65010039 กลวัชร อินทร์แป้น
65010429 ธนศักดิ์ สองศรี

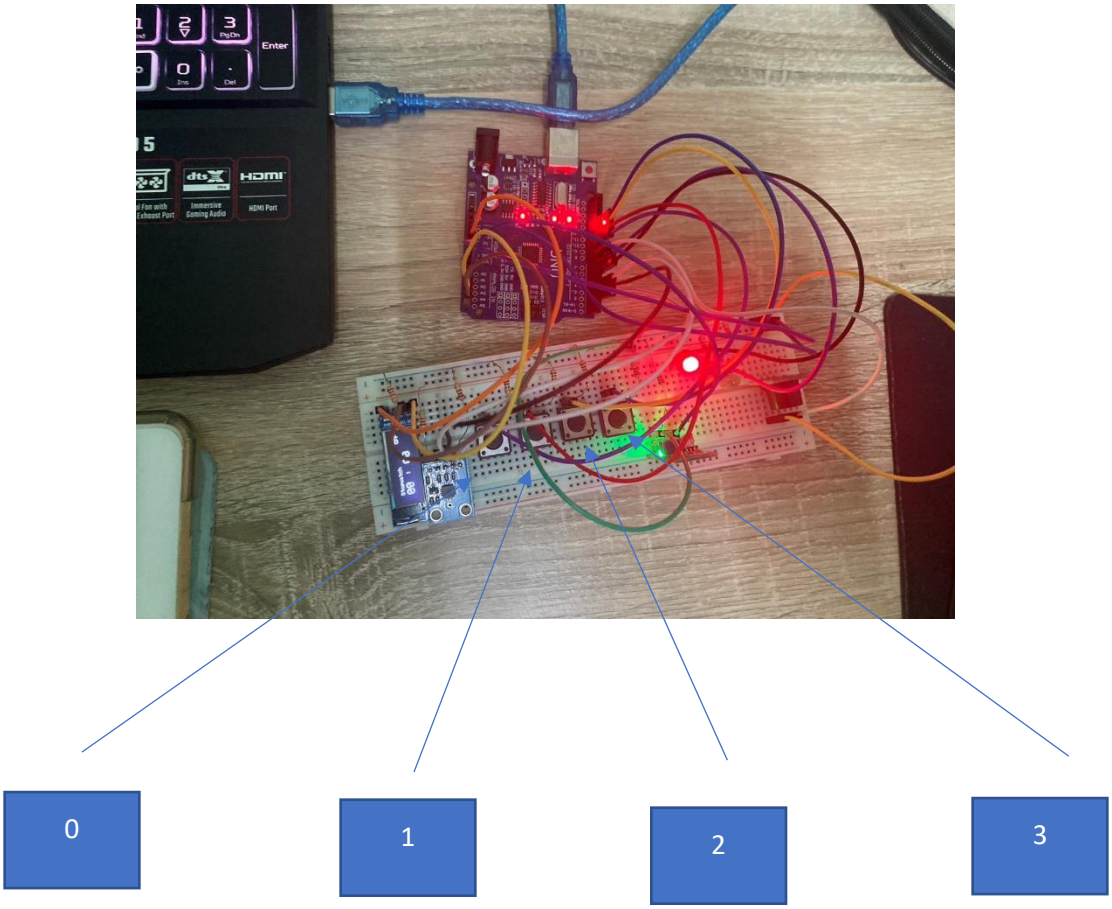
Assignment #7 : mini clock

แนวคิดการออกแบบ : เนื่องจากชอบไฟ LED เลยนำมาใช้กับนาฬิกาดิจิตัลโดยนอกจากจะใช้เพื่อตกแต่งแล้ว ยังใช้คอยเป็นตัวบอกสถานะว่าโหมดนั้นทำงานอยู่หรือไม่

ฟังก์ชันของ mini clock :

- เวลาและชื่อ Mode บนจอ OLED
- นาฬิกาจับเวลา
- นาฬิกาปลุกตั้งเวลา
- แสงไฟ LED, ลำโพงคอยบอกสถานะ
- ปรับทิศทางของหน้าจอตามความเอียง

การใช้งานโดยย่อ :



65010039 กลวัชร อินทร์แป้น
65010429 ธนศักดิ์ สองศรี

นาฬิกามี 3 โหมด โดยกดปุ่ม 3 เพื่อเปลี่ยนโหมด :

