EEL4746
Micro I

Project Title Piano Teacher: Digital Piano with Smart Features

Team Members

Name: Kevin Perez PID: 3612072

Name: Andy Alvarez PID: 6140523

Name: Antonio Garcia PID: 6188684

Name: Edixon Rosales PID: 6283389

Due date on: 12/3/2022

Florida International University

Project Objectives:

- Create a digital piano that outputs digitally generated notes
- Create software that can help the user improve piano accuracy/skills

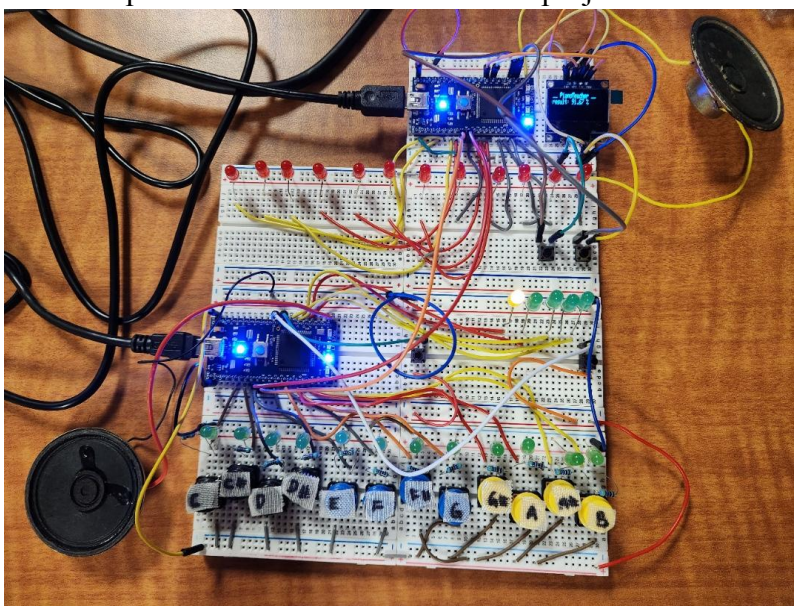
Components used

In a professional report these are needed to offer the capacity for another team to replicate the experiment conditions and verify the findings. To do so, a list of the components and the exact equipment used must be provided.

- 2 Mbed Boards
- 2 Speakers
- OLED
- 29 LEDs
- 16 Push Buttons
- Jumper Wires
- 7 Breadboards
- 12 Resistors

