i<=7; i++)
        Lcd_Chr_Cp(lcd_char[i]);
}
```

Rtc_Ds1307.c

```
#include"Main.h"
#include"Port.h"
#include"Soft_I2C.h"
#include"Rtc_Ds1307.h"

void Ds1307_Init()
{
    unsigned char tmp;
    tmp = Ds1307_Read(0x00);
    tmp &= 0x7F;
    Ds1307_Write(0x00,tmp);
}

void Ds1307_Write(unsigned char add, unsigned char dat)
{
    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(add);
    Soft_I2c_Write(dat);
    Soft_I2c_Stop();
}

unsigned char Ds1307_Read(unsigned char add)
{
    unsigned char dat;
    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(add);
    Soft_I2c_Start();
    Soft_I2c_Write(0xD1);
    dat = Soft_I2c_Read(0);
    Soft_I2c_Stop();
    return dat;
}

bit Ds1307_Read_Time(unsigned char * hour, unsigned char * minute,
    unsigned char * second, unsigned char * mode)
{
    unsigned char h_tmp, m_tmp, s_tmp;
    bit am_pm;
    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(0x00);
    Soft_I2c_Start();
    Soft_I2c_Write(0xD1);
    s_tmp = Soft_I2c_Read(1);
    m_tmp = Soft_I2c_Read(1);
    h_tmp = Soft_I2c_Read(0);
    Soft_I2c_Stop();

    s_tmp &= 0x7F;
    *second = (s_tmp>>4)*10+(s_tmp&0x0F);
    m_tmp &= 0x7F;
    *minute = (m_tmp>>4)*10+(m_tmp&0x0F);

    if(h_tmp & 0x40)        // Mode 12h
    {
        *mode = 12;
        if(h_tmp & 0x20)
        {
            am_pm = 1;      // PM
        }
        else
        {
            am_pm = 0;
        }
        h_tmp &= 0x1F;
        *hour = (h_tmp>>4)*10+(h_tmp&0x0F);
    }
    else
    {
        *mode = 24;
        h_tmp &= 0x3F;
        *hour = (h_tmp>>4)*10+(h_tmp&0x0F);
        if(*hour<12)
        {
            am_pm = 0;      // AM
        }
        else
        {
            am_pm = 1;
        }
    }
    return am_pm;
}

void Ds1307_Write_Time(unsigned char hour, unsigned minute,
    unsigned char second, unsigned char mode, bit apm)
{
    second = ((second/10)<<4)|(second%10);
    minute = ((minute/10)<<4)|(minute%10);
    hour   = ((hour  /10)<<4)|(hour  %10);
    if(mode==12)
    {
        hour |= 0x40;
        if(apm)     // PM
        {
            hour |= 0x20;
        }
    }
    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(0x00);
    Soft_I2c_Write(second);
    Soft_I2c_Write(minute);
    Soft_I2c_Write(hour);
    Soft_I2c_Stop();
}
```

```
void Ds1307_Read_Date(unsigned char * day, unsigned char * date,
    unsigned char * month, unsigned char * year)
{
    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(0x03);
    Soft_I2c_Start();
    Soft_I2c_Write(0xD1);
    *day  = Soft_I2c_Read(1);
    *date = Soft_I2c_Read(1);
    *month= Soft_I2c_Read(1);
    *year = Soft_I2c_Read(0);
    Soft_I2c_Stop();
    *day &= 0x07;
    *date &= 0x3F;
    *date = (*date>>4)*10 + (*date & 0x0F);
    *month &= 0x1F;
    *month = (*month>>4)*10 + (*month & 0x0F);
    *year = (*year>>4)*10 + (*year & 0x0F);
}

void Ds1307_Write_Date(unsigned char day, unsigned char date,
    unsigned char month, unsigned char year)
{
    date    = ((date/10)<<4)  | (date%10);
    month   = ((month/10)<<4) | (month%10);
    year    = ((year/10)<<4)  | (year%10);

    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(0x03);
    Soft_I2c_Write(day);
    Soft_I2c_Write(date);
    Soft_I2c_Write(month);
    Soft_I2c_Write(year);
    Soft_I2c_Stop();
}

void Ds1307_Write_Bytes(unsigned char add, unsigned char * buff,
    unsigned char len)
{
    unsigned char i=0;

    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(add);
    for(i=0;i<len;i++)
    {
        Soft_I2c_Write(buff[i]);
    }
    Soft_I2c_Stop();
}

void Ds1307_Read_Bytes(unsigned char add,unsigned char * buff,
    unsigned char len)
{
    unsigned char i;

    Soft_I2c_Start();
    Soft_I2c_Write(0xD0);
    Soft_I2c_Write(add);
    Soft_I2c_Start();
    Soft_I2c_Write(0xD1);
    for(i=0;i<len-1;i++)
    {
        buff[i]  = Soft_I2c_Read(1);
    }
    buff[i]  = Soft_I2c_Read(0);
    Soft_I2c_Stop();
}
```

LunarCalendar.h

```
#ifndef _LUNARCALENDAR_H_
#define _LUNARCALENDAR_H_

struct MONTH_INFO{
    unsigned int N_AL_DT_DL     :5;
    unsigned int T_AL_DT_DL     :4;
    unsigned int SN_CT_AL       :1;
    unsigned int TN_B_THT       :1;
    unsigned int SN_CT_DL       :2;
};

union LUNAR_RECORD
{
    unsigned int Word;
    struct MONTH_INFO Info;
};

void Solar2Lunar(unsigned char SolarDate, unsigned char SolarMonth,
unsigned char SolarYear,
    unsigned char * LunarDate, unsigned char * LunarMonth, char *
LunarYear);

#endif
```

Port.h

```
#ifndef _PORT_H_
#define _PORT_H_

sbit LCD_RS = P2^0;
sbit LCD_EN = P2^2;
sbit LCD_D4 = P2^4;
sbit LCD_D5 = P2^5;
sbit LCD_D6 = P2^6;
sbit LCD_D7 = P2^7;

sbit SOFT_I2C_SCL   = P1^0;
sbit SOFT_I2C_SDA   = P1^1;

sbit BTN = P3^0;

#endif
```

BÀI 4. ĐÈN GIAO THÔNG

Main.h

```
#include <REGX52.h>
#include <stdio.h>

#define elif else if
#define DECREASE_ONE(VAR) VAR = (VAR > 0 ? VAR - 1 : VAR)
#define RED 0x1
#define YELLOW 0x2
#define GREEN 0x4
#define LED_OFF 0xA
#define MANUAL 0x0
#define AUTO 0x1
#define R_DIGIT 0xB
#define Y_DIGIT 0xC
#define G_DIGIT 0xD

typedef unsigned int UINT;

static void DELAY_DISP(UINT mili_sec) { for (UINT i = 0; i < 3 * mili_sec;
i++); }
static void DELAY(UINT mili_sec) { for (UINT i = 0; i < 12 * mili_sec; i++); }

UINT RED0, YELLOW0, GREEN0, RED1, YELLOW1, GREEN1;
sbit GND0 = P2 ^ 2, GND1 = P2 ^ 3, GND2 = P2 ^ 4;
#define LED P0
sbit M_A = P3 ^ 3, R_G = P3 ^ 2;

UINT STATE_0 = RED, STATE_1 = RED, COUNT_0, COUNT_1;
UINT SINGLE_LED_DISPLAY_T = 1, RED_T = 0, GREEN_T = 0, YELLOW_T = 0;

const UINT DIGIT_CODE[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
0x7F, 0xEF, 0x0, 0x1, 0x40, 0x8};

