**BÁO CÁO CUỐI KỲ**

|  |  |
| --- | --- |
| Năm học | HK01, NH2024-2025 |
| Môn học | TT Kiến trúc và tổ chức máy tính |
| GVHD | Đậu Trọng Hiển |
| Họ và tên | Nguyễn Thanh Phú |
| MSSV | 22119211 |

# MỤC LỤC

[MỤC LỤC 2](#_Toc182960209)

[BÀI THỰC HÀNH 1 – ĐIỀU KHIỂN 8 LED 4](#_Toc182960210)

[main.c 4](#_Toc182960211)

[BÀI THỰC HÀNH 2 – ĐÈN GIAO THÔNG (SIM) 5](#_Toc182960212)

[Proteus 5](#_Toc182960213)

[main.h 5](#_Toc182960214)

[main.c 8](#_Toc182960215)

[BÀI THỰC HÀNH 3 – LỊCH VẠN NIÊN (SIM) 9](#_Toc182960216)

[Proteus 9](#_Toc182960217)

[LCD16X2\_CMDs.h 9](#_Toc182960218)

[LCD16x2.h 9](#_Toc182960219)

[DS1307.h 11](#_Toc182960220)

[LCD16x2\_DATE\_TIME.h 13](#_Toc182960221)

[main.c 14](#_Toc182960222)

[BÀI THỰC HÀNH 4 – ĐÈN GIAO THÔNG 15](#_Toc182960223)

[main.h 15](#_Toc182960224)

[main.c 18](#_Toc182960225)

[BÀI THỰC HÀNH 5 – LỊCH VẠN NIÊN 20](#_Toc182960226)

[base\_lib.h 20](#_Toc182960227)

[ThreeWiresProtocol.h 20](#_Toc182960228)

[DS1302.h 21](#_Toc182960229)

[Calendar\_OnKit.h 23](#_Toc182960230)

[main.c 25](#_Toc182960231)

[BÀI THỰC HÀNH 6 – ĐIỀU KHIỂN 03 THIẾT BỊ 26](#_Toc182960232)

[Base\_Lib.h 26](#_Toc182960233)

[LED7Seg\_OnKit.h 27](#_Toc182960234)

[IR\_Reading.h 28](#_Toc182960235)

[main.c 31](#_Toc182960236)

[BÀI THỰC HÀNH 7 – NHÀ THÔNG MINH 1 32](#_Toc182960237)

[Utilities.h 32](#_Toc182960238)

[Time.h 33](#_Toc182960239)

[XPT2046.h 34](#_Toc182960240)

[ThreeWiresProtocol.h 35](#_Toc182960241)

[DS1302.h 36](#_Toc182960242)

[IR\_Reading.h 38](#_Toc182960243)

[LED7Seg\_OnKit.h 40](#_Toc182960244)

[main.h 41](#_Toc182960245)

[main.c 46](#_Toc182960246)

[BÀI THỰC HÀNH 8 – NHÀ THÔNG MINH 2 47](#_Toc182960247)

[Utilities.h 47](#_Toc182960248)

[Time.h 47](#_Toc182960249)

[XPT2046.h 47](#_Toc182960250)

[ThreeWiresProtocol.h 47](#_Toc182960251)

[DS1302.h 47](#_Toc182960252)

[LCD\_1602.h 47](#_Toc182960253)

[String\_Ultils.h 50](#_Toc182960254)

[UART\_BLE.h 51](#_Toc182960255)

[main.h 52](#_Toc182960256)

[main.c 59](#_Toc182960257)

# BÀI THỰC HÀNH 1 – ĐIỀU KHIỂN 8 LED

A diagram of a circuit board

Description automatically generated

## main.c

|  |  |
| --- | --- |
| main.c | #include <REGX52.h>  void delay\_ms(int t){      int i;      for( i = 0; i < t\*12; i++);  }  void main(){      int LEDs = 0;      P0 = LEDs;      delay\_ms(500);      do{          LEDs = (LEDs<<1) + 1;          P0 = LEDs;          delay\_ms(500);      }while(LEDs < 0xFF);      do{          LEDs = (LEDs>>1);          P0 = LEDs;          delay\_ms(500);      }while(LEDs > 0);  } |

# BÀI THỰC HÀNH 2 – ĐÈN GIAO THÔNG (SIM)

## Proteus

A screenshot of a computer

Description automatically generated

## main.h

|  |  |
| --- | --- |
| main.h | #include <REGX52.h>  #define RED 0x21  #define YELLOW 0x12  #define GREEN 0xC  #define MANUAL 0x0  #define AUTO 0x1  #define LED\_OFF 0xa  sbit G0  = P3^6;  sbit G1  = P3^7;  sbit AutoIndicator = P3^5;  int Manual\_Auto = AUTO;  int RED\_GREEN\_SET = RED;  int STATE   = RED; // main direction  int COUNT\_0;  int COUNT\_1;  const int DIGIT[] = {0X3F, 0X06, 0X5B, 0X4F, 0X66, 0X6D, 0X7D, 0X07, 0X4F, 0X0};  #define LED P1  // Note: Active-Hight  #define TRAFFIC\_LIGHT P2  //            | traffic light 1 | traffic light 0 |  // P2: [x][x]      [G][Y][R]         [G][Y][R]  // Note: Active-Hight  void delay(unsigned int t){      unsigned int i;      for(i = 0; i < 12\*t; i++);  }  void INITIAL(){      // Config <interrupt enable> for INT0 and INT1      // NOTE:      //      INT0 - Manual/Auto      //      INT1 - RED/GREEN      EX0 = 0x1; //8'b0000\_0101      EX1 = 0x1; //8'b0000\_0101      // Config <interrupt triger> for INT0 and INT1      // NOTE: triggered at falling edge      IT0 = 0x1;      IT1 = 0x1;      // Enable all interrupts according to individual enable bits      EA  = 0x1;      // Turn on AutoIndicator      AutoIndicator = 1;  }  void eINT0\_ACTION(void) interrupt 0 {      Manual\_Auto = (Manual\_Auto==MANUAL)?(AUTO):(MANUAL);      if(Manual\_Auto == AUTO)          AutoIndicator = 0x1;      else          AutoIndicator = 0x0;  }  void eINT1\_ACTION(void) interrupt 2 {      RED\_GREEN\_SET = (RED\_GREEN\_SET==RED)?(GREEN):(RED);  }  void SET\_LED(unsigned int CODE){      //CODE = 0x0 : turn off      LED = DIGIT[CODE];  }  void DISPLAY(unsigned int second){      unsigned int i;      unsigned int k;      unsigned int delay\_t = 10;      for(k = 0; k < second; k++){          for(i = 0; i < 1200/(4\*delay\_t); i++){              G0 = 0; G1 = 0; SET\_LED((COUNT\_0/10)%10);              delay(delay\_t);              G0 = 0; G1 = 1; SET\_LED((COUNT\_1/10)%10);              delay(delay\_t);              G0 = 1; G1 = 0; SET\_LED(COUNT\_0%10);              delay(delay\_t);              G0 = 1; G1 = 1; SET\_LED(COUNT\_1);              delay(delay\_t);          }          if(COUNT\_0 > 0){              COUNT\_0 = COUNT\_0 - 1;          }          if(COUNT\_1 > 0){              COUNT\_1 = COUNT\_1 - 1;          }      }  }  void SET(int CODE){// cho huong chinh      switch (CODE){          // 0: GREEN -> YELLOW -> RED          // 1: RED   -> RED    -> GREEN          case RED:              if (STATE == RED) return;              if (STATE != YELLOW){ // -> GREEN                      TRAFFIC\_LIGHT = 0xA;// -> Change [0] to YELLOW                      COUNT\_0 = 2;                      DISPLAY(2);// wait in 2 secconds              }              TRAFFIC\_LIGHT = RED; //-> change to RED              STATE = RED;              break;            // 0: YELLOW          // 1: YELLOW          case YELLOW:              SET\_LED(10);              TRAFFIC\_LIGHT = 0x12;              STATE = YELLOW;              COUNT\_0 = 0;              COUNT\_1 = 0;              break;            // 0: RED   -> RED    -> GREEN          // 1: GREEN -> YELLOW -> RED          case GREEN:              if (STATE == GREEN) return;              if (STATE != YELLOW){// is RED                      TRAFFIC\_LIGHT = 0x11;//00 010 001                      COUNT\_1 = 2;                      DISPLAY(2);// wait in 2 secconds for Yellow              }              TRAFFIC\_LIGHT = GREEN;              STATE = GREEN;              break;      }  } |

## main.c

|  |  |
| --- | --- |
| main.c | #include "main.h"  void main(){      INITIAL();      delay(1000);      while(1){              if(Manual\_Auto == AUTO){                  SET(RED);                  COUNT\_0 = 7;                  COUNT\_1 = 5;                  DISPLAY(5);                    SET(GREEN);                  COUNT\_0 = 5;                  COUNT\_1 = 7;                  DISPLAY(5);              }else{                  LED = 0;                  if(RED\_GREEN\_SET == RED)                      SET(RED);                  else                      SET(GREEN);                  delay(100);              }      }  } |

# BÀI THỰC HÀNH 3 – LỊCH VẠN NIÊN (SIM)

## Proteus

A diagram of a circuit board

Description automatically generated

## LCD16X2\_CMDs.h

|  |
| --- |
| #define LCD\_ON\_CURSOR\_ON            0x0F  //LCD ON, cursor ON  #define CLEAR\_SCREEN                0x01  //Clear display screen  #define RETURN\_HOME                 0x02  //Return home  #define LEFT\_SHIFT\_CURSOR           0x04  //Decrement cursor (shift cursor to left)  #define RIGHT\_SHIFT\_CURSOR          0x06  //Increment cursor (shift cursor to right)  #define LEFT\_SHIFT\_DISPLAY          0x05  //Shift display right  #define RIGHT\_SHIFT\_DISPLAY         0x07  //Shift display left  #define DISPLAY\_ON\_CURSOR\_BLINKING  0x0E  //Display ON, cursor blinking  #define SET\_CURSOR\_0x\_0y            0x80  //Force cursor to beginning of first line  #define SET\_CURSOR\_1x\_0y            0xC0  //Force cursor to beginning of second line  #define LINEx2\_MAT5x7               0x38  //2 lines and 5×7 matrix  #define CMD11                       0x83  //Cursor line 1 position 3  #define ACTIVATE\_2nd\_LINE           0x3C  //Activate second line  #define LCD\_OFF\_CURSOR\_OFF          0x08  //Display OFF, cursor OFF  #define CMD14                       0xC1  //Jump to second line, position 1  #define LCD\_ON\_CURSOR\_OFF           0x0C  //Display ON, cursor OFF  #define CMD16                       0xC1  //Jump to second line, position 1  #define CMD17                       0xC2  //Jump to second line, position 2 |

## LCD16x2.h

|  |  |
| --- | --- |
| LCD16x2.h | */\**  *This lib was made to interfacing with LCD.*  *LCD's pin informatios:*  *\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_*  *|  ################################   |*  *|  ################################   |*  *| \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ |*  *| | | | | | | | | |  |  |  |  |*  *1 2 3 4 5 6 7 8 9 10 11 12 13 14*  *pin 0     - VSS - Connected to the ground of the MCU/ Power source*  *pin 1     - VDD - Connected to the supply pin of Power source*  *pin 2     - VEE - Connected to a variable POT that can source 0-5V*  *pin 3     - RS  - Toggles between Command/Data Register*  *pin 4     - RW  - Toggles the LCD between Read/Write Operation*  *pin 5     - E   - Must be held high to perform Read/Write Operation*  *Pin 7-14  - D   - Pins used to send Command or data to the LCD.*  *\*/*  *#ifndef* \_LCD16X2\_H\_  *#define* \_LCD16X2\_H\_  *#include* <stdio.h>  *#include* <REGX52.h>  *#include* "LCD16x2\_CMDs.h"  *#define* uint unsigned int  *#define* WRITE\_MODE              0x0  *#define* READ\_MODE               0x1  *#define* SEND\_CMD\_MODE           0x0  *#define* SEND\_DISPLAY\_DATA\_MODE  0x1  *//  The variables bellow can be edited bases on*  *//  your circuit.*  *//  Set your LCD is in receiving command or receriving display data.*  sbit REGISTER\_SELECT        = P1^5;  *//  Set your LCD is READ mode or WRITE mode (usually write mode, be written by your MCU)*  sbit READ\_WRITE             = P1^6;  *//  Enable your LCD by a negedge pulse*  sbit ENABLE                 = P1^7;  *//  Receive or Transfer data (parallel)*  *#define* DATA\_PORT P2  *//  Make MS\_DELAY by do "nothing"*  static void MS\_DELAY(uint *t*){      uint i;  *for*(i = 0; i < 12\**t*; i++);  }  *//  Make a MONO pulse at ENABLE pin*  *//  MONO pulse: LOW->HIGH (HIGH)\*n HIGH->LOW :))*  void ENABLE\_LCD(){  *//Enable, a high to low pulse need to enable the LCD*      ENABLE = 0x1;      MS\_DELAY(3);      ENABLE = 0x0;  }  *//  To sent command to the LCD.*  void SEND\_BYTE\_COMMAND(unsigned char *CMD*){      DATA\_PORT = *CMD*;      REGISTER\_SELECT = SEND\_CMD\_MODE;      READ\_WRITE   = WRITE\_MODE;      ENABLE\_LCD();  }  *//  To sent a byte of DISPLAY DATA to the LCD.*  void SEND\_BYTE\_DISPLAY(unsigned char *BYTE*){  *// NOTE: BYTE is displayed in ASCII.*      DATA\_PORT = *BYTE*;      REGISTER\_SELECT = SEND\_DISPLAY\_DATA\_MODE;      READ\_WRITE = WRITE\_MODE;      ENABLE\_LCD();  }  *//  To sent an array of byte of DISPLAY DATA to the LCD.*  void SEND\_BYTE\_ARRAY\_DISPLAY(unsigned char *ARR*[], uint *SIZE*){      uint i = 0;  *while*( i < *SIZE* ){          SEND\_BYTE\_DISPLAY(*ARR*[i]);          ++i;      }  }  *//  Set the position of the CURSOR in 16x2 LCD screen.*  void SET\_CURSOR\_POS(uint *ROW*, uint *COL*){  *if*(*ROW* == 0){          SEND\_BYTE\_COMMAND(SET\_CURSOR\_0x\_0y+*COL*);      }  *if*(*ROW* == 1){          SEND\_BYTE\_COMMAND(SET\_CURSOR\_1x\_0y+*COL*);      }  }  *//  Set up your LCD.*  void LCD\_INITIAL(){      SEND\_BYTE\_COMMAND(LCD\_ON\_CURSOR\_OFF);      MS\_DELAY(20);      SEND\_BYTE\_COMMAND(LINEx2\_MAT5x7);      MS\_DELAY(20);      SEND\_BYTE\_COMMAND(CLEAR\_SCREEN);  }  *#endif* |

