รายงานโครงงาน

Espunia midi controller for producer

จัดทำโดย

6410110071 **จิเฟอร์ดินานด์ เจะและ**

6410110109 **ชูศักดิ์ ขวัญรักศรี**

6410110585 **อัมมัร ไตรฐากูร**

240 - 319 **ชุดวิชานักพัฒนาระบบฝังตัว**

ตอนที่ 2 **ภาคการเรียนที่** 2 **ปีการศึกษา** 2565

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่

ทำไมถึงอยากทำ

ด้วยประสบการณ์ในการทำเพลงในวงดนตรีของเรา ทำให้พวกเราตระหนักว่าการทำเพลงโดยใช้โปรแกรมอัดเสียงโดยใช้เมาส์และคีย์บอร์ดนั้นมีความยุ่งยากและไม่เป็นมิตรกับผู้ใช้งาน โดยเฉพาะในส่วนของการอัดเครื่องดนตรีต่างๆ ดังนั้นพวกเราจึงตั้งใจที่จะสร้าง MIDI Controller for Producer ขึ้นมาเพื่อตอบสนองความต้องการของ Producer ที่ต้องการเครื่องมือที่ใช้งานง่ายและสะดวกในการสร้างสรรค์ผลงานเพลง

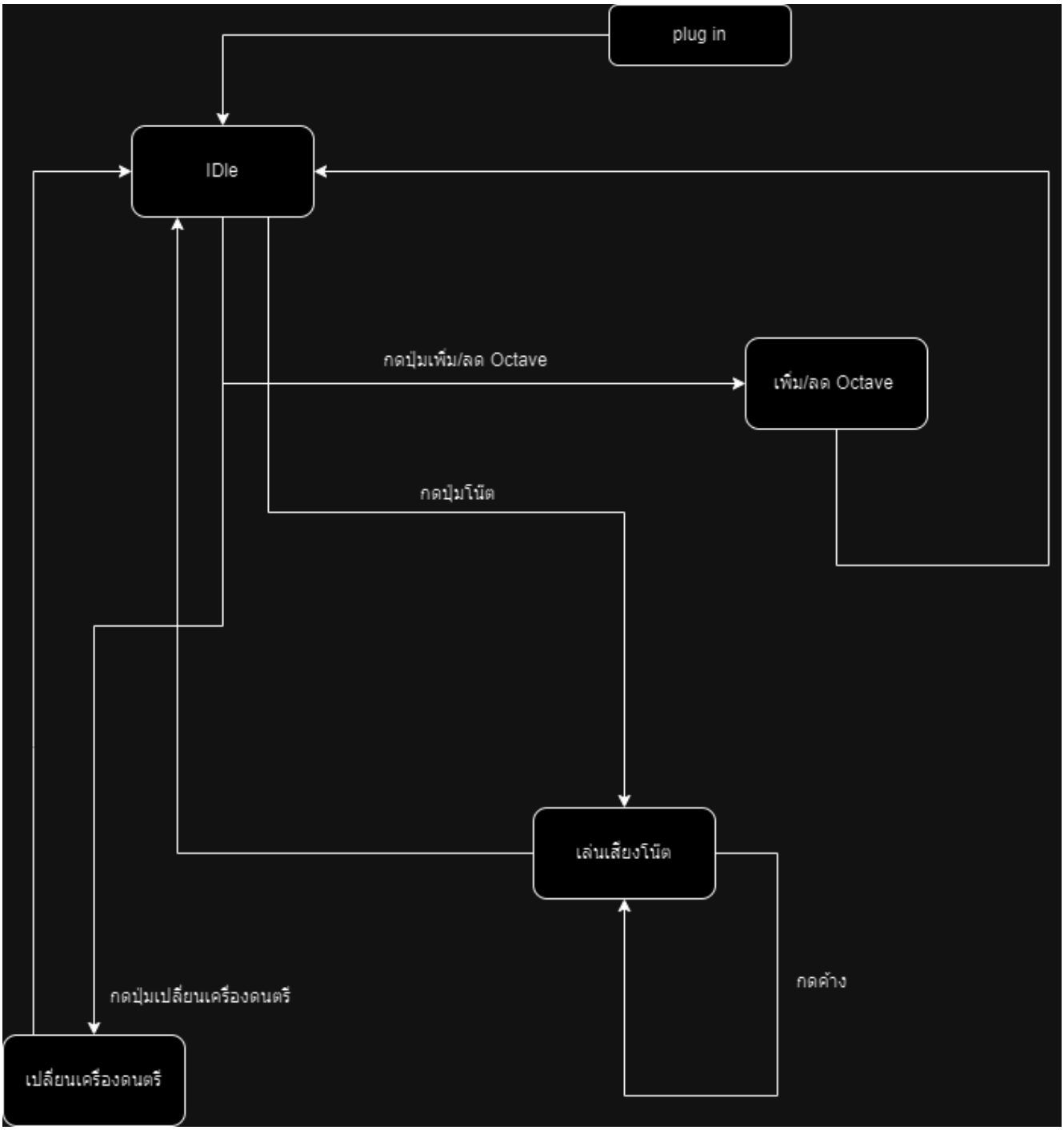
วัตถุประสงค์

MIDI Controller for Producer คืออุปกรณ์ควบคุมที่ช่วยให้ผู้ผลิตเพลงสามารถควบคุมซอฟต์แวร์การผลิตเพลงดิจิทัล ของตนได้โดยตรง คอนโทรลเลอร์ MIDI เหล่านี้มีคีย์บอร์ดแบบจำลองเครื่องดนตรีอื่น ๆ 24 ปุ่ม ปุ่มสำหรับควบคุมการเปลี่ยนเครื่องดนตรี และมีปุ่มเพิ่มลดoctave อีกทั้งยังมีจอOLED เพื่อแสดงเครื่องดนตรีนะขณะนั้นอีกทั้งยังมีโพเทนอเนกประสงค์ 4 ตัวที่สามารถกำหนดเพื่อควบคุมเราสมบัติเสียงต่าง ๆ ของเครื่องดนตรี

