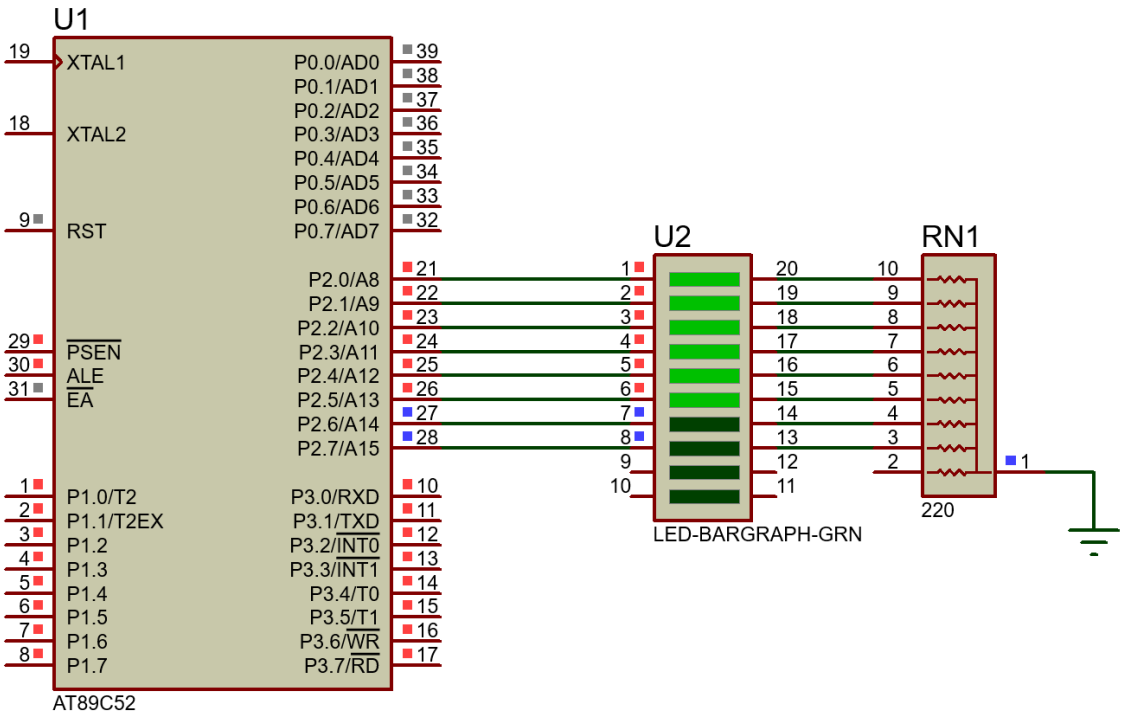
BÁO CÁO CUỐI KỲ

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

BÀI THỰC HÀNH 1 – ĐIỀU KHIỂN 8 LED



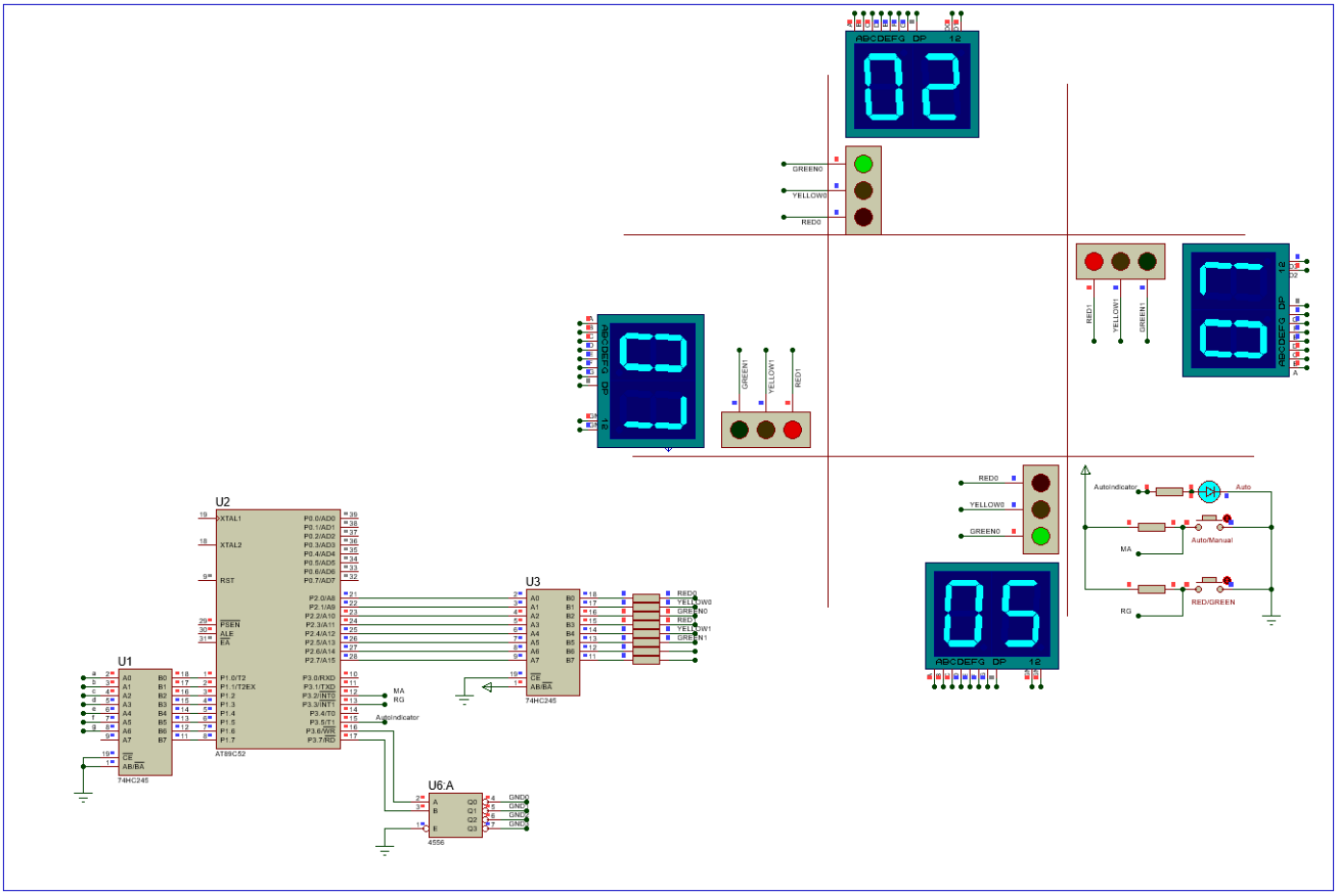
main.c

main.c

```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



main.h

main.h

```
#include <stdio.h>
#include <REGX52.h>
#define elif else if

typedef unsigned int UINT32;
#define DECREASE_ONE(VAR) VAR = (VAR>0?(VAR-1):VAR)
// Decreas VAR if VAR is greater than ZERO, else DO NOTHING!
#define GET_STATE(POS) ((POS>0)?((TRAFFIC_LIGHT & 0x38)>>0x3)*1U:(TRAFFIC_LIGHT & 0x7)*1U)
//POSITION DESCRIPTION:
//   0   : Traffic light 0
//   1   : Traffic light 1
//RETURN CODE DESCRIPTION:
//  0x1  : RED
//  0x2  : YELLOW
//  0x4  : GREEN

#define RED 0x1
#define YELLOW 0x2
#define GREEN 0x4
#define LED_OFF 0xA
#define MANUAL 0x0;
#define AUTO 0x1;

sbit M_A = P3^2;
//<INPUT> M_A: Manual/Auto mode
sbit R_G = P3^3;
//<INPUT> R_G: RED / GREEN (Only for *manual mode*)

sbit G0  = P2^2;
sbit G1  = P2^3;
sbit G2  = P2^4;

// Demultiplexer 2->4
// G0 G1   | GND3 GND2 GND1 GND0
//  0  0   |    H    H    H    L
```

```
//  0  1   |    H    H    L    H
//  1  0   |    H    L    H    H
//  1  1   |    L    H    H    H
// Note: Active-Low | MSB -> LSB

UINT32 STATE = RED;
// The state of traffic light (to prevent set the same state again)
UINT32 COUNT_0;
// Count for the main traffic light
UINT32 COUNT_1;
// Count for the order traffic light
UINT32 SINGLE_LED_DISPLAY_T = 50;
// The time use show a single 7-seg LED
UINT32 RED_T = 0;
UINT32 GREEN_T = 0;
UINT32 YELLOW_T = 3;
// The time (in seccond) for the YELLOW state while changes state
// from GREEN->RED.