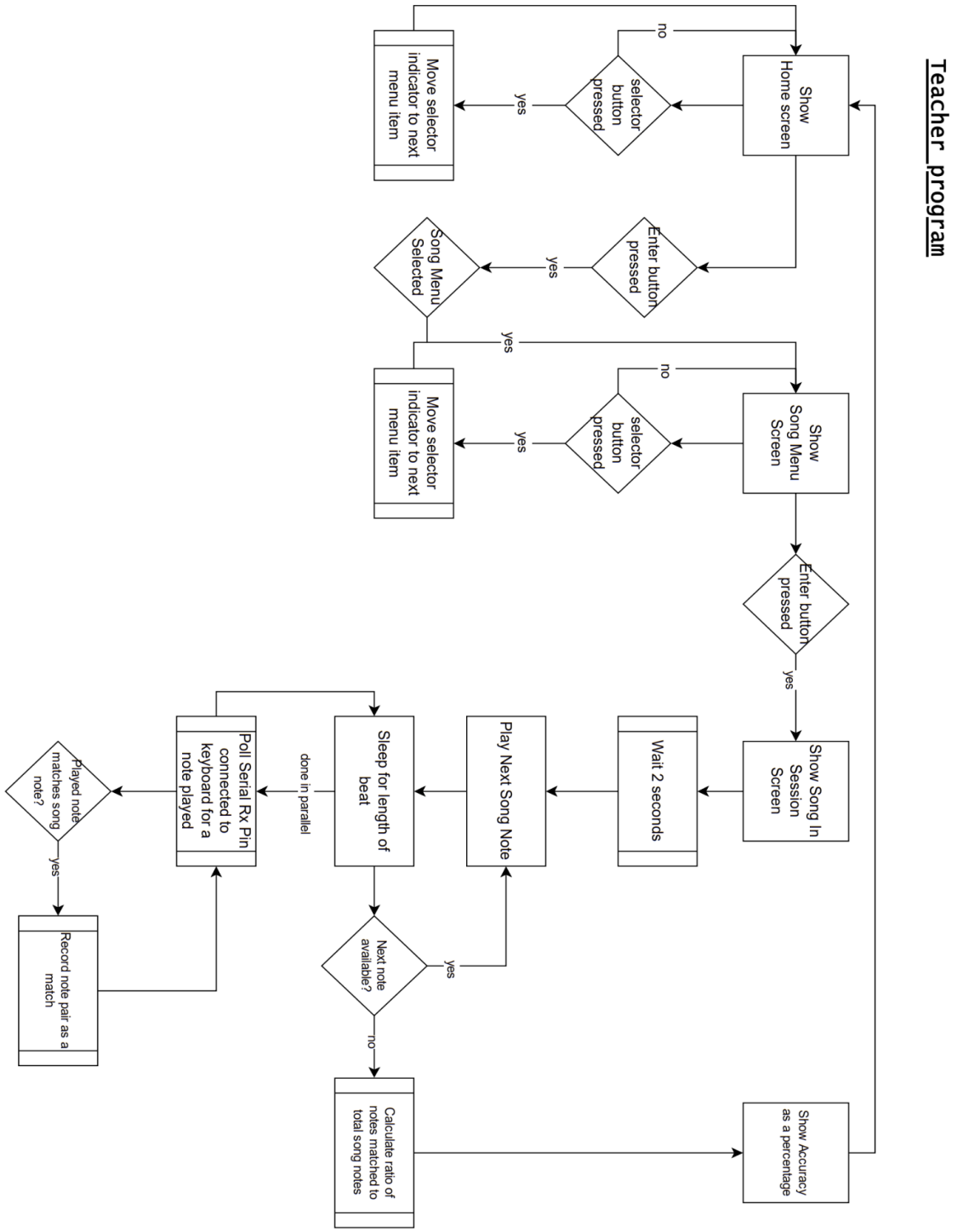
Picture or Schematic of the System

Include a picture and the schematic of the project.

