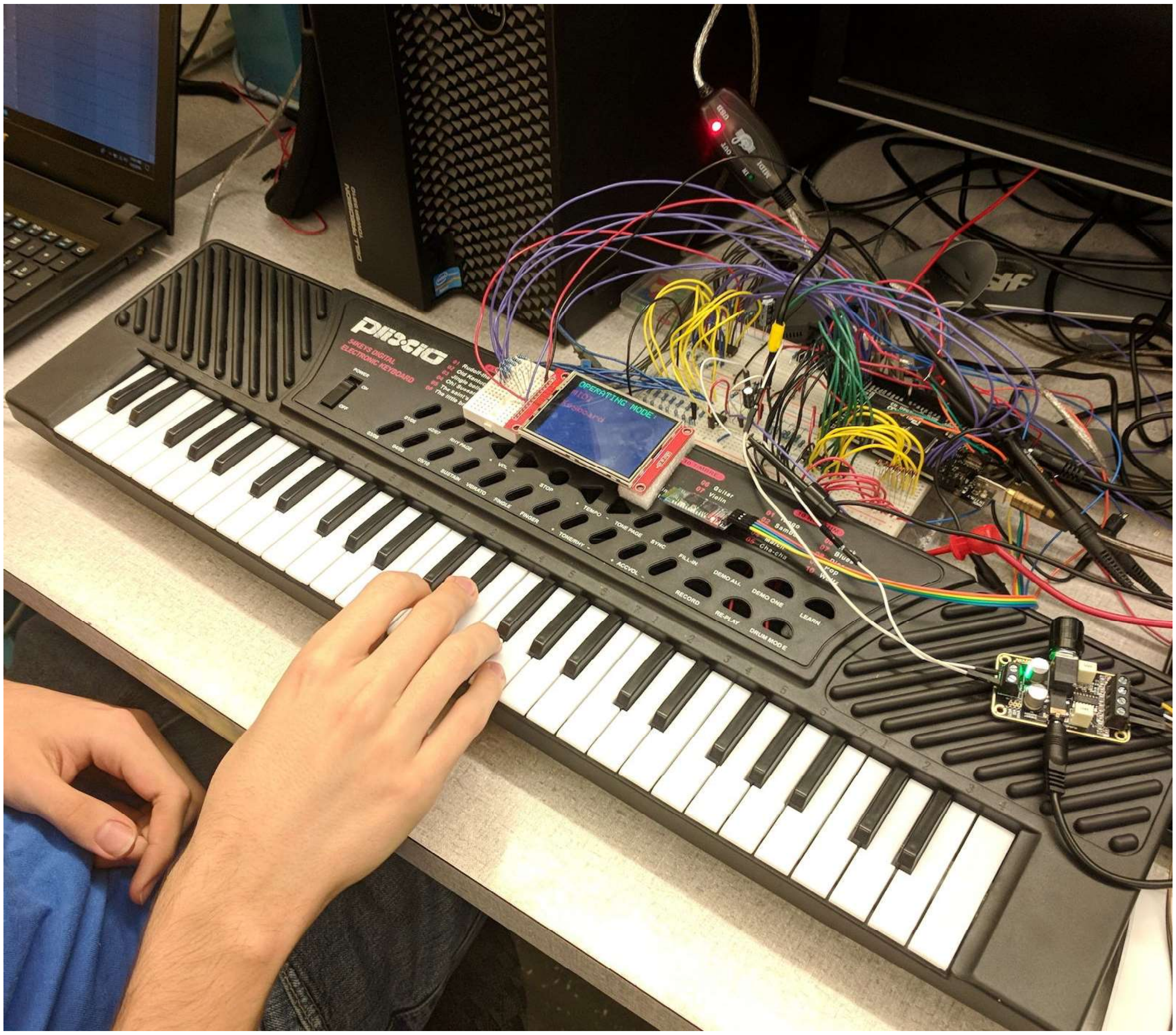

Figure 8: Final Functioning Prototype of the CDT MixMaster 8000

Full Arduino Code:

```
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

//using pins 40 - 45 for LCD
#define TFT_CS 40
#define TFT_RST 41
#define TFT_DC 42
#define TFT_MOSI 43
```

```
#define TFT_CLK 44
#define TFT_MISO 45

#define ROWS 7
#define COLS 8

// MIDI serial definitions
#define NOTE_ON_CMD 0x90
#define NOTE_OFF_CMD 0x80
#define NOTE_VELOCITY 0x7F // any value 0-127

// MIDI baud rate
#define SERIAL_RATE 31250

// Maximum number of keys that can be played at once
#define MAX_KEYS_PLAYED 12

// Number of samples in sine wave. Highly recommend using a power of 2.
