UNIVERSITY OF
WEST LONDON

TC50321E EMBEDDED SYSTEMS

Assessment A2 – Prototype MIDI Controller

BSc (Hons) Sound Engineering

21440503

Abstract

This report details the process carried out trying to design a MIDI drum pad using piezoelectric sensors. The aim is for the user to be able to trigger three different drum sounds using a corresponding piezo sensor. The piezoelectric sensor interfaces with an Arduino Micro Pro board, which processes the input from the sensor at the digital pins. Initially analog pins were used, but due to the nature of the analog signal that the piezo sensor outputs, difficulty was encountered when designing code that was robust enough to know when and when not to send MIDI messages. Using the Micro Pro's digital pins allowed for the aim to be met, as the code required to process the HIGH/LOW states of a digital pin was easier to synthesise.

Contents

Introduction

This report details how a prototype MIDI controller will be created using an Arduino Micro Pro microprocessor board, C++ code and piezoelectric sensors. Firstly, the prototype will be designed with the schematic creator TinkerCAD. This has a user-friendly interface that allows you to build your circuit in an intuitive way. A classic schematic is also created alongside. There is an IDE built in to TinkerCAD that allows you to type and load C++ code onto the prototype too. Then, the prototype will be constructed in a lab environment so it can be evaluated and seen in working operation.

There are four main chapters: Background Information; Prototype Design; C++ Code; Testing/Evaluation; and Conclusion.

Background Information

An embedded system is a combination of hardware and software that takes an input from its environment, usually through sensors and/or human input, conducts operations with the data that it receives, and then provides an output through display, motion of a motor or actuator or by sending a signal. The core of an embedded system is a microprocessor chip. This is an integrated circuit that has the facility to have code loaded and stored on it. The microcontroller board used for this assignment is the Arduino Micro Pro. This board has 20 digital I/O pins. Out of these 20 pins, 7 pins are PWM (Pulse Width Modulation) pins and, 12 pins are analog input pins. The Micro Pro has five external interrupts, which allows the instant trigger of a function when a pin goes either high or low (or both). If you attach an interrupt to an enabled pin, it triggers a specific interrupt: pin 3 maps to interrupt 0 (INT0), pin 2 is interrupt 1 (INT1), pin 0 is interrupt 2 (INT2), pin 1 is interrupt 3 (INT3), and pin 7 is interrupt 4 (INT4). Apart from this, it has an ATMega32U4 chip, a reset button, 16MHz crystal oscillator, and a micro-USB port (Wilson, 2020).

Arduino uses a version of C++ as its native coding language. It has all the language features of C++, but not all the libraries. This is because of the memory limitations of Arduino boards. Another difference between regular C++ and the Arduino version is the layout in the IDE: the main() function is replaced with the repeating loop() function, and it is preceded by a one-time setup() function.

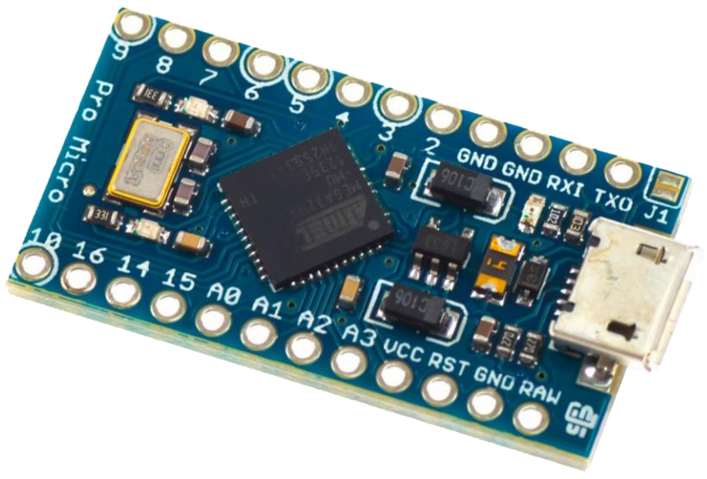

Figure 1 - Arduino Micro Pro

Prototype Design

For this assignment, a simple drum machine will be constructed, using piezoelectric sensors which will trigger drum sounds.