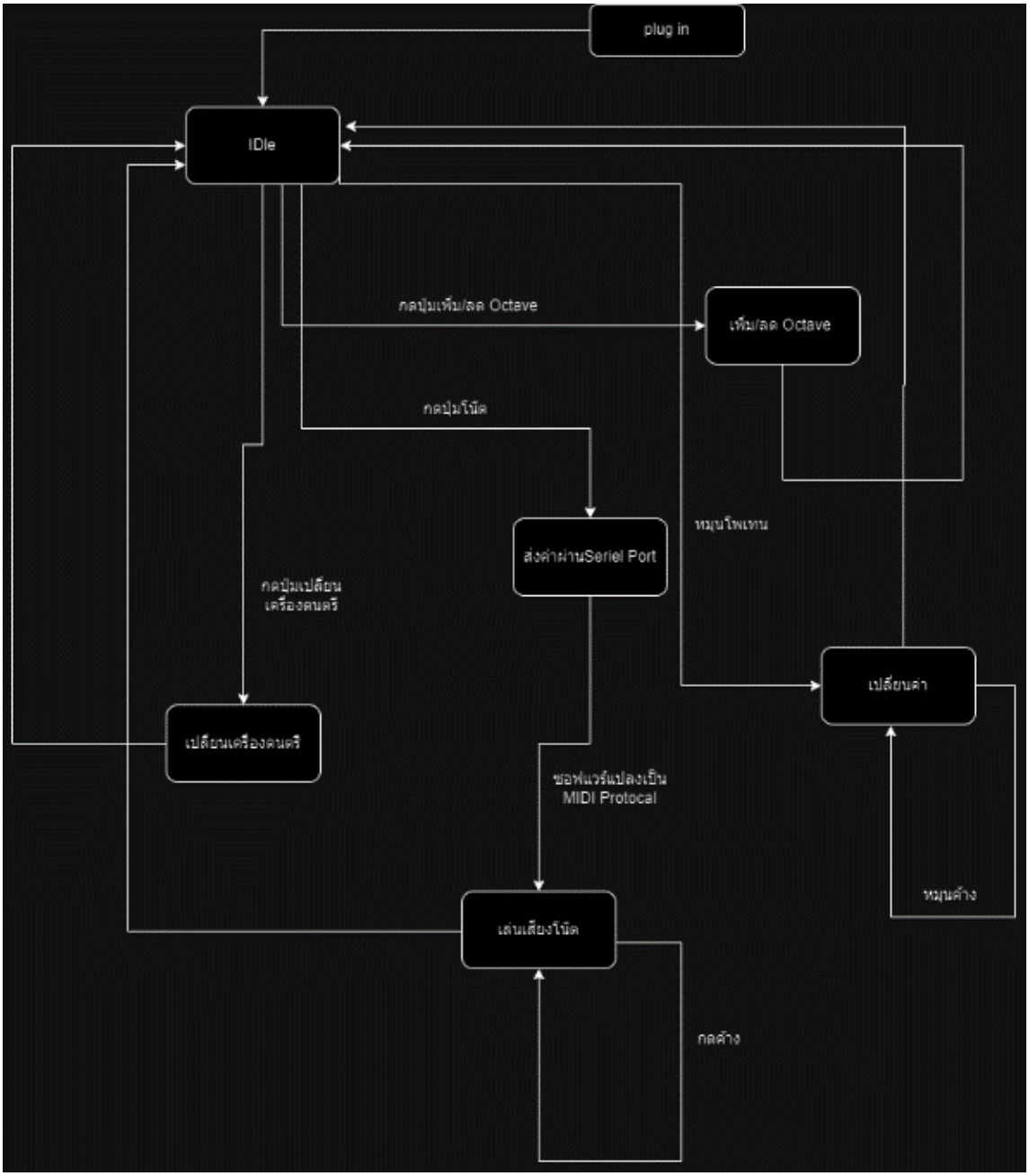
คอนโทรลเลอร์ MIDI สำหรับผู้ผลิตเพลงมีประโยชน์หลายประการ ประการแรก ช่วยให้ผู้ผลิตสามารถเล่นและบันทึกเพลงได้โดยไม่ต้องใช้เมาส์หรือคีย์บอร์ดของคอมพิวเตอร์ ประการที่สอง ช่วยให้ Producer สามารถควบคุมและปรับแต่งเสียงต่างๆ ได้อย่างสะดวก

State diagram

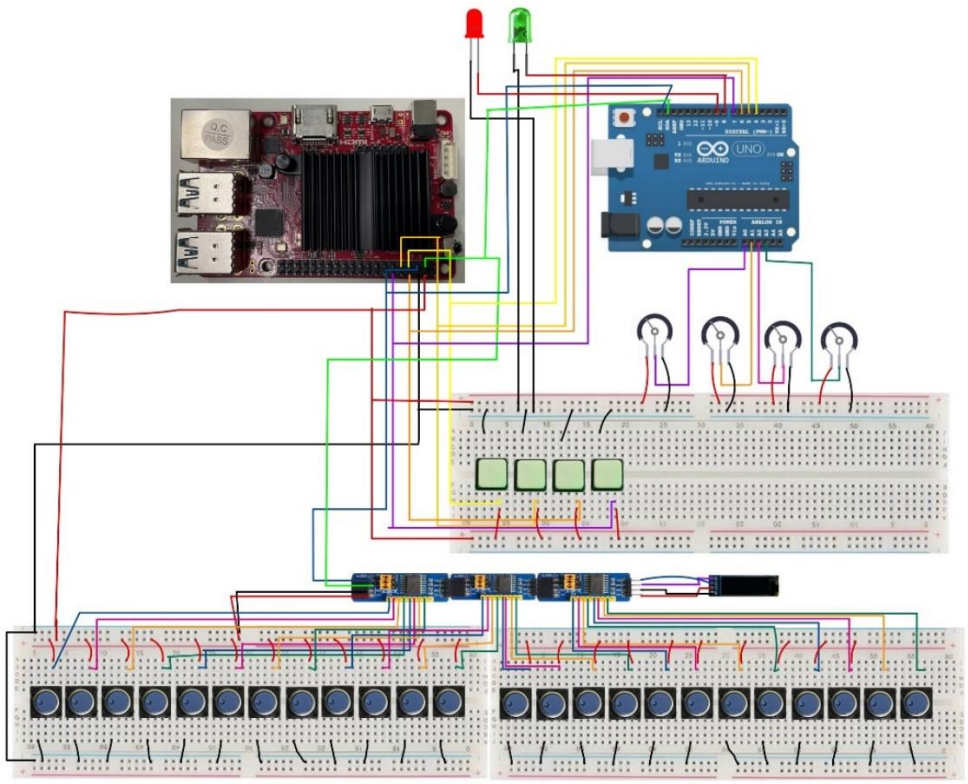
Odroid



Arduino


plug in
IDle
กดปุ่มเพิ่ม/ลด Octave
เพิ่ม/ลด Octave
กดปุ่มโน้ต
หมุนโพเทน
ส่งค่าผ่านSeriel Port
กดปุ่มเปลี่ยน
เครื่องดนตรี
เปลี่ยนค่า
เปลี่ยนเครื่องดนตรี
ซอฟแวร์แปลงเป็น
MIDI Protocal
หมุนค้าง
เล่นเสียงโน้ต
กดค้าง

Schematic Diagram



ขั้นตอนการพัฒนา

การเตรียมอุปกรณ์และวัสดุที่จำเป็น

ในขั้นตอนนี้เราต้องเตรียมอุปกรณ์และวัสดุที่จำเป็นสำหรับโปรเจคของเราดังนี้:

- Odroid: ใช้เป็นหน่วยประมวลผลหลักและสื่อสารกับ Arduino
- Arduino: ใช้สำหรับอ่านสถานะของปุ่มโน๊ตและปุ่มเพิ่ม/ลดอ็อกเทฟ
- PCF8574 (3 ตัว): ใช้สำหรับตรวจจับสถานะของปุ่มโน๊ตและส่งข้อมูลไปยัง Arduino
- จอ OLED: ใช้แสดงข้อมูลการควบคุมและสถานะของเครื่องดนตรี MIDI
- ปุ่มโน๊ต 24 ปุ่ม: ใช้สำหรับการกดแต่งคีย์โน๊ต MIDI
- ปุ่มเพิ่ม/ลด Octave 2 ปุ่ม: ใช้สำหรับการเพิ่มหรือลด Octave ของคีย์โน๊ต MIDI
- ปุ่มเปลี่ยนเครื่องดนตรี 2 ปุ่ม: ใช้สำหรับการเปลี่ยนเครื่องดนตรี MIDI