## DS1307.h

|  |  |
| --- | --- |
| DS1307.h | #ifndef \_DS1307\_H\_  #define \_DS1307\_H\_  #include <REGX52.h>  //#include "STACK\_BUFFER.h"  typedef  unsigned int uint;  typedef  unsigned char uchar;  #define logic\_inverse(x) ((x>0)?(0):1)  #define POW2(x) (1U<<(x))  #define bit\_at(x, i) (((x) & (1U<<(i)))?(1):(0))  #define MASK\_8BIT 0xFF  //  |<-ADDR->   R  //  1101\_\_000   0  #define SLAVE\_ADDR\_W 0xD0 //Slave addr of DS1307 <0x68> concat with #R bit <0>  //  |<-ADDR->   W  //  1101\_\_000   1  #define SLAVE\_ADDR\_R 0xD1 //Slave addr of DS1307 <0x68> concat with W bit <1>  #define CONTROL\_REG\_ADDR 0x07  #define SET\_SCL(LOGIC\_STATE) SCL = (LOGIC\_STATE)?(1):(0)  #define SET\_SDA(LOGIC\_STATE) SDA = (LOGIC\_STATE)?(1):(0)  #define I2C\_WRITE\_TO\_ADDR(ADDR)  ((ADDR<<1)|(0x1))  #define I2C\_READ\_FROM\_ADDR(ADDR) ((ADDR<<1)|(0x0))  enum LOGIC\_LEVEL {LOW = 0, HIGH = 1};  enum SLAVE\_STATE  {ACK = 0, NAK  = 1};  // Config two pins suit for your demand.  // I2C pins  sbit SCL = P0^6;  sbit SDA = P0^7;  //              T\_PEAK  // HIGH:        \_\_\_\_\_\_  //             /      \ T\_PEAK /  // LOW :  \_\_\_\_/        \\_\_\_\_\_\_/  uint T\_WAIT = 2;  // Do stuff things to make delay :v  static void DELAY(uint t){      uint i;      for(i = 0; i < 12\*t; i++);  }  // START CONDITION  //             |---->|  // HIGH:     \_\_\_\_\_\_\_\_  //        SCL        |  // LOW :             |\_\_\_\_  // HIGH:     \_\_\_\_\_  //        SDA     |  // LOW :          |\_\_\_\_\_\_\_  void I2C\_START(){      // DELAY(T\_WAIT);      SET\_SCL(HIGH);      SET\_SDA(HIGH);      DELAY(T\_WAIT);      SET\_SDA(LOW);      DELAY(T\_WAIT);      SET\_SCL(LOW);      DELAY(T\_WAIT);  }  // STOP CONDITION  // HIGH:                  \_\_\_\_\_\_  //        SCL            |  // LOW :             \_\_\_\_|  // HIGH:                     \_\_\_  //        SDA               |  // LOW :             \_\_\_\_\_\_\_|  void I2C\_STOP(){      DELAY(T\_WAIT);      SET\_SCL(LOW);      SET\_SDA(LOW);      DELAY(T\_WAIT);      SET\_SCL(HIGH);      DELAY(T\_WAIT);      SET\_SDA(HIGH);  }  //        |<-T1->|<-T2->|  // HIGH   \_\_\_\_\_\_\_ \_\_\_\_\_\_  // SCL:          |      |  // LOW    \_ \_ \_ \_|      |\_\_\_\_\_\_  void SCL\_MONO\_PULSE(){      DELAY(T\_WAIT);      // wait for somethings (T1)      SET\_SCL(HIGH);      // pull to high      SET\_SCL(LOW);  }  uint RECEIVE\_BIT(){      uint BIT\_DATA;      DELAY(T\_WAIT);      SET\_SDA(HIGH);  DELAY(T\_WAIT); // release SDA line      SET\_SCL(HIGH);  DELAY(T\_WAIT);      BIT\_DATA = SDA; SET\_SCL(LOW);      return BIT\_DATA;  }  uint I2C\_SEND\_BYTE(unsigned char DATA){      uint i = 0;      for( i = 0; i < 8; i++){          SET\_SDA( DATA & 0x80 );          SCL\_MONO\_PULSE();          DATA <<= 1;      }      return RECEIVE\_BIT();  }  void SEND\_ACK(){      DELAY(T\_WAIT);      SET\_SDA(LOW);  // pull SDA to low level to indicate ACK.      SCL\_MONO\_PULSE();      SET\_SDA(HIGH); // idle state  }  void SEND\_NAK(){      DELAY(T\_WAIT);      SET\_SDA(HIGH);  // pull SDA to low level to indicate No ACK.      SCL\_MONO\_PULSE();      SET\_SDA(HIGH); // idle state  }  uint I2C\_RECEIVE\_BYTE(uint ACK\_NAK){      uint i = 0, RCV\_DATA = 0;      for(i = 0; i < 8; i++){          DELAY(T\_WAIT);          RCV\_DATA <<= 1;          RCV\_DATA = RCV\_DATA | RECEIVE\_BIT();      }      if( ACK\_NAK == NAK ) SEND\_NAK();      if( ACK\_NAK == ACK) SEND\_ACK();          return RCV\_DATA;  }  void DS1307\_INIT(){      I2C\_START();      I2C\_SEND\_BYTE(SLAVE\_ADDR\_W);      I2C\_SEND\_BYTE(CONTROL\_REG\_ADDR);      I2C\_SEND\_BYTE(0x0); // Disable the SQW/OUT pin.      I2C\_STOP();  }  void DS1307\_READ(uint \*YEAR, uint \*MONTH, uint \*DAY,                  uint \*HOUR, uint \*MINUTE, uint \*SECOND){      I2C\_START();      I2C\_SEND\_BYTE(SLAVE\_ADDR\_W); // Connect to DS1307      I2C\_SEND\_BYTE(0x0);       // Request Sec RAM address at 00H      I2C\_STOP();      DELAY(T\_WAIT);      I2C\_START();      I2C\_SEND\_BYTE(SLAVE\_ADDR\_R); // Connect to DS1307      (\*SECOND) = I2C\_RECEIVE\_BYTE(ACK);      (\*MINUTE) = I2C\_RECEIVE\_BYTE(ACK);      (\*HOUR) = I2C\_RECEIVE\_BYTE(ACK);      I2C\_RECEIVE\_BYTE(ACK);      (\*DAY) = I2C\_RECEIVE\_BYTE(ACK);      (\*MONTH) = I2C\_RECEIVE\_BYTE(ACK);      (\*YEAR) = I2C\_RECEIVE\_BYTE(NAK);      I2C\_STOP();  }  #endif //\_DS1307\_H\_ |

## LCD16x2\_DATE\_TIME.h

|  |  |
| --- | --- |
| LCD16x2\_DATE\_TIME.h | #ifndef \_LCD16X2\_DATE\_TIME\_H\_  #define \_LCD16X2\_DATE\_TIME\_H\_  #include <stdio.h>  #include <REGX52.h>  #include "LCD16x2.h"  #include "DS1307.h"  #define uchar unsigned char  #define uint unsigned int  int DAY = 0;  int MONTH = 0;  int YEAR = 0;  int SECOND = 0;  int MINUTE = 0;  int HOUR   = 0;  char DATE[] ="DATE: YYYY MM DD";  char TIME[] ="TIME: HH:MM:SS";  int  DAYS\_OF\_MON[]  ={-1, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  #define SET\_DD\_MM\_YYYY(DD, MM, YYYY) {DAY = DD%31; MONTH = MM%12; YEAR = YYYY;}  #define SET\_HH\_MM\_SS(HH, MM, SS)    {HOUR = HH%24; MINUTE = MM%60; SECOND = SS%60;}  void GET\_TIME\_DATE(){      //Get date/time from ds1307      DS1307\_READ(&YEAR, &MONTH, &DAY, &HOUR, &MINUTE, &SECOND);      YEAR = (YEAR&0x0F) + (YEAR>>4)&0x0F;      MONTH = (MONTH&0x0F) + ((MONTH>>4)&0x1)\*10;      DAY = (DAY&0x0F) + ((DAY>>4)&0x0F);      HOUR = (HOUR&0xF) + ((HOUR>>4)&0x3);      MINUTE = (MINUTE&0xF) + ((MINUTE>>4)&0x3);      SECOND = (SECOND&0xF) + ((SECOND>>4)&0x3);  }  void FORMAT\_DATE(){      DATE[9]  = (uchar)(YEAR%10) + '0';      DATE[8]  = (uchar)((YEAR/10)%10) + '0';      DATE[7]  = (uchar)((YEAR/100)%10) + '0';      DATE[6]  = (uchar)((YEAR/1000)%10) + '0';      DATE[12] = (uchar)(MONTH%10) + '0';      DATE[11] = (uchar)((MONTH/10)%10) + '0';      DATE[15] = (uchar)(DAY%10) + '0';      DATE[14] = (uchar)((DAY/10)%10) + '0';  }  void FORMAT\_TIME(){      TIME[7] = (uchar)((HOUR/1)%10) + '0';      TIME[6] = (uchar)((HOUR/10)%10) + '0';      TIME[10] = (uchar)((MINUTE/1)%10) + '0';      TIME[9] = (uchar)((MINUTE/10)%10) + '0';      TIME[13] = (uchar)((SECOND/1)%10) + '0';      TIME[12] = (uchar)((SECOND/10)%10) + '0';  }  void DISPLAY(){      SET\_CURSOR\_POS(0, 0);      FORMAT\_DATE();      SEND\_BYTE\_ARRAY\_DISPLAY(DATE, 16);      SET\_CURSOR\_POS(1, 0);      FORMAT\_TIME();      SEND\_BYTE\_ARRAY\_DISPLAY(TIME, 14);  }  #endif |

## main.c

|  |  |
| --- | --- |
| main.c | #include "DS1307.h"  #include "LCD16x2\_DATE\_TIME.h"  void main(){      LCD\_INITIAL();      //SET\_HH\_MM\_SS(23, 59, 55);      //SET\_DD\_MM\_YYYY(3, 2, 2004);      DS1307\_INIT();      DISPLAY();      while(1){          GET\_TIME\_DATE();          DISPLAY();          MS\_DELAY(100);      }  } |

