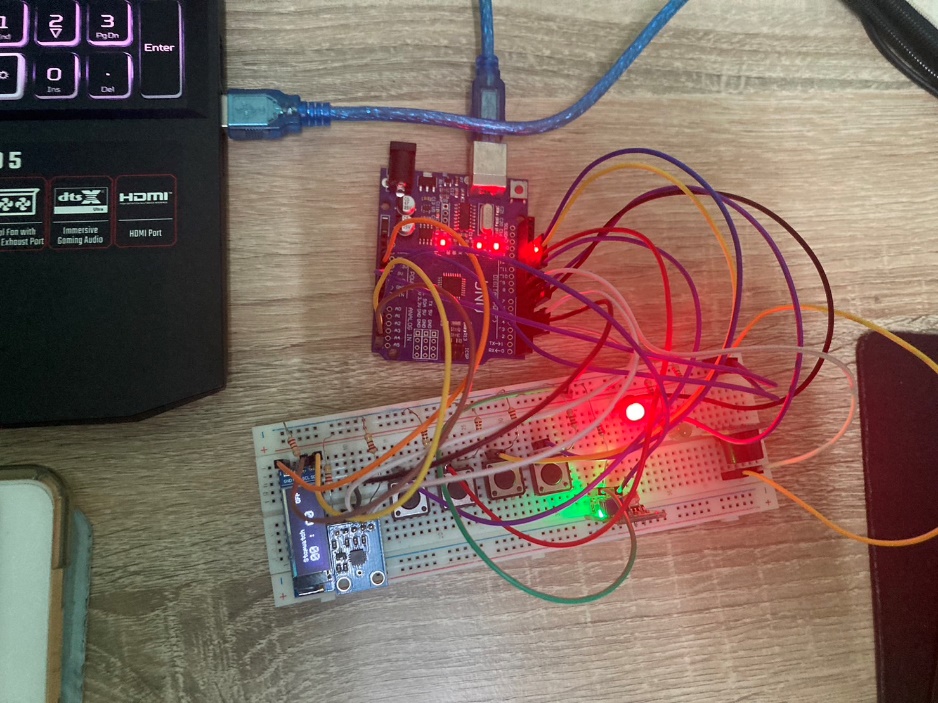
**Assignment #7 : mini clock**

**แนวคิดการออกแบบ** : เนื่องจากชอบไฟ LED เลยนำมาใช้กับนาฬิกาดิจิตัลโดยนอกจากจะใช้เพื่อตกแต่งแล้ว ยังใช้คอยเป็นตัวบอกสถานะว่าโหมดนั้นทำงานอยู่หรือไม่

**ฟังก์ชันของ mini clock** :

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

**การใช้งานโดยย่อ** :



3

2

1

0

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 ปรับ

เป็น 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)

A picture containing electronics, circuit

Description automatically generated

**โปรแกรมและการอธิบายโปรแกรมโดยย่อ :**

#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

- void loop():

แสดงชื่อ Mode, ค่าเวลา ในจอ OLED

เช็คการกดปุ่มในแต่ละ Mode แล้วทำตามเงื่อนไขใน Mode นั้น

เช็คการแสดงผลให้ตรงตามแนวที่เอียง