การเขียนโค้ด Arduino

ในส่วนนี้เราจะเขียนโค้ด Arduino สำหรับอ่านสถานะของปุ่มโน๊ต 24 ปุ่ม,ปุ่มเพิ่ม/ลดอ็อกเทฟและปุ่มเปลี่ยนเครื่องดนตรี

Code:

main.ino

```
#include <PCF8574.h>
#include <MIDI.h>

//pin
#define OCTAVE_UP 4 //
#define OCTAVE_DOWN 5 //
#define START_BTN 7
#define REC_BTN 6 //
#define STOP_BTN 7 //
#define PATTERN_UP 10
#define PATTERN_DOWN 11

//status
#define PLAY_STATUS 8
#define REC_STATUS 9

MIDI_CREATE_DEFAULT_INSTANCE();

boolean ButtonWasPressed1[] = { false, false, false, false, false, false, false, false };
boolean ButtonWasPressed2[] = { false, false, false, false, false, false, false, false };
boolean ButtonWasPressed3[] = { false, false, false, false, false, false, false, false };
```

```
boolean ButtonOctaveUpWasPressed = false;
boolean ButtonOctaveDownWasPressed = false;
boolean ButtonStartWasPressed = false;
boolean ButtonStopWasPressed = false;
boolean ButtonRecWasPressed = false;
boolean ButtonPatternUpWasPressed = false;
boolean ButtonPatternDownWasPressed = false;

int potPins[] = {A0, A1, A2, A3};
int lastPotValues[] = {0, 0, 0, 0};
int midiNotePoten[] = {4, 5, 6, 7};
const int threshold = 2;

int octave = 0;

PCF8574 PCF1(0x20);
PCF8574 PCF2(0x21);
PCF8574 PCF3(0x22);

void setupPinMode() {
  pinMode(OCTAVE_UP, INPUT);
  pinMode(OCTAVE_DOWN, INPUT);
  pinMode(START_BTN, INPUT);
  pinMode(STOP_BTN, INPUT);
  pinMode(REC_BTN, INPUT);
  pinMode(PLAY_STATUS, OUTPUT);
  pinMode(REC_STATUS, OUTPUT);
  pinMode(PATTERN_UP, INPUT);
  pinMode(PATTERN_DOWN, INPUT);

  for (int i = 0; i < 4; i++) {
    pinMode(potPins[i] , INPUT);
  }

}

void setup() {

  setupPinMode();
  MIDI.begin(MIDI_CHANNEL_OMNI);
  Serial.begin(115200);
  PCF1.begin();
  PCF2.begin();
  PCF3.begin();

}

void checkKey1(int key) {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = PCF1.readButton(key) == LOW;

  if (buttonIsPressed != ButtonWasPressed1[key] && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonWasPressed1[key] = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    int note = 24 + (12 * octave);

    if (ButtonWasPressed1[key]) {
      MIDI.sendNoteOn(key == 0 ? note : note + key, 127, 1);
    } else {
```

```
      MIDI.sendNoteOn(key == 0 ? note : note + key, 0, 1);
    }
  }
}

void checkKey2(int key) {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = PCF2.readButton(key) == LOW;

  if (buttonIsPressed != ButtonWasPressed2[key] && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonWasPressed2[key] = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    int note = 32 + (12 * octave);

    if (ButtonWasPressed2[key]) {
      MIDI.sendNoteOn(key == 0 ? note : note + key, 127, 1);
    } else {
      MIDI.sendNoteOn(key == 0 ? note : note + key, 0, 1);
    }
  }
}

void checkKey3(int key) {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = PCF3.readButton(key) == LOW;

  if (buttonIsPressed != ButtonWasPressed3[key] && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonWasPressed3[key] = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    int note = 40 + (12 * octave);

    if (ButtonWasPressed3[key]) {
      MIDI.sendNoteOn(key == 0 ? note : note + key, 127, 1);
    } else {
      MIDI.sendNoteOn(key == 0 ? note : note + key, 0, 1);
    }
  }
}

void checkButtonOctaveUp() {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = digitalRead(OCTAVE_UP) == LOW;

  if (buttonIsPressed != ButtonOctaveUpWasPressed && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonOctaveUpWasPressed = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    if (ButtonOctaveUpWasPressed) {
```

```
      if (octave < 8) {
        octave++;
        //lcdDisplayOctave(octave);
      }
    }
  }
}

void checkButtonOctaveDown() {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = digitalRead(OCTAVE_DOWN) == LOW;

  if (buttonIsPressed != ButtonOctaveDownWasPressed && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonOctaveDownWasPressed = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    if (ButtonOctaveDownWasPressed) {
      if (octave > 0) {
        octave--;
        //lcdDisplayOctave(octave);
      }
    }
  }
}

bool isPLaying = false;
bool isRecording = false;
byte pattern;
unsigned long currentTime;

void midiReadStatus() {

  if (MIDI.read()) {

    // A MIDI message was received
    byte channel = MIDI.getChannel();
    byte status = MIDI.getType();
    byte data1 = MIDI.getData1();
    byte data2 = MIDI.getData2();

    //play status
    if (data2 == 0x00) {
      digitalWrite(PLAY_STATUS, data1);
      isPLaying = data1;
    }

    //Rec status
    if (data2 == 0x01) {
      isRecording = data1;
    }

    //Stop status
    if (data2 == 0x02) {
      isPLaying = false;
    }

    //Pattern status
    if (data2 == 0x03) {
      pattern = data1;
    }
```

```
  }

  if (isPLaying && isRecording) {
    if (millis() - currentTime > 300) {
      if (millis() - currentTime > 600) {
        currentTime = millis();
      }
      digitalWrite(REC_STATUS, 1);
    } else {
      digitalWrite(REC_STATUS, 0);
    }
  } else {

    digitalWrite(REC_STATUS, isRecording);
  }
}

void checkButtonStartDown() {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = digitalRead(START_BTN) == LOW;

  if (buttonIsPressed != ButtonStartWasPressed && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonStartWasPressed = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    if (ButtonStartWasPressed) {
      MIDI.sendNoteOn(0, 127, 1);
    } else {
      MIDI.sendNoteOn(0, 0, 1);
    }
  }
}

void checkButtonRecDown() {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = digitalRead(REC_BTN) == LOW;

  if (buttonIsPressed != ButtonRecWasPressed && currentTime - ButtonStateChangeTime > DebounceTime)
{

    ButtonRecWasPressed = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    if (ButtonRecWasPressed) {
      MIDI.sendNoteOn(12, 127, 1);
    } else {
      MIDI.sendNoteOn(12, 0, 1);
    }
  }
}

void checkButtonStopDown() {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = digitalRead(STOP_BTN) == LOW;
```

```
  if (buttonIsPressed != ButtonStopWasPressed && currentTime - ButtonStateChangeTime > DebounceTime)
{

    ButtonStopWasPressed = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    if (ButtonStopWasPressed) {
      MIDI.sendNoteOn(1, 127, 1);
    } else {
      MIDI.sendNoteOn(1, 0, 1);
    }
  }
}

void checkButtonPatternUpDown() {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = digitalRead(PATTERN_UP) == LOW;

  if (buttonIsPressed != ButtonPatternUpWasPressed && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonPatternUpWasPressed = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    if (ButtonPatternUpWasPressed) {
      MIDI.sendNoteOn(13, 127, 1);
      //lcdDisplayPattern(pattern);
    } else {
      MIDI.sendNoteOn(13, 0, 1);
    }
  }
}

void checkButtonPatternDown() {

  unsigned long currentTime = millis();
  const unsigned long DebounceTime = 10;
  unsigned long ButtonStateChangeTime = 0;
  boolean buttonIsPressed = digitalRead(PATTERN_DOWN) == LOW;

  if (buttonIsPressed != ButtonPatternDownWasPressed && currentTime - ButtonStateChangeTime >
DebounceTime) {

    ButtonPatternDownWasPressed = buttonIsPressed;
    ButtonStateChangeTime = currentTime;

    if (ButtonPatternDownWasPressed) {
      MIDI.sendNoteOn(2, 127, 1);
      //lcdDisplayPattern(pattern);
    } else {
      MIDI.sendNoteOn(2, 0, 1);
    }
  }
}

void checkPoten() {

  for (int i = 0; i < 4; i++) {
    float potValue = ((1023 - analogRead(potPins[i])) / 1023.0) * 127;
    if (abs(potValue - lastPotValues[i]) >= threshold) {
      MIDI.sendNoteOn(midiNotePoten[i] , (int)potValue , 1);
```

```
      lastPotValues[i] = potValue;
    }
  }

}


void loop() {

  checkPoten();
  checkButtonOctaveUp();
  checkButtonOctaveDown();
  checkButtonStartDown();
  //checkButtonStopDown();
  checkButtonRecDown();
  //checkButtonPatternUpDown();
  //checkButtonPatternDown();
  midiReadStatus();

  for (int i = 0; i < 8; i++) {
    checkKey1(i);
    checkKey2(i);
    checkKey3(i);
  }
}
```