UINT AUTO_MANUAL() { return M_A ? AUTO : MANUAL; }
UINT RED_GREEN() { return R_G ? RED : GREEN; }
void SET_LED(UINT D) { LED = DIGIT_CODE[D]; }
void SET_DISPLAY_PERIOD(UINT T) { SINGLE_LED_DISPLAY_T = T; }
void STOP_COUNT() { COUNT_0 = COUNT_1 = 0; SET_LED(LED_OFF); }
void SET_YELLOW_TIMER(UINT _YELLOW_T) { YELLOW_T = _YELLOW_T; }
void SET_RED_GREEN_TIMER(UINT _RED_T) { COUNT_0 = RED_T = _RED_T; COUNT_1 =
GREEN_T = RED_T - YELLOW_T; }
void SET_TIMER(UINT PREVIOUS) { COUNT_0 = PREVIOUS ? GREEN_T : RED_T; COUNT_1
= PREVIOUS ? RED_T : GREEN_T; }

UINT DIGIT(UINT POS) {
    return (POS ? (RED1 ? R_DIGIT : YELLOW1 ? Y_DIGIT : G_DIGIT) : (RED0 ?
R_DIGIT : YELLOW0 ? Y_DIGIT : G_DIGIT));
}
```

```
void DISPLAY_LED() {
    for (UINT i = 0; i < 7200 / (6 * SINGLE_LED_DISPLAY_T); i++) {
        UINT segments[] = {COUNT_0 / 10 % 10, COUNT_1 / 10 % 10, DIGIT(0),
COUNT_0 % 10, COUNT_1 % 10, DIGIT(1)};
        UINT gnds[] = {1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1};
        for (UINT j = 0; j < 6; j++) {
            GND0 = gnds[j * 2], GND1 = gnds[j * 2 + 1], GND2 = (j & 1);
            SET_LED(COUNT_0 && j != 2 && j != 5 ? segments[j] : LED_OFF);
            DELAY_DISP(SINGLE_LED_DISPLAY_T);
        }
    }
}

void SET_TRAFFIC_LIGHT(UINT POS, UINT CODE) {
    if (POS) CODE <<= 3;
    if (CODE & 0x01) RED0 = 1, YELLOW0 = 0, GREEN0 = 0, STATE_0 = RED;
    elif (CODE & 0x02) RED0 = 0, YELLOW0 = 1, GREEN0 = 0, STATE_0 = YELLOW;
    elif (CODE & 0x04) RED0 = 0, YELLOW0 = 0, GREEN0 = 1, STATE_0 = GREEN;
    elif (CODE & 0x08) RED1 = 1, YELLOW1 = 0, GREEN1 = 0, STATE_1 = RED;
    elif (CODE & 0x10) RED1 = 0, YELLOW1 = 1, GREEN1 = 0, STATE_1 = YELLOW;
    elif (CODE & 0x20) RED1 = 0, YELLOW1 = 0, GREEN1 = 1, STATE_1 = GREEN;
}

UINT GET_STATE(UINT POS) { return POS ? STATE_1 : STATE_0; }

void SET_STATE(UINT CODE) {
    while ((CODE == RED && GET_STATE(0) == GREEN) || (CODE == GREEN &&
GET_STATE(0) == RED)) {
        while (COUNT_0 || COUNT_1) DISPLAY_LED(), DECREASE_ONE(COUNT_0),
DECREASE_ONE(COUNT_1);
        SET_TRAFFIC_LIGHT(0, CODE == RED ? YELLOW : GREEN);
        SET_TRAFFIC_LIGHT(1, CODE == RED ? RED : YELLOW);
        COUNT_0 = COUNT_1 = YELLOW_T;
        while (COUNT_0 || COUNT_1) DISPLAY_LED(), DECREASE_ONE(COUNT_0),
DECREASE_ONE(COUNT_1);
        SET_TRAFFIC_LIGHT(0, CODE);
        SET_TRAFFIC_LIGHT(1, CODE == RED ? GREEN : RED);
    }
}

void INITIAL() {
    SET_TRAFFIC_LIGHT(0, YELLOW);
    SET_TRAFFIC_LIGHT(1, YELLOW);
    COUNT_0 = COUNT_1 = 0;
    LED = DIGIT_CODE[LED_OFF];
    GND0 = GND1 = GND2 = 0;
}
```

Main.c

```c
#include "main.h"

void main() {
    INITIAL();
    SET_YELLOW_TIMER(5);
    SET_RED_GREEN_TIMER(17);
    SET_DISPLAY_PERIOD(12);

    while (1) {
        if (!AUTO_MANUAL()) {
            for (UINT i = 0; i < 2; ++i) {
                SET_STATE(i ? GREEN : RED);
                SET_TIMER(i);
            }
        } else {
            STOP_COUNT();
            while (!AUTO_MANUAL()) {
                SET_STATE(RED_GREEN());
                for (UINT i = 0; i < 2; ++i) {
                    GND0 = 1; GND1 = 1; GND2 = i;
                    SET_LED(DIGIT(i));
                    DELAY(SINGLE_LED_DISPLAY_T);
                }
            }
        }
    }
}
```

BÀI 5. LỊCH VẠN NIÊN

base_lib.h

```
#ifndef _BASE_LIB_H_
#define _BASE_LIB_H_

#ifndef elif
#define elif else if
#endif
#ifndef DECREASE_ONE
#define DECREASE_ONE(VAR) VAR = (VAR>0?(VAR-1):VAR)
#endif
#ifndef FOR
#define FOR(i, a, b) for(i = (a); i <= (b); ++i)//rep
#endif
#ifndef FOR_reverse
#define FOR_reverse(i, a, b) for(i = (a); i >= (b); --i)//rev
#endif

typedef unsigned char ubyte;
typedef unsigned int uint ;

static void delay_us(uint t){
    uint i = 0;
    for(i = 0; i < t; i = i + 1){
        // do nothin'
    }
}

static void delay_ms(uint t){
    uint i = 0;
    for(i = 0; i < t*12; i = i + 1){
        // do nothin'
    }
}

enum enum_STATE{ LOW  = 0, HIGH = 1 };
#endif
```

ThreeWiresProtocol.h

```
#ifndef _THREE_WIRES_PROTOCOL_H_
#define _THREE_WIRES_PROTOCOL_H_

#include <REGX52.h>
#include "base_lib.h"

sbit CE = P3^5;
sbit SCLK = P3^6;
sbit IO = P3^4;

ubyte T_PEAK = 0;
ubyte IDLE_T = 0;
ubyte READ_T = 0;

#define LH_MONO_PULSE(x) x = LOW; delay_us(T_PEAK); x = HIGH; delay_us(T_PEAK);
#define HL_MONO_PULSE(x) x = HIGH; delay_us(T_PEAK); x = LOW; delay_us(T_PEAK);

void single_byte_write(ubyte cmd, ubyte byte_data){
    ubyte nCLK = 0;
    delay_us(IDLE_T);
    CE = HIGH; SCLK = LOW;
    delay_us(T_PEAK);
    for(nCLK = 1; nCLK <= 8; nCLK++){
        IO = (cmd&0x1);
        HL_MONO_PULSE(SCLK);
        cmd = (cmd>>1);
    }
    for(nCLK = 1; nCLK <= 8; nCLK++){
        IO = (byte_data&0x1);
        HL_MONO_PULSE(SCLK);
        byte_data >>= 1;
    }