# BÀI THỰC HÀNH 4 – ĐÈN GIAO THÔNG

## main.h

|  |  |
| --- | --- |
| main.h | // #ifdef \_MAIN\_H\_  // #define \_MAIN\_H\_  //---------- Include -----------  #include <REGX52.h>  #include <stdio.h>  //---------- Macros -----------  #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  //RETURN CODE DESCRIPTION:  //  0x1  : RED  //  0x2  : YELLOW  //  0x4  : GREEN  //---------- Type defines -----  typedef unsigned int UINT;  //---------- Delay ------------  static void DELAY\_DISP(UINT mili\_sec) {    UINT i;    for (i = 0; i < 3 \* mili\_sec; i++);  }  static void DELAY(UINT mili\_sec) {    UINT i;    for (i = 0; i < 12 \* mili\_sec; i++)      ;  }  //---------- Traffic Light Ports  // sbit RED0 = P1 ^ 0;  // sbit YELLOW0 = P1 ^ 1;  // sbit GREEN0 = P1 ^ 2;  // sbit RED1 = P2 ^ 5;  // sbit YELLOW1 = P2 ^ 6;  // sbit GREEN1 = P2 ^ 7;  UINT RED0;  UINT YELLOW0;  UINT GREEN0;  UINT RED1;  UINT YELLOW1;  UINT GREEN1;  //---------- Timer Ports --------  sbit GND0 = P2 ^ 2;  sbit GND1 = P2 ^ 3;  sbit GND2 = P2 ^ 4;  #define LED P0  // P2: [x][G][F][E][D][C][B][A]  // Note: Active-Hight | MSB -> LSB | Common Anode  //---------- Timer Ports --------  sbit M\_A = P3^3;  sbit R\_G = P3^2;  //---------- State VARs  UINT STATE\_0 = RED;  UINT STATE\_1 = RED;  // The state of traffic light (to prevent set the same state again)  UINT COUNT\_0;  // Count for the main traffic light  UINT COUNT\_1;  // Count for the order traffic light  UINT SINGLE\_LED\_DISPLAY\_T = 1;  // The time use show a single 7-seg LED  UINT RED\_T = 0;  UINT GREEN\_T = 0;  UINT YELLOW\_T = 0;  // The time (in seccond) for the YELLOW state while changes state  // from GREEN->RED.  //---------- CA LED CODE  const UINT DIGIT\_CODE[] = {0X3F, 0X06, 0X5B, 0X4F, 0X66, 0X6D,                               0X7D, 0X07, 0X7F, 0XEF, 0X0, 0X1,                 0x40, 0x8};  // 7-seg LED CODE (Common Anode)  UINT AUTO\_MANUAL() {    UINT \_M\_A = M\_A;    if (\_M\_A)      return AUTO;    return MANUAL;  }  UINT RED\_GREEN() { return (R\_G) ? (RED) : (GREEN); }  void SET\_LED(UINT D) {    // CODE = 10 : turn off LED.    LED = DIGIT\_CODE[D];  }  void SET\_DISPLAY\_PERIOD(UINT T) {    // NOTE: Unit mili-second    SINGLE\_LED\_DISPLAY\_T = T;  }  void STOP\_COUNT() {    COUNT\_0 = 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){    // RED\_T = GREEN\_T + YELLOW\_T    // NOTE: Delay in second    COUNT\_0 = RED\_T = \_RED\_T;    COUNT\_1 = GREEN\_T = RED\_T - YELLOW\_T;  }  void SET\_TIMER(UINT PREVIOUS) {    // PREVIOUS DESCRIPTION    // PREVIOUS = 0 :    //  Means Traffic Light 0 is currently RED and count    // down to ZERO to change to GREEN. At the same time    // Traffic Light 1 is counting down to ZERO to change    // to YELLOW then it will change to RED.    // PREVIOUS = 1 :    //  The other side, Traffic Light 1 is currently GREEN,    // and will be changed to YELLOW, then RED.    if (PREVIOUS == 0) {      COUNT\_0 = RED\_T;      COUNT\_1 = GREEN\_T;    } else {      COUNT\_0 = GREEN\_T;      COUNT\_1 = RED\_T;    }  }  UINT DIGIT(UINT POS){    if( POS == 0){      if(RED0 == 1 && YELLOW0 == 0 && GREEN0 == 0) return R\_DIGIT;      if(RED0 == 0 && YELLOW0 == 1 && GREEN0 == 0) return Y\_DIGIT;      if(RED0 == 0 && YELLOW0 == 0 && GREEN0 == 1) return G\_DIGIT;    }else{      if(RED1 == 1 && YELLOW1 == 0 && GREEN1 == 0) return R\_DIGIT;      if(RED1 == 0 && YELLOW1 == 1 && GREEN1 == 0) return Y\_DIGIT;      if(RED1 == 0 && YELLOW1 == 0 && GREEN1 == 1) return G\_DIGIT;    }    return 0xA;  }  void DISPLAY\_LED(){    /\*This function only runs ONE SECOND\*/    UINT i = 0;    for(i = 0; i < 7200/(6\*SINGLE\_LED\_DISPLAY\_T); i++){        GND0 = 1; GND1 = 0; GND2 = 0;        if(COUNT\_0 != 0) SET\_LED((COUNT\_0/10)%10);        else SET\_LED(LED\_OFF);        DELAY\_DISP(SINGLE\_LED\_DISPLAY\_T);        GND0 = 1; GND1 = 0; GND2 = 1;        if(COUNT\_1 != 0)SET\_LED((COUNT\_1/10)%10);        else SET\_LED(LED\_OFF);        DELAY\_DISP(SINGLE\_LED\_DISPLAY\_T);          GND0 = 1; GND1 = 1; GND2 = 0; SET\_LED(DIGIT(0));        DELAY\_DISP(SINGLE\_LED\_DISPLAY\_T);          GND0 = 0; GND1 = 0; GND2 = 0;        if(COUNT\_0 != 0) SET\_LED(COUNT\_0%10);        else SET\_LED(LED\_OFF);        DELAY\_DISP(SINGLE\_LED\_DISPLAY\_T);          GND0 = 0; GND1 = 0; GND2 = 1;        if(COUNT\_1 != 0) SET\_LED(COUNT\_1%10);        else SET\_LED(LED\_OFF);        DELAY\_DISP(SINGLE\_LED\_DISPLAY\_T);        GND0 = 1; GND1 = 1; GND2 = 1; SET\_LED(DIGIT(1));        DELAY\_DISP(SINGLE\_LED\_DISPLAY\_T);    }  }  void SET\_TRAFFIC\_LIGHT(UINT POS, UINT CODE) {    // CODE DESCRIPTION:    // RED    : 0x1    // YELLOW : 0x2    // GREEN  : 0x4    // POSITION DESCRIPTION:    //    0   : Traffic light 0    //    1   : Traffic light 1    // POS = 0 --> Traffic light 0 --> 3 bits control locates at 3 last bit.    //> MSB [x][x][x][x] [x][G][Y][R] LSB    // POS = 1 --> Traffic light 1 --> 3 bits control loacates from 5th bit down    // to 3rd bit. >   MSB [x][x][G][Y] [R][x][x][x] LSB    if (POS)      CODE = (CODE << 3)&0x38;    // NOTE: MASK = 0011\_1000 in BIN equiv 0x38 in HEX    switch (CODE) {    case 0x01: //0000\_0001      RED0 = 1, YELLOW0 = 0, GREEN0 = 0, STATE\_0 = RED;      break;    case 0x02: //0000\_0010      RED0 = 0, YELLOW0 = 1, GREEN0 = 0, STATE\_0 = YELLOW;      break;    case 0x04: //0000\_0100      RED0 = 0, YELLOW0 = 0, GREEN0 = 1, STATE\_0 = GREEN;      break;    case 0x08: //0000\_1000      RED1 = 1, YELLOW1 = 0, GREEN1 = 0, STATE\_1 = RED;      break;    case 0x10: //0001\_0000      RED1 = 0, YELLOW1 = 1, GREEN1 = 0, STATE\_1 = YELLOW;      break;    case 0x20: //0010\_0000      RED1 = 0, YELLOW1 = 0, GREEN1 = 1, STATE\_1 = GREEN;      break;    }  }  UINT GET\_STATE(UINT POS){    return (POS)?(STATE\_1):(STATE\_0);  }  void SET\_STATE(UINT CODE) {    if (CODE == RED) {      if (GET\_STATE(0) == YELLOW) {        SET\_TRAFFIC\_LIGHT(0, RED);        SET\_TRAFFIC\_LIGHT(1, GREEN);      }      elif (GET\_STATE(0) == GREEN) {        while (COUNT\_0) {          DISPLAY\_LED();          DECREASE\_ONE(COUNT\_0);          DECREASE\_ONE(COUNT\_1);        }        SET\_TRAFFIC\_LIGHT(0, YELLOW);        SET\_TRAFFIC\_LIGHT(1, RED);        COUNT\_0 = YELLOW\_T;        while (COUNT\_0) {          DISPLAY\_LED();          DECREASE\_ONE(COUNT\_0);          DECREASE\_ONE(COUNT\_1);        }        SET\_TRAFFIC\_LIGHT(0, RED);        SET\_TRAFFIC\_LIGHT(1, GREEN);      }    }    if (CODE == GREEN) {      if (GET\_STATE(0) == YELLOW) {        SET\_TRAFFIC\_LIGHT(0, GREEN);        SET\_TRAFFIC\_LIGHT(1, RED);      }      elif (GET\_STATE(0) == RED) {        while (COUNT\_1) {          DISPLAY\_LED();          DECREASE\_ONE(COUNT\_0);          DECREASE\_ONE(COUNT\_1);        }        COUNT\_1 = YELLOW\_T;        SET\_TRAFFIC\_LIGHT(0, RED);        SET\_TRAFFIC\_LIGHT(1, YELLOW);        while (COUNT\_1) {          DISPLAY\_LED();          DECREASE\_ONE(COUNT\_0);          DECREASE\_ONE(COUNT\_1);        }        SET\_TRAFFIC\_LIGHT(0, GREEN);        SET\_TRAFFIC\_LIGHT(1, RED);      }    }  }  void INITIAL(){    /\*Set initial state\*/    SET\_TRAFFIC\_LIGHT(0, YELLOW);    SET\_TRAFFIC\_LIGHT(1, YELLOW);    COUNT\_0 = 0x0;    COUNT\_1 = 0x0;    LED = DIGIT\_CODE[LED\_OFF];    GND0 = GND1 = GND2 = 0;  }  // #endif |

## main.c

|  |  |
| --- | --- |
| main.c | #include "main.h"  void main(){        INITIAL();      SET\_YELLOW\_TIMER(5);      SET\_RED\_GREEN\_TIMER(17);      SET\_DISPLAY\_PERIOD(12);      while ( 0x1 ){          if(!AUTO\_MANUAL() ){              SET\_STATE(RED);              SET\_TIMER(0);              SET\_STATE(GREEN);              SET\_TIMER(1);          }else{              STOP\_COUNT();              while( !AUTO\_MANUAL() == 0x0 ){                  SET\_STATE(RED\_GREEN());                  GND0 = 1; GND1 = 1; GND2 = 0; SET\_LED(DIGIT(0));                  DELAY(SINGLE\_LED\_DISPLAY\_T);                  GND0 = 1; GND1 = 1; GND2 = 1; SET\_LED(DIGIT(1));                  DELAY(SINGLE\_LED\_DISPLAY\_T);              }          }      }  } |

# BÀI THỰC HÀNH 5 – LỊCH VẠN NIÊN

## base\_lib.h

|  |  |
| --- | --- |
| 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

|  |  |
| --- | --- |
| ThreeWiresProtocol.h | /\*      Project: Comunicate with real-time DS1302 using Three Wires Protocol      Header-File title: Three Wires Protocol      Author: Ngxx.fus      Based on: DS1302-DATASHEET-DOWNLOAD.pdf      Note: This header built for '8051 PRO' kit, to re-use the header file,            you need to edit CE, SCLK, IO pin and check the algorithm before use!  \*/  #ifndef \_THREE\_WIRES\_PROTOCOL\_H\_  #define \_THREE\_WIRES\_PROTOCOL\_H\_  #include <REGX52.h>  #include "base\_lib.h"  // type define: "usigned int" -> "uint"  // typedef unsigned int uint;  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;      //wait for sth un-finished to be finished :v      delay\_us(IDLE\_T);      //start comunication      CE = HIGH; SCLK = LOW;      //wait for sth un-finished to be finished :v      delay\_us(T\_PEAK);      // send cmd in 8 rasing edges      for(nCLK = 1; nCLK <= 8; nCLK++){          IO = (cmd&0x1);          HL\_MONO\_PULSE(SCLK);          cmd = (cmd>>1);      }      // send byte\_data in 8 rasing edges      for(nCLK = 1; nCLK <= 8; nCLK++){          IO = (byte\_data&0x1);          HL\_MONO\_PULSE(SCLK);          byte\_data >>= 1;      }      //End write process      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

|  |  |
| --- | --- |
| DS1302.h | /\*      Project: Comunicate with real-time DS1302 using Three Wires Protocol      Header-File title: DS1302      Author: Ngxx.fus      Based on: DS1302-DATASHEET-DOWNLOAD.pdf      Note: For more functions, pls read DS1302 datasheet and change the cmd, addr.  \*/  #ifndef \_DS1302\_H\_  #define  \_DS1302\_H\_  #include "base\_lib.h"  #include "ThreeWiresProtocol.h"  //typedef unsigned int uint;  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;  /\*  Read time from DS1302  SEL:  MSB  ... x    x    x    x    x    x   x   LSB           day  year mon  date hour min sec  x = 1: Choose  x = 0: Skip  \*/  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;          }      }      //date      if(SEL&0x10){          ds1302\_unlock\_reg();          byte\_data = single\_byte\_read(0x87);          x10 = ((byte\_data&0x30)>>4)\*10;          x1  = (byte\_data&0x0F);          time->DATE = x10 + x1;      }      //month      if(SEL&0x20){          ds1302\_unlock\_reg();          byte\_data = single\_byte\_read(0x89);          x10 = ((byte\_data&0x10)>>4)\*10;          x1  = (byte\_data&0x0F);          time->MONTH = x10 + x1;      }      //year      if(SEL&0x40){          ds1302\_unlock\_reg();          byte\_data = single\_byte\_read(0x87);          x10 = ((byte\_data&0xF0)>>4)\*10;          x1  = (byte\_data&0x0F);          time->YEAR = 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);      }      //date      if(SEL&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(SEL&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(SEL&0x20){          x10 = ((time->YEAR)/10)%10;          x1  = (time->YEAR)%10;          byte\_data = (x10<<4) + x1;          ds1302\_unlock\_reg();          single\_byte\_write(0x9C, byte\_data);      }      //day      if(SEL&0x40){          x1  = (time->DAY)%10;          ds1302\_unlock\_reg();          single\_byte\_write(0x9A, x1);      }  }  void ds1302\_initial(){      ThreeWiresProtocol\_Initial();  }  #endif |