โค้ดนี้ใช้ Arduino ในการควบคุมคีย์ดนตรี MIDI และตัวควบคุมอื่น ๆ ในโปรแกรมดนตรี MIDI บนคอมพิวเตอร์ของเรา

- ไดเร็กทีฟส่วนสำคัญ:
    - โค้ดใช้ไลบรารี PCF8574 และ MIDI ที่นำมาใช้ในการควบคุมดนตรี MIDI และสื่อสาร MIDI ระหว่าง Arduino และคอมพิวเตอร์.
- การกำหนดขาและค่าคงที่:
    - กำหนดขาและค่าคงที่ของพอร์ตที่ใช้ใน Arduino ซึ่งรวมถึงขาที่ใช้สำหรับปุ่ม Octave Up, Octave Down, Start, Rec, Stop, Pattern Up, และ Pattern Down ที่เป็นตัวควบคุมในโครงการ.
    - กำหนดขาที่ใช้สำหรับสถานะเล่น (PLAY_STATUS) และสถานะอัดเสียง (REC_STATUS).

-การกำหนดตัวแปร:

- กำหนดตัวแปร boolean ที่เก็บสถานะการกดปุ่มสำหรับคีย์ดนตรี MIDI แต่ละตัวและปุ่ม Octave Up/Down, Start, Rec, Stop, Pattern Up/Down.

- กำหนดตัวแปร integer ที่ใช้สำหรับอ่านค่าจากขาแบบอนาล็อกสำหรับการควบคุมค่า MIDI จากปุ่มหมุนหรือหน้าจอแตะ (potentiometer).

- กำหนดสถานะเริ่มต้น:
    - ใน setup, ทำการกำหนดขาและสถานะเริ่มต้นสำหรับการสื่อสารผ่าน MIDI และ PCF8574, รวมถึงการกำหนดขาและสถานะเริ่มต้นสำหรับการควบคุมอุปกรณ์ MIDI.
- ตรวจสอบปุ่มและส่ง MIDI:
    - ใน loop, ทำการตรวจสอบปุ่มคีย์ดนตรี MIDI แต่ละตัวด้วยฟังก์ชัน checkKey1, checkKey2, และ checkKey3 และส่งข้อมูล MIDI ตามสถานะของปุ่มที่ถูกกด.
    - ตรวจสอบปุ่ม Octave Up และ Octave Down และปรับค่า Octave ในการส่ง MIDI.
    - ตรวจสอบปุ่ม Start, Rec, Stop, Pattern Up, และ Pattern Down และส่ง MIDI ตามการกดปุ่ม.
- การอ่านสถานะ MIDI:
    - ใช้ midiReadStatus เพื่ออ่านข้อมูล MIDI จากคอมพิวเตอร์และสื่อสารสถานะการเล่น, การอัดเสียง, และการหยุดเล่น.
- การอ่านค่าจากตัวควบคุมแบบอนาล็อก:
    - ใช้ checkPoten เพื่ออ่านค่าจากตัวควบคุมแบบอนาล็อกที่กำหนดในตัวแปร potPins และส่งค่า MIDI ถ้าค่าเปลี่ยนแปลง.

โค้ดนี้ทำหน้าที่ควบคุมคีย์ดนตรี MIDI และตัวควบคุมต่าง ๆ ในโปรแกรมดนตรี MIDI บนคอมพิวเตอร์ และสามารถทำงานร่วมกับคอมพิวเตอร์เพื่อส่งค่า MIDI ที่ถูกตรวจจับผ่าน Arduino ไปยังโปรแกรมดนตรี MIDI บนคอมพิวเตอร์ของเรา.

การเขียนโค้ด Python สำหรับ Odroid

โค้ดส่วนของ Odroid ในโปรเจค MIDI Controller ทำหน้าที่ควบคุมจอ OLED, ควบคุม MIDI Synth ผ่าน FluidSynth, อ่านข้อมูลจากปุ่มที่เชื่อมต่อ, และสามารถอ่านและควบคุมตัวยูนิต PCF8574 ได้ตามต้องการของโปรเจค MIDI Controller.