    CE = LOW;
}

ubyte single_byte_read(ubyte cmd){
    ubyte nCLK;
    ubyte byte_data = 0, bit_data = 0;
    //wait for sth un-finished to be done :v
    delay_us(IDLE_T);
    //starting comunication
    CE = HIGH;SCLK = LOW;
    delay_us(T_PEAK);
    //Send command at 8 rasing edge
    for(nCLK = 1; nCLK <= 7; nCLK++){
        IO = (cmd&0x1);
        HL_MONO_PULSE(SCLK);
        cmd = (cmd>>1);
    }
    // 8th rasing edge
    IO = (cmd&0x1);
    SCLK = HIGH; delay_us(T_PEAK);
    //Receiving byte_data at 8 falling edge following
    for(nCLK = 0; nCLK <= 7; nCLK++){
        SCLK = LOW;  delay_us(READ_T);
        bit_data = IO;
        byte_data = byte_data|((bit_data&0x1)<<nCLK);
        delay_us(T_PEAK-READ_T);
        SCLK = HIGH; delay_us(T_PEAK);
    }

    //End write process
    CE = LOW;
    return byte_data;
}

void ThreeWiresProtocol_Initial(){
    IO = LOW;
    SCLK = LOW;
    CE = LOW;
}

#endif
```

DS1302.h

```
#ifndef _DS1302_H_
#define  _DS1302_H_

#include "base_lib.h"
#include "ThreeWiresProtocol.h"

enum enum_DAY{MON = 0, TUE, WED, THU, FRI, SAT, SUN};

#define ds1302_unlock_reg() single_byte_write(0x8E, 0x0)

typedef struct TIME{
    uint DAY; // mon, tue, wed, thu, ...
    uint DATE;
    uint MONTH;
    uint YEAR;
    uint HOUR;
    uint MINUTE;
    uint SECOND;
} TIME;

void ds1302_read_time(TIME* time, uint SEL){
    uint x10, x1, byte_data, AM_PM;
    //second
    if(SEL&0x1){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x81);
        x10 = ((byte_data & 0x70) >> 4)*10;
        x1  = (byte_data & 0x0F);
        time->SECOND = x1 + x10;
    }
    //minute
    if(SEL&0x2){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x83);
        x10 = ((byte_data & 0x70) >> 4)*10;
        x1  = (byte_data & 0x0F);
        time->MINUTE = x10 + x1;
    }
    //hour
    if(SEL&0x4){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x85);
        if( (byte_data & 0x80) == HIGH){
            //12-hour mode
            x10 = ((byte_data & 0x10)>>4)*10;
            x1  = (byte_data & 0x0F);
            AM_PM = (byte_data&0x20)>>5;
            time->HOUR = x10 + x1 + AM_PM * 12;
        }else{
            //24-hour mode
            uint x10 = ((byte_data & 0x30)>>4)*10;
            uint x1  = (byte_data & 0x0F);
            time->HOUR = x10 + x1;
        }
    }
}

void ds1302_write_time(TIME* const time, uint SEL){
    uint x10 = 0, x1 = 0, byte_data = 0;
    //second
    if(SEL&0x1){
        x10 = (((*time).SECOND)/10)%10;
        x1  = ((*time).SECOND)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x80, byte_data);
    }
    //minute
    if(SEL&0x2){
        x10 = ((time->MINUTE)/10)%10;
        x1  = (time->MINUTE)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x82, byte_data);
    }
    //hour
    if(SEL&0x4){
        x10 = ((time->HOUR)/10)%10;
        x1  = (time->HOUR)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x84, byte_data);
    }
}

void ds1302_initial(){
    ThreeWiresProtocol_Initial();
}
#endif
```

Calendar_OnKit.h

```
#ifndef _CALENDAR_ONKIT_H_
#define  _CALENDAR_ONKIT_H_

#include "base_lib.h"
#include "DS1302.h"
#include "LED7Seg_OnKit.h"
#include "ThreeWiresProtocol.h"

#define A_DIGIT 0x77
#define P_DIGIT 0x73

#define VIEW_DATE 0x0
#define VIEW_TIME 0x1
#define SETTING_DATE 0x2
#define SETTING_TIME 0x3

sbit TRIGGER0 = P3^2;
sbit TRIGGER1 = P3^3;

ubyte MODE = VIEW_TIME;
ubyte EDIT_POS = 1;
ubyte F_EXIT = 0;

TIME time;
```

```c
void HHMMSS_disp(){
  ds1302_read_time(&time, 0x7);
  LED[7] = DIGIT_CODE[(time.HOUR/10)%10];
  LED[6] = DIGIT_CODE[time.HOUR%10];
  LED[5] = 0x40;
  LED[4] = DIGIT_CODE[(time.MINUTE/10)%10];
  LED[3] = DIGIT_CODE[time.MINUTE%10];
  LED[2] = 0x40;
  LED[1] = DIGIT_CODE[(time.SECOND/10)%10];
  LED[0] = DIGIT_CODE[(time.SECOND)%10];
  DISP = 1;
  Disp8leds7seg();
}


void calendar_disp(){
      HHMMSS_disp();
}

void calendar_initial(){
  EA = 1; EX0 = 1; IT0 = 1;
  ds1302_initial();
  time.SECOND = 0;
  time.MINUTE = 30;
  time.HOUR = 10;
  time.DAY = TUE;
  time.DATE = 1;
  time.MONTH = 9;
  time.YEAR  = 24;
  ds1302_write_time(&time,0x7);
  set_disp_freq(48);
}

#endif
```

main.c

```c
#include "Calendar_OnKit.h"

void main(void){
    calendar_initial();
    while(0x1){
        calendar_disp();
    }
}
```

BÀI 6. ĐIỀU KHIỂN TỪ XA

Yêu cầu: điều khiển 3 thiết bị thông qua remote và nút nhấn trên kit

Base_library.h

```
#ifndef _BASE_LIBRARY_H_
#define _BASE_LIBRARY_H_

#ifndef elif
#define elif else if
#endif
#ifndef DECREASE_ONE
#define DECREASE_ONE(VAR) VAR = (VAR > 0 ? (VAR - 1) : VAR)
#endif
#ifndef REP
#define REP(i, a, b) for(i = (a); i <= (b); ++i)
#endif
#ifndef REV
#define REV(i, a, b) for(i = (a); i >= (b); --i)
#endif

typedef unsigned char uint8;
typedef unsigned int uint32;
typedef char int8;
typedef int int32;

enum enum_STATE { LOW = 0, HIGH = 1 };
enum enum_ENABLE { DISABLE = 0, ENABLE, START, STOP, MODE_16BIT,
RESET };

void delay_us(uint32 us) { for(uint32 i = 0; i < us; ++i); }

void delay_ms(uint32 ms) { for(uint32 i = 0; i < ms * 12; ++i); }

void eINT0_CTL(uint8 CONFIG) {
    EX0 = (CONFIG == ENABLE);
    IT0 = 1;
}

void eINT1_CTL(uint8 CONFIG) {
    EX1 = IT1 = (CONFIG == ENABLE);
}

#define RESET_TH 0xFC
#define RESET_TL 0x67
void TIMER0_CTL(uint8 CONFIG) {
    switch (CONFIG) {
        case ENABLE: ET0 = 1; break;
        case DISABLE: ET0 = 0; break;
        case RESET: TL0 = RESET_TL; TH0 = RESET_TH; break;
        case START: TR0 = 1; break;
        case STOP: TR0 = 0; break;
        case MODE_16BIT: TMOD |= 0x01; break;
    }
}

#define GLOBAL_INT(CONFIG) EA = (CONFIG == ENABLE)

#endif
```

LED7S_OnKit.h

```
#ifndef _LED7S_ONKIT_H_
#define _LED7S_ONKIT_H_

#include <REGX52.h>
#include "Base_library.h"

sbit GND0 = P2^2;
sbit GND1 = P2^3;
sbit GND2 = P2^4;