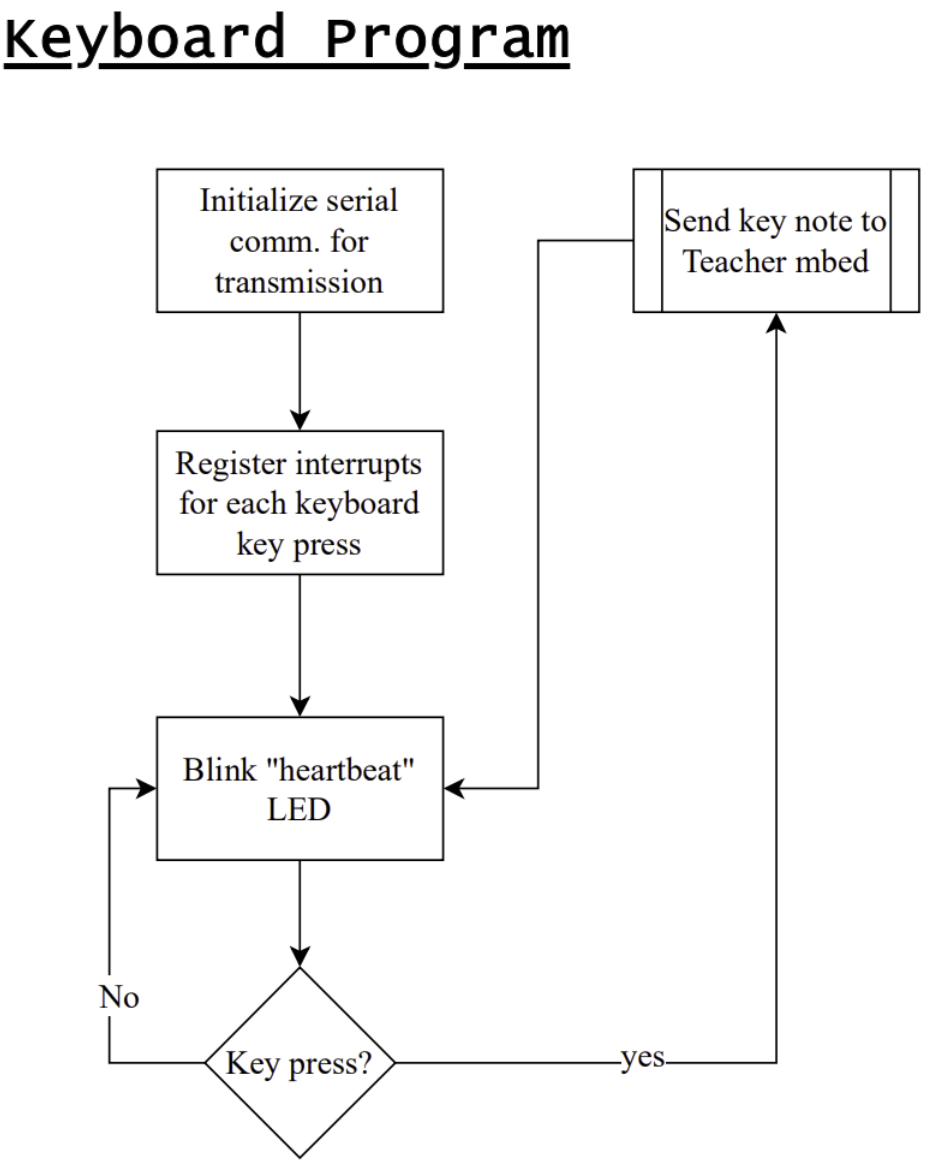


Software

“Teacher” Program Flowchart



“Keyboard” Program Flowchart



Code for the “Teacher” Mbed

```
#include "mbed.h"
char currentNote;

DigitalOut myled(LED2);
DigitalOut led1(LED1);

PwmOut buzzer(p21);
InterruptIn C(p5);//P0.9 = FIO0PIN = 0x200
InterruptIn Csharp(p6);//P0.8 = FIO0PIN =0x100
InterruptIn D(p7);//P0.7 = FIO0PIN = 0x80
InterruptIn Dsharp(p8);//P0.6 = FIO0PIN = 0x40
InterruptIn E(p11);//P0.18 = FIO0PIN = 0x40000
InterruptIn F(p12);//P0.17 = FIO1PIN = 0x20000
InterruptIn Fsharp(p13);//P0.15 = FIO1PIN = 0x8000
InterruptIn G(p14);//P0.16 = FIO2PIN = 0x10000
InterruptIn Gsharp(p15);//P0.23 = FIO2PIN = 0x800000
InterruptIn A(p16);//P0.24 = FIO2PIN = 0x1000000
InterruptIn Asharp(p17);//P0.25 = FIO2PIN = 0x2000000
InterruptIn B(p18);//P0.26 = FIO2PIN = 0x4000000
InterruptIn Increment(p30);
InterruptIn Decrement(p29);
DigitalOut lED4(p24);
DigitalOut lED5(p25);
DigitalOut lED6(p26);
DigitalOut lED7(p27);
DigitalOut lED8(p28);

//globals
int counter = 0;
float octave = 1;
void delay(void);
Serial masterDevice(p9, p10);

enum notes { cnote = 0x00, csharpnote, dnote, dsharpnote, enote, fnote, fsharpnote,
             gnote, gsharpnote, anote, asharpnote, bnote };

void sendNote()
{
    masterDevice.printf("%c", currentNote);
}

void Cnote(void){
    currentNote = cnote;
    masterDevice.printf("%c", currentNote);

    float fC = octave*261.63;
    buzzer.period(1/(2*fC)); // set PWM period
    buzzer=0.5; // set duty cycle
    wait(0.4);
    buzzer=0;
}

void Csharpnote(void){
    currentNote = csharpnote;
    masterDevice.printf("%c", currentNote);
```

```
56
57      float fCsharp = octave*277.18;
58      buzzer.period(1/(2*fCsharp)); // set PWM period
59      buzzer=0.5; // set duty cycle
60      wait(0.4);
61      buzzer=0;
62  }
63
64  void Dnote(void){
65      currentNote = dnote;
66      masterDevice.printf("%c", currentNote);
67
68      float fD = octave*293.66;
69      buzzer.period(1/(2*fD)); // set PWM period
70      buzzer=0.5; // set duty cycle
71      wait(0.4);
72      buzzer=0;
73  }
74
75  void Dsharpnote(void){
76      currentNote = dsharpnote;
77      masterDevice.printf("%c", currentNote);