Code:

buttonControl.py

```
import odroid_wiringpi as wpi
import time

wpi.wiringPiSetup()

class DebouncedButton():
  def __init__(self, pin):
    self.pin = pin
    self.last_press = 0
    wpi.pinMode(pin, 0)

  def is_pressed(self):
    if not wpi.digitalRead(self.pin) and time.time() - self.last_press > 0.7:
      self.last_press = time.time()
      return True
    else:
      return False

class LED():
  def __init__(self, pin):
    self.pin = pin
    wpi.pinMode(self.pin,1)

  def on(self):
    wpi.digitalWrite(self.pin,1)

  def off(self):
    wpi.digitalWrite(self.pin,0)
```

โค้ดนี้ใช้งานไลบรารี odroid_wiringpi เพื่อควบคุมขาของ Odroid ที่เชื่อมต่อกับตัวยูนิต PCF8574 สามตัวที่มีที่อยู่ addr1, addr2, และ addr3 โดยโค้ดนี้มีคลาส DebouncedButton และคลาส LED ที่ใช้ในการควบคุมปุ่มและ LED บน Odroid ตามลำดับ

- wpi.wiringPiSetup(): ฟังก์ชันนี้ใช้ในการเริ่มต้นการใช้งานไลบรารี odroid_wiringpi โดยกำหนดการตั้งค่าเริ่มต้นสำหรับ GPIO pins ของ Odroid.
- class DebouncedButton(): คลาสนี้ถูกสร้างขึ้นเพื่อจัดการปุ่มดิบแบบ debounce ที่เชื่อมต่อกับ Odroid ผ่านตัวยูนิต PCF8574. คลาสมีเมทอดต่าง ๆ ดังนี้:

- __init__(self, pin): เมทอดนี้ใช้ในการกำหนดตัวแปร pin และเริ่มต้นตั้งค่าสถานะล่าสุดของการกดปุ่ม (last_press) เป็น 0.
- is_pressed(self): เมทอดนี้ใช้ในการตรวจสอบสถานะของปุ่มว่าถูกกดหรือไม่ โดยใช้ debounce และหากปุ่มถูกกดและผ่านการ debounce จะส่งค่า True กลับมา มิฉะนั้นจะส่งค่า False.

- class LED(): คลาสนี้ใช้ในการควบคุม LED ที่เชื่อมต่อกับ Odroid ผ่านตัวยูนิต PCF8574. คลาสมีเมทอดต่าง ๆ ดังนี้:
  - __init__(self, pin): เมทอดนี้ใช้ในการกำหนดตัวแปร pin และเปิดการใช้งาน LED โดยตั้งค่าให้เป็นโหมด OUTPUT.
  - on(self): เมทอดนี้ใช้ในการเปิด LED โดยใช้ wpi.digitalWrite() เพื่อส่งสัญญาณ HIGH ไปยังขา GPIO ที่เชื่อมต่อกับ LED.
  - off(self): เมทอดนี้ใช้ในการปิด LED โดยใช้ wpi.digitalWrite() เพื่อส่งสัญญาณ LOW ไปยังขา GPIO ที่เชื่อมต่อกับ LED.

โค้ดนี้ทำหน้าที่ควบคุมการอ่านข้อมูลจากตัวยูนิต PCF8574 และควบคุม LED บน Odroid ตามการกดปุ่มและสถานะของตัวยูนิต PCF8574.

I2C.py

```
from pcf8574 import PCF8574

addr1 = 0x20
addr2 = 0x21
addr3 = 0x22

bus1 = PCF8574(0 , addr1)
bus2 = PCF8574(0 , addr2)
bus3 = PCF8574(0 , addr3)

def readAllBus():

  data1 = list(bus1.port)
  data2 = list(bus2.port)
  data3 = list(bus3.port)

  return [data1[::-1], data2[::-1], data3[::-1]]

def closeAllBus():
  pass
```

โค้ดนี้ใช้งานไลบรารี pcf8574 เพื่อควบคุมตัวยูนิต PCF8574 ที่เชื่อมต่อกับ Odroid ผ่านการใช้งานแบบ I2C สามตัวที่มีที่อยู่ addr1, addr2, และ addr3 โดยโค้ดนี้มีฟังก์ชันสำหรับอ่านข้อมูลทั้งหมดจาก PCF8574 และฟังก์ชันสำหรับปิดการใช้งานทั้งหมด

- from pcf8574 import PCF8574: ไลบรารี pcf8574 ถูกนำเข้าเพื่อใช้ในการสร้างอ็อบเจกต์ PCF8574 และควบคุมการสื่อสารกับตัวยูนิต PCF8574 ผ่านการใช้งานแบบ I2C.
- addr1, addr2, addr3: ตัวแปรที่กำหนดที่อยู่ของตัวยูนิต PCF8574 สามตัวที่เชื่อมต่อกับ Odroid ผ่าน I2C.
- bus1, bus2, bus3: อ็อบเจกต์ PCF8574 สามตัวที่ถูกสร้างขึ้นโดยใช้ที่อยู่ addr1, addr2, และ addr3 ตามลำดับ.
- def readAllBus(): ฟังก์ชันนี้ใช้ในการอ่านข้อมูลทั้งหมดจากตัวยูนิต PCF8574 สามตัวและส่งข้อมูลนั้นกลับเป็นรายการของข้อมูลที่ถูกกลับด้านหลัง.
- def closeAllBus(): ฟังก์ชันนี้ไม่มีการทำงานเพิ่มเติมและเป็นฟังก์ชันที่ไม่ทำอะไรเลย อาจจะถูกเพิกเฉยเนื่องจากไม่ได้ใช้งานในโค้ด.

โค้ดนี้ทำหน้าที่อ่านข้อมูลจากตัวยูนิต PCF8574 สามตัวที่เชื่อมต่อกับ Odroid ผ่านการใช้งานแบบ I2C และสามารถนำข้อมูลที่อ่านได้ไปใช้งานในส่วนอื่น ๆ ของโปรแกรมตามต้องการ เช่น ใช้ข้อมูลสถานะของปุ่มหรืออุปกรณ์ที่ต่อกับตัวยูนิต PCF8574 เพื่อควบคุมการทำงานของโปรแกรม.

midi.py

```
import fluidsynth
import I2C
from buttonControl import *
from ooled import *
running = True

preset = 0
reverb = 0.1

fs = fluidsynth.Synth()
fs.start()
sfid = fs.sfload("Arachno.sf2")
fs.program_select(0, sfid, 0, preset)

octave = 3

fs.setting('synth.gain', 1.00)

def play_note(note):
    fs.noteon(0, note, 127)
```

```
def stop_note(note):
    fs.noteoff(0, note)

def stopAllNote():
    fs.delete()


once_key1 = [False, False, False, False, False, False, False, False]
once_key2 = [False, False, False, False, False, False, False, False]
once_key3 = [False, False, False, False, False, False, False, False]

#button
presetup_btn = DebouncedButton(0)
presetdown_btn = DebouncedButton(2)

octaveup_btn = DebouncedButton(1)
octavedown_btn = DebouncedButton(4)

#LED

status_on = LED(3)

try:
  FirstDis()
  while running:

      key = I2C.readAllBus()
      status_on.on()
      #button action
      #preset


      if presetup_btn.is_pressed():
        preset += 1
        if preset > 127:
          preset = 0
        fs.program_select(0, sfid, 0, preset)

        PREDIS(F"Preset : {fs.sfpreset_name(sfid, 0, preset)}")
        print(F"Preset : {fs.sfpreset_name(sfid, 0, preset)}")

      if presetdown_btn.is_pressed():
        preset -= 1
        if preset < 0:
          preset = 127
        fs.program_select(0, sfid, 0, preset)

        PREDIS(F"Preset : {fs.sfpreset_name(sfid, 0, preset)}")
        print(F"Preset : {fs.sfpreset_name(sfid, 0, preset)}")

      #octave
      if octaveup_btn.is_pressed():



        if octave < 8:
          octave += 1

          PREDIS(F"Octave : {octave}")
          print(F"Octave : {octave}")

      if octavedown_btn.is_pressed():
```

```
        if octave > 0:
          octave -= 1

          PREDIS(F"Octave : {octave}")
          print(F"Octave : {octave}")

      #bus 1
      for i in range(len(key[0])):
        note = 24 + (12 * octave)
        if not key[0][i]:
          if not once_key1[i]:
            play_note(note + i)
            once_key1[i] = True
        else:
          if once_key1[i]:
            stop_note(note + i)
            once_key1[i] = False

      #bus2
      for i in range(len(key[1])):
        note = 32 + (12 * octave)
        if not key[1][i]:
          if not once_key2[i]:
            play_note(note + i)
            once_key2[i] = True
        else:
          if once_key2[i]:
            stop_note(note + i)
            once_key2[i] = False

      #bus3
      for i in range(len(key[2])):
        note = 40 + (12 * octave)
        if not key[2][i]:
          if not once_key3[i]:
            play_note(note + i)
            once_key3[i] = True
        else:
          if once_key3[i]:
            stop_note(note + i)
            once_key3[i] = False
except KeyboardInterrupt:
  I2C.closeAllBus()
  status_on.off()
```

