BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN KỸ THUẬT MÁY TÍNH – VIỄN THÔNG

BÁO CÁO MÔN HỌC

TỔNG HỢP MÃ NGUỒN LẬP TRÌNH VI ĐIỀU KHIỂN 8051 THỰC TẬP KIẾN TRÚC VÀ TỔ CHỨC MÁY TÍNH

GVHD: TS. ĐẬU TRỌNG HIỂN

SVTH:

PHẠM QUANG HỢP	22119178
NGUYỄN PHÚC KHANH	22119189

- Tp.HCM, tháng 12/2024 -

MỤC LỤC

1. Lập trình vi điều khiển 8051 điều khiển 8 LED đơn sáng dần, tắt dần mô phỏng trên Proteus

-Chương trình main.c-

```
#include<main.h>
//============Khai bao bien va hang================//
unsigned char x=0;
unsigned char i=0;
//====================Ham main====================//
int main () {
      P1=0;
      while(1) {            //Vong lap vo han
            //=============Sang dan=============//
            for(i=0;i<8;i++) {
                  x=(x<<1)+1;
                  P1=x;
                  delay_ms(100);
            }
            //==============Tat dan=============//
            for(i=0;i<8;i++) {
                  x=x<<1;
                  P1=x;
                  delay_ms(100);
            }
      }
}
```

-main.h-

```
#ifndef __MAIN_H
#define __MAIN_H
```

```
//==============Khai bao cac thu vien can su dung===========//
#include <stdio.h>
#include <regx52.h>
#include <math.h>
//==============Khai bao thu vien nguoi dung viet================//
#include <delay.h>
#endif
//***********Ket thuc file**************//
```

-Chương trình delay.c-

```
#include <main.h>
/*Tan so hoat dong cua vi dieu khien = Fosc(TansodaodongThachanh)/12
=> Chu ky Tosc=12/Fosc
Chon tan so Thach anh = 12MHz = 12^10^6 Hz
=> Chu ky may(Chu ki haot dong cua vi dieu khien = 12/(12^10^6 Hz)
= 1/(10^6) s = 1us
Mot vong for khoang 8 chu ky may nen = 8*1 = 8us
=> delay_ms (1) = 125*8 = 1000us */
/*================================================
 *Chuc nang: tao tre ms
 *Tham so: Time la gia tri can tan tre, gia tri 16bits
 *Gia tri tra ve: Khong co
================================================*/
void delay_ms (unsigned int Time)
{
      unsigned int i,j;
      for(i=0; i<Time; i++) {
            for(j=0; j<125; j++);
      }
}
```

```
/*================================================
 *Chuc nang: tao tre ms
 *Tham so: Time la gia tri can tan tre, gia tri 16bits
 *Gia tri tra ve: Khong co
================================================*/
void delay_us(unsigned int Time)
{
     while((--Time)!=0);
}
```

-delay.h-

```
#ifndef _DELAY_H
#define _DELAY_H
//===============KHAI BAO CAC HAM===============//
void delay_ms (unsigned int Time);
void delay_us (unsigned int Time);
#endif
```

2. Lập trình vi điều khiển 8051 mô phỏng đèn giao thông ngã tư có đếm ngược trên Proteus

-Chương trình main.c-

```
#include <main.h>
// Biến toàn cục lưu trạng thái chế độ và chuyển hướng
int chedo;              // Chế độ hoạt động (tự động = 0, thủ công = 1)
int chuyenhuong;        // Chuyển hướng giao thông (0 hoặc 1)
// Bảng mã hiển thị 7 đoạn cho các số từ 0 đến 9
unsigned char Code7seg[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
0x80, 0x90};
// Các biến phục vụ đếm thời gian
int dem;                // Biến đếm thời gian
int tanso;              // Biến tần số cho hiển thị
```

```
// Hàm delay theo đơn vị ms
void delay_ms(unsigned int time) {
    /* f = 12 MHz */
    /* f vào timer 1 = 12M / 2 = 6M */
    /* T = 1 / 6M s */
    /* 6000 lần đếm ~ 1ms */
    /* Giá trị TH1, TL1 = 59536 -> 65535 */
    while(time--) {
        TMOD = 0x01;     // Chế độ timer 1
        TH1 = 0xE8;      // Giá trị cao
        TL1 = 0x90;      // Giá trị thấp
        TR1 = 1;         // Bắt đầu đếm
        while(!TF1);     // Chờ cờ tràn
        TF1 = 0;         // Reset cờ tràn
    }
}
// Hàm điều khiển đèn giao thông tự động
void Traffic_Auto(void) {
    StatusTraffic = 0x21; // Đèn đỏ đường chính, xanh đường phụ
    // Hiển thị và đếm lùi 30s
    for (dem = 30; dem >= 3; dem--) {
        if (chedo == 1) return; // Thoát nếu chuyển sang chế độ thủ công
        for (tanso = 0; tanso < 30; tanso++) {
            // Hiển thị số giây lùi trên 7 đoạn
            dvi_DT = 1; Display = Code7seg[dem % 10];
            delay_ms(10); dvi_DT = 0;
            dvi_BN = 1; Display = Code7seg[(dem - 3) % 10];
            delay_ms(10); dvi_BN = 0;
            chuc_DT = 1; Display = Code7seg[dem / 10];
```

```
        delay_ms(10); chuc_DT = 0;
        chuc_BN = 1; Display = Code7seg[(dem - 3) / 10];
        delay_ms(10); chuc_BN = 0;
    }
}
StatusTraffic = 0x9; // Đèn vàng đường chính
// Hiển thị và đếm lùi 3s
for (dem = 2; dem >= 0; dem--) {
    if (chedo == 1) return;
    for (tanso = 0; tanso < 30; tanso++) {
        // Hiển thị số giây lùi trên 7 đoạn
        dvi_DT = 1; Display = Code7seg[dem % 10];
        delay_ms(10); dvi_DT = 0;
        chuc_DT = 1; Display = Code7seg[dem / 10];
        delay_ms(10); chuc_DT = 0;
    }
}
StatusTraffic = 0x12; // Đèn đỏ đường phụ, xanh đường chính
// Hiển thị và đếm lùi 30s
for (dem = 30; dem >= 3; dem--) {
    if (chedo == 1) return;
    for (tanso = 0; tanso < 30; tanso++) {
        // Hiển thị số giây lùi trên 7 đoạn
        dvi_BN = 1; Display = Code7seg[dem % 10];
        delay_ms(10); dvi_BN = 0;
        chuc_BN = 1; Display = Code7seg[dem / 10];
        delay_ms(10); chuc_BN = 0;
    }
}
```

```
    StatusTraffic = 0x6; // Đèn vàng đường phụ
    // Hiển thị và đếm lùi 3s
    for (dem = 2; dem >= 0; dem--) {
        if (chedo == 1) return;
        for (tanso = 0; tanso < 30; tanso++) {
            // Hiển thị số giây lùi trên 7 đoạn
            dvi_BN = 1; Display = Code7seg[dem % 10];
            delay_ms(10); dvi_BN = 0;
        }
    }
}
// Cấu hình ngắt Timer0
void Interrupt_Timer0(void)
{
    TMOD = 0x01;          // Chế độ 16-bit timer
    TH0 = 0xFC;           // Giá trị cao (1ms)
    TL0 = 0x18;           // Giá trị thấp (1ms)
    ET0 = 1;              // Cho phép ngắt Timer0
    EA = 1;               // Cho phép ngắt toàn cục
    TR0 = 1;              // Kích hoạt Timer0
}
// Hàm xử lý ngắt Timer0
void ISR_TIMER(void) interrupt 1
{
    if (ET0 == 1) {
        if (SW_chedo == 0) {   // Kiểm tra nút nhấn chuyển chế độ
            delay_ms(100);
            if (SW_chedo == 1) chedo++;
            if (chedo > 1) chedo = 0;
```

```
        }
        // Reset giá trị Timer0 sau mỗi lần ngắt
        TH0 = 0xFC;
        TL0 = 0x18;
    }
}
// Hàm chính
void main(void)
{
    Interrupt_Timer0(); // Cấu hình ngắt Timer0
    while (1) {
        if (chedo == 1) {  // Chế độ thủ công
            if (SW == 0) { // Kiểm tra nút nhấn chuyển hướng
                delay_ms(100);
                if (SW == 1) chuyenhuong++;
                if (chuyenhuong > 1) chuyenhuong = 0;
            }
            if (chuyenhuong) {
                if (P1_1 == 1) { StatusTraffic = 0x6; delay_ms(2000); }
                StatusTraffic = 0x21;
            } else {
                if (P1_0 == 1) { StatusTraffic = 0x9; delay_ms(2000); }
                StatusTraffic = 0x12;
            }
        }
        if (chedo == 0) Traffic_Auto(); // Chế độ tự động
    }
}
```

-main.h-

```
#include <AT89X52.h>
#define StatusTraffic P1         /*do0, do1, vang0, vang1, xanh0, xanh1*/
sbit SW_chedo = P0^4;            // Nút nhấn chuyển chế độ
sbit SW = P0^5;                  // Nút nhấn chuyển hướng
sbit dvi_DT = P0^0;              // Chân điều khiển đơn vị phía 0.
sbit chuc_DT = P0^1;             // Chân điều khiển hàng chục phía 0.
sbit dvi_BN = P0^2;              // Chân điều khiển đơn vị của phía 1.
sbit chuc_BN = P0^3;             // Chân điều khiển hàng chục phía 1.
#define Display P2
```