## Calendar\_OnKit.h

|  |  |
| --- | --- |
| 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;  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 YYMMDD\_disp(){    ds1302\_read\_time(&time, 0x38);    LED[7] = DIGIT\_CODE[(time.YEAR/10)%10];    LED[6] = DIGIT\_CODE[time.YEAR%10];    LED[5] = 0x40;    LED[4] = DIGIT\_CODE[(time.MONTH/10)%10];    LED[3] = DIGIT\_CODE[time.MONTH%10];    LED[2] = 0x40;    LED[1] = DIGIT\_CODE[(time.DATE/10)%10];    LED[0] = DIGIT\_CODE[(time.DATE)%10];    DISP = 1;    Disp8leds7seg();  }  void calendar\_disp(){    switch (MODE) {      case VIEW\_TIME:        HHMMSS\_disp();        break;      case VIEW\_DATE:        YYMMDD\_disp();        break;    }  }  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,0x7F);    set\_disp\_freq(48);  }  void Interrupt0\_Action(void) interrupt 0 {    MODE=(MODE+1)%2;    DISP = 0;  }  #endif |

## main.c

|  |  |
| --- | --- |
| main.c | #include "Calendar\_OnKit.h"  void main(void){      calendar\_initial();      while(0x1){          calendar\_disp();      }  } |

# BÀI THỰC HÀNH 6 – ĐIỀU KHIỂN 03 THIẾT BỊ

Điều khiển 03 thiết bị thông qua điều khiển và ma trận nút nhấn.

## Base\_Lib.h

|  |  |
| --- | --- |
| Base\_Lib.h | /\*      Note:          Base\_Lib.h is a lib that include all define, typedef,          base function, ... It can be reused in many following project.      Autor:          Nguyen Thanh Phu      Version:          0.0.2  \*/  #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 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){      uint32 i = 0;      for(i = 0; i < us; i = i + 1){          // do nothin'      }  }  void delay\_ms(uint32 ms){      uint32 i = 0;      for(i = 0; i < ms\*12; i = i + 1){          // do nothin'      }  }  void eINT0\_CTL(uint8 CONFIG){      if( CONFIG == ENABLE){          EX0 = 1;          IT0 = 1;      }      if( CONFIG == DISABLE){          EX0 = 0;          IT0 = 1;      }  }  void eINT1\_CTL(uint8 CONFIG){      if( CONFIG == ENABLE){          // Configure INT1 falling edge interrupt          IT1 = 1;          // Enable the INT1 External Interrupt          EX1 = 1;      }      if( CONFIG == DISABLE){          // Configure INT1 falling edge interrupt          IT1 = 0;          // Enable the INT1 External Interrupt          EX1 = 0;      }  }  #define RESET\_TH 0xFC  #define RESET\_TL 0x67  void TIMER0\_CTL(uint8 CONFIG){      switch (CONFIG) {          case ENABLE:              ET0 = 1;            return;          case DISABLE:              ET0 = 0;            return;          case RESET:              TL0 = RESET\_TL;              TH0 = RESET\_TH;     return;          case START:              TR0 = 1;            return;          case STOP:              TR0 = 0;            return;          case MODE\_16BIT:              TMOD = TMOD|0x01;   return;      }  }  #define GLOBAL\_INT(CONFIG) EA=(CONFIG==ENABLE)?(1):(0)  #endif |

## LED7Seg\_OnKit.h

|  |  |
| --- | --- |
| LED7Seg\_OnKit.h | //version: 0.1.3  //---------- Include -----------  #ifndef \_LED7SEG\_ONKIT\_H\_  #define \_LED7SEG\_ONKIT\_H\_  #include <REGX52.h>  // Thư viện cơ sở cho AT89C52  #include "Base\_Lib.h"  //---------- Macros -----------  //Định nghĩa lại kiểu dữ liệu  //typedef unsigned int  uint;  //Các chân chọn vị trí LED.  sbit GND0 = P2^2;  sbit GND1 = P2^3;  sbit GND2 = P2^4;  //Chân điều khiển từng LED trong LED7seg  #define LED\_7SEG P0  const uint8 DIGIT\_CODE[] = {0X3F, 0X06, 0X5B, 0X4F, 0X66, 0X6D,                             0X7D, 0X07, 0X7F, 0X6F, /\*A\*/0x77, 0xFC,                             0x58, 0x5E, 0x79, 0x71};  uint8 LED[8] = {0, 0, 0, 0, 0, 0, 0, 0};  /\*    8   7   6   5   4   3   2   1    \_   \_   \_   \_   \_   \_   \_   \_   |\_| |\_| |\_| |\_| |\_| |\_| |\_| |\_|   |\_|.|\_|.|\_|.|\_|.|\_|.|\_|.|\_|.|\_|.  Hàm này sẽ chọn LED ở vị trí POS, mã hiển thị là CODE.  VD: Hiển thị số "1", CODE = 0x06  \*/  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;      }  }  /\*  Hiển thị trong ms\_disp\_t cả 8 LED7Seg.  Nội dung hiển thị của LED thứ i tuỳ thuộc vào  giá trị chứa trong LED[i].  Giới hạn:  hz\_freq =   24 Hz  --> 100Hz  ms\_disp\_t = 50 ms  --> 2500 ms  \*/  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 |

## IR\_Reading.h

|  |  |
| --- | --- |
| IR\_Reading.h | #ifndef \_IF\_READING\_  #define \_IF\_READING\_  #include "REGX52.h"  #include "Base\_Lib.h"  #include "Matrix\_Button.h"  /\*      Refs:          https://embeddedflakes.com/interrupt-handling-in-8051/          https://exploreembedded.com/wiki/NEC\_IR\_Remote\_Control\_Interface\_with\_8051          https://embeddedflakes.com/8051-timers-vs-counters/          https://exploreembedded.com/wiki/6.8051\_Interrupts  \*/  /\*      Delay computation:          System specifications:              Clock Freq  = 11.0592 Mhz                          = 11059200 Hz                          = 11059200 clock\_period/second              ---> Clock period =  1/11059200              Machine Cycle = 12clock\_period          Computation:              16bit -> Max value of timer: (2<<16)-1 = 65535              Delay=1ms <-> 0.001second \* 11059200(clock\_period/second)                      <-> 11059.2 clock\_period.              Bcz the machine\_cycle = 12 clock\_period              So              Delay=1ms <-> 11059.2clock\_period / 12clock\_period                      <-> 921.6 machine\_cycle (e.g ADD)                      <-> count from 64615 -> 65535                      <-> count from 0xFC67 -> 0xFFFF              Keep in mind that 16bit counter mode has been devide into              2 registers 4bit call 8bit HIGH (TH0) and 8bit LOW (TL0).  \*/  /\*          To count the number of mili-second, we use Overflow Timer Interrupt.      It means when you count to 0xFFFF (16bit counter mode), and continue count up,      TF0 is set to HIGH, and Overflow Timer 0 Interrupt function is called. We      at this time increase one into mili-second counter variable.  \*/  /\*      When a key is pressed on the remote controller, the message transmitted      consists of the following, in order:          A 9ms leading pulse burst (16 times the pulse burst length used for          a logical data bit)          A 4.5ms space          The 8-bit address for the receiving device          The 8-bit logical inverse of the address          The 8-bit command          The 8-bit logical inverse of the command          A final 562.5µs pulse burst to signify the end of message transmission.  \*/  /\*  NEC IR Remote Codes (Size: 3bytes data)      0xFFA25D: CH-      0xFF629D: CH      0xFFE21D: CH+      0xFF22DD: PREV      0xFF02FD: NEXT      0xFFC23D: PLAY/PAUSE      0xFFE01F: VOL-      0xFFA857: VOL+      0xFF906F: EQ      0xFF6897: 0      0xFF9867: 100+      0xFFB04F: 200+      0xFF30CF: 1      0xFF18E7: 2      0xFF7A85: 3      0xFF10EF: 4      0xFF38C7: 5      0xFF5AA5: 6      0xFF42BD: 7      0xFF4AB5: 8      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;  // Based on the diagram of "8051 Pro" Kit  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;  // Final data\_frame  uint32 data\_frame = 0;  // Temporary storing unfinished  data-frame while receiving.  uint32 buffer = 0;  // Mili-second count  uint8 ms\_count = 0;  // bit-count  int8 negedge\_count = 0;  //Manual\_Remote  enum enum\_MR{ MANUAL=0, REMOTE=1 };  uint8 manual\_remote = REMOTE;  void IR\_Reading\_Initial(){      IndicatorLED = 1;      DataRcv = 1;      buffer = 0;      data\_frame = 0;      negedge\_count = 0;      GLOBAL\_INT(ENABLE);      eINT0\_CTL(ENABLE);      eINT1\_CTL(ENABLE);      TIMER0\_CTL(ENABLE);      TIMER0\_CTL(MODE\_16BIT);      TIMER0\_CTL(START);      TIMER0\_CTL(RESET);  }  //Yeah, this function need to declare in main.h  //but, merge into IR\_Reading.h for reducing  //working tree  void Initial(){      // Run initial      IR\_Reading\_Initial();      // SET all LED-7seg OFF by set a->g = L.      P0 = 0;      // SET all LED OFF by set Port 2 = H.      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);      //A data-frame isn't longer than 67.5mili-sec.      if(ms\_count<67) ms\_count = ms\_count + 1;  }  void External1\_Interrupt() interrupt 2 {      //Toggle mode      manual\_remote = (manual\_remote==MANUAL)?(REMOTE):(MANUAL);      MR = manual\_remote;  }  void Manual\_Control(){      uint32 btn\_matrix = 0;      if(manual\_remote == MANUAL){          btn\_matrix =  Get\_BTN\_MATRIX();          if(btn\_matrix & 0x2)              L0 = ~L0;          if(btn\_matrix & 0x40)              L1 = ~L1;          if(btn\_matrix & 0x800)              L2 = ~L2;          //Prevent continuos toggle stata :v          while( btn\_matrix == Get\_BTN\_MATRIX())              delay\_us(1000);      }  }  void External0\_Interrupt() interrupt 0 {      uint32 current\_mscount = 0;      if(manual\_remote == MANUAL) return;      current\_mscount = ms\_count;      TIMER0\_CTL(RESET);      ms\_count=0;      negedge\_count +=1;      DataRcv = ~DataRcv;      // this neg-edge by SOF (start of frame)?      if(current\_mscount >= 67){          negedge\_count = -2;          RESET\_BUFFER();      }else{          if( negedge\_count < 0)              /\*Do nothing, skip this neg-edge\*/;          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(1000);              LED\_Show(data\_frame);              FrameExtracted=1;          }      }    }  #endif |

## main.c

|  |  |
| --- | --- |
| main.c | #include "IR\_Reading.h"  void main(void){      Initial();      while(0x1){        Manual\_Control();      }  } |

# BÀI THỰC HÀNH 7 – NHÀ THÔNG MINH 1

Điều khiển 03 thiết bị, trong đó có

* 01 thiết bị có thể điều khiển từ xa bằng hồng ngoại.
* 01 thiết bị có thể hẹn giờ tắt/bật.
* 01 thiết bị có thể tắt/bật tự động theo ánh sáng.