โค้ดนี้เป็นส่วนหลักของโปรแกรมที่ใช้ใน Arduino เพื่อควบคุมการเล่นเสียงดนตรี MIDI ด้วยโมดูล FluidSynth และควบคุมแสดงข้อมูลบนหน้าจอ OLED ที่เชื่อมต่อกับ Odroid ผ่าน I2C และการควบคุมปุ่มสำหรับเปลี่ยนเสียงดนตรี และปุ่มเพิ่ม/ลดโอคเทฟ (octave) ด้วยตัวแปรต่าง ๆ รวมถึงการใช้งาน LED สถานะ

- import fluidsynth: ไลบรารีนี้เอาไว้ใช้งานโมดูล FluidSynth ที่จะทำหน้าที่ในการเล่นเสียงดนตรี MIDI.
- import I2C: ไลบรารีนี้ไม่ได้รวมอยู่ในโค้ดนี้ แต่น่าจะเป็นโมดูลที่ใช้ในการควบคุมการอ่านข้อมูลจากตัวยูนิต PCF8574 ใน Odroid ผ่าน I2C.

- from buttonControl import *: ไลบรารีนี้อาจเป็นโมดูลที่ใช้ในการควบคุมการอ่านข้อมูลจากปุ่มที่เชื่อมต่อกับ Arduino ซึ่งไม่ได้รวมในโค้ดที่ให้มา ในกรณีนี้คงใช้เพื่อการอ่านปุ่ม.
- from ooled import *: ไลบรารีนี้อาจเป็นโมดูลที่ใช้ในการควบคุมหน้าจอ OLED ที่เชื่อมต่อกับ Odroid ซึ่งไม่ได้รวมในโค้ดที่ให้มา ในกรณีนี้คงใช้เพื่อควบคุมการแสดงข้อมูลบนหน้าจอ OLED.
- preset = 0: ตัวแปร preset ใช้เก็บค่าสำหรับเลือกเสียงดนตรี MIDI โดยเริ่มต้นที่ 0.
- reverb = 0.1: ตัวแปร reverb ใช้เก็บค่าสำหรับการตั้งค่าเอฟเฟกท์ Reverb ของเสียง.
- fs = fluidsynth.Synth(): สร้างอ็อบเจกต์ FluidSynth สำหรับควบคุมการเล่นเสียงดนตรี MIDI.
- fs.start(): เริ่มทำงานของ FluidSynth.
- sfid = fs.sfload("Arachno.sf2"): โหลดไฟล์ SoundFont (sf2) และระบุไฟล์ที่ใช้เสียงดนตรี MIDI ด้วยที่อยู่ของไฟล์ SoundFont.
- fs.program_select(0, sfid, 0, preset): กำหนดค่าสำหรับการเล่นเสียงดนตรี MIDI โดยระบุโหมด (mode), ที่อยู่ของไฟล์ SoundFont, และค่า preset โดยใช้ตัวแปรที่กำหนดไว้.
- octave = 3: ตัวแปร octave ใช้เก็บค่าโอคเทฟ (octave) สำหรับการปรับความสูงของเสียง.
- fs.setting('synth.gain', 1.00): ตั้งค่าระดับเสียงให้เป็น 1.00.
- def play_note(note): ฟังก์ชันนี้ใช้ในการเล่นเสียงโน๊ต MIDI โดยระบุโน๊ตที่ต้องการเล่น.
- def stop_note(note): ฟังก์ชันนี้ใช้ในการหยุดเสียงโน๊ต MIDI โดยระบุโน๊ตที่ต้องการหยุด.
- def stopAllNote(): ฟังก์ชันนี้ใช้ในการหยุดการเล่นเสียงทั้งหมด.
- once_key1, once_key2, once_key3: ตัวแปรที่ใช้สำหรับการตรวจสอบสถานะของปุ่มในการเล่นคีย์บอร์ดสามชุด.
- ปุ่ม: มีการกำหนดและกำหนดค่าให้กับปุ่มต่าง ๆ ที่ใช้ในการควบคุมการเล่นเสียงดนตรี MIDI และปรับค่าโอคเทฟ (octave).
- status_on: ตัวแปร LED สถานะ ที่ใช้ในการแสดงสถานะของโปรแกรม.
- โค้ดในบล็อก try, while running, และ except KeyboardInterrupt: ใช้ในการรันการทำงานหลักของโปรแกรม โดยตรวจสอบสถานะของปุ่มและควบคุมการเล่นเสียงดนตรี MIDI และควบคุมการแสดงข้อมูลบนหน้าจอ OLED รวมถึงการแสดงสถานะ LED สถานะ. โค้ดจะสั่งให้โปรแกรมหยุดเมื่อมีการกด Ctrl+C หรือเกิดการยกเลิก.

โค้ดนี้ทำหน้าที่ควบคุมการเล่นเสียงดนตรี MIDI และควบคุมการแสดงข้อมูลบนหน้าจอ OLED รวมถึงรองรับการควบคุมด้วยปุ่มและการแสดงสถานะผ่าน LED สถานะ.