#define SAMPLES 4096



//-----------------------MIDI-------------------------
// column pins
const int col1Pin = 29;
const int col2Pin = 30;
const int col3Pin = 31;
const int col4Pin = 32;
const int col5Pin = 33;
const int col6Pin = 34;
const int col7Pin = 35;
const int col8Pin = 36;
const int colPins[COLS] =
{
  col1Pin,
  col2Pin,
  col3Pin,
  col4Pin,
  col5Pin,
  col6Pin,
  col7Pin,
  col8Pin
};

// row pins
const int row1Pin = 22;
const int row2Pin = 23;
const int row3Pin = 24;
const int row4Pin = 25;
const int row5Pin = 26;
const int row6Pin = 27;
const int row7Pin = 28;
const int rowPins[ROWS] =
{
  row1Pin,
  row2Pin,
  row3Pin,
  row4Pin,
  row5Pin,
  row6Pin,
  row7Pin
};

boolean keyPressed[ROWS][COLS];
uint8_t keyToMidiMap[ROWS][COLS];
```

```
//--------------BLUETOOTH-----------------------------------
//SoftwareSerial BT(46, 47); // RX, TX --> can be any digital pins
volatile char command; // data from HC-06 Bluetooth device

//-----------------LCD--------------------------------------
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK,
TFT_RST, TFT_MISO);
int tracker1;
int tracker2;
int tracker3;
int tracker4;
int tracker5;

//----------------INDEPENDENT-------------------------------
//const int speaker = 37;
//int timer;


double frequencyChart[] =
{
  32.70, 34.65, 36.71, 38.89, 41.20, 43.65, 46.25, 49.00,
  51.91, 55.00, 58.27, 61.74, 65.41, 69.30, 73.42, 77.78,
  82.41, 87.31, 92.50, 98.00, 103.83, 110.00, 116.54, 123.47,
  130.81, 138.59, 146.83, 155.56, 164.81, 174.61, 185.00, 196.00,
  207.65, 220.0, 223.08, 246.94, 261.63, 277.18, 293.66, 311.13, 329.63,
  349.23, 369.99, 392.00, 415.30, 440.00, 466.16, 493.88, 523.25, 554.37,
  587.33, 622.25, 659.25, 698.46,
};
byte storedWave[SAMPLES];  // Stored waveform

unsigned long t = 0;  // Counter used to iterate through the waveform
int buf;  // Buffer for the next value to load into the DAC.
```

```
unsigned int count;  // Count of the loop. Used to limit the number of
function calls per loop.