78
79      float fDsharp = octave*311.13;
80      buzzer.period(1/(2*fDsharp)); // set PWM period
81      buzzer=0.5; // set duty cycle
82      wait(0.4);
83      buzzer=0;
84  }
85
86  void Enote(void){
87      currentNote = enote;
88      masterDevice.printf("%c", currentNote);
89
90      float fE = octave*329.63;
91      buzzer.period(1/(2*fE)); // set PWM period
92      buzzer=0.5; // set duty cycle
93      wait(0.4);
94      buzzer=0;
95  }
96
97  void Fnote(void){
98      currentNote = fnote;
99      masterDevice.printf("%c", currentNote);
100
101     float fF = octave*349.23;
102     buzzer.period(1/(2*fF)); // set PWM period
103     buzzer=0.5; // set duty cycle
104     wait(0.4);
105     buzzer=0;
106 }
107
108 void Fsharpnote(void){
109     currentNote = fsharpnote;
110     masterDevice.printf("%c", currentNote);
111
```

```
    float fFsharp = octave*369.99;
    buzzer.period(1/(2*fFsharp)); // set PWM period
    buzzer=0.5; // set duty cycle
    wait(0.4);
    buzzer=0;
}

void Gnote(void){
    currentNote = gnote;
    masterDevice.printf("%c", currentNote);

    float fG = octave*392.00;
    buzzer.period(1/(2*fG)); // set PWM period
    buzzer=0.5; // set duty cycle
    wait(0.4);
    buzzer=0;
}

void Gsharpnote(void){
    currentNote = gsharpnote;
    masterDevice.printf("%c", currentNote);

    float fGsharp = octave*415.30;
    buzzer.period(1/(2*fGsharp)); // set PWM period
    buzzer=0.5; // set duty cycle
    wait(0.4);
    buzzer=0;
}

void Anote(void){
    currentNote = anote;
    masterDevice.printf("%c", currentNote);

    float fA = octave*440.00;
    buzzer.period(1/(2*fA)); // set PWM period
    buzzer=0.5; // set duty cycle
    wait(0.4);
    buzzer=0;
}

void Asharpnote(void){
    currentNote = asharpnote;
    masterDevice.printf("%c", currentNote);

    float fAsharp = octave*466.16;
    buzzer.period(1/(2*fAsharp)); // set PWM period
    buzzer=0.5; // set duty cycle
    wait(0.4);
    buzzer=0;
}

void Bnote(void){
    currentNote = bnote;
    masterDevice.printf("%c", bnote);

    float fB = octave*493.88;
```

```
168     buzzer.period(1/(2*fB)); // set PWM period
169     buzzer=0.5; // set duty cycle