oled.py

```
from luma.core.interface.serial import i2c
from luma.core.render import canvas
from luma.oled.device import ssd1306, ssd1325, ssd1331, sh1106
from time import sleep
from PIL import ImageFont, ImageDraw, Image
import time
import datetime

serial = i2c(port=0, address=0x3C)
# device = sh1106(serial, rotate=0)
device = ssd1306(serial, rotate=2)


def FirstDis():
    text = "Preset : Grand Piano"
    width = device.width
    height = device.height
    image = Image.new("1", (width, height))
    draw = ImageDraw.Draw(image)
    fontsize = 15
    font = ImageFont.truetype("/home/odroid/arial_bold.ttf", fontsize)  # Replace with your font
path and size
    max_text_width = 128


    display_width = device.width
    display_height = device.height

    lines = []
    line = ""
    for word in text.split():
        if font.getsize(line + word)[0] <= display_width:
            line += word + " "
        else:
            lines.append(line)
            line = word + " "
    lines.append(line)

    x = 0
    y = 0


    for line in lines:
        draw.text((x, y), line, font=font, fill=255)
        y += font.getsize(line)[1]


    device.display(image)

def PREDIS(t):

    device.clear()
    text = t
    width = device.width
    height = device.height
```

```
    image = Image.new("1", (width, height))
    draw = ImageDraw.Draw(image)
    fontsize = 15
    font = ImageFont.truetype("/home/odroid/arial_bold.ttf", fontsize)  # Replace with your font
path and size
    max_text_width = 128


    display_width = device.width
    display_height = device.height

    lines = []
    line = ""
    for word in text.split():
        if font.getsize(line + word)[0] <= display_width:
            line += word + " "
        else:
            lines.append(line)
            line = word + " "
    lines.append(line)

    x = 0
    y = 0


    for line in lines:
        draw.text((x, y), line, font=font, fill=255)
        y += font.getsize(line)[1]


    device.display(image)

def OCDIS(t):

    device.clear()
    text = t
    width = device.width
    height = device.height
    image = Image.new("1", (width, height))
    draw = ImageDraw.Draw(image)
    fontsize = 15
    font = ImageFont.truetype("/home/odroid/arial_bold.ttf", fontsize)  # Replace with your font
path and size
    max_text_width = 128


    display_width = device.width
    display_height = device.height

    lines = []
    line = ""
    for word in text.split():
        if font.getsize(line + word)[0] <= display_width:
            line += word + " "
        else:
            lines.append(line)
            line = word + " "
    lines.append(line)

    x = 0
    y = 0


    for line in lines:
        draw.text((x, y), line, font=font, fill=255)
```

```
        y += font.getsize(line)[1]

    device.display(image)
```

โค้ดแรกเป็นส่วนหนึ่งของโปรแกรมที่ใช้ใน Odroid เพื่อควบคุมการแสดงข้อมูลบนหน้าจอ OLED ด้วยการเรียกใช้ไลบรารี Luma และ Pillow รวมถึงการกำหนดข้อความและแสดงข้อมูลที่ต้องการบนหน้าจอ OLED ในรูปแบบของตัวอักษรและตัวเลข รวมถึงแสดงผลหลายบรรทัด

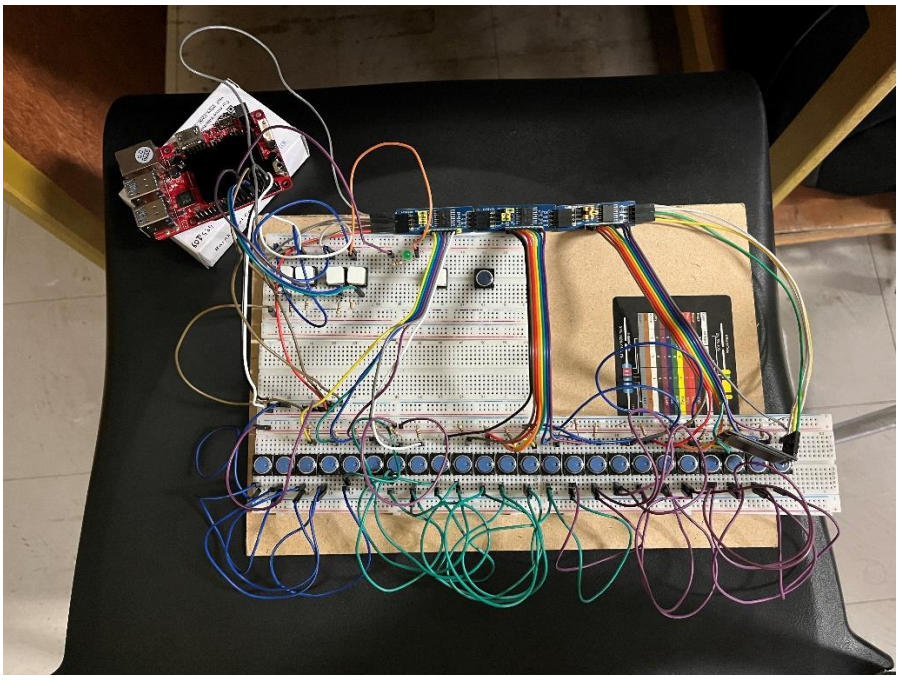
- from luma.core.interface.serial import i2c: ไลบรารีเพื่อใช้งานอินเตอร์เฟซ I2C สำหรับการเชื่อมต่อกับหน้าจอ OLED.
- from luma.core.render import canvas: ไลบรารีสำหรับการเรียกใช้เพื่อเขียนข้อมูลลงบนหน้าจอ OLED.
- from luma.oled.device import ssd1306, ssd1325, ssd1331, sh1106: ไลบรารีเพื่อรองรับหลายรุ่นของหน้าจอ OLED เช่น SSD1306, SSD1325, SSD1331, และ SH1106.
- from PIL import ImageFont, ImageDraw, Image: ไลบรารีสำหรับการใช้งานภาพและการวาดตัวอักษร/รูปภาพบนหน้าจอ OLED.
- serial = i2c(port=0, address=0x3C): สร้างอินเตอร์เฟซ I2C ที่กำหนดพอร์ต 0 และที่อยู่ I2C 0x3C สำหรับการเชื่อมต่อกับหน้าจอ OLED.
- device = ssd1306(serial, rotate=2): สร้างอ็อบเจกต์ของหน้าจอ OLED และกำหนดว่าใช้หน้าจอรุ่น SSD1306 และมีการหมุนหน้าจอทีซ้าย 90 องศา (rotate=2).
- def FirstDis(): ฟังก์ชันนี้ใช้ในการแสดงข้อความแรกบนหน้าจอ OLED และกำหนดตัวอักษรและขนาดของข้อความแรก.
- def PREDIS(t): ฟังก์ชันนี้ใช้ในการแสดงข้อความบนหน้าจอ OLED โดยรับข้อความที่ต้องการแสดงและปรับแสดงข้อความตามข้อความที่รับเข้ามา.
- def OCDIS(t): ฟังก์ชันนี้ใช้ในการแสดงข้อความบนหน้าจอ OLED โดยรับข้อความที่ต้องการแสดงและปรับแสดงข้อความตามข้อความที่รับเข้ามา.
- ฟังก์ชัน def displayLOGO(): ใช้ในการลบข้อมูลบนหน้าจอ OLED และแสดงข้อความ "ESPUNIA" บนหน้าจอ.