3. Lập trình vi điều khiển 8051 mô phỏng lịch vạn niên hiển thị ngày, tháng, năm; giờ, phút, giây trên Proteus

Chương trình main.c

```
#include <reg51.h>        // Thư viện cho vi điều khiển 8051
#include "lcd.h"          // Thư viện điều khiển LCD
#include "ds1307.h"       // Thư viện điều khiển RTC DS1307
#include "i2c.h"          // Thư viện giao tiếp I2C
#include "delay.h"        // Thư viện tạo độ trễ

/* -----------------main program------------------- */
void main()
{
   unsigned char sec, min, hour, day, month, year;
   lcd_Init();             /* Khởi tạo LCD */
   ds1307_Init();          /* Khởi tạo RTC (DS1307) */

   /* Thiết lập thời gian và ngày tháng (chỉ thực hiện một lần để đồng bộ) */
   ds1307_SetTimeAuto();  // Thiết lập thời gian tự động
   ds1307_SetDateAuto();  // Thiết lập ngày tháng tự động
```

```
    /* Hiển thị "Time:" trên dòng đầu tiên của LCD */
    lcd_DisplayString("Time: ");

    /* Hiển thị "Date:" trên dòng thứ hai của LCD */
    lcd_GoToLineTwo();
    lcd_DisplayString("Date: ");

    /* Hiển thị thời gian và ngày tháng liên tục */
    while(1)
    {
        /* Đọc thời gian từ RTC DS1307 */
        ds1307_GetTime(&hour, &min, &sec);

        /* Hiển thị thời gian trên dòng đầu tiên, bắt đầu từ vị trí thứ 7 */
        lcd_GoToXY(0, 6);
        lcd_DisplayRtcTime(hour, min, sec);

        /* Đọc ngày tháng từ RTC DS1307 */
        ds1307_GetDate(&day, &month, &year);

        /* Hiển thị ngày tháng trên dòng thứ hai, bắt đầu từ vị trí thứ 7 */
        lcd_GoToXY(1, 6);
        lcd_DisplayRtcDate(day, month, year);
    }
}
```

-delay.c-

```
#include<reg51.h>
#include "delay.h"
void delay_us(unsigned int us_count) {
```

```
     while(us_count!=0) {
          us_count--;
  }
}
```

-delay.h-

```
// the #ifndef prevents the file from being included more than once
#ifndef __DELAY_H__
#define __DELAY_H__
void delay_us(unsigned int us_count);
#endif
```

-ds1307.c-

```
#include<reg51.h>
#include "ds1307.h"
#include "i2c.h"
#include "delay.h"
// Các tệp header hỗ trợ điều khiển RTC DS1307, giao tiếp I2C, và hàm trễ.

#define DS1307_ID 0xD0          // Địa chỉ I2C của chip DS1307.
#define SEC_ADDRESS   0x00 // Địa chỉ thanh ghi lưu trữ giây trong DS1307.
#define DATE_ADDRESS  0x04 // Địa chỉ thanh ghi lưu trữ ngày trong DS1307.
#define control       0x07 // Địa chỉ thanh ghi điều khiển DS1307.

void ds1307_Init()         // Khởi tạo DS1307:
{
    i2c_Start();           // Bắt đầu giao tiếp I2C.
    ds1307_Write(DS1307_ID);
    ds1307_Write(control);
    ds1307_Write(0x00);    // Cấu hình thanh ghi điều khiển.
```

```
    i2c_Stop();                         // Kết thúc giao tiếp I2C.
}
void ds1307_Write(unsigned char dat)  // Gửi 1 byte dữ liệu qua I2C.
{
    i2c_Write(dat);
    i2c_Clock();                        // Gửi tín hiệu xung clk để xác nhận.
}
unsigned char ds1307_Read()           // Đọc dữ liệu từ DS1307 qua I2C.
{
    unsigned char dat;
    dat = i2c_Read();
    return(dat);
}
void ds1307_SetTime(unsigned char hh, unsigned char mm, unsigned char ss)
// Cài đặt thời gian (giờ, phút, giây) cho DS1307.
{
    i2c_Start();
    ds1307_Write(DS1307_ID);
    ds1307_Write(SEC_ADDRESS);
    ds1307_Write(ss);
    ds1307_Write(mm);
    ds1307_Write(hh);
    i2c_Stop();
}
void ds1307_SetTimeAuto(void)
// Đặt tự động (giá trị cố định hoặc mặc định).
{
    i2c_Start();
    ds1307_Write(DS1307_ID);
```

```
    ds1307_Write(SEC_ADDRESS);
    i2c_Stop();
}
void ds1307_SetDate(unsigned char dd, unsigned char mm, unsigned char yy)
// Cài đặt ngày (ngày, tháng, năm) cho DS1307.
{
    i2c_Start();
    ds1307_Write(DS1307_ID);
    ds1307_Write(DATE_ADDRESS);
    ds1307_Write(dd);
    ds1307_Write(mm);
    ds1307_Write(yy);
    i2c_Stop();
}
void ds1307_SetDateAuto(void)
// Đặt ngày tự động (giá trị cố định hoặc mặc định).
{
    i2c_Start();
    ds1307_Write(DS1307_ID);
    ds1307_Write(DATE_ADDRESS);
    i2c_Stop();
}
void ds1307_GetTime(unsigned char *h_ptr,unsigned char *m_ptr,unsigned char
                                                                    *s_ptr)
// Đọc giờ, phút, giây từ DS1307.
{
    i2c_Start();
    ds1307_Write(DS1307_ID);
    ds1307_Write(SEC_ADDRESS);
```

```
    i2c_Stop();

    i2c_Start();
    ds1307_Write(0xD1);
    *s_ptr = ds1307_Read(); i2c_Ack();
    *m_ptr = ds1307_Read(); i2c_Ack();
    *h_ptr = ds1307_Read(); i2c_NoAck();
    i2c_Stop();
}
void ds1307_GetDate(unsigned char *d_ptr,unsigned char *m_ptr,unsigned char
                                                                    *y_ptr)
// Đọc ngày, tháng, năm từ DS1307.
{
    i2c_Start();
    ds1307_Write(DS1307_ID);
    ds1307_Write(DATE_ADDRESS);
    i2c_Stop();

    i2c_Start();
    ds1307_Write(0xD1);
    *d_ptr = ds1307_Read(); i2c_Ack();
    *m_ptr = ds1307_Read(); i2c_Ack();
    *y_ptr = ds1307_Read(); i2c_NoAck();
    i2c_Stop();
}
```

-ds1307.h-

```
#ifndef __DS1307_H__
#define __DS1307_H__
void ds1307_Init();
```

```
void ds1307_Write(unsigned char dat);
void ds1307_SetTime(unsigned char hh, unsigned char mm, unsigned char ss);
void ds1307_SetDate(unsigned char dd, unsigned char mm, unsigned char yy);
void ds1307_GetTime(unsigned char *h_ptr,unsigned char *m_ptr,unsigned char
                                                                *s_ptr);
void ds1307_GetDate(unsigned char *d_ptr,unsigned char *m_ptr,unsigned char
                                                                *y_ptr);
#endif
```

-i2c.c-

```
#include<reg51.h>
#include "delay.h"
#include "i2c.h"

sbit SCL=P2^0;      // Định nghĩa chân SCL (Clock Line) nối với P2.0
sbit SDA=P2^1;      // Định nghĩa chân SDA (Data Line) nối với P2.1

void i2c_Clock(void)
{
    delay_us(50);   // Đợi một khoảng thời gian
    SCL = 1;        // Kéo SCL lên mức cao (tín hiệu xung clock)

    delay_us(50);   // Giữ mức cao
    SCL = 0;        // Kéo SCL xuống mức thấp (kết thúc xung clock)
}

void i2c_Start()
{
    SCL = 0;        // Kéo SCL xuống thấp để bắt đầu
```

```
    SDA = 1;        // Đặt SDA ở mức cao
    delay_us(50);

    SCL = 1;        // Kéo SCL lên cao
    delay_us(50);

    SDA = 0;        // Kéo SDA xuống thấp, tạo điều kiện bắt đầu
    delay_us(50);

    SCL = 0;        // Kéo SCL xuống thấp để hoàn tất
}
```

-i2c.h-

```
#ifndef __I2C_H__
#define __I2C_H__

void i2c_Clock(void);
void i2c_Start();
void i2c_Stop(void);
void i2c_Write(unsigned char );
unsigned char i2c_Read(void);
void i2c_Ack();
void i2c_NoAck();

#endif
```