#define LED_7SEG P0

const uint8 DIGIT_CODE[] = {
    0X3F, 0X06, 0X5B, 0X4F, 0X66, 0X6D, 0X7D, 0X07, 0X7F, 0X6F,
    0x77, 0xFC, 0x58, 0x5E, 0x79, 0x71
};

uint8 LED[8] = {0, 0, 0, 0, 0, 0, 0, 0};

void led7seg_disp(uint8 POS, uint8 CODE) {
    GND0 = POS & 0x1;
    GND1 = (POS >> 1) & 0x1;
    GND2 = (POS >> 2) & 0x1;
    LED_7SEG = CODE;
}

void Disp8leds7seg(uint32 ms_disp_t) {
    uint32 j;
    REP(j, 1, ms_disp_t)
        REP(uint8 i, 0, 7) {
            led7seg_disp(i + 1, LED[i]);
            delay_us(5);
            LED_7SEG = 0x0;
        }
}

#endif
```

```
#ifndef _IF_READING_
#define _IF_READING_

#include "REGX52.h"
#include "Base_library.h"
```

IR_reading.h

```
#include "Matrix_Button.h"

// Reset timer tại 0xFC67
#define PUSH_BIT_1() buffer |= (1UL << (31 - negedge_count))
#define RESET_BUFFER() buffer = negedge_count = 0
#define EXTRACT_FRAME() data_frame = buffer; RESET_BUFFER()

sbit IR_RCV_PIN = P3^2;
sbit IndicatorLED = P2^7;
sbit DataRcv = P2^6;
sbit FrameExtracted = P2^5;
sbit MR = P2^4;
sbit L0 = P2^0;
sbit L1 = P2^1;
sbit L2 = P2^2;

uint32 data_frame = 0, buffer = 0;
uint8 ms_count = 0, manual_remote = REMOTE;
int8 negedge_count = 0;

void IR_Reading_Initial() {
    IndicatorLED = DataRcv = 1;
    RESET_BUFFER();
    GLOBAL_INT(ENABLE);
    eINT0_CTL(ENABLE);
    eINT1_CTL(ENABLE);
    TIMER0_CTL(ENABLE);
    TIMER0_CTL(MODE_16BIT);
    TIMER0_CTL(START);
    TIMER0_CTL(RESET);
}

void Initial() {
    IR_Reading_Initial();
    P0 = 0; P2 = 0xFF;
}

void LED_Show(uint32 CODE) {
    switch (CODE) {
        case 0xFF30CF: L0 = ~L0; break;
        case 0xFF18E7: L1 = ~L1; break;
        case 0xFF7A85: L2 = ~L2; break;
        default: P2 = 0xFF;
    }
}

void Timer0_OverFlow_Interrupt() interrupt 1 {
    IndicatorLED = ~IndicatorLED;
    TIMER0_CTL(RESET);
    ms_count = (ms_count < 67) ? (ms_count + 1) : ms_count;
}

void External1_Interrupt() interrupt 2 {
    MR = manual_remote = (manual_remote == MANUAL) ? REMOTE : MANUAL;
}

void Manual_Control() {
    if (manual_remote == MANUAL) {
        uint32 btn_matrix = Get_BTN_MATRIX();
        if (btn_matrix & 0x2) L0 = ~L0;
        if (btn_matrix & 0x40) L1 = ~L1;
        if (btn_matrix & 0x800) L2 = ~L2;
        while (btn_matrix == Get_BTN_MATRIX()) delay_us(1000);
    }
}

void External0_Interrupt() interrupt 0 {
    if (manual_remote == MANUAL) return;

    uint32 current_mscount = ms_count;
    TIMER0_CTL(RESET);
    ms_count = 0;
    negedge_count++;
    DataRcv = ~DataRcv;

    if (current_mscount >= 67) {
        negedge_count = -2;
        RESET_BUFFER();
    } else if (negedge_count >= 0 && negedge_count <= 31) {
        if (current_mscount >= 2) PUSH_BIT_1();
    } else if (negedge_count >= 32) {
        EXTRACT_FRAME();
        FrameExtracted = 0;
        delay_ms(1000);
        LED_Show(data_frame);
        FrameExtracted = 1;
    }
}

#endif
```

Main.c

```
#include "REGX52.h"
#include "Base_library.h"
#include "IR_reading.h"

void main() {
    Initial();
    while (1) {
        Manual_Control();
```

	} }

BÀI 7. SMART HOME (-CONTROL DEVICE – ĐÓNG NGẮT ĐÈN TỰ ĐỘNG - TIMING)

Utilities.h

```
#ifndef _UTILITIES_H_
#define _UTILITIES_H_

#include <REGX52.h>

#define elif else if
#define DECREASE_ONE(VAR) VAR = (VAR>0?(VAR-1):VAR)
#define REP(i, a, b) for(i = (a); i <= (b); ++i)
#define REV(i, a, b) for(i = (a); i >= (b); --i)
#define true 0x1
#define false 0x0
#define bool uint8
#define min_val(A, B)  (((A)<(B))?(A):(B))
#define max_val(A, B)  (((A)>(B))?(A):(B))
#define nth_bit(num, k)  (num&(1<<(k)))  //check n-th bit is 1-bit or 0-bit
#define bool_casting(x)  ((x)?(1):(0))

typedef unsigned char   uint8;
typedef unsigned short   uint16;
typedef unsigned int    uint32;

enum enum_STATE_1{ ON  = 0, OFF = 1, NONE = 255 };
enum enum_STATE_2{ LOW  = 0, HIGH = 1, Z = 255 };
enum enum_ENABLE{ DISABLE=0, ENABLE, START,
    STOP, MODE_16BIT, RESET
};

void delay_us(uint32 us){
    uint32 i = 0;
    for(i = 0; i < us; i = i + 1){
    }
}

void delay_ms(uint32 ms){
    uint32 i = 0;
    uint32 j = 0;
    for(i = 0; i < ms*19; i = i + 1){
    }
}



#endif
```

Time.h

```
#ifndef _TIME_H_
#define _TIME_H_

#include "Utilities.h"

#ifndef _STRUCT_TIME_
#define _STRUCT_TIME_
    typedef struct TIME{
        uint8 DAY;
        uint8 DATE;
        uint8 MONTH;
        uint8 YEAR;
        uint8 HOUR;
        uint8 MINUTE;
        uint8 SECOND;
    } TIME;
#endif

uint8 time_equal_cmp(TIME a, TIME b, uint8 mask){
    if( ((mask&0x1)!=0) && (a.SECOND!=b.SECOND) )
        return false;
    if( ((mask&0x2)!=0) && (a.MINUTE!=b.MINUTE) )
        return false;
    if( ((mask&0x4)!=0) && (a.HOUR!=b.HOUR) )
        return false;
    if( ((mask&0x8)!=0) && (a.DATE!=b.DATE) )
        return false;
    if( ((mask&0x10)!=0) && (a.MONTH!=b.MONTH) )
        return false;
    if( ((mask&0x20)!=0) && (a.YEAR!=b.YEAR) )
        return false;
    return true;
}


#endif
```

XPT2046.h

```
#ifndef   __XPT2046_H_
#define   __XPT2046_H_

#include "Utilities.h"

sbit D_OUT = P3^7;
sbit D_IN  = P3^4;
sbit S_CLK  = P3^6;
sbit C_S   = P3^5;


void SPI_Initial(void)
{
    S_CLK = 0;
    C_S  = 1;
    D_IN = 1;
    S_CLK = 1;
    C_S  = 0;
}

void SPI_Write(uint8 __data)
{
    uint8 i;
    S_CLK = 0;
    for(i=0; i<8; i++)
    {
        D_IN = __data >> 7;
        __data <<= 1;
        S_CLK = 0;
        delay_us(5);
        S_CLK = 1;