A piezoelectric sensor uses a quartz crystal disc that, when force is applied, has a charge across it in a certain direction. This compression allows for a potential difference to be measured across it (Regtien and Dertien, 2018). These values can be used to send MIDI messages over USB in real time.

These sensors will be connected onto a breadboard, which will allow them to interface with the Arduino Micro Pro, utilising the MIDIUSB library (Arduino, 2022). Unfortunately, TinkerCAD only supports the use of the Arduino Uno board, so it will be used for schematic purposes, but the physical prototype will be built with the Micro Pro. The schematic can be seen below.

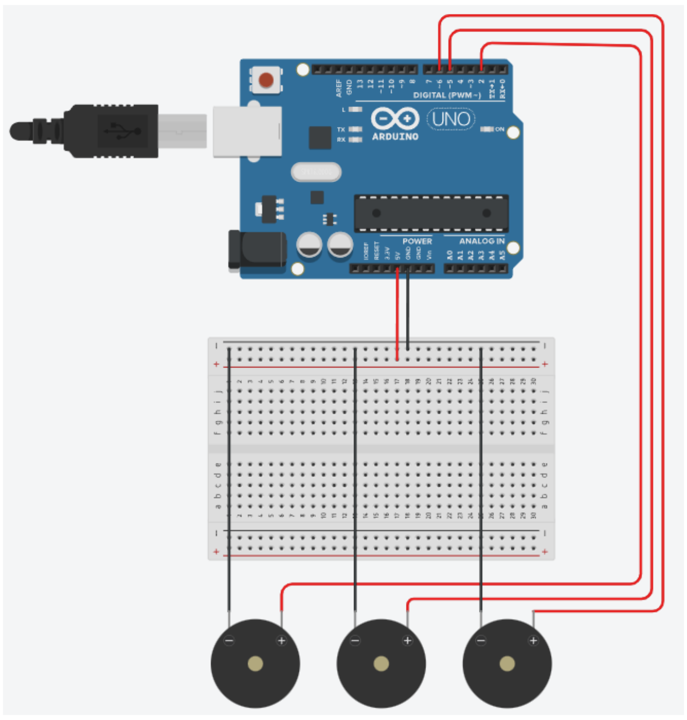

Figure 2 - MIDI drum pad schematic

C++ Code

```
#include "MIDIUSB.h" // MIDIUSB library

int Speed = 100; // millisecond buffer

const int Pin1 = 2;
const int Pin2 = 5;
const int Pin3 = 6; // piezo pins are defined

unsigned long Millis1 = 0;
unsigned long Millis2 = 0;
unsigned long Millis3 = 0; // variables that store when an event happens

int Active1 = 1;
int Active2 = 1;
int Active3 = 1; // variables to show whether a piezo sensor is ‘active’ or not

int Read1;
int Read2;
int Read3; // variables to read piezo sensors

unsigned long currentMillis = 0; // variable that stores current millis() value

void setup() {


}

void loop() {

    Read1 = digitalRead(Pin1); // reads piezo
    if (Read1 == LOW && Active1 == 1) { // if piezo is triggered, LOW is read

        Millis1 = millis(); // time of event recorded

        Serial.println("Sending note on"); // note on message
        noteOn(0, 48, 127);   // Channel 0, middle C, max velocity
        MidiUSB.flush();

        Serial.print("Pin1: ");
        Serial.println(Read1);

        Serial.println("Sending note off"); // note off message
        noteOff(0, 48, 127);  // Channel 0, middle C, max velocity
        MidiUSB.flush();

        Active1 = 0; // piezo temporarily deactivated
        Read1 = HIGH; // read variable is reset
    }

    currentMillis = millis(); // timestamp
    if (currentMillis > Millis1 + Speed) {
        Active1 = 1;
    } // buffer waits until time within variable “Speed” has passed



    Read2 = digitalRead(Pin2); // REPEAT FOR PIN 2
    if (Read2 == LOW && Active2 == 1) {
        Millis2 = millis();

        Serial.println("Sending note on");
        noteOn(0, 50, 127);   // Channel 0, middle D, max velocity
        MidiUSB.flush();

        Serial.print("Pin2: ");
        Serial.println(Read2);

        Serial.println("Sending note off");
        noteOff(0, 50, 127);  // Channel 0, middle D, max velocity
        MidiUSB.flush();

        Active2 = 0;
        Read2 = HIGH;
    }

    currentMillis = millis();
    if (currentMillis > Millis2 + Speed) {
        Active2 = 1;
    }



    Read3 = digitalRead(Pin3); // REPEAT FOR PIN 3
    if (Read3 == LOW && Active3 == 1) {

        Millis3 = millis();

        Serial.println("Sending note on");
        noteOn(0, 52, 127);   // Channel 0, middle E, max velocity
        MidiUSB.flush();

        Serial.print("Pin3: ");
        Serial.println(Read3);

        Serial.println("Sending note off");
        noteOff(0, 52, 127);  // Channel 0, middle E, max velocity
        MidiUSB.flush();

        Active3 = 0;
        Read3 = HIGH;
    }

    currentMillis = millis();
    if (currentMillis > Millis3 + Speed) {
        Active3 = 1;
    }

}

void noteOn(byte channel, byte pitch, byte velocity) {
    midiEventPacket_t noteOn = { 0x09, 0x90 | channel, pitch, velocity };
    MidiUSB.sendMIDI(noteOn);
} // MIDIUSB Note On Function

void noteOff(byte channel, byte pitch, byte velocity) {
    midiEventPacket_t noteOff = { 0x08, 0x80 | channel, pitch, velocity };
    MidiUSB.sendMIDI(noteOff);
} // MIDIUSB Note Off Function
```