## Utilities.h

|  |  |
| --- | --- |
| Utilities.h | /\*      Note:          From 10/11/2024, base\_lib.h/Base\_Lib.h has been changed to Utilities.h          Utilities.h.h is a lib that include all define, typedef,          base function, ... It can be reused in many following project.      Autor:          Nguyen Thanh Phu      Version:          0.1.5  \*/  #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;  typedef char    int8;  typedef short    int16;  typedef int     int32;  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  };  static void delay\_us(uint32 us){      uint32 i = 0;      for(i = 0; i < us; i = i + 1){          // do nothin'      }  }  void delay\_ms(uint32 ms){      uint32 i = 0;      uint32 j = 0;      for(i = 0; i < ms\*19; i = i + 1){          // do nothin'      }  }  void eINT0\_CTL(uint8 CONFIG){      if( CONFIG == ENABLE){          EX0 = 1;          IT0 = 1;      }      if( CONFIG == DISABLE){          EX0 = 0;          IT0 = 0;      }  }  // void eINT1\_CTL(uint8 CONFIG){  //     if( CONFIG == ENABLE){  //         // Configure INT1 falling edge interrupt  //         IT1 = 1;  //         // Enable the INT1 External Interrupt  //         EX1 = 1;  //     }  //     if( CONFIG == DISABLE){  //         // Configure INT1 falling edge interrupt  //         IT1 = 0;  //         // Enable the INT1 External Interrupt  //         EX1 = 0;  //     }  // }  #define RESET\_TH 0xFC  #define RESET\_TL 0x67  void TIMER0\_CTL(uint8 CONFIG){      switch (CONFIG) {          case ENABLE:              ET0 = 1;            return;          case DISABLE:              ET0 = 0;            return;          case RESET:              TL0 = RESET\_TL;              TH0 = RESET\_TH;     return;          case START:              TR0 = 1;            return;          case STOP:              TR0 = 0;            return;          case MODE\_16BIT:              TMOD = TMOD|0x01;   return;      }  }  // #define GLOBAL\_INT(CONFIG)  void GLOBAL\_INT(uint8 CONFIG){      EA=(CONFIG==ENABLE)?(1):(0);  }  #endif |

## Time.h

|  |  |
| --- | --- |
| Time.h | #ifndef \_TIME\_H\_  #define \_TIME\_H\_  #include "Utilities.h"  #ifndef \_STRUCT\_TIME\_  #define \_STRUCT\_TIME\_      typedef struct TIME{          uint8 DAY; // mon, tue, wed, thu, ...          uint8 DATE;          uint8 MONTH;          uint8 YEAR;          uint8 HOUR;          uint8 MINUTE;          uint8 SECOND;      } TIME;  #endif  // uint8 time\_copy(TIME\* scr, TIME\* dest, uint8 mask){  //     if((mask&0x1)!=0) dest->SECOND = scr->SECOND;  //     if((mask&0x2)!=0) dest->MINUTE = scr->MINUTE;  //     if((mask&0x4)!=0) dest->HOUR = scr->HOUR;  //     if((mask&0x8)!=0) dest->DATE = scr->DATE;  //     if((mask&0x10)!=0) dest->MONTH = scr->MONTH;  //     if((mask&0x20)!=0) dest->YEAR  = scr->YEAR;  // }  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

|  |  |
| --- | --- |
| 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

|  |  |
| --- | --- |
| ThreeWiresProtocol.h | /\*      Project: Comunicate with real-time DS1302 using Three Wires Protocol      Header-File title: Three Wires Protocol      Author: Ngxx.fus      Based on: DS1302-DATASHEET-DOWNLOAD.pdf      Note: This header built for '8051 PRO' kit, to re-use the header file,            you need to edit CE, SCLK, IO pin and check the algorithm before use!  \*/  #ifndef \_THREE\_WIRES\_PROTOCOL\_H\_  #define \_THREE\_WIRES\_PROTOCOL\_H\_  #include "Utilities.h"  // type define: "usigned int" -> "uint"  // typedef unsigned int uint;  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;      //wait for sth un-finished to be finished :v      delay\_us(IDLE\_T);      //start comunication      CE = HIGH; SCLK = LOW;      //wait for sth un-finished to be finished :v      delay\_us(T\_PEAK);      // send cmd in 8 rasing edges      for(nCLK = 1; nCLK <= 8; nCLK++){          IO = (cmd&0x1);          HL\_MONO\_PULSE(SCLK);          cmd = (cmd>>1);      }      // send byte\_data in 8 rasing edges      for(nCLK = 1; nCLK <= 8; nCLK++){          IO = (byte\_data&0x1);          HL\_MONO\_PULSE(SCLK);          byte\_data >>= 1;      }      //End write process      CE = LOW;  }  uint8 single\_byte\_read(uint8 cmd){      uint8 nCLK;      uint8 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

|  |  |
| --- | --- |
| DS1302.h | /\*      Project: Comunicate with real-time DS1302 using Three Wires Protocol      Header-File title: DS1302      Author: Ngxx.fus      Based on: DS1302-DATASHEET-DOWNLOAD.pdf      Note: For more functions, pls read DS1302 datasheet and change the cmd, addr.  \*/  #ifndef \_DS1302\_H\_  #define  \_DS1302\_H\_  #include "Time.h"  #include "Utilities.h"  #include "ThreeWiresProtocol.h"  //typedef unsigned int uint32;  enum enum\_DAY{MON = 0, TUE, WED, THU, FRI, SAT, SUN};  #define ds1302\_unlock\_reg() single\_byte\_write(0x8E, 0x0)  /\*  Read time from DS1302  mask:  MSB  ... x    x    x    x    x    x   x   LSB           day  year mon  date hour min sec  x = 1: Choose  x = 0: Skip  \*/  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;      }      //month      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;      }      //year      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;      }      // //day      // if(mask&0x40){      //     x1  = (time->DAY)%10;      //     ds1302\_unlock\_reg();      //     single\_byte\_write(0x9A, 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

|  |  |
| --- | --- |
| IR\_Reading.h | #ifndef \_IF\_READING\_  #define \_IF\_READING\_  #include "Utilities.h"  #include "DS1302.h"  #include "LED7Seg\_OnKit.h"  //#include "Matrix\_Button.h"  /\*      Refs:          https://embeddedflakes.com/interrupt-handling-in-8051/          https://exploreembedded.com/wiki/NEC\_IR\_Remote\_Control\_Interface\_with\_8051          https://embeddedflakes.com/8051-timers-vs-counters/          https://exploreembedded.com/wiki/6.8051\_Interrupts  \*/  #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;  // Based on the diagram of "8051 Pro" Kit  // sbit IR\_RCV\_PIN = P3^2;  // sbit IndicatorLED = P2^7;  // sbit DataRcv = P2^6;  sbit FrameExtracted = P2^0;  // sbit MR = P2^4;  // Final data\_frame  uint32 data\_frame = 0;  // Temporary storing unfinished  data-frame while receiving.  uint32 buffer = 0;  // Mili-second count  uint8 ms\_count = 0;  // bit-count  int8 negedge\_count = 0;  // //check if we have a new data\_frame or not?  // uint8 new\_dataframe(){  //     return (data\_frame!=0)?1:0;  // }  //clear frame after read!  uint32 read\_extracted\_frame(){      uint32 frame = data\_frame;      data\_frame = 0;      return frame;  }  void IR\_Reading\_Initial(){      // IndicatorLED = 1;      // DataRcv = 1;      buffer = 0;      data\_frame = 0;      negedge\_count = 0;      GLOBAL\_INT(ENABLE);      eINT0\_CTL(ENABLE);      TIMER0\_CTL(ENABLE);      TIMER0\_CTL(MODE\_16BIT);      TIMER0\_CTL(START);      TIMER0\_CTL(RESET);  }  void Timer0\_OverFlow\_Interrupt() interrupt 1 {      // IndicatorLED = ~IndicatorLED;      TIMER0\_CTL(RESET);      //A data-frame isn't longer than 67.5mili-sec.      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;      // DataRcv = ~DataRcv;      // this neg-edge by SOF (start of frame)?      if(current\_mscount >= 67){          negedge\_count = -2;          RESET\_BUFFER();      }else{          if( negedge\_count < 0)              /\*Do nothing, skip this neg-edge\*/;          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

|  |  |
| --- | --- |
| LED7Seg\_OnKit.h | //version: 0.1.3  //---------- Include -----------  #ifndef \_LED7SEG\_ONKIT\_H\_  #define \_LED7SEG\_ONKIT\_H\_  #include "Utilities.h"  //---------- Macros -----------  //Định nghĩa lại kiểu dữ liệu  //typedef unsigned int  uint;  //Các chân chọn vị trí LED.  sbit GND0 = P2^2;  sbit GND1 = P2^3;  sbit GND2 = P2^4;  //Chân điều khiển từng LED trong LED7seg  #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};  /\*    8   7   6   5   4   3   2   1    \_   \_   \_   \_   \_   \_   \_   \_   |\_| |\_| |\_| |\_| |\_| |\_| |\_| |\_|   |\_|.|\_|.|\_|.|\_|.|\_|.|\_|.|\_|.|\_|.  Hàm này sẽ chọn LED ở vị trí POS, mã hiển thị là CODE.  VD: Hiển thị số "1", CODE = 0x06  \*/  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;      }  }  /\*  Hiển thị trong ms\_disp\_t cả 8 LED7Seg.  Nội dung hiển thị của LED thứ i tuỳ thuộc vào  giá trị chứa trong LED[i].  Giới hạn:  hz\_freq =   24 Hz  --> 100Hz  ms\_disp\_t = 50 ms  --> 2500 ms  \*/  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

|  |  |
| --- | --- |
| 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  };  //Current screen :v  uint32 CURRENT\_INDX = 0;  // uint8 MODE\_TREE[250];  //Wait for yes\_no?  uint8 WAIT\_YES\_NO = false;  /\*  MSB  ... 2 1 0 LSB           x x x      H: Enable      L: Disable  \*/  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;  /\*  MSB  ... 1 0 LSB           | |           | |           | off           on timer      H: Enable      L: Disable  \*/  uint8 timer\_enable = 0;  // code received from IR REMOTE  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;      // POS = 0: exit      // POS = 1: set on minute \_x1      // POS = 2: set on minute \_x10      // POS = 3: set on hour \_x1      // POS = 4: set on hour \_x10      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

|  |  |
| --- | --- |
| 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 THỰC HÀNH 8 – NHÀ THÔNG MINH 2

Điều khiển 05 thiết bị, trong đó có

* 01 thiết bị có thể điều khiển từ xa bằng hồng ngoại.
* 01 thiết bị có thể hẹn giờ tắt/bật.
* 01 thiết bị có thể tắt/bật tự động theo ánh sáng.
* 01 thiết bị có thể điều khiển từ diện thoại thông minh.

## Utilities.h

|  |  |
| --- | --- |
|  | Đã định nghĩa ở [BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1](#_BÀI_THỰC_HÀNH) |

## Time.h

|  |  |
| --- | --- |
|  | Đã định nghĩa ở [BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1](#_BÀI_THỰC_HÀNH) |

## XPT2046.h

|  |  |
| --- | --- |
|  | Đã định nghĩa ở [BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1](#_BÀI_THỰC_HÀNH) |

## ThreeWiresProtocol.h

|  |  |
| --- | --- |
|  | Đã định nghĩa ở [BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1](#_BÀI_THỰC_HÀNH) |

## DS1302.h

|  |  |
| --- | --- |
|  | Đã định nghĩa ở [BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1](#_BÀI_THỰC_HÀNH) |