170     wait(0.4);
171     buzzer=0;
172 }
173
174 void UpFreq(void){
175     counter += 1;
176     if (counter == 0){
177         octave = 1; LED4 = 1; LED5 = 0; LED6 = 0; LED7 = 0; LED8 = 0;
178     }
179     else if (counter == 1){
180         octave = 2; LED4 = 0; LED5 = 1; LED6 = 0; LED7 = 0; LED8 = 0;
181     }
182     else if (counter == 2){
183         octave = 4; LED4 = 0; LED5 = 0; LED6 = 1; LED7 = 0; LED8 = 0;
184     }
185     else if (counter == 3){
186         octave = 8; LED4 = 0; LED5 = 0; LED6 = 0; LED7 = 1; LED8 = 0;
187     }
188     else if (counter >= 4){
189         counter = 4;octave = 16; LED4 = 0; LED5 = 0; LED6 = 0; LED7 = 0; LED8 = 1;
190     }
191     wait(0.1);
192 }
193
194 void DownFreq(void){
195     counter -= 1;
196     if (counter <= 0){
197         counter = 0; octave = 1; LED4 = 1; LED5 = 0; LED6 = 0; LED7 = 0; LED8 = 0;
198     }
199     else if (counter == 1){
200         octave = 2; LED4 = 0; LED5 = 1; LED6 = 0; LED7 = 0; LED8 = 0;
201     }
202     else if (counter == 2){
203         octave = 4; LED4 = 0; LED5 = 0; LED6 = 1; LED7 = 0; LED8 = 0;
204     }
205     else if (counter == 3){
206         octave = 8; LED4 = 0; LED5 = 0; LED6 = 0; LED7 = 1; LED8 = 0;
207     }
208     else if (counter == 4){
209         octave = 16; LED4 = 0; LED5 = 0; LED6 = 0; LED7 = 0; LED8 = 1;
210     }
211     wait(0.1);
212 }
213
214 int main() {
215     masterDevice.baud(115200);
216
217     // setup interrupt functions
218     C.rise(&Cnote);
219     Csharp.rise(&Csharpnote);
220     D.rise(&Dnote);
221     Dsharp.rise(&Dsharpnote);
222     E.rise(&Enote);
```

```
223     F.rise(&Fnote);
224     Fsharp.rise(&Fsharpnote);
225     G.rise(&Gnote);
226     Gsharp.rise(&Gsharpnote);
227     A.rise(&Anote);
228     Asharp.rise(&Asharpnote);
229     B.rise(&Bnote);
230     Increment.rise(&UpFreq);
231     Decrement.rise(&DownFreq);
232 
233     lED4 = 1;
234 
235 
236     while(1) {
237         myled = 1;
238         wait(0.2);
239         myled = 0;
240         wait(0.2);
241     }
242 }
243 
244 void delay(void){
245     int j; //loop variable j
246     for (j=0;j<100000000;j++) {
247         j++; j--; //waste time
248     }
249 }
```