    }
}

uint32 SPI_Read(void)
{
    uint32 i, __data=0;
    S_CLK = 0;
    for(i=0; i<12; i++)
    {
        __data <<= 1;

        S_CLK = 1;
        S_CLK = 0;

        __data |= D_OUT;

    }
    return __data;
}

uint32 Read_AD_Data(uint8 __command)
{
    uint8 i;
    uint32 AD_Value;
    S_CLK = 0;
    C_S  = 0;
    SPI_Write(__command);
    for(i=6; i>0; i--);
    S_CLK = 1;
    S_CLK = 0;
    AD_Value=SPI_Read();
    C_S = 1;
    return AD_Value;
}

#endif
```

ThreeWiresProtocol.h

```
#ifndef _THREE_WIRES_PROTOCOL_H_
#define _THREE_WIRES_PROTOCOL_H_

#include "Utilities.h"

sbit CE = P3^5;
sbit SCLK = P3^6;
sbit IO = P3^4;

uint8 T_PEAK = 0;
uint8 IDLE_T = 0;
uint8 READ_T = 0;

#define LH_MONO_PULSE(x) x = LOW; delay_us(T_PEAK); x = HIGH; delay_us(T_PEAK);
#define HL_MONO_PULSE(x) x = HIGH; delay_us(T_PEAK); x = LOW; delay_us(T_PEAK);

void single_byte_write(uint8 cmd, uint8 byte_data){
    uint8 nCLK = 0;
    delay_us(IDLE_T);
    CE = HIGH; SCLK = LOW;
    delay_us(T_PEAK);
    for(nCLK = 1; nCLK <= 8; nCLK++){
        IO = (cmd&0x1);
        HL_MONO_PULSE(SCLK);
        cmd = (cmd>>1);
    }
    for(nCLK = 1; nCLK <= 8; nCLK++){
        IO = (byte_data&0x1);
        HL_MONO_PULSE(SCLK);
        byte_data >>= 1;
    }

    CE = LOW;
}

uint8 single_byte_read(uint8 cmd){
    uint8 nCLK;
    uint8 byte_data = 0, bit_data = 0;
    delay_us(IDLE_T);
    CE = HIGH;SCLK = LOW;
    delay_us(T_PEAK);
    for(nCLK = 1; nCLK <= 7; nCLK++){
        IO = (cmd&0x1);
        HL_MONO_PULSE(SCLK);
        cmd = (cmd>>1);
    }
    IO = (cmd&0x1);
    SCLK = HIGH; delay_us(T_PEAK);
    for(nCLK = 0; nCLK <= 7; nCLK++){
        SCLK = LOW;  delay_us(READ_T);
        bit_data = IO;
        byte_data = byte_data|((bit_data&0x1)<<nCLK);
        delay_us(T_PEAK-READ_T);
        SCLK = HIGH; delay_us(T_PEAK);
    }

    CE = LOW;
    return byte_data;
}

void ThreeWiresProtocol_Initial(){
    IO = LOW;
    SCLK = LOW;
    CE = LOW;
}

#endif
```

DS1302.h

```
#ifndef _DS1302_H_
#define   _DS1302_H_

#include "Time.h"
#include "Utilities.h"
#include "ThreeWiresProtocol.h"


enum enum_DAY{MON = 0, TUE, WED, THU, FRI, SAT, SUN};

#define ds1302_unlock_reg() single_byte_write(0x8E, 0x0)
void DS1302_Read_Time(TIME* time, uint8 mask){
    uint8 x10, x1, byte_data, AM_PM;
    //second
    if(mask&0x1){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x81);
        x10 = ((byte_data & 0x70) >> 4)*10;
        x1  = (byte_data & 0x0F);
        time->SECOND = x1 + x10;
    }
    //minute
    if(mask&0x2){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x83);
        x10 = ((byte_data & 0x70) >> 4)*10;
        x1  = (byte_data & 0x0F);
        time->MINUTE = x10 + x1;
    }
    //hour
    if(mask&0x4){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x85);
        if( (byte_data & 0x80) == HIGH){
            //12-hour mode
            x10 = ((byte_data & 0x10)>>4)*10;
            x1  = (byte_data & 0x0F);
            AM_PM = (byte_data&0x20)>>5;
            time->HOUR = x10 + x1 + AM_PM * 12;
        }else{
            //24-hour mode
            uint8 x10 = ((byte_data & 0x30)>>4)*10;
            uint8 x1  = (byte_data & 0x0F);
            time->HOUR = x10 + x1;
        }
    }
    //date
    if(mask&0x8){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x87);
        x10 = ((byte_data&0x30)>>4)*10;
        x1  = (byte_data&0x0F);
        time->DATE = x10 + x1;
    }
    if(mask&0x10){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x89);
        x10 = ((byte_data&0x10)>>4)*10;
        x1  = (byte_data&0x0F);
        time->MONTH = x10 + x1;
    }
    if(mask&0x20){
        ds1302_unlock_reg();
        byte_data = single_byte_read(0x8D);
        x10 = ((byte_data&0xF0)>>4)*10;
        x1  = (byte_data&0x0F);
        time->YEAR = x10 + x1;
    }

}
}

void DS1302_Write_Time(TIME* const time, uint8 mask){
    uint8 x10 = 0, x1 = 0, byte_data = 0;
    //second
    if(mask&0x1){
        x10 = (((*time).SECOND)/10)%10;
        x1  = ((*time).SECOND)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x80, byte_data);
    }
    //minute
    if(mask&0x2){
        x10 = ((time->MINUTE)/10)%10;
        x1  = (time->MINUTE)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x82, byte_data);
    }
    //hour
    if(mask&0x4){
        x10 = ((time->HOUR)/10)%10;
        x1  = (time->HOUR)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x84, byte_data);
    }
    //date
    if(mask&0x8){
        x10 = ((time->DATE)/10)%10;
        x1  = (time->DATE)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x86, byte_data);
    }
    //month
    if(mask&0x10){
        x10 = ((time->MONTH)/10)%10;
        x1  = (time->MONTH)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x88, byte_data);
    }
    //year
    if(mask&0x20){
        x10 = ((time->YEAR)/10)%10;
        x1  = (time->YEAR)%10;
        byte_data = (x10<<4) + x1;
        ds1302_unlock_reg();
        single_byte_write(0x8C, byte_data);
    }
    //day
    if(mask&0x40){
        x1  = (time->DAY)%10;
        ds1302_unlock_reg();
        single_byte_write(0x9A, x1);
    }
}

void DS1302_Initial(){
    ThreeWiresProtocol_Initial();
}
#endif
```

IR_Reading.h

```
#ifndef _IF_READING_
#define _IF_READING_

#include "Utilities.h"
#include "DS1302.h"
#include "LED7Seg_OnKit.h"
//#include "Matrix_Button.h"



#define ON_OFF 0xA25D
#define MODE  0x629D
#define MUTE  0xE21D
#define PREV  0x02FD  // PREV
#define NEXT  0xC23D  // NEXT
#define PLAY_PAUSE  0x22DD  // PLAY/PAUSE
#define VOL_DOWN  0xA857  // VOL-
#define VOL_UP  0x906F  // VOL+
#define EQ  0xE01F  // EQ
#define __0  0xFF6897  // 0
#define __1  0xFF30CF  // 1
#define __2  0xFF18E7  // 2
#define __3  0xFF7A85  // 3
#define __4  0xFF10EF  // 4
#define __5  0xFF38C7  // 5
#define __6  0xFF5AA5  // 6
#define __7  0xFF42BD  // 7
#define __8  0xFF4AB5  // 8
#define __9  0xFF52AD  // 9