void setup() {
  count = 0;

  //setting up LCD
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 0;
  tracker4 = 0;
  tracker5 = 0;

  tft.begin();
  tft.setRotation(3); // rotate screen orientation
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_GREEN);
  tft.setTextSize(3);
  tft.println("WELCOME TO...");
  delay(2000);
  tft.setTextColor(ILI9341_RED);
  tft.setTextSize(4);
  tft.println();
  tft.println("The CDT");
  tft.println("MixMaster");
  tft.println("8000!!!");
  delay(3500); // display this message for a bit
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_GREEN);
  tft.setTextSize(3);
  tft.println("OPERATING MODE:");
```

```
// boolean matrix that checks for keys pressed
for (int i = 0; i < COLS; ++i) {
    for (int j = 0; j < ROWS; ++j) {
    keyPressed[j][i] = false;
    }
}

// maps correct MIDI notes to each key
int note = 0x24; //c2 = note 36
for (int i = 0; i < ROWS ; ++i) {
    for (int j = 0; j < COLS; ++j) {
    keyToMidiMap[i][j] = note;
    note++;
    }
}

// pin mode setup

pinMode(col1Pin, OUTPUT);
pinMode(col2Pin, OUTPUT);
pinMode(col3Pin, OUTPUT);
pinMode(col4Pin, OUTPUT);
pinMode(col5Pin, OUTPUT);
pinMode(col6Pin, OUTPUT);
pinMode(col7Pin, OUTPUT);
pinMode(col8Pin, OUTPUT);

//pinMode(speaker, OUTPUT);

pinMode(row1Pin, INPUT);
pinMode(row2Pin, INPUT);
pinMode(row3Pin, INPUT);
pinMode(row4Pin, INPUT);
pinMode(row5Pin, INPUT);
pinMode(row6Pin, INPUT);
pinMode(row7Pin, INPUT);

Serial.begin(2000000);
Serial1.begin(SERIAL_RATE);

// The HC-06 defaults to 9600 according to the datasheet.
Serial3.begin(9600);
command = '0';

 // DAC Pins. PORT D.
pinMode(10, OUTPUT);
pinMode(11, OUTPUT);
pinMode(12, OUTPUT);
pinMode(13, OUTPUT);
pinMode(50, OUTPUT);
pinMode(51, OUTPUT);
pinMode(52, OUTPUT);
pinMode(53, OUTPUT);
// Setup interrupts for signal output to DAC.
cli();//stop interrupts

//set timer2 interrupt at 40kHz
TCCR2A = 0;// set entire TCCR2A register to 0
TCCR2B = 0;// same for TCCR2B
TCNT2  = 0;//initialize counter value to 0
// set compare match register for 40khz increments
OCR2A = 49;//(84*10000) / ((8*10)-1)
// turn on CTC mode
TCCR2A |= (1 << WGM21);
// Set CS21 bit for 8 prescaler
```

```
  TCCR2B |= (1 << CS21);
  // enable timer compare interrupt
  TIMSK2 |= (1 << OCIE2A);
  sei();//allow interrupts

  // Adjust frequencies to proper values
  // 10 is used based on the 4000 samples of the array
  // 2 is used to increase the notes by 1 octave
  for (int j = 0; j < 54; j++) {
      frequencyChart[j] = (frequencyChart[j] / 10.0) * 2;
  }

  // Initialize the stored wave to a sine wave.
  //generateSine();
}

// Smallest possible implementation of scanKeys(), smallest being time increment
int bigRowCtr = 0;
int bigColCtr = 0;
int bigRowValue[ROWS];
void scanKeys() {
  digitalWrite(colPins[bigColCtr], HIGH);

  bigRowValue[bigRowCtr] = digitalRead(rowPins[bigRowCtr]);

  if (bigRowValue[bigRowCtr] != 0 && !keyPressed[bigRowCtr][bigColCtr]) {
      keyPressed[bigRowCtr][bigColCtr] = true;
  }
  if (bigRowValue[bigRowCtr] == 0 && keyPressed[bigRowCtr][bigColCtr]) {
      keyPressed[bigRowCtr][bigColCtr] = false;
  }
  digitalWrite(colPins[bigColCtr], LOW);
  bigRowCtr++;
  if (bigRowCtr >= ROWS) {
      bigRowCtr = 0;
      bigColCtr++;
      if (bigColCtr >= COLS) {
      bigColCtr = 0;
      }
  }
}

void noteOn(int row, int col) {
  Serial1.write(NOTE_ON_CMD);
  Serial1.write(keyToMidiMap[row][col]);
  Serial1.write(NOTE_VELOCITY);
}

void noteOff(int row, int col) {
  Serial1.write(NOTE_OFF_CMD);
  Serial1.write(keyToMidiMap[row][col]);
  Serial1.write(NOTE_VELOCITY);
}

void drawRects() {
  /*
  for (int j = 80; j < 120; j += 20) {
      for (int i = 0; i < 320; i += 20) {