-lcd.c-

```
#include<reg51.h>
#include "lcd.h"
#include "delay.h"
```

```
#define dataport P1       // Định nghĩa PORT1 làm cổng dữ liệu cho LCD
sbit rs = dataport^0;     // Chân RS kết nối với P1.0 (chọn thanh ghi: 0 –
                                                    lệnh, 1 - dữ liệu)
sbit rw = dataport^1;     // Chân RW kết nối với P1.1 (chọn chế độ đọc/ghi:
                                                    0 - ghi, 1 - đọc)
sbit en = dataport^2;     // Chân EN kết nối với P1.2 (xung kích hoạt)

/* Định nghĩa thông số LCD */
#define LCDMaxLines 2     // LCD có 2 dòng
#define LCDMaxChars 16    // LCD có 16 ký tự mỗi dòng
#define LineOne 0x80      // Địa chỉ bắt đầu của dòng 1
#define LineTwo 0xC0      // Địa chỉ bắt đầu của dòng 2
#define BlankSpace ' '    // Ký tự khoảng trắng

void LCD_Init()           // Khởi tạo LCD
{
    delay_us(5000);       // Chờ LCD ổn định sau khi khởi động
    lcd_WriteCmd(0x02);   // Khởi tạo LCD ở chế độ 4-bit
    lcd_WriteCmd(0x28);   // Chế độ 4-bit, 2 dòng, 5x7 ký tự
    lcd_WriteCmd(0x0C);   // Bật hiển thị, tắt con trỏ
    lcd_WriteCmd(0x01);   // Xóa màn hình
    lcd_WriteCmd(0x80);   // Đưa con trỏ về đầu dòng 1
}

void lcd_WriteCmd(char a)  // Hàm gửi lệnh đến LCD
{
    dataport = (a & 0xF0); // Gửi nibble cao (4 bit đầu)
    rs = 0;                // Chọn chế độ lệnh
    rw = 0;                // Chọn chế độ ghi
```

```
    en = 1;                 // Tạo xung kích hoạt
    delay_us(1);
    en = 0;

    dataport = ((a << 4) & 0xF0); // Gửi nibble thấp (4 bit cuối)
    rs = 0;
    rw = 0;
    en = 1;
    delay_us(1);
    en = 0;
}

void lcd_Writedata(char a)      // Hàm gửi dữ liệu đến LCD
{
    dataport = (a & 0xF0);      // Gửi nibble cao
    rs = 1;                     // Chọn chế độ dữ liệu
    rw = 0;                     // Chọn chế độ ghi
    en = 1;                     // Tạo xung kích hoạt
    delay_us(1);
    en = 0;

    dataport = ((a << 4) & 0xF0); // Gửi nibble thấp
    rs = 1;
    rw = 0;
    en = 1;
    delay_us(1);
    en = 0;
}
```

```
void lcd_GoToLineTwo()
{
    lcd_WriteCmd(LineTwo); // Di chuyển con trỏ đến dòng 2, cột 1
}

void lcd_GoToXY(char row, char col){  // Hàm chuyển con trỏ đến vị trí X (dòng), Y (cột)
    char pos;
    if(row < LCDMaxLines)             // Kiểm tra dòng hợp lệ
    {
        pos = LineOne | (row << 6);   // Xác định dòng
        if(col < LCDMaxChars)         // Kiểm tra cột hợp lệ
            pos = pos + col;          // Xác định vị trí
        lcd_WriteCmd(pos);            // Di chuyển con trỏ
    }
}

void lcd_DisplayString(char *string_ptr) // Hiển thị chuỗi ký tự
{
    while(*string_ptr)                // Lặp qua từng ký tự của chuỗi
        lcd_Writedata(*string_ptr++); // Hiển thị từng ký tự
}

void lcd_DisplayRtcTime(char hour, char min, char sec) {
// Hàm hiển thị thời gian từ RTC
    lcd_Writedata(((hour >> 4) & 0x0F) + 0x30); // Hiển thị hàng chục giờ
    lcd_Writedata((hour & 0x0F) + 0x30);        // Hiển thị hàng đơn vị giờ
    lcd_Writedata(':');
```

```
    lcd_Writedata(((min >> 4) & 0x0F) + 0x30);  // Hiển thị hàng chục phút
    lcd_Writedata((min & 0x0F) + 0x30);         // Hiển thị hàng đơn vị phút
    lcd_Writedata(':');

    lcd_Writedata(((sec >> 4) & 0x0F) + 0x30);  // Hiển thị hàng chục giây
    lcd_Writedata((sec & 0x0F) + 0x30);         // Hiển thị hàng đơn vị giây
}

void lcd_DisplayRtcDate(char day, char month, char year) {
// Hàm hiển thị ngày từ RTC
    lcd_Writedata(((day >> 4) & 0x0F) + 0x30);  // Hiển thị hàng chục ngày
    lcd_Writedata((day & 0x0F) + 0x30);         // Hiển thị hàng đơn vị ngày
    lcd_Writedata('/');

    lcd_Writedata(((month >> 4) & 0x0F) + 0x30); // Hiển thị hàng chục tháng
    lcd_Writedata((month & 0x0F) + 0x30);        // Hiển thị đơn vị tháng
    lcd_Writedata('/');

    lcd_Writedata(((year >> 4) & 0x0F) + 0x30);  // Hiển thị hàng chục năm
    lcd_Writedata((year & 0x0F) + 0x30);         // Hiển thị hàng đơn vị năm
}
```

-lcd.h-

```
#ifndef __LCD_H__
#define __LCD_H__

void lcd_Init();                            // Hàm khởi tạo LCD
void lcd_WriteCmd( char cmd);               // Hàm gửi lệnh đến LCD
void lcd_Writedata( char a);                // Hàm gửi dữ liệu đến LCD
```

```
void lcd_DisplayString(char *string_ptr);   // Hiển thị chuỗi ký tự
void lcd_DisplayNumber(unsigned int num);
void lcd_ScrollMessage(char *msg_ptr);
void lcd_DisplayRtcTime(char hour,char min,char sec);
                                            // Hàm hiển thị thời gian từ RTC
void lcd_DisplayRtcDate(char day,char month,char year);
                                            // Hàm hiển thị ngày từ RTC
void lcd_Clear();
void lcd_GoToLineOne();
void lcd_GoToLineTwo();
void lcd_GoToXY(char row, char col);        // Hàm chuyển con trỏ đến vị trí
                                        X (dòng), Y (cột)
#endif
```

4. Lập trình mô phỏng trên Kit 8051 pro đèn giao thông có đếm ngược

-main.c-

```
#include <AT89X52.h>

sbit SW_chedo = P3 ^ 2;          // Chân P3.2 nhận tín hiệu chuyển chế độ.
sbit SW = P3 ^ 3;                // Chân P3.3 nhận tín hiệu chuyển hướng.

#define Display P0   // Cổng P0 dùng để xuất dữ liệu điều khiển LED 7 đoạn.
#define chonLED P2   // Cổng P2 điều khiển chọn LED 7 đoạn thông qua các bit
                                                        P2.4, P2.3, P2.2.
int chedo = 1;        // Biến lưu trạng thái chế độ (1: thủ công, 0: tự động).
int chuyenhuong = 1; // Biến lưu trạng thái hướng (1:đi lên, 0:đi xuống).
int chuyenVang = 0;  // Biến lưu trạng thái đèn vàng chuyển đổi.
unsigned char code Code7segCatot[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d,0x7d,
0x07, 0x7f, 0x6f, 0x77};
```

```
// Bảng mã hiển thị số trên LED 7 đoạn (Cathode chung).
/*du lieu trang thai den*/
/*4 trang thai*/
/*do0 sang, vang0 tat, xanh0 tat, do1 tat, vang1 tat, xanh1 sang*/
/*do0 sang, vang0 tat, xanh0 tat, do1 tat, vang1 sang, xanh1 tat*/
/*do0 tat, vang0 tat, xanh0 sang, do1 sang, vang1 tat, xanh1 tat*/
/*do0 tat, vang0 sang, xanh0 tat, do1 sang, vang1 tat, xanh1 tat*/

unsigned char code StatusLED[] = {0x81, 0x82, 0x24, 0x44};
// Trạng thái của các đèn giao thông (Đỏ, Vàng, Xanh).

unsigned char code BNsochay[] = {0x00, 0x18, 0x04, 0x1c},
                  DTsochay[] = {0x18, 0x00, 0x1c, 0x04};
// Dữ liệu chọn LED 7 đoạn cho hai hướng đi lên và đi xuống.

void delay_ms(unsigned int time) {
  // Tạo độ trễ bằng Timer 1 (1ms).
  while (time--) {
    TMOD = 0x01;   // Chế độ timer 16 bit.
    TH1 = 0xE8;    // Giá trị nạp để tạo trễ 1ms.
    TL1 = 0x90;
    TR1 = 1;       // Kích hoạt Timer 1.
    while (!TF1);  // Đợi cờ tràn TF1.
    TF1 = 0;       // Xóa cờ tràn.
  }
}

void HienThiDen(char Status) {
  // Điều khiển trạng thái đèn giao thông qua IC 74HC595.
```

```
  int i;
  P3_6 = 0;  // Đặt mức thấp cho chân clock.
  P3_5 = 0;  // Đặt mức thấp cho chân chốt dữ liệu.
  for (i = 0; i < 8; i++) {
    P3_4 = Status >> 7;  // Gửi từng bit của trạng thái đèn.
    Status <<= 1;        // Dịch trái để gửi bit tiếp theo.
    delay_ms(1);
    P3_6 = 1;  // Tạo cạnh lên cho clock.
    delay_ms(1);
    P3_6 = 0;
  }
  P3_5 = 1;  // Tạo cạnh lên để xuất dữ liệu ra.
}

void HienThiSoChay(char *Huong, int demA, int demB) {
  // Hiển thị số đếm chạy ngược (dùng cho Đỏ/Xanh).
  for (int dem = demA; dem >= demB; dem--) {
    if (chedo == 1) return; // Thoát nếu chuyển sang chế độ thủ công.
    for (int tanso = 0; tanso < 60; tanso++) {
      chonLED = Huong[0];
      Display = Code7segCatot[dem % 10];  // Hàng đơn vị.
      delay_ms(3);
      chonLED = Huong[2];
      Display = Code7segCatot[dem / 10];  // Hàng chục.
      delay_ms(3);
    }
  }
}
```

```
void HienThiSoVang(char *Huong, int demA, int demB) {
  // Hiển thị số đếm cố định (dùng cho đèn Vàng).
  for (int dem = demA; dem >= demB; dem--) {
    if (chedo == 1) return;
    for (int tanso = 0; tanso < 60; tanso++) {
      chonLED = Huong[0];
      Display = Code7segCatot[dem % 10]; // Hàng đơn vị.
      delay_ms(3);
      chonLED = Huong[2];
      Display = Code7segCatot[dem / 10]; // Hàng chục.
      delay_ms(3);
    }
  }
}

void Traffic_Auto(void) {
  // Chương trình điều khiển giao thông tự động.
  HienThiDen(StatusLED[0]);           // Đèn đỏ.
  HienThiSoChay(BNsochay, 30, 3);     // Đếm 30 giây.
  HienThiDen(StatusLED[1]);           // Đèn vàng.
  HienThiSoVang(BNsochay, 3, 0);      // Đếm 3 giây.
  HienThiDen(StatusLED[2]);           // Đèn xanh.
  HienThiSoChay(DTsochay, 30, 3);     // Đếm 30 giây.
  HienThiDen(StatusLED[3]);           // Đèn vàng.
  HienThiSoVang(DTsochay, 3, 0);      // Đếm 3 giây.
}

void Interrupt_Timer0(void) {
  // Cấu hình Timer 0 để tạo ngắt 1ms.
```

```
  TMOD = 0x01;  // Timer 0 chế độ 16 bit.
  TH0 = 0xFC;   // Giá trị nạp để tạo trễ 1ms.
  TL0 = 0x18;
  ET0 = 1;      // Bật ngắt Timer 0.
  EA = 1;       // Cho phép ngắt toàn cục.
  TR0 = 1;      // Kích hoạt Timer 0.
}

void ISR_TIMER(void) interrupt 1 {
  // Xử lý ngắt Timer 0 (kiểm tra nút SW_chedo).
  if (SW_chedo == 0) {
    delay_ms(100); // Chống dội phím.
    if (SW_chedo == 1) chedo = !chedo;
  }
  TH0 = 0xFC; // Nạp lại giá trị trễ 1ms.
  TL0 = 0x18;
}

void main(void) {
  Interrupt_Timer0();  // Bật ngắt kiểm tra chế độ.
  while (1) {
    if (chedo) {  // Chế độ thủ công.
      if (SW == 0) {  // Kiểm tra nút chuyển hướng.
        delay_ms(100); // Chống dội phím.
        if (SW == 1) chuyenhuong = !chuyenhuong;
      }
      if (chuyenhuong) {
        HienThiDen(StatusLED[0]);  // Đèn đỏ.
      } else {
```

```
        HienThiDen(StatusLED[2]);  // Đèn xanh.
      }
    } else {
      Traffic_Auto();  // Chạy chế độ tự động.
    }
  }
}
```