// Reset timer at 0xFC67
#define PUSH_BIT_1() buffer |=(uint32)1<<(31-negedge_count);
#define PUSH_BIT_0() /*do nothing*/;
#define RESET_BUFFER() buffer=0;
#define EXTRACT_FRAME() data_frame=buffer; buffer = 0; negedge_count = 0;

sbit FrameExtracted = P2^0;
uint32 data_frame = 0;
uint32 buffer = 0;
uint8 ms_count = 0;
int8 negedge_count = 0;

uint32 read_extracted_frame(){
    uint32 frame = data_frame;
    data_frame = 0;
    return frame;
}

void IR_Reading_Initial(){
    buffer = 0;
    data_frame = 0;
    negedge_count = 0;

    EA = 1; EX0=1; ET0 = 1; TMOD |= 0x1; TR0=1; TL0=0x67; TL1=0xFC;
}

void Timer0_OverFlow_Interrupt() interrupt 1 {
    TIMER0_CTL(RESET);
    if(ms_count<67) ms_count = ms_count + 1;
}

void External0_Interrupt() interrupt 0 {
    uint32 current_mscount = 0;

    current_mscount = ms_count;
    TIMER0_CTL(RESET);
    ms_count=0;
    negedge_count +=1;
    if(current_mscount >= 67){
        negedge_count = -2;
        RESET_BUFFER();
    }else{
        if( negedge_count < 0)
        if(0 <= negedge_count && negedge_count <= 31){
            if( current_mscount >= 2){
                PUSH_BIT_1();
            }else{
                PUSH_BIT_0();
            }
        }else if(negedge_count >= 32){
            EXTRACT_FRAME();
            FrameExtracted=0;
            delay_ms(1);
            FrameExtracted=1;
        }
    }
}

#endif
```

LED7Seg_OnKit.h

```
#ifndef _LED7SEG_ONKIT_H_
#define _LED7SEG_ONKIT_H_
#include "Utilities.h"

//---------- Macros -----------
sbit GND0 = P2^2;
sbit GND1 = P2^3;
sbit GND2 = P2^4;

#define LED_7SEG P0

const uint8 DIGIT_CODE[] = {0X3F, 0X06, 0X5B, 0X4F, 0X66, 0X6D,
                           0X7D, 0X07, 0X7F, 0X6F, /*A*/0x77, 0x7C,
                           0x58, 0x5E, 0x79, 0x71};

uint8 LED[8] = {0, 0, 0, 0, 0, 0, 0, 0};
void led7seg_disp(uint8  POS, uint8  CODE){
    switch (POS) {
        case 0x1:
            { GND0 = 0; GND1 = 0; GND2 = 0; LED_7SEG = CODE; return;}
        case 0x2:
            { GND0 = 1; GND1 = 0; GND2 = 0; LED_7SEG = CODE; return;}
        case 0x3:
            { GND0 = 0; GND1 = 1; GND2 = 0; LED_7SEG = CODE; return;}
        case 0x4:
            { GND0 = 1; GND1 = 1; GND2 = 0; LED_7SEG = CODE; return;}
        case 0x5:
            { GND0 = 0; GND1 = 0; GND2 = 1; LED_7SEG = CODE; return;}
        case 0x6:
            { GND0 = 1; GND1 = 0; GND2 = 1; LED_7SEG = CODE; return;}
        case 0x7:
            { GND0 = 0; GND1 = 1; GND2 = 1; LED_7SEG = CODE; return;}
        case 0x8:
            { GND0 = 1; GND1 = 1; GND2 = 1; LED_7SEG = CODE; return;}
        default:
            LED_7SEG = 0x0;
    }
}
void Disp8leds7seg(uint32 ms_disp_t){
    uint8 i = 0;
    uint32 j = 0;
    REP(j, 1, ms_disp_t)
        REP(i, 0, 7){
            led7seg_disp(i+1, LED[i]);
            delay_us(5);
            LED_7SEG = 0x0;
        }
}
#endif
```

main.h

```
#include "Utilities.h"
#include "DS1302.h"
#include "XPT2046.h"
#include "IR_Reading.h"
#include "LED7Seg_OnKit.h"

sbit dev0 = P2^5;
sbit dev1 = P2^6;
sbit dev2 = P2^7;

enum enum_modes{
    NORMAL_MODE = 0,
    SETUP_MODE  = 1,
    TIME_SETUP_MODE = 3,
    DEV_CONTROL_MODE = 4,
    SYS_TIME_SETUP_MODE = 9,
    SYS_TIME_SETUP = 27,
    ON_TIME_SETUP_MODE = 10,
    ON_TIME_SETUP = 30,
    OFF_TIME_SETUP_MODE = 11,
    OFF_TIME_SETUP = 33,
    DEV0_SETUP_MODE = 12,
    DEV1_SETUP_MODE = 13,
    DEV2_SETUP_MODE = 14,
    DEV0_ON_OFF =36,
    DEV1_ON_OFF =39,
    DEV2_ON_OFF =42
```

```
};

uint32 CURRENT_INDX = 0;
uint8 WAIT_YES_NO = false;
uint8 dev0_user_ctl = 0;
uint8 dev1_user_ctl = 0;
uint8 dev2_user_ctl = 0;
uint8 dev0_syst_ctl = 0;
uint8 dev1_syst_ctl = 0;
uint8 dev2_syst_ctl = 0;
uint8 timer_enable = 0;
uint32 IR_data = 0;
//System time
TIME system_time = {0, 0, 0, 0, 0, 0, 0};
//Turn-on device time
TIME time_on = {0, 0, 0, 0, 0, 0, 0};
//Turn-off device time
TIME time_off = {0, 0, 0, 0, 0, 0, 0};

//Remote code to number
uint8 CODE2NUM(uint32 CODE){
    switch (CODE) {
        case __0: return 0;
        case __1: return 1;
        case __2: return 2;
        case __3: return 3;
        case __4: return 4;
        case __5: return 5;
        case __6: return 6;
        case __7: return 7;
        case __8: return 8;
        case __9: return 9;
    }
    return 0;
}

void clear(){
    LED[0] = 0x0;
    LED[1] = 0x0;
    LED[2] = 0x0;
    LED[3] = 0x0;
    LED[4] = 0x0;
    LED[5] = 0x0;
    LED[6] = 0x0;
    LED[7] = 0x0;
}

uint8 YES_NO(){
    uint32 CODE = 0;
    while(0x1){
        CODE = read_extracted_frame();
        clear();
        LED[7] = 0x6E; LED[6] = 0x37;
        Disp8leds7seg(1);
        switch (CODE) {
            //extend for more options :v
            case PLAY_PAUSE: return 1;
            case MODE: return 0;
            case ON_OFF: return 0;
        }
    }
    return 0;
}

uint8 SET_TIMER(TIME* t){
    uint8 POS = 0;
    uint32 CODE = 0;
    TIME tmp;
    // tmp = *t;
    DS1302_Read_Time(&tmp, 0x6);
    while(0x1){
        CODE = read_extracted_frame();
        if(CODE == PLAY_PAUSE) break;
        if(CODE == PREV) POS = (POS+1 + 2)%2;
        if(CODE == NEXT) POS = (POS-1 + 2)%2;
        if(CODE == ON_OFF) return 0;
        if(CODE == MODE) return 0;
        switch (POS) {
            case 0:
                tmp.MINUTE += CODE2NUM(CODE)%10; tmp.MINUTE%=60; break;
            case 1:
                tmp.HOUR   += CODE2NUM(CODE); tmp.HOUR%=24; break;
        }
        LED[0] = DIGIT_CODE[tmp.MINUTE%10] + ((POS==0)?(0x80):(0));
        LED[1] = DIGIT_CODE[tmp.MINUTE/10];
        LED[2] = DIGIT_CODE[tmp.HOUR%10]   + ((POS==1)?(0x80):(0));
        LED[3] = DIGIT_CODE[tmp.HOUR/10];
        Disp8leds7seg(50);
    }
    if(YES_NO()){
        *t = tmp;
        return 1;
    }
    return 0;
}