## LCD\_1602.h

|  |  |
| --- | --- |
| LCD\_1602.h | #ifndef \_LCD\_1602\_H\_  #define \_LCD\_1602\_H\_  /\*  DOCUMENTATIONS:      https://developer.arm.com/documentation/101655/0961/Cx51-User-s-Guide/Language-Extensions/Data-Types/Special-Function-Registers/sbit      https://circuitdigest.com/article/16x2-lcd-display-module-pinout-datasheet  \*/  /\*  This lib was made to interfacing with LCD.  LCD's pin informatios:    \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_  |  ################################   |  |  ################################   |  | \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ |      | | | | | | | | | |  |  |  |  |      1 2 3 4 5 6 7 8 9 10 11 12 13 14  pin 0     - VSS - Connected to the ground of the MCU/ Power source  pin 1     - VDD - Connected to the supply pin of Power source  pin 2     - VEE - Connected to a variable POT that can source 0-5V  pin 3     - RS  - Toggles between Command/Data Register  pin 4     - RW  - Toggles the LCD between Read/Write Operation  pin 5     - E   - Must be held high to perform Read/Write Operation  Pin 7-14  - D   - Pins used to send Command or data to the LCD.  macro  \*/  #include <stdio.h>  #include <REGX52.h>  #include "Utilities.h"  #define LCD\_ON\_CURSOR\_ON            0x0F  //LCD ON, cursor ON  #define CLEAR\_SCREEN                0x01  //Clear display screen  #define RETURN\_HOME                 0x02  //Return home  #define LEFT\_SHIFT\_CURSOR           0x04  //Decrement cursor (shift cursor to left)  #define RIGHT\_SHIFT\_CURSOR          0x06  //Increment cursor (shift cursor to right)  #define LEFT\_SHIFT\_DISPLAY          0x05  //Shift display right  #define RIGHT\_SHIFT\_DISPLAY         0x07  //Shift display left  #define DISPLAY\_ON\_CURSOR\_BLINKING  0x0E  //Display ON, cursor blinking  #define SET\_CURSOR\_0x\_0y            0x80  //Force cursor to beginning of first line  #define SET\_CURSOR\_1x\_0y            0xC0  //Force cursor to beginning of second line  #define LINEx2\_MAT5x7               0x38  //2 lines and 5×7 matrix  #define CMD11                       0x83  //Cursor line 1 position 3  #define ACTIVATE\_2nd\_LINE           0x3C  //Activate second line  #define LCD\_OFF\_CURSOR\_OFF          0x08  //Display OFF, cursor OFF  #define CMD14                       0xC1  //Jump to second line, position 1  #define LCD\_ON\_CURSOR\_OFF           0x0C  //Display ON, cursor OFF  #define CMD16                       0xC1  //Jump to second line, position 1  #define CMD17  #define WRITE\_MODE              0x0  #define READ\_MODE               0x1  #define SEND\_CMD\_MODE           0x0  #define SEND\_DISPLAY\_DATA\_MODE  0x1  //  The variables bellow can be edited bases on  //  your circuit.  //  Set your LCD is in receiving command or receriving display data.  sbit RS         = P2^6; //RS pin  //  Set your LCD is READ mode or WRITE mode (usually write mode, be written by your MCU)  sbit RW         = P2^5; //  //  Enable your LCD by a negedge pulse  sbit EN         = P2^7;  //  Receive or Transfer data (parallel)  #define DATA\_PORT P0  //  Make a MONO pulse at EN pin  //  MONO pulse: LOW->HIGH (HIGH)\*n HIGH->LOW :))  void LCD\_ENABLE(){      //Enable, a high to low pulse need to enable the LCD      EN = 0x1;      delay\_us(50);      EN = 0x0;  }  //  To sent command to the LCD.  void LCD\_SEND\_BYTE\_COMMAND(unsigned char CMD){      DATA\_PORT = CMD;      delay\_us(50);      RS = SEND\_CMD\_MODE;      delay\_us(50);      RW   = WRITE\_MODE;      delay\_us(50);      LCD\_ENABLE();  }  //  To sent a byte of DISPLAY DATA to the LCD.  void LCD\_SEND\_BYTE\_DISPLAY(unsigned char BYTE){      // NOTE: BYTE is displayed in ASCII.      DATA\_PORT = BYTE;      delay\_us(50);      RS = SEND\_DISPLAY\_DATA\_MODE;      delay\_us(50);      RW = WRITE\_MODE;      delay\_us(50);      LCD\_ENABLE();  }  //  To sent an array of byte of DISPLAY DATA to the LCD.  void LCD\_SEND\_BYTE\_ARRAY\_DISPLAY(char ARR[], uint8 SIZE){      uint32 i = 0;      if(SIZE<0)          while(\*ARR){              LCD\_SEND\_BYTE\_DISPLAY(\*ARR);              ARR++;          }      else          while( i < SIZE ){              LCD\_SEND\_BYTE\_DISPLAY(ARR[i]);              ++i;          }  }  //  Set the position of the CURSOR in 16x2 LCD screen.  void LCD\_SET\_CURSOR\_POS(uint32 ROW, uint32 COL){      if(ROW == 0){          LCD\_SEND\_BYTE\_COMMAND(SET\_CURSOR\_0x\_0y+COL);      }      if(ROW == 1){          LCD\_SEND\_BYTE\_COMMAND(SET\_CURSOR\_1x\_0y+COL);      }  }  /\*  Set TEXT in LCD.  Limits:      size : [0-->31]      row  : [0-->1]      col  : [0-->15]      warp\_text   : [0->1]      clear\_screen : [0->1]      row\_offset  : [0->1]      col\_offset  : [0->15]  \*/  void LCD\_Set\_Text(          char str[], uint8 str\_size,          uint8 row\_offset, uint8 col\_offset      ){      uint8 displayed = 0;          //un-warp text          LCD\_SET\_CURSOR\_POS(row\_offset, col\_offset);          LCD\_SEND\_BYTE\_ARRAY\_DISPLAY(str, str\_size);  }  //Simple to set TEXT which to be displayed in LCD WITHOUT CLEAR previous screen  void LCD\_Simple\_Set\_Text\_1(          char str[], uint8 str\_size,          uint8 row\_offset, uint8 col\_offset      ){      if(str\_size == 0) { while(str[str\_size++]); --str\_size;}      LCD\_Set\_Text(str, str\_size, row\_offset, col\_offset);  }  //Simple to set TEXT which to be displayed in LCD WITH CLEAR previous screen  void LCD\_Simple\_Set\_Text\_2(          char str[], uint8 str\_size,          uint8 row\_offset, uint8 col\_offset      ){      LCD\_SEND\_BYTE\_COMMAND(0x01);      if(str\_size == 0) { while(str[str\_size++]); --str\_size;} // Cannot disp '\0' --> remove it!      LCD\_Set\_Text(str, str\_size, row\_offset, col\_offset);  }  // void LCD\_Clear\_Screen(){  //  LCD\_SEND\_BYTE\_COMMAND(CLEAR\_SCREEN);  // }  //  Set up your LCD.  void LCD\_Initial(){      delay\_us(50);      LCD\_SEND\_BYTE\_COMMAND(LINEx2\_MAT5x7);      LCD\_SEND\_BYTE\_COMMAND(LCD\_ON\_CURSOR\_OFF);      LCD\_SEND\_BYTE\_COMMAND(RIGHT\_SHIFT\_CURSOR);  }  #endif |

## String\_Ultils.h

|  |  |
| --- | --- |
| String\_Ultils.h | #ifndef \_STRING\_ULTILS\_H\_  #define \_STRING\_ULTILS\_H\_  #include "Utilities.h"  #define digit2char(x) ((x)+'0')  #define str\_len(x) sizeof(x)  #define x1\_digit(x) ((x)%10)  #define x10\_digit(x) (((x)/10)%10)  #define x100\_digit(x) (((x)/100)%10)  #define x1000\_digit(x) (((x)/1000)%10)  #define \_is\_lower\_case(x) ('a' <= (x) && (x) <= 'z')  #define \_is\_upper\_case(x) ('A' <= (x) && (x) <= 'Z')  #define \_not\_digit(x) (!('0' <= (x) && (x) <= '9'))  #define \_not\_dollar\_sign(x) (!((x)!='$'))  #define \_not\_equal\_sign(x) (!((x)!='='))  #define \_not\_underscore(x) (!((x)=='\_'))  #define \_is\_new\_line(x) ((x)=='\n')  #define \_is\_carriage\_return(x) ((x)=='\r')  uint8 \_string\_equal\_compare(char str1[], char str2[], uint8 cmp\_size, uint8 str2\_offset){      //Warning: str2\_offset+cmp\_size < size\_of(str2)      uint8 i;      if(cmp\_size < 1) return 0;      REP(i, 0, cmp\_size-1){          if( str1[i] != str2[i+str2\_offset] ) return 0;      }      return 1;  }  // void \_string\_to\_upper\_case(char str[], uint8 str\_size){  //     if(str\_size == 0) return;  //     while(str\_size--){  //         if(\_is\_lower\_case(str[str\_size])) str[str\_size]-= 'a'-'A';  //     }  // }  // void \_string\_to\_lower\_case(char str[], uint8 str\_size){  //     if(str\_size == 0) return;  //  while(str\_size--){  //      if(\_is\_upper\_case(str[str\_size])) str[str\_size]+= 'a'-'A';  //  }  // }  // void \_string\_copy(char dest[], char scr[], uint8 cp\_size, uint8 offset){  //     uint8 i;  //     if(cp\_size < 1 ) return;  //     REP(i, offset, cp\_size-1)  //         dest[i] = scr[i];  // }  uint8 \_string\_find\_pattern(char pattern[], uint8 pat\_size, char text[], uint8 txt\_size, uint8 offset){      // Note: pat\_size <= txt\_size < 256      // Note: Return the first found position from range [offset, txt\_size)      uint8 i;      if(pat\_size>txt\_size) return txt\_size;      // This algorithm run with O-complexity: O(pat\_size\*txt\_size)      REP(i, offset, txt\_size-pat\_size){          if( \_string\_equal\_compare(pattern, text+i, pat\_size, 0)){              return i;          }      }      return txt\_size;  }  // void \_string\_num2text(uint32 num, char text[], uint8 text\_size){  //     uint8 i;  //     REP(i, 0, text\_size-1){  //         text[(text\_size-1)-i] = num%10 + '0';  //         num/=10;  //     }  //     text[text\_size]='\0';  // }  #endif |

## UART\_BLE.h

|  |  |
| --- | --- |
| UART\_BLE.h | // refs: https://www.electronicwings.com/8051/8051-uart  // refs: https://embetronicx.com/tutorials/microcontrollers/8051/8051-uart-tutorial-serial-communication/  #ifndef \_BLUETOOTH\_UART\_H\_  #define \_BLUETOOTH\_UART\_H\_  #include "LCD\_1602.h"  #include "Utilities.h"  #include "String\_Ultils.h"  /\*  Valid msg from Bluetooth module:  NOTE: flip dev-state (1->0, 0->1) if the dev is controlled by USER      inv1      inv2      inv3      inv4      inv5  \*/  #define \_max\_buffer\_size 8  #define \_bounded\_y(x, y) ((x)%(y))  uint8 ble\_rcv\_size;  char  ble\_rcv\_data[\_max\_buffer\_size];  void UART\_Byte\_Transmit(char transmit\_data){      // Copy data to BUFFER      SBUF = transmit\_data;      // while for another Transmission until it end      while (TI==0);      // Reset TI flag to start this Transmission      TI = 0;  }  void UART\_Bytes\_Transmit(char transmit\_data[], int32 transmit\_data\_size){      int32 i;      if(transmit\_data\_size == 0){          while(\*transmit\_data){              UART\_Byte\_Transmit(\*transmit\_data);              transmit\_data++;          }      }else{          REP(i, 1, transmit\_data\_size){              UART\_Byte\_Transmit(transmit\_data[i-1]);          }      }  }  // uint8 UART\_Read\_Data(char uart\_data[], char read\_size){  //  if(read\_size == 0 || ble\_rcv\_size == 0) return false; // for read failed  //  read\_size = min\_val(read\_size, ble\_rcv\_size);  //  \_string\_copy(uart\_data, ble\_rcv\_data, read\_size, 0);  //  //reset buffer size (aka ble\_rcv\_data)  //  ble\_rcv\_size = 0;  //  return true; // for read successful  // }  uint8 ble\_has\_contained(char pattern[], uint8 pat\_size){      if(ble\_rcv\_size<1) return 0;      return \_string\_find\_pattern(pattern, pat\_size, ble\_rcv\_data, ble\_rcv\_size, 0) < ble\_rcv\_size;  }  void Bluetooth\_UART\_Initial(){      //UART initial      GLOBAL\_INT(ENABLE);      ES = 1; //Serial interrupt      TMOD |= 0x20; //Set timer 2 mode 8bit      TH1 = 0xFD; //load value for baud rate = 9600      TL1 = 0xFD; //load value for baud rate = 9600      SCON = 0x50;      TR1 = 1; // start timer 1        //Bluetooth initial      ble\_rcv\_size = 0;  }  void UART\_Received() interrupt 4 {      char temp\_char;      if(RI == 1){          temp\_char = SBUF;          if(              \_is\_carriage\_return(temp\_char)              || \_is\_new\_line(temp\_char)          ){                  RI = 0;                  return;          }          if(\_is\_upper\_case(temp\_char)) temp\_char -= 'a'-'A';          ble\_rcv\_size = \_bounded\_y(ble\_rcv\_size, \_max\_buffer\_size)+1;          ble\_rcv\_data[ble\_rcv\_size-1] = temp\_char;          RI = 0;      }else{          // TI = 0;      }  }  #endif |