      tft.fillRect(i, j, 20, 20, ILI9341_BLACK);
      }
  }
  */
  tft.fillRect(55, 80, 265, 40, ILI9341_BLACK);
}
```

```
void sineWave() {
  if (tracker1 == 0) {
      generateSine();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Sine Wave");

  }
  tracker1 = 1;
  tracker2 = 0;
  tracker3 = 0;
  tracker4 = 0;
  tracker5 = 0;

  incrementedScan();
  buf = loadBuffer();

}

void squareWave () {
  if (tracker3 == 0) {
      generateSquare();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
```

```
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Square Wave");

  }
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 1;
  tracker4 = 0;
  tracker5 = 0;

  incrementedScan();
  buf = loadBuffer();
}

void sawtoothWave () {
  if (tracker4 == 0) {
      generateSawtooth();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Sawtooth Wave");

  }
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 0;
  tracker4 = 1;
  tracker5 = 0;

  incrementedScan();
  buf = loadBuffer();
}

void triangleWave () {
  if (tracker5 == 0) {
      generateTriangle();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Triangle Wave");

  }
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 0;
  tracker4 = 0;
  tracker5 = 1;

  incrementedScan();
  buf = loadBuffer();
}

void MIDIOp() {
  if (tracker2 == 0) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("MIDI");
      tft.println();
      tft.println("Keyboard");
  }
  tracker2 = 1;
  tracker1 = 0;
  tracker3 = 0;
  tracker4 = 0;
  tracker5 = 0;

  for (int colCtr = 0; colCtr < COLS; ++colCtr) {
      digitalWrite(colPins[colCtr], HIGH);

      int rowValue[ROWS]; // moved this to inside the loop

      // get the row values of the scanned columns
      rowValue[0] = digitalRead(row1Pin);
      rowValue[1] = digitalRead(row2Pin);
      rowValue[2] = digitalRead(row3Pin);
      rowValue[3] = digitalRead(row4Pin);
      rowValue[4] = digitalRead(row5Pin);
      rowValue[5] = digitalRead(row6Pin);
      rowValue[6] = digitalRead(row7Pin);

      // process keys pressed
      for(int rowCtr=0; rowCtr < ROWS; ++rowCtr) {
      if (rowValue[rowCtr] != 0 && !keyPressed[rowCtr][colCtr]) {
      keyPressed[rowCtr][colCtr] = true;
      noteOn(rowCtr,colCtr);
      } else if (rowValue[rowCtr] == 0 && keyPressed[rowCtr][colCtr]) {
      keyPressed[rowCtr][colCtr] = false;
      noteOff(rowCtr,colCtr);
      }
      }
      digitalWrite(colPins[colCtr], LOW);
  }
}

// Returns the correct value to load into the buffer
int loadBuffer() {
  unsigned int result = 0;
  unsigned int num = 0;
  unsigned int pos = t;
  double freq = 40;
  unsigned int leftshift = 0;
  unsigned int rightshift = 0;

  for (int i = 0; i < COLS; ++i) {
      for (int j = 0; j < ROWS; ++j) {
      if (keyPressed[j][i]) {
      //freq = frequencyChart[(i*8) + j];
      freq = frequencyChart[i + (j*8)];
      /*
      leftshift = analogRead(0);
```

```
    rightshift = analogRead(1);

    Serial.print("leftshift: ");
    Serial.println(leftshift);
    Serial.print("rightshift: ");
    Serial.print(rightshift);

    if (leftshift > 350 && leftshift < 610) {
          freq = freq / ((0.0125 * (double) leftshift) + 1);
    } else if (rightshift > 350 && rightshift < 610) {
          freq = freq * ((0.0125 * (double) rightshift) + 1);
    }
    */
    //Serial.println((i*8) + j);
    //Serial.print("leftshift: ");
    //Serial.println(leftshift);
    //Serial.print("rightshift: ");
    //Serial.print(rightshift);

    pos = freq * t;
    pos = pos & 0xFFF;

    result += storedWave[pos];  //library here
    num++;
    }
    }
}

// Normalize the wave. Consider optimizing. Division is very expensive.