const UINT32 DIGIT_CODE[] = {0X3F, 0X06, 0X5B, 0X4F, 0X66, 0X6D, 0X7D,
                             0X07, 0X7F, 0XEF, 0X0};
// 7-seg LED CODE (Common Anode)

#define LED P0

// P2: [x][G][F][E][D][C][B][A]
// Note: Active-Hight | MSB -> LSB | Common Anode

#define TRAFFIC_LIGHT P2
//            | traffic light 1 | traffic light 0 |
// P2: [x][x]      [G][Y][R]          [G][Y][R]
// Note: Active-Hight | MSB -> LSB | x : "don't care!"

void INITIAL(){
    /*Set initial state*/
    TRAFFIC_LIGHT = 0x12;
    COUNT_0 = 0x0;
    COUNT_1 = 0x0;
    LED = DIGIT_CODE[LED_OFF];
    G0 = G1 = 0;
}

UINT32 AUTO_MANUAL(){
    if (M_A) return AUTO;
    return MANUAL;
}

UINT32 RED_GREEN(){
    return (R_G)?(RED):(GREEN);
}

void DELAY(UINT32 t){
    /*Delay in mili-second*/
    UINT32 i;
    for(i = 0; i < 12*t; i++);
}

void SET_LED(UINT32 D){
    //CODE = 10 : turn off LED.
    LED = DIGIT_CODE[D];
}

void SET_YELLOW_TIMER(UINT32 _YELLOW_T){
    YELLOW_T = _YELLOW_T;
}

void SET_RED_GREEN_TIMER(UINT32 _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_DISPLAY_PERIOD(UINT32 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_TIMER(UINT32 REV){
    //REV DESCRIPTION
    //REV = 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.
    //REV = 1 :
    //  The other side, Traffic Light 1 is currently GREEN,
    //and will be changed to YELLOW, then RED.
    if(REV == 0){
        COUNT_0 = RED_T;
        COUNT_1 = GREEN_T;
    }else{
        COUNT_0 = GREEN_T;
        COUNT_1 = RED_T;
    }

}

void DISPLAY_LED(){
    /*This function only runs ONE SECOND*/
    UINT32 i = 0;
    for(i = 0; i < 1400/(4*SINGLE_LED_DISPLAY_T); i++){
            G0 = 1; G1 = 0; G2 = 0;
            if(COUNT_0 != 0) SET_LED((COUNT_0/10)%10);
            else SET_LED(LED_OFF);
            DELAY(SINGLE_LED_DISPLAY_T);

            G0 = 1; G1 = 0; G2 = 1;
            if(COUNT_1 != 0)SET_LED((COUNT_1/10)%10);
            else SET_LED(LED_OFF);
            DELAY(SINGLE_LED_DISPLAY_T);

            G0 = 0; G1 = 0; G2 = 0;
            if(COUNT_0 != 0) SET_LED(COUNT_0%10);
            else SET_LED(LED_OFF);
            DELAY(SINGLE_LED_DISPLAY_T);

            G0 = 0; G1 = 0; G2 = 1;
            if(COUNT_1 != 0) SET_LED(COUNT_1%10);
            else SET_LED(LED_OFF);
            DELAY(SINGLE_LED_DISPLAY_T);
    }
}

void SET_TRAFFIC_LIGHT(UINT32 POS, UINT32 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

    UINT32 TL = TRAFFIC_LIGHT;
    UINT32 CURRENT_STATE = GET_STATE(POS) & 0x7;
    CODE = CODE & 0x7;
    // Standardizing CODE

    if( (CODE & CURRENT_STATE) == 0 ){
        //New state is the same with prev state --> abort!
        if(POS == 0)
            TRAFFIC_LIGHT = (TL & 0xF8) | CODE;
        else
            TRAFFIC_LIGHT = (TL & 0xC7) | (CODE << 0x3);
    }
}