1. นาฬิกาหลัก : เวลาจะไหลไปเรื่อยๆ และ บันทึกค่าใน EEPROM

   - กดปุ่ม 0 จะเพิ่มชั่วโมงทีละ 1

   - กดปุ่ม 1 จะเพิ่มนาทีทีละ 1

   - กดปุ่ม 2 จะปรับตัวคูณการเพิ่มของของปุ่ม 0,1 โดยจะมีค่าในช่วง 0-3 เช่น ปรับเป็น 2 เมื่อกดปุ่ม 1 จะเพิ่มชั่วโมงทีละ 2

2. นาฬิกาจับเวลา : จะจับเวลาไปเรื่อยๆ

   - กดปุ่ม 0 จะเริ่มจับเวลา กดอีกครั้งจะหยุดจับเวลา

   - กดปุ่ม 1 จะหยุดการจับเวลา และรีเซ็ทค่า

   - กดปุ่ม 2 จะเริ่มจับเวลาถอยหลัง กดอีกครั้งจะหยุดจับเวลา

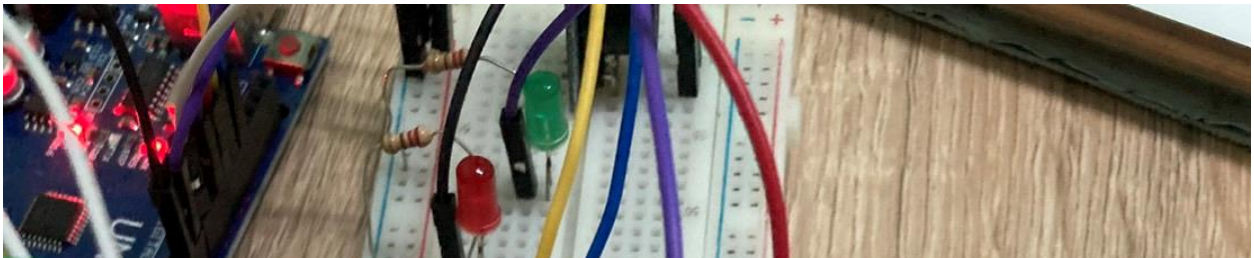
   - โดยถ้าเวลายังเดินอยู่จะโชว์ LED เขียว, ถ้าเวลาหยุดเดินจะโชว์ LED แดง

3. นาฬิกาจับเวลา : Set เวลาที่ต้องการ เมื่อถึงเวลาจะมีเสียงขึ้น

   - กดปุ่ม 0 จะเพิ่มชั่วโมงทีละ 1

   - กดปุ่ม 1 จะเพิ่มนาทีทีละ 1

   - กดปุ่ม 2 จะเปิด/ปิด นาฬิกาปลุก ถ้าเปิด(LEDเขียว)/ปิด(LEDแดง)



65010039 กลวัชร อินทร์แป้น
65010429 ธนศักดิ์ สองศรี

โปรแกรมและการอธิบายโปรแกรมโดยย่อ :

65010039 กลวัชร อินทร์แป้น
65010429 ธนศักดิ์ สองศรี

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <EEPROM.h>
#include <TimerOne.h>

struct Time {
  int hour;
  int min;
  int sec;
};

// LED Settings
#define LED_RED 6
#define LED_GREEN 7

// OLED Settings
#define OLED_RESET -1
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 OLED(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Button Settings
#define BTN_1 10
#define BTN_2 11
#define BTN_3 12
#define BTN_4 13
int button[4] = { BTN_1, BTN_2, BTN_3, BTN_4 };
int reading[4];
int buttonState[4];
int lastButtonState[4] = { HIGH, HIGH, HIGH, HIGH };
unsigned long long int debounceDelay = 50;
unsigned long long int lastDebounceTime[4];

#define BUZZER_PIN 8

Time Clock = { 0, 0, 0 };

Time Stopwatch = { 0, 0, 0 };
int Stopwatch_Pause = 1;
int Countdown_Pause = 1;

Time Alarm = { 0, 0, 0 };
Time Alarm_Set = { 0, 0, 0 };
```

```
int Alarm_On = 0;