if (num) {
    //result = result >> 3;
    result = result / num;
}
```

```
  return result;
}

// Generate sine wave and place values in storedWave
void generateSine() {
   // Generate sine signals
  for (int i = 0; i < SAMPLES; i++) {
      storedWave[i] = 127 + (127 * sin((6.28*i)/SAMPLES));
      //Serial.println(storedWave[i]);
  }
}

// Generate sawtooth wave and place values in storedWave
void generateSawtooth() {
  for (int i = 0; i < SAMPLES; i++) {
      storedWave[i] = (i * 256) / SAMPLES;
  }
}

// Generate square wave and place values in storedWave
void generateSquare() {
  for (int i = 0; i < SAMPLES; i++) {
      if (i < (SAMPLES / 2)) {
      storedWave[i] = 255;
      } else {
      storedWave[i] = 0;
      }
  }
}

// Generate triangle wave and place values in storedWave
void generateTriangle() {
  for (int i = 0; i < SAMPLES; i++) {
      if (i < (SAMPLES/2)) {
      storedWave[i] = (i * 256 * 2) / SAMPLES;
      } else {
      storedWave[i] = 255 - (((i - (SAMPLES / 2)) * 256 * 2) / SAMPLES);
      }
  }
}

void incrementedScan() {
   // Scan the keys once every 10 loops
  count++;
  if (count >= 10) {
      count = 0;
  }
  if (!count) {
      scanKeys();
  }
}

void loop() {

  if (Serial3.available()) { // if there is bluetooth data, update command
      while(Serial3.available()) { // While there is more to be read, keep
reading.
      command = Serial3.read();
      }
  }
  if (command == 'm') { //MIDI OPERATION
      MIDIOp();
  } else if (command == 'i')  { //INDEPENDENT OPERATION
      sineWave();
  } else if (command == 'q') {
      squareWave();
  } else if (command == 'a') {
      sawtoothWave();
  } else if (command == 't') {
      triangleWave();
  } else {
      sineWave();
  }

}

// Interrupt code for timer 2
ISR(TIMER2_COMPA_vect){
  PORTB = buf;
  t++;
}//#include <SoftwareSerial.h>//bluetooth header
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

//using pins 40 - 45 for LCD
#define TFT_CS 40
#define TFT_RST 41
#define TFT_DC 42
#define TFT_MOSI 43
#define TFT_CLK 44
#define TFT_MISO 45

#define ROWS 7
#define COLS 8

// MIDI serial definitions
#define NOTE_ON_CMD 0x90
#define NOTE_OFF_CMD 0x80
#define NOTE_VELOCITY 0x7F // any value 0-127
```

```
// MIDI baud rate
#define SERIAL_RATE 31250

// Maximum number of keys that can be played at once
#define MAX_KEYS_PLAYED 12

// Number of samples in sine wave. Highly recommend using a power of 2.