Testing/Evaluation

When the design was complete, building (including testing) commenced. Many issues were encountered during testing that involved lots of code alterations. Seen below is an iteration of the code before the final version was created.

```
#include "MIDIUSB.h"

int THRESHOLD = 10;

int val_1 = 0;
int valB_1 = 0;
int val_2 = 0;
int valB_2 = 0;

void setup() {

    Serial.begin(115200);
}



void loop() {

    val_1 = analogRead(A1);  // function to read analog voltage from sensor 1
    //if(valB_1 > (val_1) && val_1 > THRESHOLD){
      //sensOne();
    //}
    Serial.print("A1: ");
    Serial.println((val_1 * (-1)) + 1023);

    valB_1 = val_1;


    //val_2 = analogRead(A2);  // function to read analog voltage from sensor 1
    //if(valB_2 > (val_2) && val_2 > THRESHOLD){
      //sensTwo();
    //}
    //Serial.print("A2: ");
    //Serial.println(val_2);

    //valB_2 = val_2;
}



void sensOne() {

    Serial.println("Sending note on");
    noteOn(0, 48, (val_1 / 8));   // Channel 0, middle C, normal velocity
    MidiUSB.flush();
    delay(50);

    Serial.print("A1_Pin: ");
    Serial.println(val_1 / 8);

    Serial.println("Sending note off");
    noteOff(0, 48, (val_1 / 8));  // Channel 0, middle C, normal velocity
    MidiUSB.flush();
    delay(50);

    while (val_1 > THRESHOLD) {
        val_1 = analogRead(A1);

    }
}

void sensTwo() {

    Serial.println("Sending note on");
    noteOn(0, 48, (val_2 / 8));   // Channel 0, middle C, normal velocity
    MidiUSB.flush();
    delay(50);

    Serial.print("A2_Pin: ");
    Serial.println(val_2 / 8);

    Serial.println("Sending note off");
    noteOff(0, 48, (val_2 / 8));  // Channel 0, middle C, normal velocity
    MidiUSB.flush();
    delay(50);

    while (val_2 > THRESHOLD) {
        val_2 = analogRead(A2);

    }
}

void noteOn(byte channel, byte pitch, byte velocity) {
    midiEventPacket_t noteOn = { 0x09, 0x90 | channel, pitch, velocity };
    MidiUSB.sendMIDI(noteOn);
}

void noteOff(byte channel, byte pitch, byte velocity) {
    midiEventPacket_t noteOff = { 0x08, 0x80 | channel, pitch, velocity };
    MidiUSB.sendMIDI(noteOff);
}
```

This iteration of the code did not give out a reliable reading. It used the analog pins on the Micro Pro, which in the end proved too problematic to carry on with. This was because the analog reading gave out a value between 0-1023 which couldn’t be relied upon. Difficulty was encountered when trying to turn this continuous reading into an ‘impulse’ that could trigger a single MIDI message. Often, the code would trigger multiple MIDI messages, due to the analog nature of the input signal. Digital pins were eventually used, and a working design was created.