-main.h-

```
#include <AT89X52.h>
/*#define StatusTraffic /*do0, do1, vang0, vang1, xanh0, xanh1*/
sbit SW_chedo = P3^2;
sbit SW = P3^3;
#define Display P0 /*xuat du lieu led 7 doan*/
#define chonLED P2 /*P2.4,P2.3,P2.2 dieukhien 8 led 7 doan*/
```

5. Lập trình mô phỏng trên Kit 8051 pro lịch vạn niên hiển thị ngày, tháng, năm; giờ, phút, giây

-main.c-

```
#include<reg51.h>
#include"ds1302.h"      // Thư viện giao tiếp với module thời gian thực DS1302
#define DIG P0           // Cổng P0 dùng để xuất dữ liệu cho LED 7 đoạn
sbit LSA = P2^2;         // Chân điều khiển vị trí LED 7 đoạn (bit A)
sbit LSB = P2^3;         // Chân điều khiển vị trí LED 7 đoạn (bit B)
sbit LSC = P2^4;         // Chân điều khiển vị trí LED 7 đoạn (bit C)
sbit toggle = P3^2;      // Nút chuyển đổi chế độ hiển thị (giờ hoặc ngày)

unsigned char code DIG_CODE[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07, 0x7f, 0x6f};
// Mã hóa các số từ 0-9 cho LED 7 đoạn (chuẩn mã hóa common anode)
```

```
unsigned char Num = 0;   // Biến đếm để xác định LED nào đang được điều khiển
unsigned int disp[8] = {0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f};
// Bộ nhớ lưu dữ liệu hiển thị cho 8 LED 7 đoạn

void main()
{
    Ds1302Init();          // Khởi tạo giao tiếp với DS1302
    Timer0Configuration(); // Cấu hình bộ Timer 0 để điều khiển quét LED

    while(1)
    {
        Ds1302ReadTime();  // Đọc thời gian từ module DS1302

        if(toggle == 1)    // Nếu nút `toggle` được nhấn
        {
            // Hiển thị giờ: phút: giây (hh:mm:ss)
            disp[7] = DIG_CODE[TIME[0] & 0x0f];  // Hàng đơn vị của giờ
            disp[6] = DIG_CODE[TIME[0] >> 4];    // Hàng chục của giờ
            disp[5] = 0x40;                      // Dấu ':'
            disp[4] = DIG_CODE[TIME[1] & 0x0f];  // Hàng đơn vị của phút
            disp[3] = DIG_CODE[TIME[1] >> 4];    // Hàng chục của phút
            disp[2] = 0x40;                      // Dấu ':'
            disp[1] = DIG_CODE[TIME[2] & 0x0f];  // Hàng đơn vị của giây
            disp[0] = DIG_CODE[TIME[2] >> 4];    // Hàng chục của giây
        } else {
            // Hiển thị ngày: tháng: năm (dd:mm:yy)
            disp[7] = DIG_CODE[TIME[6] & 0x0f];  // Hàng đơn vị của ngày
            disp[6] = DIG_CODE[TIME[6] >> 4];    // Hàng chục của ngày
```

```
            disp[5] = 0x40;                      // Dấu '/'
            disp[4] = DIG_CODE[TIME[4] & 0x0f];  // Hàng đơn vị của tháng
            disp[3] = DIG_CODE[TIME[4] >> 4];    // Hàng chục của tháng
            disp[2] = 0x40;                      // Dấu '/'
            disp[1] = DIG_CODE[TIME[3] & 0x0f];  // Hàng đơn vị của năm
            disp[0] = DIG_CODE[TIME[3] >> 4];    // Hàng chục của năm
        }
    }
}
void Timer0Configuration() //Cấu hình Timer0
{
    TMOD = 0x02;    // Chế độ 2: Tự động nạp lại
    TH0 = 0xFE;     // Giá trị khởi tạo cho Timer 0 (tạo ngắt định kỳ nhanh)
    TL0 = 0xFE;     // Giá trị thấp ban đầu
    ET0 = 1;        // Cho phép ngắt Timer 0
    EA = 1;         // Cho phép tất cả các ngắt
    TR0 = 1;        // Bật Timer 0
}

void DigDisplay() interrupt 1     // Hàm ngắt hiển thị LED
{
    DIG = 0;         // Tắt LED trước khi cập nhật dữ liệu

    // Chọn LED tương ứng thông qua LSA, LSB, LSC
    switch(Num)
    {
        case 7: LSA = 0; LSB = 0; LSC = 0; break;
        case 6: LSA = 1; LSB = 0; LSC = 0; break;
        case 5: LSA = 0; LSB = 1; LSC = 0; break;
```

```
        case 4: LSA = 1; LSB = 1; LSC = 0; break;
        case 3: LSA = 0; LSB = 0; LSC = 1; break;
        case 2: LSA = 1; LSB = 0; LSC = 1; break;
        case 1: LSA = 0; LSB = 1; LSC = 1; break;
        case 0: LSA = 1; LSB = 1; LSC = 1; break;
    }

    DIG = disp[Num];  // Cập nhật dữ liệu LED hiện tại
    Num++;            // Chuyển sang LED tiếp theo

    if(Num > 7)       // Reset sau khi hoàn thành vòng quét
        Num = 0;
}
```

-ds1302.c-

```
#include "ds1302.h"

uchar code READ_RTC_ADDR[7] = {0x81, 0x83, 0x85, 0x87, 0x89, 0x8b, 0x8d};
```

// Địa chỉ đọc thời gian từ các thanh ghi của DS1302 (giờ, phút, giây, ngày, tháng, năm, thứ).

```
uchar code WRITE_RTC_ADDR[7] = {0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c};
```

// Địa chỉ ghi thời gian vào các thanh ghi của DS1302 (giờ, phút, giây, ngày, tháng, năm, thứ).

```
uchar TIME[7] = {0x00, 0x03, 0x16, 0x07, 0x10, 0x24, 0x24};
```