#define SAMPLES 4096



//-----------------------MIDI-------------------------
// column pins
const int col1Pin = 29;
const int col2Pin = 30;
const int col3Pin = 31;
const int col4Pin = 32;
const int col5Pin = 33;
const int col6Pin = 34;
const int col7Pin = 35;
const int col8Pin = 36;
const int colPins[COLS] =
{
  col1Pin,
  col2Pin,
  col3Pin,
  col4Pin,
  col5Pin,
  col6Pin,
  col7Pin,
  col8Pin
};
```

```
// row pins
const int row1Pin = 22;
const int row2Pin = 23;
const int row3Pin = 24;
const int row4Pin = 25;
const int row5Pin = 26;
const int row6Pin = 27;
const int row7Pin = 28;
const int rowPins[ROWS] =
{
  row1Pin,
  row2Pin,
  row3Pin,
  row4Pin,
  row5Pin,
  row6Pin,
  row7Pin
};

boolean keyPressed[ROWS][COLS];
uint8_t keyToMidiMap[ROWS][COLS];


//--------------BLUETOOTH-----------------------------------
//SoftwareSerial BT(46, 47); // RX, TX --> can be any digital pins
volatile char command; // data from HC-06 Bluetooth device

//-----------------LCD--------------------------------------
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST, TFT_MISO);
int tracker1;
int tracker2;
int tracker3;
int tracker4;
int tracker5;

//----------------INDEPENDENT-------------------------------
//const int speaker = 37;
//int timer;


double frequencyChart[] =
{
  32.70, 34.65, 36.71, 38.89, 41.20, 43.65, 46.25, 49.00,
  51.91, 55.00, 58.27, 61.74, 65.41, 69.30, 73.42, 77.78,
  82.41, 87.31, 92.50, 98.00, 103.83, 110.00, 116.54, 123.47,
  130.81, 138.59, 146.83, 155.56, 164.81, 174.61, 185.00, 196.00,
  207.65, 220.0, 223.08, 246.94, 261.63, 277.18, 293.66, 311.13, 329.63,
  349.23, 369.99, 392.00, 415.30, 440.00, 466.16, 493.88, 523.25, 554.37,
  587.33, 622.25, 659.25, 698.46,
};
byte storedWave[SAMPLES];  // Stored waveform

unsigned long t = 0;  // Counter used to iterate through the waveform
int buf;  // Buffer for the next value to load into the DAC.
unsigned int count;  // Count of the loop. Used to limit the number of
function calls per loop.

void setup() {
  count = 0;

  //setting up LCD
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 0;
```

```
tracker4 = 0;
tracker5 = 0;

tft.begin();
tft.setRotation(3); // rotate screen orientation
tft.fillScreen(ILI9341_BLACK);
tft.setCursor(0, 0);
tft.setTextColor(ILI9341_GREEN);
tft.setTextSize(3);
tft.println("WELCOME TO...");
delay(2000);
tft.setTextColor(ILI9341_RED);
tft.setTextSize(4);
tft.println();
tft.println("The CDT");
tft.println("MixMaster");
tft.println("8000!!!");
delay(3500); // display this message for a bit
tft.fillScreen(ILI9341_BLACK);
tft.setCursor(0, 0);
tft.setTextColor(ILI9341_GREEN);
tft.setTextSize(3);
tft.println("OPERATING MODE:");

// boolean matrix that checks for keys pressed
for (int i = 0; i < COLS; ++i) {
    for (int j = 0; j < ROWS; ++j) {
    keyPressed[j][i] = false;
    }
}

// maps correct MIDI notes to each key
int note = 0x24; //c2 = note 36
```

```
for (int i = 0; i < ROWS ; ++i) {
    for (int j = 0; j < COLS; ++j) {
    keyToMidiMap[i][j] = note;
    note++;
    }
}

// pin mode setup

pinMode(col1Pin, OUTPUT);
pinMode(col2Pin, OUTPUT);
pinMode(col3Pin, OUTPUT);
pinMode(col4Pin, OUTPUT);
pinMode(col5Pin, OUTPUT);
pinMode(col6Pin, OUTPUT);
pinMode(col7Pin, OUTPUT);
pinMode(col8Pin, OUTPUT);

//pinMode(speaker, OUTPUT);

pinMode(row1Pin, INPUT);
pinMode(row2Pin, INPUT);
pinMode(row3Pin, INPUT);
pinMode(row4Pin, INPUT);
pinMode(row5Pin, INPUT);
pinMode(row6Pin, INPUT);
pinMode(row7Pin, INPUT);

Serial.begin(2000000);
Serial1.begin(SERIAL_RATE);

// The HC-06 defaults to 9600 according to the datasheet.