uint8 SET_ON_OFF_NONE(uint8 *val, uint8 dev){
    uint8 tmp = 2;
    uint32 CODE = 0;
    while(0x1){
        CODE = read_extracted_frame();
        if(CODE == PLAY_PAUSE) break;
        if(CODE == PREV) tmp = (tmp+1 + 3)%3;
        if(CODE == NEXT) tmp = (tmp-1 + 3)%3;
        if(CODE == ON_OFF) return 0;
        if(CODE == MODE) return 0;
        switch (tmp) {
            case 0:
                LED[7] = DIGIT_CODE[13];
                LED[6] = DIGIT_CODE[dev];
                LED[5] = 0;
                LED[4] = 0;
                LED[3] = DIGIT_CODE[0];
                LED[2] = DIGIT_CODE[15];
                LED[1] = DIGIT_CODE[15];
                LED[0] = 0;
                break;
            case 1:
                LED[7] = DIGIT_CODE[13];
                LED[6] = DIGIT_CODE[dev];
                LED[5] = 0;
                LED[4] = 0;
                LED[3] = DIGIT_CODE[0];
                LED[2] = 0x37;
                LED[1] = 0x0;
                LED[0] = 0x0;
                break;
            case 2:
                LED[7] = DIGIT_CODE[13];
                LED[6] = DIGIT_CODE[dev];
                LED[5] = 0;
                LED[4] = 0;
                LED[3] = 0x37;
                LED[2] = DIGIT_CODE[0];
                LED[1] = 0x37;
                LED[0] = DIGIT_CODE[14];
                break;
        }
        Disp8leds7seg(50);
    }
    if(YES_NO()){
        *val = (tmp == 0 || tmp == 1)?(tmp):(Z);
        return 1;
    }
    return 0;
}

void read_system_time(){
    DS1302_Read_Time(&system_time, 0x7);
}

void update_dev_state(){
    if(dev0_user_ctl == Z)
        dev0 = (dev0_syst_ctl)?0:1;
    else
        dev0 = (dev0_user_ctl)?0:1;
    if(dev1_user_ctl == Z)
        dev1 = (dev1_syst_ctl)?0:1;
    else
        dev1 = (dev1_user_ctl)?0:1;
    if(dev2_user_ctl == Z)
        dev2 = (dev2_syst_ctl)?0:1;
    else
        dev2 = (dev2_user_ctl)?0:1;
}

uint32 have_daylight(){
    Read_AD_Data(0xA4);
    if( (Read_AD_Data(0xA4)%1000) > 30)
        return true;
    return false;
}

uint32 get_up_index(uint32 indx){
    if(indx == 0) return 1;
    return (indx/3);
}

uint32 get_down_index(uint32 indx){
    if(indx*3 > 42) return indx;
    return (indx*3);
}

uint32 get_left_index(uint32 indx){
    switch (indx) {
        case 3: return 4;
        case 4: return 3;
        case 10: return 9;
        case 11: return 10;
        case 9: return 11;
        case 12: return 14;
        case 13: return 12;
        case 14: return 13;
    }
    return indx;
}

uint32 get_right_index(uint32 indx){
    switch (indx) {
        case 3: return 4;
        case 4: return 3;
        case 9: return 10;
        case 10: return 11;
        case 11: return 9;
        case 12: return 13;
        case 13: return 14;
        case 14: return 12;
    }
    return indx;
}

void code_proc(uint32 CODE){
    switch (CODE) {
        case ON_OFF:
            dev0_syst_ctl = (dev0_syst_ctl)?(0):(1);
            return;
        case MODE:
            CURRENT_INDX = get_up_index(CURRENT_INDX);
            break;
        case PLAY_PAUSE:
            CURRENT_INDX = get_down_index(CURRENT_INDX);
            break;
        case PREV:
            CURRENT_INDX = get_left_index(CURRENT_INDX);
            break;
        case NEXT:
            CURRENT_INDX = get_right_index(CURRENT_INDX);
            break;
    }
    switch (CURRENT_INDX) {
        case NORMAL_MODE:
            LED[0] = DIGIT_CODE[(system_time.SECOND)%10];
            LED[1] = DIGIT_CODE[(system_time.SECOND/10)%10];
            LED[2] = 0x40;
            LED[3] = DIGIT_CODE[(system_time.MINUTE)%10];
            LED[4] = DIGIT_CODE[(system_time.MINUTE/10)%10];
            LED[5] = 0x40;
            LED[6] = DIGIT_CODE[(system_time.HOUR)%10];
            LED[7] = DIGIT_CODE[(system_time.HOUR/10)%10];
            return;
        case SETUP_MODE:
            LED[7] = DIGIT_CODE[5];
            LED[6] = DIGIT_CODE[14];
            LED[5] = 0x7;
            LED[4] = 0x3E;
            LED[3] = 0x73;
            LED[2] = 0x0;
            LED[1] = 0x0;
            LED[0] = 0;
            return;
        case TIME_SETUP_MODE:
            LED[7] = 0x31;
            LED[6] = 0x40;
            LED[5] = 0x39;
            LED[4] = 0x31;
            LED[3] = 0x38;
            LED[2] = 0x0;
            LED[1] = 0x0;
            LED[0] = 0x0;
            return;

        case SYS_TIME_SETUP_MODE:
            LED[7] = DIGIT_CODE[5];
            LED[6] = 0x6E;
            LED[5] = DIGIT_CODE[5];
            LED[4] = 0x0;
            LED[3] = 0x0;
            LED[2] = 0x0;
            LED[1] = 0x0;
            LED[0] = 0x0;
            return;

        case SYS_TIME_SETUP:
            if( SET_TIMER(&system_time))
                DS1302_Write_Time(&system_time, 0x7F);
            CURRENT_INDX = get_up_index(CURRENT_INDX);
            return;

        case ON_TIME_SETUP_MODE:
            LED[7] = DIGIT_CODE[0];
            LED[6] = 0x37;
            LED[5] = 0;
            LED[4] = 0;
            LED[3] = 0x0;
            LED[2] = 0x0;
            LED[1] = 0x0;
            LED[0] = 0x0;
            return;

        case ON_TIME_SETUP:
            SET_TIMER(&time_on);
            CURRENT_INDX = get_up_index(CURRENT_INDX);
            return;

        case OFF_TIME_SETUP_MODE:
            LED[7] = DIGIT_CODE[0];
            LED[6] = DIGIT_CODE[15];
            LED[5] = DIGIT_CODE[15];
            LED[4] = 0;
            LED[3] = 0x0;
            LED[2] = 0x0;
            LED[1] = 0x0;
            LED[0] = 0x0;
            return;

        case OFF_TIME_SETUP:
            SET_TIMER(&time_off);
            CURRENT_INDX = get_up_index(CURRENT_INDX);
            return;

        case DEV_CONTROL_MODE:
            LED[7] = DIGIT_CODE[13];
            LED[6] = DIGIT_CODE[14];
            LED[5] = 0x3E;
            LED[4] = 0x39;
            LED[3] = 0x31;
            LED[2] = 0x38;
            LED[1] = 0x0;
            LED[0] = 0x0;
            return;

        case DEV0_SETUP_MODE:
            LED[7] = DIGIT_CODE[5];
            LED[6] = DIGIT_CODE[14];
            LED[5] = 0x7;
            LED[4] = 0x3E;
            LED[3] = 0x73;
            LED[2] = 0x0;
            LED[1] = DIGIT_CODE[13];
            LED[0] = DIGIT_CODE[0];
            return;