// Bộ nhớ lưu trữ thời gian hiện tại (giây, phút, giờ, ngày, tháng, năm, thứ) dưới dạng mã BCD.

```
void Ds1302Write(uchar addr, uchar dat)
{
```

```
// Ghi dữ liệu `dat` vào địa chỉ thanh ghi `addr` của DS1302.
EA = 0;          // Tắt ngắt để đảm bảo giao tiếp an toàn.
RST = 0;         // Reset DS1302.
_nop_();         // Chờ ngắn.
SCLK = 0;        // Đặt xung clock về 0.
_nop_();
RST = 1;         // Bật DS1302 để bắt đầu giao tiếp.
_nop_();

// Gửi địa chỉ ghi.
for (uchar n = 0; n < 8; n++) {
    DSIO = addr & 0x01;  // Gửi bit thấp nhất của địa chỉ.
    addr >>= 1;          // Dịch địa chỉ sang phải 1 bit.
    SCLK = 1;            // Tăng xung clock.
    _nop_();
    SCLK = 0;            // Giảm xung clock.
    _nop_();
}

// Gửi dữ liệu ghi.
for (uchar n = 0; n < 8; n++) {
    DSIO = dat & 0x01;  // Gửi bit thấp nhất của dữ liệu.
    dat >>= 1;          // Dịch dữ liệu sang phải 1 bit.
    SCLK = 1;
    _nop_();
    SCLK = 0;
    _nop_();
}
```

```
    RST = 0;         // Kết thúc giao tiếp.
    _nop_();
    EA = 1;          // Bật lại ngắt.
}

uchar Ds1302Read(uchar addr) {
    // Đọc dữ liệu từ địa chỉ thanh ghi `addr` của DS1302.
    uchar n, dat, dat1;
    EA = 0;          // Tắt ngắt.
    RST = 0;         // Reset DS1302.
    _nop_();
    SCLK = 0;        // Đặt xung clock về 0.
    _nop_();
    RST = 1;         // Bật DS1302.
    _nop_();

    // Gửi địa chỉ đọc.
    for (n = 0; n < 8; n++) {
        DSIO = addr & 0x01;  // Gửi bit thấp nhất của địa chỉ.
        addr >>= 1;
        SCLK = 1;           // Tăng xung clock.
        _nop_();
        SCLK = 0;           // Giảm xung clock.
        _nop_();
    }
    _nop_();

    // Đọc dữ liệu từ DS1302.
    for (n = 0; n < 8; n++) {
```

```
        dat1 = DSIO;        // Đọc bit từ DSIO.
        dat = (dat >> 1) | (dat1 << 7);  // Ghép các bit thành byte dữ liệu.
        SCLK = 1;           // Tăng xung clock.
        _nop_();
        SCLK = 0;           // Giảm xung clock.
        _nop_();
    }

    RST = 0;         // Kết thúc giao tiếp.
    _nop_();
    SCLK = 1;
    _nop_();
    DSIO = 0;        // Đặt DSIO ở chế độ cao.
    _nop_();
    DSIO = 1;
    _nop_();
    EA = 1;          // Bật lại ngắt.
    return dat;      // Trả về dữ liệu đọc được.
}
void Ds1302Init() {
    // Khởi tạo DS1302 và thiết lập thời gian ban đầu.
    uchar n;
    Ds1302Write(0x8E, 0x00);  // Tắt bảo vệ ghi.
    for (n = 0; n < 7; n++) {
        Ds1302Write(WRITE_RTC_ADDR[n], TIME[n]);  // Ghi thời gian ban đầu vào
                                                  các thanh ghi.
    }
    Ds1302Write(0x8E, 0x80);  // Bật bảo vệ ghi.
}
```

```
void Ds1302ReadTime() {
    // Đọc thời gian hiện tại từ DS1302 và lưu vào mảng `TIME`.
    uchar n;
    for (n = 0; n < 7; n++) {
        TIME[n] = Ds1302Read(READ_RTC_ADDR[n]);  // Đọc thanh ghi thời gian.
    }
}
```

-ds1302.h-

```
#ifndef __DS1302_H_
#define __DS1302_H_
#include <reg51.h>
#include <intrins.h>  // Thư viện cho các hàm nội tại của 8051, như `_nop_()`.

#ifndef uchar
#define uchar unsigned char
#endif
// Định nghĩa kiểu dữ liệu `uchar` là `unsigned char`.

#ifndef uint
#define uint unsigned int
#endif
// Định nghĩa kiểu dữ liệu `uint` là `unsigned int`.

sbit DSIO = P3^4;  // Chân dữ liệu I/O của DS1302, nối với bit thứ 4 của P3.
sbit RST = P3^5;   // Chân reset của DS1302, được nối với bit thứ 5 của P3.
sbit SCLK = P3^6;  // Chân xung clock (SCLK) của DS1302, được nối với bit thứ
                                                        6 của port P3.
```

```
void Ds1302Write(uchar addr, uchar dat);
// Hàm ghi dữ liệu `dat` vào địa chỉ `addr` của DS1302.

uchar Ds1302Read(uchar addr);
// Hàm đọc dữ liệu từ địa chỉ `addr` của DS1302.

void Ds1302Init();
// Hàm khởi tạo DS1302, thiết lập thời gian ban đầu.

void Ds1302ReadTime();
// Hàm đọc thời gian hiện tại từ DS1302 và lưu vào biến `TIME`.

extern uchar TIME[7];
// Mảng lưu trữ thời gian (giây, phút, giờ, ngày, tháng, năm, thứ) dưới dạng mã BCD.
// Biến này được định nghĩa bên ngoài tệp này, nhưng có thể được sử dụng ở đây nhờ từ khóa `extern`.

#endif
```

6. Lập trình trên Kit 8051 pro demo điều khiển từ xa 3 thiết bị bằng hồng ngoại

```
/*
0xFF30CF: 1
0xFF18E7: 2
0xFF7A85: 3
0xFF10EF: 4
0xFF38C7: 5
0xFF5AA5: 6
*/
```

```
#include <REG51.H>

unsigned char irKey = 0;          // Lưu mã phím đã giải mã
unsigned long bitPattern = 0;     // Lưu mẫu bit nhận được từ tín hiệu IR
unsigned long newKey = 0;         // Lưu phím mới nhận được
unsigned long pre_newKey = 0;     // Lưu phím nhận được trước đó
unsigned char timerValue;         // Giá trị thời gian giữa các xung
unsigned char msCount = 0;        // Biến đếm thời gian (ms)
char pulseCount = 0;              // Đếm số xung IR

void Timer0_Init(void);           // Hàm khởi tạo Timer0
unsigned char Decode_IRKey(unsigned char key); // Hàm giải mã tín hiệu IR

sbit led1 = P2^0;                 // Định nghĩa LED 1 tại P2.0
sbit led2 = P2^2;                 // Định nghĩa LED 2 tại P2.2
sbit led3 = P2^4;                 // Định nghĩa LED 3 tại P2.4

void main(){
    P2 = 0xEA;                    // Cài đặt giá trị mặc định cho cổng P2
    Timer0_Init();                // Khởi tạo Timer0
    EA = 1;                       // Cho phép ngắt toàn cục

    while (1) {
        if ((newKey != 0) && (pre_newKey != newKey)) {
            // Nếu nhận phím mới và khác phím trước đó
            irKey = Decode_IRKey(newKey); // Giải mã phím điều khiển IR

            switch (irKey) {              // Điều khiển LED theo mã phím
                case 1: led1 = 1; break; // Bật LED 1
```

```
                case 2: led2 = 1; break; // Bật LED 2
                case 3: led3 = 1; break; // Bật LED 3
                case 4: led1 = 0; break; // Tắt LED 1
                case 5: led2 = 0; break; // Tắt LED 2
                case 6: led3 = 0; break; // Tắt LED 3
                default: break; // Không làm gì nếu mã phím không hợp lệ
            }
        }
        pre_newKey = newKey;            // Cập nhật phím trước đó
    }
}

unsigned char Decode_IRKey(unsigned char key){
    unsigned char returnValue = 0;
    switch (key) {
        case 0xFF30CF: returnValue = 1; break; // Mã bật LED 1
        case 0xFF18E7: returnValue = 2; break; // Mã bật LED 2
        case 0xFF7A85: returnValue = 3; break; // Mã bật LED 3
        case 0xFF10EF: returnValue = 4; break; // Mã tắt LED 1
        case 0xFF38C7: returnValue = 5; break; // Mã tắt LED 2
        case 0xFF5AA5: returnValue = 6; break; // Mã tắt LED 3
    }
    return returnValue;
}

void timer0_isr() interrupt 1 {       // Xử lý ngắt Timer0
    if (msCount < 50) msCount++;      // Tăng biến đếm nếu chưa đạt 50ms
    TH0 = 0xFC;                       // Nạp lại giá trị Timer0
    TL0 = 0x67;
```

```
}

void externalIntr0_ISR() interrupt 0 { // Xử lý ngắt ngoài (INTO)
    timerValue = msCount;             // Lưu giá trị thời gian giữa các xung
    msCount = 0;                      // Reset đếm thời gian
    TH0 = 0xFC;                       // Nạp lại giá trị Timer0
    TL0 = 0x67;
    pulseCount++;                     // Tăng số xung

    if (timerValue >= 50) {           // Nếu độ rộng xung > 50ms
        pulseCount = -2;              // Đánh dấu S0F và bỏ qua 2 xung đầu
        bitPattern = 0;               // Reset mẫu bit
    } else if ((pulseCount >= 0) && (pulseCount < 32)) { // Lưu 32 bit dữ liệu
        if (timerValue >= 2)          // Nếu xung >= 2ms, ghi logic 1
            bitPattern |= (unsigned long)1 << (31 - pulseCount);
    } else if (pulseCount >= 32) {    // Khi nhận đủ 32 bit
        newKey = bitPattern;          // Lưu mã phím mới
        pulseCount = 0;               // Reset số xung
    }
}
void Timer0_Init() {
    TMOD |= 0x01;                     // Cấu hình Timer0 chế độ 16-bit
    TH0 = 0xFC;                       // Nạp giá trị ban đầu cho Timer0
    TL0 = 0x67;
    TR0 = 1;                          // Bắt đầu Timer0
    ET0 = 1;                          // Bật ngắt Timer0
    IT0 = 1;                          // Cấu hình ngắt ngoài cạnh xuống
    EX0 = 1;                          // Bật ngắt ngoài INTO
}
```