Code for the “Piano Keyboard” Mbed

This is the MCU that controls the OLED display and receives input from the MCU connected to the keyboard keys.

```
#include "mbed.h"
#include "Adafruit_SSD1306.h"
#include <vector>
using namespace std;

Serial slaveRED(p9, p10);

PwmOut buzzer(p21);
DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

DigitalIn enterButton(p24);
DigitalIn moveButton(p23);

// Globals
float x = 0.4;
char data_send_RED;// word we will send from slave to master
char data_receive_RED;// word we will receive from master
char menu_RED = 0x00;
//Define addresses of digital i/o control registers, as pointers to volatile data
//OUTPUTs
#define FIO0DIR (*(volatile unsigned int *)(0x2009C000))
#define FIO0PIN (*(volatile unsigned int *)(0x2009C014))

// Function Prototypes
void delay(void);
void OrangesLemons(void);
void TestSong(void);
void SongSelectionMenu();
void Heading();
void SessionMenu();
void songSessionMenu();

enum Notes { cnote = 0x00, csharpnote, dnote, dsharpnote, enote, fnote, fsharpnote,
             gnote, gsharpnote, anote, asharpnote, bnote };
Notes currentPlaybackNote;
Notes currentKeyboardNote;

// vector to store the keyboard notes that match playback notes
vector<char> noteMatchRecord;

// keyboard key press state
char checkKeyboardState = 0; // 1 if a keyboard key has been pressed

class State
{
public:
    uint16_t delay;
    void (*func)();
    State* nextState;
};
```

```
54
55 // an I2C sub-class that provides a constructed default
56 class I2CPreInit : public I2C
57 {
58 public:
59     I2CPreInit(PinName sda, PinName scl) : I2C(sda, scl)
60     {
61         frequency(400000);
62         start();
63     };
64 };
65
66 I2CPreInit gI2C(p28,p27); // SDA, SCL
67 #define DISPLAY_ADDRESS 0x78
68 Adafruit_SSD1306_I2c gOled2(gI2C,p22);
69
70 void songSelectionMenu()
71 {
72     gOled2.clearDisplay();
73     gOled2.display();
74
75     char songSelected = 'a';
76     gOled2.setTextCursor(0,0);
77     Heading();
78     char* songOneName3 = "> Test Song";
79     char* songTwoName3 = " Orag-Lem";
80     gOled2.printf("%s\n", songOneName3);
81     gOled2.printf("%s\n", songTwoName3);
82     gOled2.display();
83
84     bool buttonPressed = 0;
85     wait(2.0); // wait until ENTER button has been released
86
87     while(1)
88     {
89         wait(0.2);
90         if(moveButton == 1 && buttonPressed == 0)
91         {
92             buttonPressed = 1;
93             ++songSelected;
94             if(songSelected > 'b') // circle
95                 songSelected = 'a';
96
97             switch(songSelected) {
98                 case 'a':
99                     gOled2.clearDisplay();
100                    gOled2.display();
101                    gOled2.setTextCursor(0,0);
102                    Heading();
103                    char* songOneName = "> Test Song";
104                    char* songTwoName = " Orag-Lem";
105                    gOled2.printf("%s\n", songOneName);
106                    gOled2.printf("%s\n", songTwoName);
107                    gOled2.display();
108                    break;
```

```
109                 case 'b':
110                     gOled2.clearDisplay();
111                     gOled2.display();
112                     gOled2.setTextCursor(0,0);
113                     Heading();
114                     char* songOneName2 = " Test Song";
115                     char* songTwoName2 = "> Orag-Lem";
116                     gOled2.printf("%s\n", songOneName2);
117                     gOled2.printf("%s\n", songTwoName2);
118                     gOled2.display();
119                     break;
120             }
121         }
122         else if(moveButton == 0 && buttonPressed == 1)
123         {
124             buttonPressed = 0;
125         }
126         else if(enterButton == 1)
127         {
128             songSessionMenu();
129             if(songSelected == 'a')
130                 TestSong();
131             else
132                 OrangesLemons();
133 
134             gOled2.clearDisplay();
135             gOled2.setTextCursor(0,0);
136             gOled2.display();
137 
138             int totalNotesMatched = 0;
139 
140             for(int i = 0; i < noteMatchRecord.size(); ++i)
141             {
142                 if(noteMatchRecord[i] != CHAR_MAX)
143                     ++totalNotesMatched;
144             }
145 
146             gOled2.setTextCursor(0,0);
147             if(totalNotesMatched == 0) {
148                 Heading();
149                 gOled2.printf("result: %.2f %%\n", 0.0f);
150             }
151             else {
152                 //gOled2.printf("result: %f\n", (float)totalNotesMatched);
153                 //gOled2.printf("result: %f\n", (float)noteMatchRecord.size());
154                 Heading();