// Mode Setting
#define CLOCK_MODE 0
#define STOPWATCH_MODE 1
#define ALARM_MODE 2
int MODE = CLOCK_MODE;

// Accelermeter Settings
const unsigned int X_AXIS_PIN = A0;
const unsigned int Y_AXIS_PIN = A1;
const unsigned int Z_AXIS_PIN = A2;
const unsigned int NUM_AXES = 3;
const unsigned int PINS[NUM_AXES] = {
  X_AXIS_PIN, Y_AXIS_PIN, Z_AXIS_PIN
};
const unsigned int BUFFER_SIZE = 16;
int buffer[NUM_AXES][BUFFER_SIZE];
int buffer_pos[NUM_AXES] = { 0 };

// ACCELEROMETER
int get_axis(const int axis) {
  delay(1);
  buffer[axis][buffer_pos[axis]] = analogRead(PINS[axis]);
  buffer_pos[axis] = (buffer_pos[axis] + 1) % BUFFER_SIZE;
  long sum = 0;
  for (unsigned int i = 0; i < BUFFER_SIZE; i++)
    sum += buffer[axis][i];
  return round(sum / BUFFER_SIZE);
}

int get_x() {
  return get_axis(0);
}
int get_y() {
  return get_axis(1);
}
int get_z() {
  return get_axis(2);
}

// Debounce
int debounce(int i) {
  int isChange = 0;
```

```
  reading[i] = digitalRead(button[i]);
  if (reading[i] != lastButtonState[i]) {
    lastDebounceTime[i] = millis();
  }
  if ((millis() - lastDebounceTime[i]) >= debounceDelay) {
    if (reading[i] != buttonState[i]) {
      buttonState[i] = reading[i];
      isChange = 1;
    }
  }
  lastButtonState[i] = reading[i];
  return isChange;
}

void LED_Status(int num, int State){
  digitalWrite(num, State);
}

// Timer One
void interruptClock() {
  timer();
  Stopwatch_Timer();
  Alarm_Start();
}

int Clock_Mul = 1;
// Main Clock
void timer() {
  Clock.sec += 1;
  Clock.min += Clock.sec / 60;
  Clock.sec %= 60;
  Clock.hour += Clock.min / 60;
  Clock.min %= 60;
  Clock.hour %= 24;
  if (Clock.sec == 0) {
    EEPROM.update(20, Clock.hour);
    EEPROM.update(21, Clock.min);
  }
}
void change_hour() {
  Clock.hour += 1 * Clock_Mul;
  EEPROM.update(20, Clock.hour);
  Clock.hour %= 24;
}
void change_min() {
```

```
  Clock.min += 1 * Clock_Mul;
  Clock.hour += Clock.min / 60;
  EEPROM.update(21, Clock.min);
  Clock.min %= 60;
}

// Stopwatch
void Stopwatch_Timer() {
  if (!Stopwatch_Pause && Countdown_Pause == 1) {
    Stopwatch.sec += 1;
    Stopwatch.min += Stopwatch.sec / 60;
    Stopwatch.sec %= 60;
    Stopwatch.hour += Stopwatch.min / 60;
    Stopwatch.min %= 60;
  }
  else if(!Countdown_Pause && Stopwatch_Pause == 1 && Stopwatch.sec > 0){
    Stopwatch.sec -= 1;
    Stopwatch.min -= Stopwatch.sec / 60;
    Stopwatch.sec %= 60;
    Stopwatch.hour -= Stopwatch.min / 60;
    Stopwatch.min %= 60;
    if(Stopwatch.sec == 0 && Countdown_Pause == 0){
      tone(BUZZER_PIN, 1000, 200);
      Stopwatch_Pause = 1;
      Countdown_Pause = 1;
    }
  }
}

void Stopwatch_Reset() {
  Stopwatch.hour = 0;
  Stopwatch.min = 0;
  Stopwatch.sec = 0;
}

// Alarm
void change_hour_alarm() {
  Alarm_Set.hour += 1 * Clock_Mul;
  //EEPROM.update(20, Alarm_Set.hour);
  Alarm_Set.hour %= 24;
}
void change_min_alarm() {
  Alarm_Set.min += 1 * Clock_Mul;
  //EEPROM.update(21, Alarm_Set.min);
  Alarm_Set.min %= 60;
```