7. Lập trình trên Kit 8051 pro demo Smarthome (điều khiển thiết bị, đóng ngắt đèn tự động, hẹn giờ mở thiết bị)

-main.c-

```
#include "reg51.h"
#include "XPT2046.h"        // Thư viện giao tiếp với module ADC XPT2046
/*
0xFF30CF: 1                 // Mã hồng ngoại cho phím 1
0xFF18E7: 2                 // Mã hồng ngoại cho phím 2
*/
sbit button = P3^3;         // Nút nhấn kết nối chân P3.3
#define Display P0          // LED 7 đoạn kết nối với cổng P0

// Mã hiển thị số 0-9 và chữ A-F trên LED 7 đoạn (cực âm chung)
unsigned char code Code7segCatot[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d,
                                      0x7d, 0x07, 0x7f, 0x6f, 0x77};
//----------- Biến lưu thời gian hoặc dữ liệu mẫu-------------
unsigned char Time[6] = {0x10, 0x01, 0x16, 0x24, 0x10, 0x07};

// -------Các biến dùng trong xử lý tín hiệu hồng ngoại-------
unsigned char irKey = 0;
unsigned long bitPattern = 0;
unsigned long newKey = 0;
unsigned long pre_newKey = 0;
// -----------Biến dùng trong xử lý timer và đếm--------------
unsigned char timerValue;
unsigned char msCount = 0;
// -----------Biến dùng trong đọc ADC và đếm xung-------------
uint temp, count;
char pulseCount = 0;
```

```
// -----------------Khai báo hàm-----------------
void Timer0_Init(void);
unsigned char Decode_IRKey(unsigned char key);
void Timer1_Init(void);
void delay(int time);
void HienThiSoChay(int demA, int demB);

// --------------Định nghĩa các chân LED-------------
sbit led1 = P2^0;               // LED 1 kết nối P2.0
sbit led2 = P2^2;               // LED 2 kết nối P2.2
sbit led3 = P2^4;               // LED 3 kết nối P2.4

void main(void){
    P2 = 0xff;                  // Cấu hình tất cả chân cổng P2 làm đầu ra
    Timer0_Init();              // Khởi tạo Timer 0
    EA = 1;                     // Bật ngắt toàn cục
    while (1) {
        // Đọc giá trị từ ADC XPT2046 sau mỗi 50 chu kỳ
        if (count == 50) {
            count = 0;
            temp = Read_AD_Data(0xA4); // Đọc dữ liệu ADC từ kênh 0xA4
        }
        count++;
        // Điều khiển led2 theo giá trị ADC
        if (temp <= 10)
            led2 = 0;           // Tắt LED 2 nếu giá trị ADC <= 10
        else
            led2 = 1;           // Bật LED 2 nếu giá trị ADC > 10
```

```
        // Xử lý tín hiệu hồng ngoại
        if ((newKey != 0) && (pre_newKey != newKey)) {
            irKey = Decode_IRKey(newKey);   // Giải mã tín hiệu hồng ngoại
            switch (irKey) {
                case 1:
                    led1 = 1;               // Bật LED 1 khi nhận mã IR là 1
                    break;
                case 2:
                    led1 = 0;               // Tắt LED 1 khi nhận mã IR là 2
                default:
                    break;
            }
        }
        pre_newKey = newKey;                // Lưu giá trị mã IR mới nhất
        // Xử lý nút nhấn
        if (button == 1)
            delay(200);                     // Chống dội nút
        if (button == 0)
            HienThiSoChay(5, 0);  // Hiển thị số từ 5 đến 0 trên LED 7 đoạn
    }
}
void HienThiSoChay(int demA, int demB) {
    int dem;
    for (dem = demA; dem >= demB; dem--) {
        Display = Code7segCatot[dem];  // Hiển thị số trên LED 7 đoạn
        delay(100000);                 // Thời gian chờ giữa các số
    }
    led3 = ~led3;                     // Đảo trạng thái LED 3 sau mỗi chu kỳ
}
```

```
void delay(int time) {
    while (--time);                    // Vòng lặp tạo trễ
}

unsigned char Decode_IRKey(unsigned char key){
    unsigned char returnValue = 0;
    switch (key)
    {
        case 0xFF30CF:                 // Mã IR tương ứng phím 1
            returnValue = 1;
            break;
        case 0xFF18E7:                 // Mã IR tương ứng phím 2
            returnValue = 2;
            break;
    }
    return returnValue;                // Trả về giá trị giải mã
}

// Ngắt Timer 0
void timer0_isr() interrupt 1 {
    if (msCount < 50)
        msCount++;                     // Tăng biến đếm msCount
    TH0 = 0xFC;                        // Reload giá trị ban đầu
    TL0 = 0x67;
}

// Ngắt ngoài 0 để xử lý tín hiệu IR
void externalIntr0_ISR() interrupt 0 {
    timerValue = msCount;              // Lưu thời gian giữa các xung
```

```
    msCount = 0;                        // Reset msCount
    TH0 = 0xFC;                         // Reload giá trị Timer 0
    TL0 = 0x67;
    pulseCount++;                       // Đếm số xung nhận được

    // Kiểm tra và xử lý bit tín hiệu
    if (timerValue >= 50) {             // Reset khi thời gian vượt ngưỡng
        pulseCount = -2;
        bitPattern = 0;
    } else if (pulseCount >= 0 && pulseCount < 32) {
        if (timerValue >= 2)            // Ghi nhận bit tín hiệu
            bitPattern |= (unsigned long)1 << (31 - pulseCount);
    } else if (pulseCount >= 32) {      // Khi nhận đủ 32 bit
        newKey = bitPattern;            // Lưu mã IR vào newKey
        pulseCount = 0;
    }
}

// Khởi tạo Timer 0
void Timer0_Init() {
    TMOD |= 0x01;                       // Cấu hình Timer 0 ở chế độ 16-bit
    TH0 = 0xFC;                         // Giá trị reload
    TL0 = 0x67;
    TR0 = 1;                            // Bật Timer 0
    ET0 = 1;                            // Bật ngắt Timer 0
    IT0 = 1;                            // Ngắt ngoài 0 sườn âm
    EX0 = 1;                            // Bật ngắt ngoài 0
}
```

-XPT2046.c-

```
#include "XPT2046.h"        // Thư viện giao tiếp với module ADC XPT2046

// Hàm khởi động giao thức SPI
void SPI_Start(void)
{
    CLK = 0;                // Đặt xung clock về mức thấp
    CS  = 1;                // Đưa chân chip select về mức cao
    DIN = 1;                // Đặt chân dữ liệu đầu vào về mức cao
    CLK = 1;                // Đưa xung clock lên mức cao
    CS  = 0;                // Kích hoạt thiết bị ngoại vi bằng cách đưa CS=0
}

// Hàm ghi một byte dữ liệu qua giao thức SPI
void SPI_Write(uchar dat)
{
    uchar i;
    CLK = 0;                 // Đặt xung clock về mức thấp
    for (i = 0; i < 8; i++)  // Lặp qua 8 bit của dữ liệu
    {
        DIN = dat >> 7;      // Gửi bit cao nhất của byte dữ liệu
        dat <<= 1;           // Dịch trái dữ liệu để chuẩn bị gửi bit tiếp theo
        CLK = 0;         // Đặt CLK về mức thấp để chuẩn bị cho xung tiếp theo

        CLK = 1;         // Đưa xung clock lên mức cao để xác nhận bit đã gửi
    }
}
```

```
// Hàm đọc dữ liệu 12 bit từ thiết bị qua giao thức SPI
uint SPI_Read(void)
{
    uint i, dat = 0;
    CLK = 0;                    // Đặt xung clock về mức thấp
    for (i = 0; i < 12; i++)    // Lặp qua 12 bit để đọc dữ liệu
    {
        dat <<= 1;      // Dịch trái để chuẩn bị nhận bit mới

        CLK = 1;        // Đưa xung clock lên mức cao để lấy bit từ DOUT
        CLK = 0;        // Đặt CLK về mức thấp để chuẩn bị cho bit tiếp theo

        dat |= DOUT;    // Ghi nhận bit từ chân dữ liệu đầu ra (DOUT)
    }
    return dat;         // Trả về giá trị đã đọc
}

// Hàm đọc giá trị ADC từ XPT2046 với lệnh chỉ định
uint Read_AD_Data(uchar cmd)
{
    uchar i;
    uint AD_Value;
    CLK = 0;          // Đặt xung clock về mức thấp
    CS  = 0;          // Kích hoạt thiết bị bằng cách đưa CS về mức thấp
    SPI_Write(cmd);   // Gửi lệnh chỉ định đến ADC

    for (i = 6; i > 0; i--);  // Trễ một khoảng ngắn đảm bảo tín hiệu ổn định

    CLK = 1;          // Đưa xung clock lên mức cao
```

```
    _nop_();          // Lệnh không làm gì, giúp tạo trễ nhỏ
    _nop_();
    CLK = 0;          // Đặt xung clock về mức thấp
    _nop_();
    _nop_();
    AD_Value = SPI_Read();  // Đọc giá trị ADC từ thiết bị
    CS = 1;           // Ngắt kích hoạt thiết bị bằng cách đưa CS về mức cao
    return AD_Value;  // Trả về giá trị ADC đã đọc
}
```