155                 gOled2.display();
156                 gOled2.printf("result: %.2f %%\n",
157                     100.0f * ((float)totalNotesMatched/(float)noteMatchRecord.size()));
158                 gOled2.display();
159             }
160             gOled2.display();
161         }
162     }
163 }
```

```
164
165 void songSessionMenu()
166 {
167     gOled2.clearDisplay();
168     gOled2.display();
169     gOled2.setTextCursor(0,0);
170     gOled2.printf("song in session\n");
171     gOled2.display();
172 }
173
174 int main() {
175     slaveRED.baud(115200);
176     FIO0DIR = 0x078783C0;
177
178     char menuPage = 1;
179
180     gOled2.clearDisplay();
181     gOled2.display();
182
183     // Display song selection menu
184     gOled2.setTextCursor(0,0);
185     Heading();
186     gOled2.printf("> Song Menu\n");
187     gOled2.display();
188
189     bool buttonPressed = 0;
190
191     while(1)
192     {
193         if(moveButton == 1 && buttonPressed == 0)
194         {
195             buttonPressed = 1;
196             ++menuPage;
197             if(menuPage > 1)
198                 menuPage = 1;
199         }
200         else if(moveButton == 0 && buttonPressed == 1)
201         {
202             buttonPressed = 0;
203         }
204         else if(enterButton == 1)
205         {
206             switch(menuPage) {
207             case 1:
208                 songSelectionMenu();
209                 break;
210             }
211         }
212     }
213 }
214
215 void SongSelectionMenu()
216 {
217     gOled2.printf("> Song One\r\n");
218 }
219
```

```
220 void Heading()
221 {
222     gOled2.printf("__ PianoTeacher __\r\n");
223 }
224
225 void SessionMenu()
226 {
227     gOled2.printf("Start Playing\r\n");
228 }
229
230 void toggleKeyboardInputState()
231 {
232     if(checkKeyboardState == 0)
233         checkKeyboardState = 1;
234 }
235
236 void checkKeyboard()
237 {
238     if(checkKeyboardState == 1)
239         return;
240
241     if(slaveRED.readable()) {
242         char keyboardNote = slaveRED.getc();
243         currentKeyboardNote = (enum Notes) keyboardNote;
244         // prevent matching more than 1 keypress to a playback note
245         if(keyboardNote == currentPlaybackNote) {
246             led1 = !led1;
247             noteMatchRecord.push_back(keyboardNote);
248         }
249         else if(keyboardNote == CHAR_MAX){
250             noteMatchRecord.push_back(CHAR_MAX);
251         }
252         toggleKeyboardInputState();
253     }
254 }
255
256 // Song Functions
257 void OrangesLemons(void){
258     Notes notes[] = {enote, csharpnote, enote, csharpnote, anote,
259         bnote, csharpnote, dnote, bnote, enote, csharpnote, anote};
260     float frequency[]={659,554,659,554,440,494,554,587,494,659,554,440};
261     //E,C#,E,C#,A,B,C#,D,B,E,C#,A
262     float beat[]={1,1,1,1,1,0.5,0.5,1,1,1,1,2}; //beat array
263     int RedLED[] = {0x40000,0x100,0x40000,0x100,0x1000000,0x4000000,
264         0x100,0x80,0x4000000,0x40000,0x100,0x1000000};
265     noteMatchRecord.clear();
266     noteMatchRecord.resize(12, CHAR_MAX);
267
268     Ticker ticker;
269     ticker.attach(&checkKeyboard, 0.0001);
270
271     for (int i=0;i<=11;i++) {
272         currentPlaybackNote = notes[i];
273         FIO0PIN = RedLED[i];
274         buzzer.period(1/(2*frequency[i])); // set PWM period
```

```
        buzzer=0.5; // set duty cycle
        wait(0.7*beat[i]);
        if(checkKeyboardState == 0)
        {
            noteMatchRecord.push_back(CHAR_MAX);
        }
    }

    ticker.detach();
    buzzer=0;
    FIO0PIN = 0x00000000;
    wait(0.1);
    buzzer=0.0;
}

void TestSong()
{
    Notes notes[] = {cnote, csharpnote, dnote, dsharpnote, enote, fnote,
        fsharpnote, gnote, gsharpnote, anote, asharpnote, bnote };
    float frequency[]={523, 554, 587, 622, 659, 698, 739, 783, 830, 880, 932, 987};
    //E,C#,E,C#,A,B,C#,D,B,E,C#,A
    float beat[]={1,1,1,1,1,1,1,1,1,1,1,1}; //beat array
    int RedLED[] = {0x200, 0x100, 0x80, 0x40, 0x40000, 0x20000, 0x8000,
        0x10000, 0x800000, 0x1000000, 0x2000000, 0x4000000};
    Ticker ticker;
    ticker.attach(&checkKeyboard, 0.0001);

    for (int i=0;i<=11;i++) {
        checkKeyboardState = 0;
        currentPlaybackNote = notes[i];
        FIO0PIN = RedLED[i];
        buzzer.period(1/(2*frequency[i])); // set PWM period
        buzzer=0.5; // set duty cycle
        wait(0.7*beat[i]);
        // record that no note was played
        if(checkKeyboardState == 0)
        {
            noteMatchRecord.push_back(CHAR_MAX);
        }
    }

    ticker.detach();
    buzzer=0;
    FIO0PIN = 0x00000000;
    wait(0.1);
    buzzer=0.0;
}

//delay function
void delay(void){
    int j; //loop variable j
    for (j=0;j<100000000;j++) {
        j++; j--; //waste time
    }
}
```