```
}
void Alarm_Start() {
  if (Clock.sec == 0) {
    Alarm_Set.min = Clock.min;
  }
  if (Clock.min == 0) {
    Alarm_Set.hour = Clock.hour;
  }
}

// Clock
void change_mode() {
  MODE += 1;
  MODE %= 3;
  /*if (MODE == ALARM_MODE && Alarm_On == 0) {
    Alarm.hour = Clock.hour;
    Alarm.min = Clock.min;
  }*/
}

// Time Text
String clockText = "00 : 00";
String clockText_Sec = "00";
String stopwatchText = "00 : 00";
String alarmText = "00 : 00";
String alarm_setText = "00 : 00";
void time_text() {
  clockText[0] = (Clock.hour / 10) + '0';
  clockText[1] = (Clock.hour % 10) + '0';

  clockText[5] = (Clock.min / 10) + '0';
  clockText[6] = (Clock.min % 10) + '0';

  clockText_Sec[0] = (Clock.sec / 10) + '0';
  clockText_Sec[1] = (Clock.sec % 10) + '0';

  stopwatchText[0] = (Stopwatch.min / 10) + '0';
  stopwatchText[1] = (Stopwatch.min % 10) + '0';

  stopwatchText[5] = (Stopwatch.sec / 10) + '0';
  stopwatchText[6] = (Stopwatch.sec % 10) + '0';

  alarmText[0] = (Alarm.hour / 10) + '0';
  alarmText[1] = (Alarm.hour % 10) + '0';
```

```
  alarmText[5] = (Alarm.min / 10) + '0';
  alarmText[6] = (Alarm.min % 10) + '0';

  alarm_setText[0] = (Alarm_Set.hour / 10) + '0';
  alarm_setText[1] = (Alarm_Set.hour % 10) + '0';

  alarm_setText[5] = (Alarm_Set.min / 10) + '0';
  alarm_setText[6] = (Alarm_Set.min % 10) + '0';
}

void Display_Text_OLED(int x, int y, int TextSize, String text){
  OLED.setTextSize(TextSize);
  OLED.setCursor(x, y);
  OLED.println(text);
}

void setup() {
  Serial.begin(9600);
  if (!OLED.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("SSD1306 allocation failed");
  } else {
    Serial.println("All OLED Start Work !!!");
  }
  for (int i = 0; i < 4; i++) {
    pinMode(button[i], INPUT_PULLUP);
  }
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  Timer1.initialize(1000000);
  Timer1.attachInterrupt(interruptClock);

  Clock.hour = EEPROM.read(20);
  Clock.min = EEPROM.read(21);

  /*Alarm_Set.hour = EEPROM.read(20);
  Alarm_Set.hour = EEPROM.read(21);*/
}

void loop() {
  Alarm.hour = Clock.hour;
  Alarm.min = Clock.min;
  Alarm.sec = Clock.sec;

  OLED.clearDisplay();
  OLED.setTextColor(WHITE);
```

```
if (debounce(0)) {
  if (!digitalRead(button[0])) {
    switch (MODE) {
      case CLOCK_MODE:
        change_hour();
        break;
      case STOPWATCH_MODE:
        Countdown_Pause = 1;
        Stopwatch_Pause = !Stopwatch_Pause;
        tone(BUZZER_PIN, 100, 100);
        break;
      case ALARM_MODE:
        change_hour_alarm();
        break;
    }
  }
}

if (debounce(1)) {
  if (!digitalRead(button[1])) {
    switch (MODE) {
      case CLOCK_MODE:
        change_min();
        break;
      case STOPWATCH_MODE:
        Stopwatch_Pause = 1;
        Stopwatch_Reset();
        break;
      case ALARM_MODE:
        change_min_alarm();
        break;
    }
  }
}

if (debounce(2)) {
  if (!digitalRead(button[2])) {
    switch (MODE) {
      case CLOCK_MODE:
        Clock_Mul += 1;
        Clock_Mul %= 4;
        break;
      case STOPWATCH_MODE:
        Stopwatch_Pause = 1;
```