void SET_STATE(UINT32 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);
        }
    }
}
```

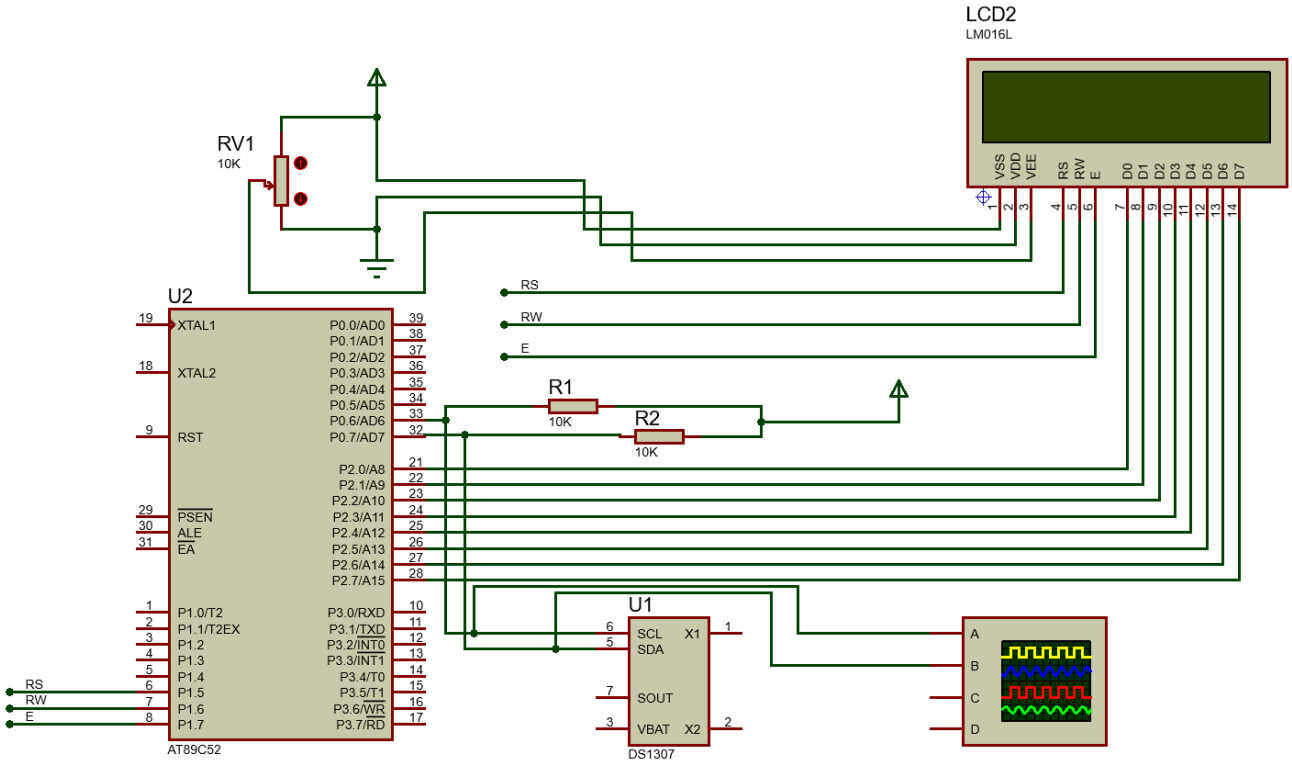
main.c

main.c

```
#include "main.h"

void main(){

    INITIAL();
    SET_YELLOW_TIMER(5);
    SET_RED_GREEN_TIMER(15);
    SET_DISPLAY_PERIOD(12);

    while ( 0x1 ){
        if( AUTO_MANUAL() ){
            SET_STATE(RED);
            SET_TIMER(0);
            SET_STATE(GREEN);
            SET_TIMER(1);
        }else{
            STOP_COUNT();
            SET_STATE(RED_GREEN());
        }
    }
}
```

BÀI THỰC HÀNH 3 – LỊCH VẠN NIÊN (SIM)

Proteus



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 0x0FF

//  |<-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)
#endif
#ifndef FOR_reverse
#define FOR_reverse(i, a, b) for(i = (a); i >= (b); --i)
#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

```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
};

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;
    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

Time.h

	Đã định nghĩa ở BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1

XPT2046.h

	Đã định nghĩa ở BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1

ThreeWiresProtocol.h

	Đã định nghĩa ở BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1

DS1302.h

	Đã định nghĩa ở BÀI THỰC HÀNH SỐ 7 – NHÀ THÔNG MINH 1

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;
}
```