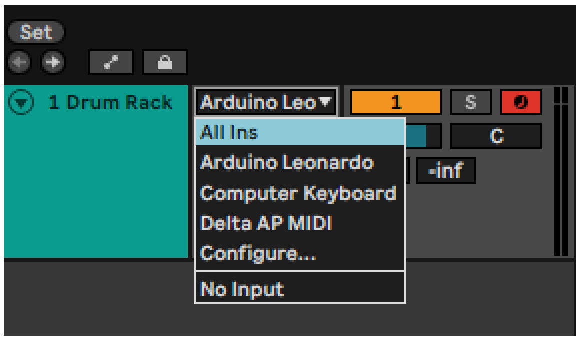

Figure 3 - Ableton recognising Arduino as a MIDI device

Shown above is Ableton recognising the Micro Pro as a MIDI device, although it recognises it as a Leonardo board. This still operates as expected.

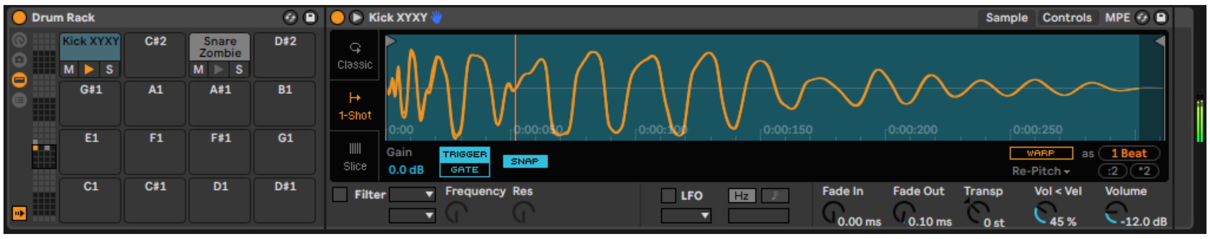

Figure 4 - Arduino triggers a sample

The piezo sensor is struck, and the kick sample which is appropriately mapped is triggered.

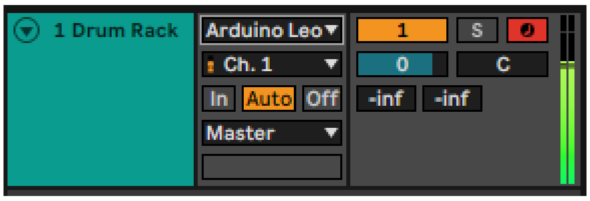

Figure 5 - MIDI message in Channel 1

A MIDI message can be seen on MIDI Ch. 1, and the amplitude of the kick sample registers on the meter of '1 Drum Rack'.

Conclusion

Overall, the assignment was a success, as a solution was found to the initial aim; finding a way to trigger MIDI messages with piezo sensors. Initially, it seemed intuitive to use the Micro Pro's analog pins to read the analog signals of the piezo sensors, but after much trial and error, the digital pins were used to simplify the process. This was mainly due to the variability of the analog signals. This made it difficult to design code that was robust enough to handle these signals. The simple HIGH/LOW state of the digital pins made it easier to design code that could reliably send a MIDI message when the user tapped a piezo sensor. This change of direction was vital to completing the aim and creating a successful design.

Bibliography

Arduino (2022) *MIDIUSB Library.* Available at: https://www.arduino.cc/reference/en/libraries/midiusb/ (Accessed: 05/06/2021).

Regtien, P. and Dertien, E. (2018) 'Piezoelectric sensors', .

Wilson, J. (2020) *Introduction to Arduino Pro Micro.* Available at: https://www.theengineeringprojects.com/2020/12/introduction-to-arduino-pro-micro.html (Accessed: .

Appendix 1 – User Operation Video

MIDI Controller
Operation.mov