-XPT2046.h-

```
#ifndef __XPT2046_H_
#define __XPT2046_H_  // Định nghĩa để tránh việc bao gồm nhiều lần file header
//--- Các thư viện cần thiết ---//
#include <reg51.h>
#include <intrins.h>    // Thư viện cung cấp các hàm nội tại của vi điều khiển

//--- Định nghĩa kiểu dữ liệu để tiện sử dụng ---//
#ifndef uchar
#define uchar unsigned char  // Định nghĩa kiểu `uchar` là `unsigned char`
#endif

#ifndef uint
#define uint unsigned int    // Định nghĩa kiểu `uint` là `unsigned int`
#endif

#ifndef ulong
#define ulong unsigned long  // Định nghĩa kiểu `ulong` là `unsigned long`
#endif
```

```
//--- Định nghĩa các chân IO sử dụng cho giao tiếp SPI ---//
sbit DOUT = P3^7;  // Chân xuất dữ liệu từ ADC
sbit CLK  = P3^6;  // Chân xung clock để điều khiển truyền/nhận dữ liệu
sbit DIN  = P3^4;  // Chân nhận dữ liệu từ vi điều khiển
sbit CS   = P3^5;  // Chân chip select để kích hoạt hoặc ngắt ADC

//--- Khai báo các hàm giao tiếp với module XPT2046 ---//
uint Read_AD_Data(uchar cmd);  // Hàm đọc dữ liệu từ ADC với lệnh chỉ định
uint SPI_Read(void);           // Hàm đọc dữ liệu từ ADC qua giao thức SPI
void SPI_Write(uchar dat);     // Hàm ghi dữ liệu tới ADC qua giao thức SPI

#endif                         // Kết thúc định nghĩa file header
```