```
          if(Stopwatch.sec == 0 && Stopwatch.min == 0 && Stopwatch.hour == 0){
            Countdown_Pause = 1;
          }
          else{
            Countdown_Pause = !Countdown_Pause;
          }

          tone(BUZZER_PIN, 100, 100);
        case ALARM_MODE:
          Alarm_On = Alarm_On == 0 ? 1 : 0;
          break;
      }
    }
  }

  if (debounce(3)) {
    if (!digitalRead(button[3])) {
      change_mode();
    }
  }

  if ((Alarm_On) && (Alarm_Set.hour == Clock.hour) && (Alarm_Set.min ==
Clock.min) && Clock.sec == 0) {
    Alarm_On = 0;
    tone(BUZZER_PIN, 100, 1000);
  }

  Serial.print(get_x());
  Serial.print(" ");
  Serial.print(get_y());
  Serial.print(" ");
  Serial.println(get_z());

  time_text();

  String Clock_Mul_Str = "x ";
  Clock_Mul_Str[1] = Clock_Mul + '0';

  switch (MODE) {
    case CLOCK_MODE:
      LED_Status(LED_RED, 0);
      LED_Status(LED_GREEN, 0);
      Display_Text_OLED(10, 0, 1, "1 Clock");
      Display_Text_OLED(10, 18, 2, clockText);
      Display_Text_OLED(115, 0, 1, Clock_Mul_Str);
```

```
      Display_Text_OLED(110, 25, 1, clockText_Sec);
      break;
    case STOPWATCH_MODE:
      Display_Text_OLED(10, 0, 1, "2 Stopwatch");
      Display_Text_OLED(10, 12, 2, stopwatchText);
      if(Stopwatch_Pause && Countdown_Pause){
        Display_Text_OLED(110, 0, 1, "OFF");
        LED_Status(LED_RED, 1);
        LED_Status(LED_GREEN, 0);
      }
      else{
        Display_Text_OLED(110, 0, 1, "ON");
        LED_Status(LED_RED, 0);
        LED_Status(LED_GREEN,1);
      }
      break;
    case ALARM_MODE:
      Display_Text_OLED(10, 0, 1, "3 Alarm");
      Display_Text_OLED(10, 25, 1, Alarm_On == 0 ? "OFF" : "ON");
      Display_Text_OLED(60, 12, 1, "NOW");
      Display_Text_OLED(86, 12, 1, alarmText);
      Display_Text_OLED(60, 25, 1, "SET");
      Display_Text_OLED(86, 25, 1, alarm_setText);
      //Display_Text_OLED(100, 0, 1, clockText_Sec);
      if(Alarm_On){
        LED_Status(LED_RED, 0);
        LED_Status(LED_GREEN, 1);
      }else{
        LED_Status(LED_RED, 1);
        LED_Status(LED_GREEN, 0);
      }
      break;
  }
  //Song();
  if(get_x() > 300){
    OLED.setRotation(0);
  }
  else{
    OLED.setRotation(2);
  }
  OLED.display();
}
```

คำอธิบาย Code :

- get_axis() :

ใช้ส่งค่าพิกัด ของ OLED

- debounce(int i):

ใช้ตรวจสอบการกดปุ่มหากกดค้างให้ส่งคืนค่าเพียง 1 ครั้ง

- LED_Status(int num, int State):

ใช้แสดงค่า LED ตัวที่ num ว่า ติด/ดับ

- interruptClock():

ทำให้ฟังก์ชันภายใน เกิดการ interrupt ทุกๆ 1 วินาที

- timer():

ฟังก์ชัน Main Clock มีค่า ชั่วโมง:นาที:วินาที และบันทึกลง EEPROM

- Stopwatch_timer():

ฟังก์ชัน Stopwatch แสดงค่า นาที:วินาที สามารถจับเวลา/จับเวลาถอยหลังได้

- change_mode():

กดเพื่อเปลี่ยนโหมด

- time_text():

แปลงตัวเลขจากนาฬิกาแต่ละโหมดให้เป็นข้อความ

- Display_Text_OLED():

แสดงข้อความที่ตำแหน่ง x, y, ขนาด, ข้อความ

- void setup():

Setup อุปกรณ์, การดีเลย์, การอ่านค่า EEPROM

65010039 กลวัชร อินทร์แป้น
65010429 ธนศักดิ์ สองศรี

- void loop():

แสดงชื่อ Mode, ค่าเวลา ในจอ OLED

เช็คการกดปุ่มในแต่ละ Mode แล้วทำตามเงื่อนไขใน Mode นั้น

เช็คการแสดงผลให้ตรงตามแนวที่เอียง