Serial3.begin(9600);
```

```
command = '0';

 // DAC Pins. PORT D.
pinMode(10, OUTPUT);
pinMode(11, OUTPUT);
pinMode(12, OUTPUT);
pinMode(13, OUTPUT);
pinMode(50, OUTPUT);
pinMode(51, OUTPUT);
pinMode(52, OUTPUT);
pinMode(53, OUTPUT);
// Setup interrupts for signal output to DAC.
cli();//stop interrupts

//set timer2 interrupt at 40kHz
TCCR2A = 0;// set entire TCCR2A register to 0
TCCR2B = 0;// same for TCCR2B
TCNT2  = 0;//initialize counter value to 0
// set compare match register for 40khz increments
OCR2A = 49;//(84*10000) / ((8*10)-1)
// turn on CTC mode
TCCR2A |= (1 << WGM21);
// Set CS21 bit for 8 prescaler
TCCR2B |= (1 << CS21);
// enable timer compare interrupt
TIMSK2 |= (1 << OCIE2A);
sei();//allow interrupts

// Adjust frequencies to proper values
// 10 is used based on the 4000 samples of the array
// 2 is used to increase the notes by 1 octave
for (int j = 0; j < 54; j++) {
    frequencyChart[j] = (frequencyChart[j] / 10.0) * 2;
```

```
  }

  // Initialize the stored wave to a sine wave.
  //generateSine();
}

// Smallest possible implementation of scanKeys(), smallest being time
increment
int bigRowCtr = 0;
int bigColCtr = 0;
int bigRowValue[ROWS];
void scanKeys() {
  digitalWrite(colPins[bigColCtr], HIGH);

  bigRowValue[bigRowCtr] = digitalRead(rowPins[bigRowCtr]);

  if (bigRowValue[bigRowCtr] != 0 && !keyPressed[bigRowCtr][bigColCtr]) {
      keyPressed[bigRowCtr][bigColCtr] = true;
  }
  if (bigRowValue[bigRowCtr] == 0 && keyPressed[bigRowCtr][bigColCtr]) {
      keyPressed[bigRowCtr][bigColCtr] = false;
  }
  digitalWrite(colPins[bigColCtr], LOW);
  bigRowCtr++;
  if (bigRowCtr >= ROWS) {
      bigRowCtr = 0;
      bigColCtr++;
      if (bigColCtr >= COLS) {
      bigColCtr = 0;
      }
  }
}
```

```
void noteOn(int row, int col) {
  Serial1.write(NOTE_ON_CMD);
  Serial1.write(keyToMidiMap[row][col]);
  Serial1.write(NOTE_VELOCITY);
}

void noteOff(int row, int col) {
  Serial1.write(NOTE_OFF_CMD);
  Serial1.write(keyToMidiMap[row][col]);
  Serial1.write(NOTE_VELOCITY);
}

void drawRects() {
  /*
  for (int j = 80; j < 120; j += 20) {
      for (int i = 0; i < 320; i += 20) {
      tft.fillRect(i, j, 20, 20, ILI9341_BLACK);
      }
  }
  */
  tft.fillRect(55, 80, 265, 40, ILI9341_BLACK);
}

void sineWave() {
  if (tracker1 == 0) {
      generateSine();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Sine Wave");

  }
  tracker1 = 1;
  tracker2 = 0;
  tracker3 = 0;
  tracker4 = 0;
  tracker5 = 0;

  incrementedScan();
  buf = loadBuffer();

}

void squareWave () {
  if (tracker3 == 0) {
      generateSquare();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Square Wave");

  }
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 1;
  tracker4 = 0;
  tracker5 = 0;

  incrementedScan();
  buf = loadBuffer();
}

void sawtoothWave () {
  if (tracker4 == 0) {
      generateSawtooth();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Sawtooth Wave");

  }
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 0;
  tracker4 = 1;
  tracker5 = 0;

  incrementedScan();
  buf = loadBuffer();
}

void triangleWave () {
  if (tracker5 == 0) {
      generateTriangle();
      if (tracker2 == 1) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      } else {
      drawRects();
      }
      tft.setCursor(0, 40);
      tft.setTextSize(3);
      tft.setTextColor(ILI9341_RED);
      tft.println("Normal Keyboard");
      tft.println();