- def setup_lcd(): ฟังก์ชันนี้ใช้ในการกำหนดและเริ่มต้นการใช้งานหน้าจอ OLED รวมถึงแสดงข้อความ "OLED Start Work !!!" บนหน้าจอ.
- def lcdDisplayOctave(int octave): ฟังก์ชันนี้ใช้ในการแสดงข้อมูลเกี่ยวกับค่าโอคเทฟบนหน้าจอ OLED โดยรับค่าตัวเลข octave และแสดงข้อความ "OC" ตามด้วยค่า octave บนหน้าจอ.
- def lcdDisplayPattern(int pattern): ฟังก์ชันนี้ใช้ในการแสดงข้อมูลเกี่ยวกับรูปแบบการเล่นบนหน้าจอ OLED โดยรับค่าตัวเลข pattern และแสดงข้อความ "P" ตามด้วยค่า pattern บนหน้าจอ.
- โค้ดในบล็อก def FirstDis(), def PREDIS(t), และ def OCDIS(t): จะดำเนินการกำหนดข้อความที่ต้องการแสดงบนหน้าจอ OLED ในรูปแบบที่เหมาะสมและแสดงข้อมูลนั้นบนหน้าจอ OLED.

โค้ดนี้ใช้ในการควบคุมแสดงข้อมูลบนหน้าจอ OLED และแสดงข้อมูลเกี่ยวกับค่าโอคเทฟและรูปแบบการเล่นบนหน้าจอ OLED โดยรับข้อมูลและสร้างข้อความที่ต้องการแสดงบนหน้าจอ.

การรวมโค้ดและทดสอบระบบ

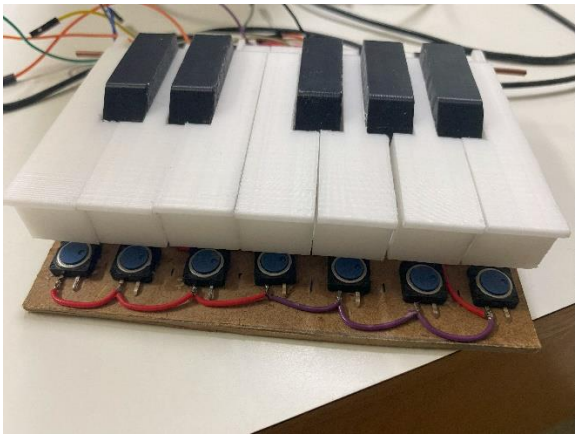
หลังจากที่เราเขียนโค้ด Arduino และ Python สำหรับ Odroid เสร็จสิ้นแล้ว เราจะต้องรวมโค้ดและทดสอบระบบโดยการกดปุ่ม MIDI และตรวจสอบว่าข้อมูลถูกส่งมาแสดงบนจอ OLED ได้อย่างถูกต้อง.

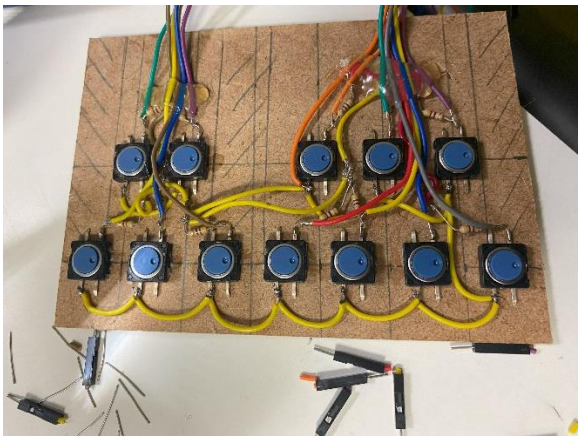


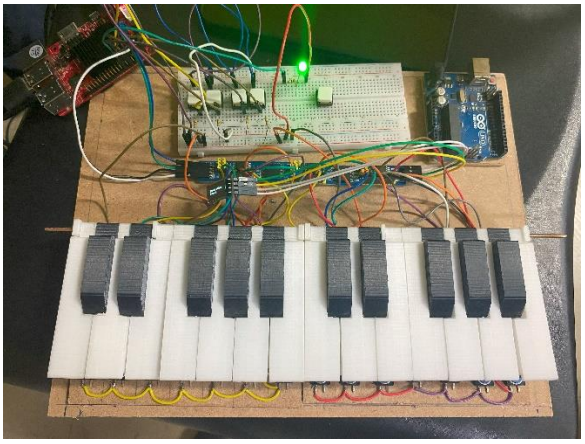
การทดสอบและปรับปรุง

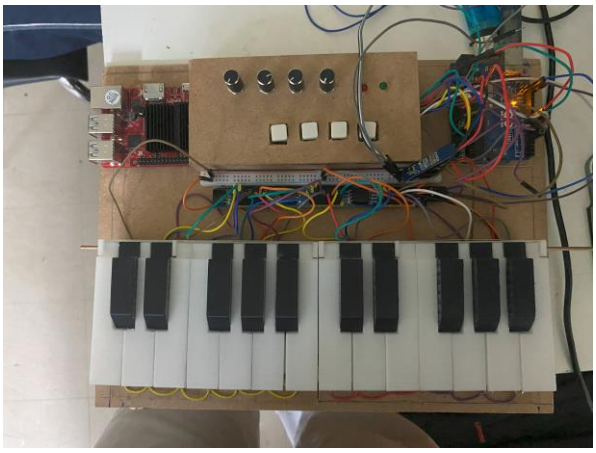
หลังจากที่เรารวมโค้ดและทดสอบระบบเสร็จสิ้นแล้ว เราควรทดสอบโปรเจคในสถานการณ์จริง โดยการใช้งาน MIDI Controller ในการควบคุมเครื่องดนตรี MIDI และตรวจสอบความถูกต้องของการแสดงข้อมูลบนจอ OLED และการทำงานของปุ่มต่าง ๆ

ปรับแต่งให้สวยและใช้สะดวกขึ้น









สรุปผล

โครงการ MIDI Controller ด้วย Odroid, Arduino, และ PCF8574 เป็นโครงการที่ทำให้เราสามารถควบคุมเครื่องดนตรี MIDI ได้อย่างอิสระและสร้างเสียงที่ต้องการได้อย่างอิสระ โดยใช้ปุ่มโน๊ต MIDI 24 ปุ่ม และปุ่มเพิ่ม/ลด Octave 2 ปุ่ม และ ปุ่มเปลี่ยนเครื่องดนตรี 2 ปุ่ม เราสามารถปรับแต่งคีย์โน๊ต MIDI และควบคุม Octave ได้อย่างราบรื่น และสร้างเสียงที่ต้องการได้อย่างอิสระ

ในส่วนของ Arduino, มันถูกเชื่อมต่อกับคอมพิวเตอร์เพื่อใช้กับซอฟต์แวร์ FL Studio ที่อนุญาตให้เราอัดเสียงและเล่นเสียงได้ นี่คือส่วนของโครงการที่ช่วยให้เราสร้างเสียงดนตรีที่เราต้องการได้อย่างอิสระผ่านส่วน Arduino

ส่วนของ Odroid ทำให้เราสามารถเล่นเสียงได้อย่างรวดเร็วและอิสระ และสามารถพกพาและเล่นเสียงที่เราต้องการที่ไหนก็ได้เพียงแค่มีการเชื่อมต่อปลั๊กเพื่อให้ไฟแก่ Odroid โครงการนี้ช่วยให้นักดนตรีและผู้สนใจดนตรีสามารถสร้างเสียงดนตรีดิจิตอลและควบคุมการเล่นดนตรีได้อย่างครบถ้วนและสร้างผลงานดนตรีตามความต้องการของตนเองได้อย่างสะดวกและอิสระ.