## main.h

|  |  |
| --- | --- |
| main.h | #include "Time.h"  #include "String\_Ultils.h"  #include "UART\_BLE.h"  #include "DS1302.h"  #include "Utilities.h"  #include "XPT2046.h"  #include "UART\_BLE.h"  #include "LCD\_1602.h"  #include "LCD\_1602.h"  #include "IR\_Reading.h"  #define \_1st\_bit\_mask 0x01  #define \_2nd\_bit\_mask 0x02  #define \_3rd\_bit\_mask 0x04  #define \_4th\_bit\_mask 0x08  #define \_5th\_bit\_mask 0x10  #define \_6th\_bit\_mask 0x20  #define \_7th\_bit\_mask 0x40  #define \_8th\_bit\_mask 0x80  #define DEV1\_MASK 0x003  #define DEV2\_MASK 0x00C  #define DEV3\_MASK 0x030  #define DEV4\_MASK 0x0C0  #define DEV5\_MASK 0x300  #define MODE\_UP() DISP\_MODE/=5  #define MODE\_DOWN() if(DISP\_MODE<170) DISP\_MODE\*=5  #define MODE\_NORMAL 0  #define MODE\_SETUP 1  #define MODE\_SETUP\_TIME 5  #define MODE\_SETUP\_TIME\_SYSTEM\_DISP  25  #define MODE\_SETUP\_TIME\_SYSTEM\_ACTION  125  #define MODE\_SETUP\_TIME\_TIMER\_1\_DISP 26  #define MODE\_SETUP\_TIME\_TIMER\_1\_ACTION 130  #define MODE\_SETUP\_TIME\_TIMER\_2\_DISP 27  #define MODE\_SETUP\_TIME\_TIMER\_2\_ACTION 135  #define MODE\_SETUP\_DEVICE 6  #define MODE\_SETUP\_DEVICE\_1\_DISP 30  #define MODE\_SETUP\_DEVICE\_1\_ACTION 150  #define MODE\_SETUP\_DEVICE\_2\_DISP 31  #define MODE\_SETUP\_DEVICE\_2\_ACTION 155  #define MODE\_SETUP\_DEVICE\_3\_DISP 32  #define MODE\_SETUP\_DEVICE\_3\_ACTION 160  #define MODE\_SETUP\_DEVICE\_4\_DISP 33  #define MODE\_SETUP\_DEVICE\_4\_ACTION 165  #define MODE\_SETUP\_DEVICE\_5\_DISP 34  #define MODE\_SETUP\_DEVICE\_5\_ACTION 170  sbit DEV1 = P2^0;  sbit DEV2 = P2^1;  sbit DEV3 = P2^2;  sbit DEV4 = P2^3;  sbit DEV5 = P2^4;  uint8 gp\_reg; //general purpose register  uint8 HOLD = 0;  uint8 DISP\_MODE = 0;  uint8 SYST\_TRIGGER = 0;  uint8 USER\_DEV\_CTL = 0;  uint8 SYS\_CTL\_INV = 0;  uint16 CTL\_DEV\_SEL = 0;  char time\_disp\_1[]="TIME:   XX:XX:XX";  char time\_disp\_2[]="DATE:   XX/XX/XX";  TIME    sys\_t = {7  , 12, 11, 24, 21, 05, 45},          timer\_1 = {7  , 12, 11, 24, 21, 05, 55},          timer\_2 = {7  , 12, 11, 24, 21, 06, 5};  void Time\_Data\_Display(char label\_1[], char label\_2[], TIME t, uint8 SEL){      time\_disp\_1[0] = label\_1[0];      time\_disp\_1[1] = label\_1[1];      time\_disp\_1[2] = label\_1[2];      time\_disp\_1[3] = label\_1[3];      time\_disp\_1[4] = label\_1[4];      time\_disp\_1[5] = label\_1[5];      time\_disp\_1[6] = label\_1[6];      time\_disp\_1[7] = label\_1[7];      time\_disp\_2[0] = label\_2[0];      time\_disp\_2[1] = label\_2[1];      time\_disp\_2[2] = label\_2[2];      time\_disp\_2[3] = label\_2[3];      time\_disp\_2[4] = label\_2[4];      time\_disp\_2[5] = label\_2[5];      time\_disp\_2[6] = label\_2[6];      time\_disp\_2[7] = label\_2[7];      if(SEL & \_1st\_bit\_mask){          time\_disp\_1[8] = digit2char(x10\_digit(t.HOUR));          time\_disp\_1[9] = digit2char(x1\_digit(t.HOUR));          time\_disp\_1[11] = digit2char(x10\_digit(t.MINUTE));          time\_disp\_1[12] = digit2char(x1\_digit(t.MINUTE));          time\_disp\_1[14] = digit2char(x10\_digit(t.SECOND));          time\_disp\_1[15] = digit2char(x1\_digit(t.SECOND));          LCD\_Simple\_Set\_Text\_1(time\_disp\_1, 16, 0, 0);      }      if(SEL & \_2nd\_bit\_mask){          time\_disp\_2[8] = digit2char(x10\_digit(t.DATE));          time\_disp\_2[9] = digit2char(x1\_digit(t.DATE));          time\_disp\_2[11] = digit2char(x10\_digit(t.MONTH));          time\_disp\_2[12] = digit2char(x1\_digit(t.MONTH));          time\_disp\_2[14] = digit2char(x10\_digit(t.YEAR));          time\_disp\_2[15] = digit2char(x1\_digit(t.YEAR));          LCD\_Simple\_Set\_Text\_1(time\_disp\_2, 16, 1, 0);      }  }  uint8 YES\_NO(){      uint8 ans = 0;      LCD\_Simple\_Set\_Text\_2("CONFIRM?[YES/NO]", 0, 0, 0);      while(0x1){          if(ans)              LCD\_Simple\_Set\_Text\_1("> YES", 0, 1, 0);          else              LCD\_Simple\_Set\_Text\_1("> NO ", 0, 1, 0);          // delay\_ms(1000);          switch (read\_extracted\_frame()) {              case PREV: ans=(ans+1)%2; break;              case NEXT: ans=(ans+1)%2; break;              case PLAY\_PAUSE: return ans;              case MODE:  return 0;              case ON\_OFF:  return 0;          }      }      return 0;  }  uint8 Setup\_Device(){      /\*      Map of return value:      MSB           4  3  2  1  0  LSB          [x][x][x][x][x][x][x][x]                    |  |  |  ^~~~~                    |  |  |  Control mode (USER/LIGHT/TIMER\_1/TIMER\_2)                    |  |  ^~                    |  |  Invert trigger                    |  ^~                    |  Pre-set dev-state (ON/OFF)                    ^~                    Skip pre-set dev-state      \*/      uint8 pos = 0, res = 0;      uint32 rm\_code;      LCD\_Simple\_Set\_Text\_2("Setup device:", 0, 0, 0);      while(0x1){          rm\_code = read\_extracted\_frame();          if(rm\_code == MODE) return 0x8F;;          if(rm\_code == PLAY\_PAUSE) break;          if(rm\_code == NEXT) pos=(pos+1)%4;          if(rm\_code == PREV) pos=(pos+3)%4;          ble\_rcv\_size = 0;          switch (pos) {              case 0:                  LCD\_Simple\_Set\_Text\_1("> MODE: USER   ", 0, 1, 0);                  break;              case 1:                  LCD\_Simple\_Set\_Text\_1("> MODE: LIGHT  ", 0, 1, 0);                  break;              case 2:                  LCD\_Simple\_Set\_Text\_1("> MODE: TIMER\_1", 0, 1, 0);                  break;              case 3:                  LCD\_Simple\_Set\_Text\_1("> MODE: TIMER\_2", 0, 1, 0);                  break;          }      }      res = pos;      pos=1;      if(res)          while(0x1){              rm\_code = read\_extracted\_frame();              if(rm\_code == MODE) return 0x8F;;              if(rm\_code == PLAY\_PAUSE) break;              if(rm\_code == NEXT) pos=(pos+1)%2;              if(rm\_code == PREV) pos=(pos+1)%2;              ble\_rcv\_size = 0;              switch (pos) {                  case 0:                      LCD\_Simple\_Set\_Text\_1("> TRIG: NON-INV ", 0, 1, 0);                      break;                  case 1:                      LCD\_Simple\_Set\_Text\_1("> TRIG: INVERTED", 0, 1, 0);                      break;              }          }      res |= (pos<<2);      pos=2;      while(0x1){          rm\_code = read\_extracted\_frame();          if(rm\_code == MODE) return 0x8F;;          if(rm\_code == PLAY\_PAUSE) break;          if(rm\_code == NEXT) pos=(pos+1)%3;          if(rm\_code == PREV) pos=(pos+2)%3;          ble\_rcv\_size = 0;          switch (pos) {              case 0:                  LCD\_Simple\_Set\_Text\_1("> STATE: ON     ", 0, 1, 0);                  break;              case 1:                  LCD\_Simple\_Set\_Text\_1("> STATE: OFF    ", 0, 1, 0);                  break;              case 2:                  LCD\_Simple\_Set\_Text\_1("> STATE: SKIP   ", 0, 1, 0);                  break;          }      }      res |= (pos<<3);      return res;  }  uint8 Setup\_Time(TIME\* t){      TIME temp;      uint8 pos = 0;      uint8 found\_pos = 0;      uint32 REMOTE\_CODE = 0;      DS1302\_Read\_Time(&temp, 0x3F);      while(0x1){          // Direct          REMOTE\_CODE = read\_extracted\_frame();          if(REMOTE\_CODE == PLAY\_PAUSE) break;          elif(REMOTE\_CODE == MODE) return 0;          elif(REMOTE\_CODE == NEXT) pos = (pos+1)%6;          elif(REMOTE\_CODE == PREV)  pos = (pos-1+6)%6;          switch (pos) {              case 0:                  temp.SECOND = (temp.SECOND+CODE2NUM(REMOTE\_CODE)+found\_pos)%60;                  break;              case 1:                  temp.MINUTE = (temp.MINUTE+CODE2NUM(REMOTE\_CODE)+found\_pos)%60;                  break;              case 2:                  temp.HOUR = (temp.HOUR+CODE2NUM(REMOTE\_CODE)+found\_pos)%24;                  break;              case 3:                  temp.DATE = (temp.DATE+CODE2NUM(REMOTE\_CODE)+found\_pos)%31;                  break;              case 4:                  temp.MONTH = (temp.MONTH+CODE2NUM(REMOTE\_CODE)+found\_pos)%12;                  break;              case 5:                  temp.YEAR = (temp.YEAR+CODE2NUM(REMOTE\_CODE)+found\_pos)%100;                  break;          }          // Current config pos          (pos==0)?Time\_Data\_Display("SET     ", "SECCOND ", temp, 0x3):          (pos==1)?Time\_Data\_Display("SET     ", "MINUTE  ", temp, 0x3):          (pos==2)?Time\_Data\_Display("SET     ", "HOUR    ", temp, 0x3):          (pos==3)?Time\_Data\_Display("SET     ", "DATE    ", temp, 0x3):          (pos==4)?Time\_Data\_Display("SET     ", "MONTH   ", temp, 0x3):          /\*pos=5\*/Time\_Data\_Display("SET     ", "YEAR    ", temp, 0x3);      }      ble\_rcv\_size = 0;      // UART\_Bytes\_Transmit("\nPLS confirm!\n your config!", 0);      pos = YES\_NO();      if(pos) \*t = temp;      return pos;  }  void Send\_Report\_To\_Smartphone(uint8 bypass){      if(bypass || (HOLD == 0 && sys\_t.SECOND==0)){              UART\_Bytes\_Transmit("\n-------------------------", 0);              UART\_Bytes\_Transmit("\nSystem time:\n", 0);              UART\_Bytes\_Transmit(time\_disp\_1, 0);              UART\_Byte\_Transmit('\n');              UART\_Bytes\_Transmit(time\_disp\_2, 0);              UART\_Bytes\_Transmit("\nSystem status:", 0);              UART\_Bytes\_Transmit("\nDevice 1: ", 0); UART\_Bytes\_Transmit((DEV1)?"OFF":"ON", 0);              UART\_Bytes\_Transmit("\nDevice 2: ", 0); UART\_Bytes\_Transmit((DEV2)?"OFF":"ON", 0);              UART\_Bytes\_Transmit("\nDevice 3: ", 0); UART\_Bytes\_Transmit((DEV3)?"OFF":"ON", 0);              UART\_Bytes\_Transmit("\nDevice 4: ", 0); UART\_Bytes\_Transmit((DEV4)?"OFF":"ON", 0);              UART\_Bytes\_Transmit("\nDevice 5: ", 0); UART\_Bytes\_Transmit((DEV5)?"OFF":"ON", 0);              UART\_Byte\_Transmit('\n');      }      HOLD=(sys\_t.SECOND)?0:HOLD+1;  }  void Fetch\_System\_Time(){      DS1302\_Read\_Time(&sys\_t, 0x3F);  }  void Fetch\_User\_Control(){      if(data\_frame==ON\_OFF){          // USER\_DEV\_CTL = (USER\_DEV\_CTL&0xFE) | ((USER\_DEV\_CTL&\_1st\_bit\_mask)?0:1);          USER\_DEV\_CTL = (USER\_DEV\_CTL)?(0):(0xFF);      }      // \_string\_to\_lower\_case(ble\_rcv\_data, ble\_rcv\_size);      if(ble\_has\_contained("inv1", 4))          ble\_rcv\_size = 0,          USER\_DEV\_CTL = (USER\_DEV\_CTL&0x1)?(USER\_DEV\_CTL&0xFE):(USER\_DEV\_CTL|0x1);      elif(ble\_has\_contained("inv2", 4))          ble\_rcv\_size = 0,          USER\_DEV\_CTL = (USER\_DEV\_CTL&0x2)?(USER\_DEV\_CTL&0xFD):(USER\_DEV\_CTL|0x2);      elif(ble\_has\_contained("inv3", 4))          ble\_rcv\_size = 0,          USER\_DEV\_CTL = (USER\_DEV\_CTL&0x4)?(USER\_DEV\_CTL&0xFB):(USER\_DEV\_CTL|0x4);      elif(ble\_has\_contained("inv4", 4))          ble\_rcv\_size = 0,          USER\_DEV\_CTL = (USER\_DEV\_CTL&0x8)?(USER\_DEV\_CTL&0xF7):(USER\_DEV\_CTL|0x8);      elif(ble\_has\_contained("inv5", 4))          ble\_rcv\_size = 0,          USER\_DEV\_CTL = (USER\_DEV\_CTL&0x10)?