        case DEV1_SETUP_MODE:
            LED[7] = DIGIT_CODE[5];
            LED[6] = DIGIT_CODE[14];
            LED[5] = 0x7;
            LED[4] = 0x3E;
            LED[3] = 0x73;
            LED[2] = 0x0;
            LED[1] = DIGIT_CODE[13];
            LED[0] = DIGIT_CODE[1];
            return;

        case DEV2_SETUP_MODE:
            LED[7] = DIGIT_CODE[5];
            LED[6] = DIGIT_CODE[14];
            LED[5] = 0x7;
            LED[4] = 0x3E;
            LED[3] = 0x73;
            LED[2] = 0x0;
            LED[1] = DIGIT_CODE[13];
            LED[0] = DIGIT_CODE[2];
            return;

        case DEV0_ON_OFF:
            SET_ON_OFF_NONE(&dev0_user_ctl, 0);
            CURRENT_INDX = get_up_index(CURRENT_INDX);
            update_dev_state();
            return;

        case DEV1_ON_OFF:
            SET_ON_OFF_NONE(&dev1_user_ctl, 1);
            CURRENT_INDX = get_up_index(CURRENT_INDX);
            update_dev_state();
            return;

        case DEV2_ON_OFF:
            SET_ON_OFF_NONE(&dev2_user_ctl, 2);
            CURRENT_INDX = get_up_index(CURRENT_INDX);
            update_dev_state();
            return;

    }
}

void main_intial(){
    IR_Reading_Initial();
    DS1302_Initial();
    dev0_user_ctl = Z;
    dev1_user_ctl = Z;
    dev2_user_ctl = Z;
    CURRENT_INDX = 0;
    DS1302_Write_Time(&system_time, 0x7F);

}
```

main.c

```
#include "main.h"
#include "IR_Reading.h"

int main(){
    main_intial();
    while(true){
        read_system_time();
        if(time_equal_cmp(system_time, time_on, 0x6))
            dev2_syst_ctl = HIGH;
        if(time_equal_cmp(system_time, time_off, 0x6))
            dev2_syst_ctl = LOW;
        if(have_daylight()){
            dev1_syst_ctl = LOW;
        }else{
            dev1_syst_ctl = HIGH;
        }
        update_dev_state();
        code_proc(read_extracted_frame());
        Disp8leds7seg(10);
    }
    return 0;
}
```

BÀI 8. SNAKE GAME

Snake.h

```
#ifndef SNAKE_H
#define SNAKE_H

#include <matrix.h>
#include <REG51.h>

sbit UE = P3^2;
sbit SHITA = P3^3;
sbit HIDARI = P3^0;
sbit MIGI = P3^1;

unsigned char snake[64];
unsigned char food;
unsigned char direction;
unsigned char snake_length;
unsigned char display_buffer[8];
unsigned char last_direction;
unsigned int Speed = 20;

void init_game()
{
    unsigned char i;
    for (i = 0; i < 64; i++)
    {
        snake[i] = 0;
    }
    snake[0] = 28;
    snake[1] = 27;
    food = 10;
    direction = 3;
    last_direction = 3;
    snake_length = 2;
}

void check_direction()
{
    unsigned char new_direction = last_direction;

    if (UE == 0 && last_direction != 1) { new_direction = 0; }
    if (SHITA == 0 && last_direction != 0) { new_direction = 1; }
    if (HIDARI == 0 && last_direction != 3) { new_direction = 2; }
    if (MIGI == 0 && last_direction != 2) { new_direction = 3; }

    if (new_direction != last_direction)
    {
        direction = new_direction;
        last_direction = new_direction;
    }
}

void update_snake()
{
    unsigned char i;
    for (i = snake_length; i > 0; i--)
    {
        snake[i] = snake[i - 1];
    }

    switch (direction)
    {
        case 0: snake[0] = (snake[0] - 8 + 64) % 64; break;
        case 1: snake[0] = (snake[0] + 8) % 64; break;
        case 2: snake[0] = (snake[0] % 8 == 0) ? snake[0] + 7 :
snake[0] - 1; break;
        case 3: snake[0] = (snake[0] % 8 == 7) ? snake[0] - 7 :
snake[0] + 1; break;
    }
}

void generate_new_food()
{
    unsigned char i;
    unsigned char is_valid;
    do
    {
        is_valid = 1;
        food = (food + 17) % 64;
        for (i = 0; i < snake_length; i++)
        {
            if (food == snake[i]) { is_valid = 0; break; }
        }
    } while (!is_valid);
}

void check_collision()
{
    unsigned char i;
    for (i = 1; i < snake_length; i++)
    {
        if (snake[0] == snake[i])
        {
            init_game();
            return;
        }
    }

    if (snake[0] == food)
    {
        if (snake_length < 63)
        {
            snake_length++;
        }
        generate_new_food();
    }
}

void update_display_buffer()
{
    unsigned char i, row, col;
    for (i = 0; i < 8; i++) { display_buffer[i] = 0x00; }

    for (i = 0; i < snake_length; i++)
    {
        row = snake[i] / 8;
        col = snake[i] % 8;
        display_buffer[row] |= (1 << col);
    }

    row = food / 8;
    col = food % 8;
    display_buffer[row] |= (1 << col);
}

void Running_Game()
{
    check_direction();
    update_snake();
    check_collision();
    update_display_buffer();
}

#endif
```

Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#include <REG51.h>

sbit SRCLK = P3^6;
sbit RCLK = P3^5;
sbit SER = P3^4;

unsigned char code Cols[8] = {0x7f, 0xbf, 0xdf, 0xef, 0xf7, 0xfb,
0xfd, 0xfe};

void delay(unsigned int time)
{
    unsigned int i, j;
    for (i = 0; i < time; i++)
    {
        for (j = 0; j < 121; j++);
    }
}

void Hc595SendByte(unsigned char dat)
{
    unsigned char a;
    SRCLK = 0;
    RCLK = 0;
    for (a = 0; a < 8; a++)
    {
        SER = (dat & 0x80) >> 7;
        dat <<= 1;
        SRCLK = 1;
        SRCLK = 0;
    }
    RCLK = 1;
    RCLK = 0;
}

#endif
```

Main.c

```
#include <matrix.h>
#include <REG51.h>
#include <Snake.h>

#define COMMONPORTS P0

void main()
{
    unsigned char tab, i;
    init_game();

    while(1)
    {
        Running_Game();
        for (i = 0; i < Speed; i++)
        {
            for (tab = 0; tab < 8; tab++)
            {
                Hc595SendByte(0x00);
                COMMONPORTS = Cols[tab];
                Hc595SendByte(display_buffer[tab]);
                delay(2);
            }
        }
    }
}
```

```
void showGameOverScreen()
{
    matrix_clear();
    matrix_display_text("GAME", 0);
    delay(1000);
    matrix_clear();
    matrix_display_text("OVER", 0);
    delay(1000);
    matrix_clear();
    char scoreText[16];
    snprintf(scoreText, sizeof(scoreText), "Score:%d", score);
    matrix_display_text(scoreText, 0);
    delay(2000);
    matrix_clear();
    matrix_display_text("R:Restart", 0);
    delay(1000);
    matrix_display_text("Q:Quit", 0);

    char choice;
    do {
        choice = getchar();
        if (choice == 'R' || choice == 'r') {
            resetGame();
            break;
        } else if (choice == 'Q' || choice == 'q') {
            exit(0);
        }
    } while (choice != 'R' && choice != 'r' && choice != 'Q' && choice !=
'q');
    matrix_clear();
}
```