Troubleshooting

Issue #1: A Simple Miswiring

We were unable to get UART communication to work; and the reason why is because we miswired the RX/TX pins. We had the RX->RX and TX->TX. We learned that the issue may not always be software and that the hardware configuration should be verified periodically during regression tests.

Issue #2: Unable to use I2C

Prior to using UART communication between both mbed boards, we tried I2C. We were unable to transmit data between the two boards. We attempted multiple implementations such as: (1) not using pull-up resistors in the connection of the SDA/SCL pins and instead setting the internal pin mode to pull-up, (2) configuring the mbed connection to be master-to-slave, (3) and manually setting the baud rate. To troubleshoot we printed the data received by the Teacher mbed. Since no data was received, the communication did not work.

Issue #3: Serial Communication Receive Issue

We had some difficulty getting UART to work properly. The issue was that the data received by the Teacher mbed was appearing as garbage when printed on the OLED display. The issue had something to do with the Teacher mbed not recognizing the input; though this may have been around the same time when the RX/TX pins were improperly connected. Anyway, after the RX/TX pins were correctly connected, we implemented a procedure that causes the Teacher mbed to poll the UART RX pin while it is playing a musical note. If the Teacher mbed detects a note was at any point during the beat's interval, then a match was registered.

Issue #4: Piano LED Lighting Issue

To light the piano LEDs, a resistor ladder was attempted in simulation. The core reason a resistor ladder was suggested to be used was to keep the amount of input pins to a minimum, and with a resistor ladder connected to an analog input pin, this would be possible. The simulation was done with a 50 ohm resistor connected in series with each LED, this allowed for different voltage readings depending on the key that is pushed. However, because of a lack of 50 ohm resistors, the ladder was attempted with 470 ohm resistors instead. This proved to fail because the resistance was lowering the voltage by a considerable amount and made some LEDs extremely dim. So instead, it was decided to just use 12 input pins for the piano. In the end, this proved to be a better solution because this would, in theory, allow the input of multiple signals at once or at least with quite less delay—something not possible with a resistor ladder.

Recommendations and/or Conclusions

To conclude, this presented a completed product of a smart digital piano that can be used as a downscaled version of a regular piano. Although this product does not have all the 7 octaves a standard piano has, it has 5 octaves that the user can manually change with buttons. The piano keys also come with a speaker and LEDs that light up when the user presses a key so that they can see and hear what key they pressed. The smart piano also has a feature that the team calls a 'Piano Teacher.' This feature consists of an OLED screen that allows the user to select a song from a library. When the song is selected, the music will begin playing, and the row of red LEDs will light up depending on which note is supposed to be played. The purpose of these LEDs is to guide the user through the song so they can learn or practice. Once the song ends, the user's performance will be displayed on the OLED. The performance is shown as a percentage of notes that were pressed by the user when they were supposed to be pressed as indicated by the LEDs. The purpose of this is to inform the user of their performance and encourage them to continue practicing to beat their score.

While this product has many features, there is still much room for improvement. For one, the library of songs should contain more than the 2 songs available now. The more songs, the more practice the user would receive. Another thing that needs improvement is the number of notes that can be played simultaneously. Many notes can be pressed simultaneously with the current setup, and the LEDs work accordingly. However, only one note can be heard at a time. This means that it would be unable to play more complex songs that require more than one note to be pressed.

Another very important improvement to this project that would boost its teaching ability is adding an option to lower the speed at which notes appear. This would allow a player to learn the song slowly instead of being overwhelmed with the rate at which notes appear. An improvement that would work in tandem with the previous one is adding an indicator for when each note would appear, this would let the player get better timing for a song.