(USER\_DEV\_CTL&0xEF):(USER\_DEV\_CTL|0x10);      if(ble\_has\_contained("report", 6)){          ble\_rcv\_size = 0;          Send\_Report\_To\_Smartphone(1);      }  }  void Fetch\_System\_Trigger(){      /\*      SYST\_TRIGGER      MSB  7  6  5  4  3  2  1  0  LSB          [x][x][x][x][x][x][x][x]                          |  |  ^~                          |  |  Trigger from Light-dependent resistor                          |  ^~                          |  Trigger from timer\_1 (exist for a second)                          ^~                          Trigger from timer\_2 (exist for a second)      \*/      Read\_AD\_Data(0xA4);      SYST\_TRIGGER = (SYST\_TRIGGER&0xFE)|((Read\_AD\_Data(0xA4)%1000) > 30);      if(time\_equal\_cmp(sys\_t, timer\_1, 0x3F))          SYST\_TRIGGER |= 0x2;      else          SYST\_TRIGGER &= 0xFD;      if(time\_equal\_cmp(sys\_t, timer\_2, 0x3F))          SYST\_TRIGGER |= 0x4;      else          SYST\_TRIGGER &= 0xFB;  }  uint8 Update\_A\_Device\_State(uint8 dev, uint8 dev\_mask, uint16 clt\_dev\_sel\_mask){      uint8 res = 0x80; //No-changed      if((CTL\_DEV\_SEL&clt\_dev\_sel\_mask) == 0){          res = bool\_casting(USER\_DEV\_CTL&dev\_mask);      }      elif((CTL\_DEV\_SEL&clt\_dev\_sel\_mask) == (0x1<<((dev-1)\*2))){          if(SYST\_TRIGGER&\_1st\_bit\_mask)//DAYLIGHT              res = (SYS\_CTL\_INV&dev\_mask)?1:0;          else              res = (SYS\_CTL\_INV&dev\_mask)?0:1;      }      elif((CTL\_DEV\_SEL&clt\_dev\_sel\_mask) == (0x2<<((dev-1)\*2))){          if((SYST\_TRIGGER&\_2nd\_bit\_mask))//TIMER 1              res = (SYS\_CTL\_INV&dev\_mask)?1:0;      }      elif((CTL\_DEV\_SEL&clt\_dev\_sel\_mask) == (0x3<<((dev-1)\*2))){          if((SYST\_TRIGGER&\_3rd\_bit\_mask))//TIMER 2              res = (SYS\_CTL\_INV&dev\_mask)?1:0;      }      return res;  }  void Update\_Device\_State(){      gp\_reg = Update\_A\_Device\_State(1, \_1st\_bit\_mask, DEV1\_MASK);      DEV1 = (gp\_reg&0x80)?(DEV1):(gp\_reg&0x1);      gp\_reg = Update\_A\_Device\_State(2, \_2nd\_bit\_mask, DEV2\_MASK);      DEV2 = (gp\_reg&0x80)?(DEV2):(gp\_reg&0x1);      gp\_reg = Update\_A\_Device\_State(3, \_3rd\_bit\_mask, DEV3\_MASK);      DEV3 = (gp\_reg&0x80)?(DEV3):(gp\_reg&0x1);      gp\_reg = Update\_A\_Device\_State(4, \_4th\_bit\_mask, DEV4\_MASK);      DEV4 = (gp\_reg&0x80)?(DEV4):(gp\_reg&0x1);      gp\_reg = Update\_A\_Device\_State(5, \_5th\_bit\_mask, DEV5\_MASK);      DEV5 = (gp\_reg&0x80)?(DEV5):(gp\_reg&0x1);  }  void MODE\_RIGHT(){      switch (DISP\_MODE) {          case MODE\_NORMAL: return;          case MODE\_SETUP: return;          case MODE\_SETUP\_TIME: DISP\_MODE = MODE\_SETUP\_DEVICE; return;          case MODE\_SETUP\_DEVICE: DISP\_MODE = MODE\_SETUP\_TIME; return;          case MODE\_SETUP\_DEVICE\_1\_DISP: ++DISP\_MODE; return;          case MODE\_SETUP\_DEVICE\_2\_DISP: ++DISP\_MODE; return;          case MODE\_SETUP\_DEVICE\_3\_DISP: ++DISP\_MODE; return;          case MODE\_SETUP\_DEVICE\_4\_DISP: ++DISP\_MODE; return;          case MODE\_SETUP\_DEVICE\_5\_DISP: DISP\_MODE = MODE\_SETUP\_DEVICE\_1\_DISP; return;          case MODE\_SETUP\_TIME\_SYSTEM\_DISP: ++DISP\_MODE; return;          case MODE\_SETUP\_TIME\_TIMER\_1\_DISP: ++DISP\_MODE; return;          case MODE\_SETUP\_TIME\_TIMER\_2\_DISP: DISP\_MODE = MODE\_SETUP\_TIME\_SYSTEM\_DISP; return;      }  }  void MODE\_LEFT(){      switch (DISP\_MODE) {          case MODE\_NORMAL: return;          case MODE\_SETUP: return;          case MODE\_SETUP\_TIME: DISP\_MODE = MODE\_SETUP\_DEVICE; return;          case MODE\_SETUP\_DEVICE: DISP\_MODE = MODE\_SETUP\_TIME; return;          case MODE\_SETUP\_DEVICE\_1\_DISP: DISP\_MODE = MODE\_SETUP\_DEVICE\_5\_DISP; return;          case MODE\_SETUP\_DEVICE\_2\_DISP: --DISP\_MODE; return;          case MODE\_SETUP\_DEVICE\_3\_DISP: --DISP\_MODE; return;          case MODE\_SETUP\_DEVICE\_4\_DISP: --DISP\_MODE; return;          case MODE\_SETUP\_DEVICE\_5\_DISP: --DISP\_MODE; return;          case MODE\_SETUP\_TIME\_SYSTEM\_DISP: DISP\_MODE = MODE\_SETUP\_TIME\_TIMER\_2\_DISP; return;          case MODE\_SETUP\_TIME\_TIMER\_1\_DISP: --DISP\_MODE; return;          case MODE\_SETUP\_TIME\_TIMER\_2\_DISP: --DISP\_MODE; return;      }  }  void Mode\_Change(){      //uint8 found;      uint32 REMOTE\_CODE = read\_extracted\_frame();      if(REMOTE\_CODE){          switch (REMOTE\_CODE) {              case PLAY\_PAUSE:                  if(DISP\_MODE != MODE\_NORMAL)                      MODE\_DOWN(); break;              case MODE:                  if(DISP\_MODE == MODE\_NORMAL){                      DISP\_MODE = MODE\_SETUP; break;                  } elif(DISP\_MODE == MODE\_SETUP){                      DISP\_MODE = MODE\_NORMAL; break;                  } else{                      MODE\_UP(); break;                  }              case NEXT: MODE\_RIGHT(); break;              case PREV: MODE\_LEFT(); break;          }      }  void Mode\_Process(){      switch (DISP\_MODE) {          case MODE\_NORMAL: Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x3); return;          case MODE\_SETUP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP MODE\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP MODE      ", 0, 1, 0);              return;          case MODE\_SETUP\_TIME:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP TIME\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP TIME      ", 0, 1, 0);              return;          case MODE\_SETUP\_TIME\_SYSTEM\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP SYSTEM\_TIME\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP SYS\_TIME  ", 0, 1, 0);              return;          case MODE\_SETUP\_TIME\_SYSTEM\_ACTION:              MODE\_UP();              if(Setup\_Time(&sys\_t)) DS1302\_Write\_Time(&sys\_t, 0x37);              return;          case MODE\_SETUP\_TIME\_TIMER\_1\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP TIMER\_1\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP TIMER\_1   ", 0, 1, 0);              return;          case MODE\_SETUP\_TIME\_TIMER\_1\_ACTION:              MODE\_UP();              Setup\_Time(&timer\_1);              return;          case MODE\_SETUP\_TIME\_TIMER\_2\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP TIMER\_2\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP TIMER\_2   ", 0, 1, 0);              return;          case MODE\_SETUP\_TIME\_TIMER\_2\_ACTION:              MODE\_UP();              Setup\_Time(&timer\_2);              return;          case MODE\_SETUP\_DEVICE:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP DEVICE\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP DEVICE    ", 0, 1, 0);              return;          case MODE\_SETUP\_DEVICE\_1\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP DEVICE 1\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP  DEVICE 1 ", 0, 1, 0);              return;          case MODE\_SETUP\_DEVICE\_1\_ACTION:              //Call set up function!              MODE\_UP();              gp\_reg = Setup\_Device();              if(gp\_reg&0x80) return;              CTL\_DEV\_SEL = (CTL\_DEV\_SEL&0xFFFC) | (gp\_reg&0x3);              SYS\_CTL\_INV = (SYS\_CTL\_INV&0xFE) | ((gp\_reg>>2)&0x1);              if(gp\_reg&0x10) return; //not set dev-state              USER\_DEV\_CTL = (USER\_DEV\_CTL&0xFE) | ((gp\_reg>>3)&0x1);              return;          case MODE\_SETUP\_DEVICE\_2\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP DEVICE 2\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP  DEVICE 2 ", 0, 1, 0);              return;          case MODE\_SETUP\_DEVICE\_2\_ACTION:              //Call set up function!              MODE\_UP();              gp\_reg = Setup\_Device();              if(gp\_reg&0x80) return;              CTL\_DEV\_SEL = (CTL\_DEV\_SEL&0xFFF3) | ((gp\_reg&0x3)<<2);              SYS\_CTL\_INV = (SYS\_CTL\_INV&0xFD) | ((gp\_reg>>1)&0x2);              if(gp\_reg&0x10) return; //not set dev-state              USER\_DEV\_CTL = (USER\_DEV\_CTL&0xFD) | ((gp\_reg>>2)&0x2);              return;          case MODE\_SETUP\_DEVICE\_3\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP DEVICE 3\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP  DEVICE 3 ", 0, 1, 0);              return;          case MODE\_SETUP\_DEVICE\_3\_ACTION:              MODE\_UP();              gp\_reg = Setup\_Device();              if(gp\_reg&0x80) return;              CTL\_DEV\_SEL = (CTL\_DEV\_SEL&0xFFCF) | ((gp\_reg&0x3)<<4);              SYS\_CTL\_INV = (SYS\_CTL\_INV&0xFB) | ((gp\_reg)&0x4);              if(gp\_reg&0x10) return; //not set dev-state              USER\_DEV\_CTL = (USER\_DEV\_CTL&0xFB) | ((gp\_reg>>1)&0x4);              return;          case MODE\_SETUP\_DEVICE\_4\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP DEVICE 4\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP  DEVICE 4 ", 0, 1, 0);              return;          case MODE\_SETUP\_DEVICE\_4\_ACTION:              MODE\_UP();              gp\_reg = Setup\_Device();              if(gp\_reg&0x80) return;              CTL\_DEV\_SEL = (CTL\_DEV\_SEL&0xFF3F) | ((gp\_reg&0x3)<<6);              SYS\_CTL\_INV = (SYS\_CTL\_INV&0xF7) | ((gp\_reg<<1)&0x8);              if(gp\_reg&0x10) return; //not set dev-state              USER\_DEV\_CTL = (USER\_DEV\_CTL&0xF7) | ((gp\_reg)&0x8);              return;          case MODE\_SETUP\_DEVICE\_5\_DISP:              Time\_Data\_Display("TIME:   ", "DATE:   ", sys\_t, 0x1);              // UART\_Bytes\_Transmit("\nSETUP DEVICE 5\n", 0);              LCD\_Simple\_Set\_Text\_1("SETUP  DEVICE 5 ", 0, 1, 0);              return;          case MODE\_SETUP\_DEVICE\_5\_ACTION:              MODE\_UP();              gp\_reg = Setup\_Device();              if(gp\_reg&0x80) return;              CTL\_DEV\_SEL = (CTL\_DEV\_SEL&0xFCFF) | ((gp\_reg&0x3)<<8);              SYS\_CTL\_INV = (SYS\_CTL\_INV&0xEF) | ((gp\_reg<<2)&0x10);              if(gp\_reg&0x10) return; //not set dev-state              USER\_DEV\_CTL = (USER\_DEV\_CTL&0xEF) | ((gp\_reg<<1)&0x10);              return;          default:              UART\_Bytes\_Transmit("\nSYSTEM FAULT!!!\nPLS PRESS RESET!!!\n", 0);              LCD\_Simple\_Set\_Text\_2("Wrong MODE...", 0, 0, 0);              LCD\_Simple\_Set\_Text\_1("PLS press RESET", 0, 1, 0);      }  }  void Hello(){      UART\_Bytes\_Transmit("\nHello!\nFrom ngxx.fus!\n", 0);      LCD\_Simple\_Set\_Text\_2("Hello!",0, 0, 0);      LCD\_Simple\_Set\_Text\_1("From NGXXFUS :>",0, 1, 0);      delay\_ms(2000);  }  void Main\_Initial(){      LCD\_Initial();      DS1302\_Initial();      IR\_Reading\_Initial();      Bluetooth\_UART\_Initial();      SPI\_Initial();      //--------------//      Hello();      //--------------//      DS1302\_Write\_Time(&sys\_t, 0x7F);      //--------------//      CTL\_DEV\_SEL=0x0E4;      SYS\_CTL\_INV=0x2;      USER\_DEV\_CTL=0x0;  } |

## main.c

|  |  |
| --- | --- |
| main.c | #include "main.h"  int main(void){      Main\_Initial();      while(0x1){          Fetch\_System\_Time();          Send\_Report\_To\_Smartphone(0);          Fetch\_User\_Control();          Fetch\_System\_Trigger();          Update\_Device\_State();          Mode\_Change();          Mode\_Process();          // delay\_ms(300);      }      return 0;  } |