8. <u>Bài thi cuối kỳ</u>: Lập trình trên Kit 8051 pro demo game rắn săn mồi (sử dụng led ma trận 8x8, và ma trận phím 4x4)

```
#include <regx52.h>
#include <stdlib.h>        // Thư viện hỗ trợ các hàm ngẫu nhiên và tiện ích

/* Định nghĩa độ khó */
#define TIMER0_COUNTER_MAX 10  //Giớihạn Timer0 để điều khiển tốc độ trò chơi
#define SNAKE_MAX_LENGTH 64    // Độ dài tối đa của con rắn
#define GAME_OVER 1            // Trạng thái trò chơi kết thúc
#define GAME_PLAYING 0         // Trạng thái trò chơi đang chạy

/* Định nghĩa các chân kết nối của IC 74HC595 */
sbit CLOCK = P3^6;             // Chân CLOCK điều khiển tín hiệu đồng hồ
sbit LATCH = P3^5;             // Chân LATCH chốt dữ liệu
sbit DATA = P3^4;              // Chân DATA truyền dữ liệu
```

```
/* Biến đếm cho các bộ đếm thời gian */
unsigned char timer2_counter = 0;          // Biến đếm cho Timer2
unsigned char timer0_counter = 0;          // Biến đếm cho Timer0

/* Bộ đệm hiển thị 8x8 LED */
unsigned char xdata displayBuffer[8] = {0, 0, 0, 0, 0, 0, 0, 0};
// Lưu trạng thái của từng hàng LED

/* Ma trận giá trị của bàn phím 4x4 */
char code keypad[17] = {
    10, 'u', 10, 10,    // Hàng 1
    'l', 's', 'r', 10,  // Hàng 2
    10, 'd', 10, 10,    // Hàng 3
    10, 10, 10, 10,     // Hàng 4
    10                  // Phần dư không dùng
};

/* Cấu trúc biểu diễn 1 điểm trên màn hình */
typedef struct {
    unsigned char x;       // Tọa độ x
    unsigned char y;       // Tọa độ y
} Point;

/* Đối tượng trong trò chơi */
Point xdata snake[SNAKE_MAX_LENGTH];  // Danh sách các điểm tạo thành con rắn
Point apple;                          // Vị trí của quả táo
unsigned char snakeLength = 0;        // Chiều dài của con rắn
unsigned char snakeDirection = 0;     // Hướng di chuyển của con rắn
bit gameState = GAME_OVER;            // Trạng thái trò chơi
```

```
/* Khai báo các hàm trò chơi */
void Snake_Display();                 // Hiển thị con rắn
void Snake_Step();                    // Cập nhật trạng thái con rắn
unsigned char isSnakeHitItself();     // Kiểm tra rắn tự va chạm
void Apple_Create();                  // Tạo quả táo mới
void Apple_Display();                 // Hiển thị quả táo
void Display_Clear();                 // Xóa bộ đệm hiển thị
void Display_Point(Point);            // Hiển thị 1 điểm trên LED
void Display_Point2(unsigned x, unsigned y);  // Hiển thị 1 điểm với tọa độ
void Keypad_Read();                   // Đọc bàn phím
char Keypad_Check();                  // Kiểm tra trạng thái bàn phím
char Keypad_Position(unsigned short value);  // Lấy vị trí nhấn trên bàn phím

/* Các hàm khác */
void Matrix_StartDisplay();           // Bắt đầu hiển thị LED ma trận
void Game_Init();                     // Khởi tạo trò chơi
void Game_Start();                    // Bắt đầu trò chơi
void Game_Stop();                     // Dừng trò chơi

void main () {
    // Biến dùng để xử lý bàn phím
    unsigned char counter2 = 0;
    char key_clicked = 0;             // Phím vừa nhấn
    char pre_key_clicked = 0;         // Phím nhấn trước đó

    /* Thiết lập ban đầu */
    P1 = 0xff;                  // Thiết lập các chân của Port 1 làm đầu vào
    P2 = 0;                     // Xóa dữ liệu Port 2
```

```
    P0 = 0;                         // Xóa dữ liệu Port 0
    EA = 1;                         // Cho phép ngắt toàn cục
    LATCH = 0;                      // Chân LATCH ở mức thấp
    DATA = 0;                       // Chân DATA ở mức thấp
    CLOCK = 0;                      // Chân CLOCK ở mức thấp

    /* Bắt đầu hiển thị LED ma trận */
    Matrix_StartDisplay();

    /* Khởi tạo trò chơi */
    Game_Init();

    /* Vòng lặp chính để xử lý bàn phím */
    while(1) {
        key_clicked = keypad[Keypad_Check()];    // Lấy phím nhấn hiện tại
        if((pre_key_clicked != key_clicked) && (key_clicked != 10)) {
            // Khi trò chơi chưa bắt đầu và nhấn phím 's'
            if ((gameState == GAME_OVER) && (key_clicked == 's')) {
                Game_Init();                    // Khởi tạo lại trò chơi
                Game_Start();                   // Bắt đầu trò chơi
            }
            else if (gameState == GAME_PLAYING) {
                snakeDirection = key_clicked;
                // Cập nhật hướng di chuyển của rắn
            }
        }
        pre_key_clicked = key_clicked;          // Lưu phím nhấn trước đó
    }
}
```

```
void Snake_Display(){
    unsigned char counter2 = 0;
    Display_Clear();            // Xóa màn hình hiển thị (các điểm trước đó)
    Apple_Display();            // Hiển thị vị trí của táo

    // Duyệt qua tất cả các phần của rắn và hiển thị chúng
    for(counter2 = 0; counter2 < snakeLength; counter2++){
        // Hiển thị mỗi phần của rắn
        Display_Point(snake[counter2]);
    }
}

void Snake_Step(){
    unsigned char counter2 = 0;
    // Lưu vị trí đuôi rắn
    Point snake_tail;
    snake_tail.x = snake[snakeLength - 1].x;
    snake_tail.y = snake[snakeLength - 1].y;

    switch (snakeDirection)
    {
        // Di chuyển rắn sang phải
        case 'r':
            for(counter2 = snakeLength - 1; counter2 > 0; counter2--){
                // Di chuyển từng phần của rắn từ đuôi về đầu
                snake[counter2].x = snake[counter2 - 1].x;
                snake[counter2].y = snake[counter2 - 1].y;
            }
            // Di chuyển đầu rắn sang phải
```

```
    snake[0].x = snake[0].x + 1;
    if(snake[0].x == 8){
        // Nếu rắn va vào cạnh phải, đưa đầu rắn về phía trái
        snake[0].x = 0;
    }
    break;

// Di chuyển rắn sang trái
case 'l':
    for(counter2 = snakeLength - 1; counter2 > 0; counter2--){
        // Di chuyển từng phần của rắn từ đuôi về đầu
        snake[counter2].x = snake[counter2 - 1].x;
        snake[counter2].y = snake[counter2 - 1].y;
    }
    // Di chuyển đầu rắn sang trái
    snake[0].x--;
    if(snake[0].x == 255){
        // Nếu rắn va vào cạnh trái, đưa đầu rắn về phía phải
        snake[0].x = 7;
    }
    break;

// Di chuyển rắn lên trên
case 'u':
    for(counter2 = snakeLength - 1; counter2 > 0; counter2--){
        // Di chuyển từng phần của rắn từ đuôi về đầu
        snake[counter2].x = snake[counter2 - 1].x;
        snake[counter2].y = snake[counter2 - 1].y;
    }
```

```
        // Di chuyển đầu rắn lên trên
        snake[0].y++;
        if(snake[0].y == 8){
            // Nếu rắn va vào cạnh trên, đưa đầu rắn về phía dưới
            snake[0].y = 0;
        }
        break;

    // Di chuyển rắn xuống dưới
    case 'd':
        for(counter2 = snakeLength - 1; counter2 > 0; counter2--){
            // Di chuyển từng phần của rắn từ đuôi về đầu
            snake[counter2].x = snake[counter2 - 1].x;
            snake[counter2].y = snake[counter2 - 1].y;
        }
        // Di chuyển đầu rắn xuống dưới
        snake[0].y--;
        if(snake[0].y == 255){
            // Nếu rắn va vào cạnh dưới, đưa đầu rắn về phía trên
            snake[0].y = 7;
        }
        break;

    default:
        break;
}

// Kiểm tra nếu rắn ăn được táo
if((snake[0].x == apple.x) && (snake[0].y == apple.y)){
```

```
        // Tăng chiều dài của rắn: thêm phần đuôi mới vào vị trí của đuôi cũ
        snake[snakeLength].x = snake_tail.x;
        snake[snakeLength].y = snake_tail.y;
        snakeLength++;                // Tăng độ dài rắn
        Apple_Create();               // Tạo lại táo ở vị trí mới
    }

    // Kiểm tra nếu rắn đụng phải chính nó
    if(isSnakeHitItself()){
        Game_Stop();                  // Dừng trò chơi nếu rắn tự va vào mình
    }

    // Cập nhật hiển thị của rắn trên màn hình
    Snake_Display();
}

unsigned char isSnakeHitItself() {
    unsigned char returnValue = 0;
    unsigned char counter2 = 0;
    // Duyệt qua tất cả các phần của rắn, bắt đầu từ phần thứ hai

    for(counter2 = 1; counter2 < snakeLength; counter2++){
        // Kiểm tra đầu rắn có trùng vị trí với phần nào khác của rắn không
        returnValue = (snake[0].x == snake[counter2].x) && (snake[0].y == snake[counter2].y);
        if (returnValue) break; // Nếu có va chạm, dừng vòng lặp
    }
    return returnValue; // Trả về 1 nếu rắn va vào chính nó, ngược lại về 0
}
```

```
void Apple_Display(){      // Hiển thị táo tại vị trí hiện tại của nó
    Display_Point(apple);
}

void Apple_Create(){
    unsigned char check = 1;
    unsigned char counter2 = 0;
    // Lặp lại cho đến khi tạo một vị trí táo không trùng với phần nào của rắn
    do
    {
        // Tạo một vị trí ngẫu nhiên cho táo trong phạm vi từ 0 đến 7
        apple.x = ((unsigned int) rand()) % 8;
        apple.y = ((unsigned int) rand()) % 8;

        // Kiểm tra xem táo có bị trùng với bất kỳ phần nào của rắn không
        for (counter2 = 0; counter2 < snakeLength; counter2++)
        {
            check = (apple.x == snake[counter2].x) && (apple.y == snake[counter2].y);
            if(check) break; // Nếu có trùng, thoát khỏi vòng lặp kiểm tra
        }
    }
    // Tiếp tục nếu táo trùng vị trí với rắn
    while (check);
}

void Display_Clear(){
    unsigned char counter2 = 0;
    // Duyệt qua tất cả các vị trí trên màn hình
```

```
    for (counter2 = 0; counter2 < 8; counter2++) {
        // Đặt tất cả các điểm trên màn hình về 0 (tức là làm mới màn hình)
        displayBuffer[counter2] = 0;
    }
}
/* Hiển thị một điểm trên màn hình */
void Display_Point(Point point){
    unsigned char x = point.x;
    unsigned char y = point.y;
    // Cập nhật bộ nhớ đệm hiển thị (displayBuffer) bằng cách đặt bit tại vị trí (x, y) thành 1
    displayBuffer[7 - y] = displayBuffer[7 - y] | (1 << (7 - x));
}

/* Kiểm tra vị trí phím trên bàn phím ma trận */
char Keypad_Check(){
    unsigned short clickPos = 0;
    // Cài đặt P1 với các giá trị để quét các cột của bàn phím
    P1 = 0xef;  // Chọn cột 1
    clickPos = (clickPos << 4) | (P1 & 0x0f); // Lưu trạng thái của cột 1
    P1 = 0xdf;  // Chọn cột 2
    clickPos = (clickPos << 4) | (P1 & 0x0f); // Lưu trạng thái của cột 2
    P1 = 0xbf;  // Chọn cột 3
    clickPos = (clickPos << 4) | (P1 & 0x0f); // Lưu trạng thái của cột 3
    P1 = 0x7f;  // Chọn cột 4
    clickPos = (clickPos << 4) | (P1 & 0x0f); // Lưu trạng thái của cột 4
    // Trả về vị trí của phím đã nhấn
    return Keypad_Position(clickPos);
}
```

```
/* Xác định vị trí phím trong bàn phím ma trận */
char Keypad_Position(unsigned short value){
    char returnvalue = 16;
    // Kiểm tra giá trị của `value` (là trạng thái của bàn phím sau khi quét)
    switch (value & 0xff)
    {
        case 0xf7: returnvalue = 0; break;
        case 0xfb: returnvalue = 1; break;
        case 0xfd: returnvalue = 2; break;
        case 0xfe: returnvalue = 3; break;
        case 0x7f: returnvalue = 4; break;
        case 0xbf: returnvalue = 5; break;
        case 0xdf: returnvalue = 6; break;
        case 0xef: returnvalue = 7; break;
        default:
            value = value >> 8;  // Dịch 8 bit kiểm tra phần trên của `value`
            switch (value & 0xff)
            {
                case 0xf7: returnvalue = 8; break;
                case 0xfb: returnvalue = 9; break;
                case 0xfd: returnvalue = 10; break;
                case 0xfe: returnvalue = 11; break;
                case 0x7f: returnvalue = 12; break;
                case 0xbf: returnvalue = 13; break;
                case 0xdf: returnvalue = 14; break;
                case 0xef: returnvalue = 15; break;
            }
            break;
    }
```

```
    return returnvalue;
}
void Timer0_Isr(void) interrupt 1
{
    TL0 = 0x00;            // Đặt lại giá trị thấp của bộ đếm Timer0
    TH0 = 0x04;            // Đặt lại giá trị cao của bộ đếm Timer0
    timer0_counter++;      // Tăng giá trị của biến đếm Timer0
    if(timer0_counter == TIMER0_COUNTER_MAX){
        timer0_counter = 0;  // Nếu giá trị đếm đạt giới hạn, reset lại bộ đếm
        Snake_Step();      // Gọi hàm xử lý bước di chuyển của rắn
    }
    // Đặt lại giá trị của Timer0 để tiếp tục đếm
}
void Game_Init(){
    snakeLength = 3;                  // Khởi tạo độ dài rắn
    apple.x = 5;                      /* Vị trí của quả táo */
    apple.y = 5;

    snake[2].x = 3;                   /* Vị trí của các phần rắn */
    snake[2].y = 3;
    snake[1].x = 4;
    snake[1].y = 3;
    snake[0].x = 5;
    snake[0].y = 3;
    /* Hướng di chuyển ban đầu */
    snakeDirection = 'r';             // 'r' là hướng phải
    gameState = GAME_OVER;       // Trạng thái trò chơi bắt đầu là GAME_OVER
    Snake_Display();                  // Hiển thị rắn
}
```

```
void Game_Start(void)
{
    TMOD &= 0xF0;          // Thiết lập chế độ hoạt động của Timer0
    TMOD |= 0x01;          // Chọn chế độ 16-bit cho Timer0
    TL0 = 0x00;            // Đặt lại giá trị thấp của Timer0
    TH0 = 0x04;            // Đặt lại giá trị cao của Timer0
    TF0 = 0;               // Xóa cờ ngắt TF0
    TR0 = 1;               // Bắt đầu Timer0

    ET0 = 1;               // Bật ngắt Timer0

    // Trạng thái trò chơi được chuyển sang GAME_PLAYING
    gameState = GAME_PLAYING;
}

void Game_Stop(){
    TR0 = 0;                // Dừng Timer0
    gameState = GAME_OVER;  // Đặt trạng thái trò chơi là GAME_OVER
}

void Timer2_Isr(void) interrupt 5
{
    TF2 = 0;                    // Xóa cờ ngắt Timer2
    DATA = timer2_counter == 0;  // Thiết lập DATA tùy thuộc vào giá trị đếm
    CLOCK = 1;                  // Cập nhật tín hiệu CLOCK
    // Hiển thị một giá trị từ displayBuffer trên cổng P0
    P0 = ~displayBuffer[7 - timer2_counter];
    LATCH = 1;                  // Cập nhật tín hiệu LATCH
    timer2_counter++;           // Tăng giá trị đếm của Timer2
```

```
    // Reset giá trị đếm khi đạt đến 8
    if(timer2_counter == 8) timer2_counter = 0;
    CLOCK = 0;                  // Tắt tín hiệu CLOCK
    LATCH = 0;                  // Tắt tín hiệu LATCH
}

void Matrix_StartDisplay(void)
{
    T2MOD = 0;                  // Đặt lại các thanh ghi điều khiển của Timer2
    T2CON = 0;                  // Đặt lại các thanh ghi cấu hình Timer2
    TL2 = 0xCD;                 // Đặt giá trị thấp của Timer2
    TH2 = 0xF8;                 // Đặt giá trị cao của Timer2
    RCAP2L = 0xCD;              // Đặt giá trị nạp lại thấp của Timer2
    RCAP2H = 0xF8;              // Đặt giá trị nạp lại cao của Timer2
    TR2 = 1;                    // Bật Timer2
    ET2 = 1;                    // Bật ngắt Timer2
}
```