      tft.println("--> Triangle Wave");

  }
  tracker1 = 0;
  tracker2 = 0;
  tracker3 = 0;
  tracker4 = 0;
  tracker5 = 1;

  incrementedScan();
  buf = loadBuffer();
}

void MIDIOp() {
  if (tracker2 == 0) {
      tft.fillRect(0, 30, 320, 90, ILI9341_BLACK);
      tft.setCursor(0, 40);
```

```
        tft.setTextSize(3);
        tft.setTextColor(ILI9341_RED);
        tft.println("MIDI");
        tft.println();
        tft.println("Keyboard");
    }
    tracker2 = 1;
    tracker1 = 0;
    tracker3 = 0;
    tracker4 = 0;
    tracker5 = 0;

    for (int colCtr = 0; colCtr < COLS; ++colCtr) {
        digitalWrite(colPins[colCtr], HIGH);

        int rowValue[ROWS]; // moved this to inside the loop

        // get the row values of the scanned columns
        rowValue[0] = digitalRead(row1Pin);
        rowValue[1] = digitalRead(row2Pin);
        rowValue[2] = digitalRead(row3Pin);
        rowValue[3] = digitalRead(row4Pin);
        rowValue[4] = digitalRead(row5Pin);
        rowValue[5] = digitalRead(row6Pin);
        rowValue[6] = digitalRead(row7Pin);

        // process keys pressed
        for(int rowCtr=0; rowCtr < ROWS; ++rowCtr) {
        if (rowValue[rowCtr] != 0 && !keyPressed[rowCtr][colCtr]) {
        keyPressed[rowCtr][colCtr] = true;
        noteOn(rowCtr,colCtr);
        } else if (rowValue[rowCtr] == 0 && keyPressed[rowCtr][colCtr]) {
        keyPressed[rowCtr][colCtr] = false;
```

```
      noteOff(rowCtr,colCtr);
      }
      }
      digitalWrite(colPins[colCtr], LOW);
  }
}

// Returns the correct value to load into the buffer
int loadBuffer() {
  unsigned int result = 0;
  unsigned int num = 0;
  unsigned int pos = t;
  double freq = 40;
  unsigned int leftshift = 0;
  unsigned int rightshift = 0;

  for (int i = 0; i < COLS; ++i) {
      for (int j = 0; j < ROWS; ++j) {
      if (keyPressed[j][i]) {
      //freq = frequencyChart[(i*8) + j];
      freq = frequencyChart[i + (j*8)];
      /*
      leftshift = analogRead(0);
      rightshift = analogRead(1);

      Serial.print("leftshift: ");
      Serial.println(leftshift);
      Serial.print("rightshift: ");
      Serial.print(rightshift);

      if (leftshift > 350 && leftshift < 610) {
            freq = freq / ((0.0125 * (double) leftshift) + 1);
      } else if (rightshift > 350 && rightshift < 610) {
            freq = freq * ((0.0125 * (double) rightshift) + 1);
      }
      */
      //Serial.println((i*8) + j);
      //Serial.print("leftshift: ");
      //Serial.println(leftshift);
      //Serial.print("rightshift: ");
      //Serial.print(rightshift);

      pos = freq * t;
      pos = pos & 0xFFF;

      result += storedWave[pos];  //library here
      num++;
      }
      }
  }

  // Normalize the wave. Consider optimizing. Division is very expensive.
  if (num) {
      //result = result >> 3;
      result = result / num;
  }
  return result;
}

// Generate sine wave and place values in storedWave
void generateSine() {
   // Generate sine signals
  for (int i = 0; i < SAMPLES; i++) {
      storedWave[i] = 127 + (127 * sin((6.28*i)/SAMPLES));
      //Serial.println(storedWave[i]);
  }
```

```
}

// Generate sawtooth wave and place values in storedWave
void generateSawtooth() {
  for (int i = 0; i < SAMPLES; i++) {
      storedWave[i] = (i * 256) / SAMPLES;
  }
}

// Generate square wave and place values in storedWave
void generateSquare() {
  for (int i = 0; i < SAMPLES; i++) {
      if (i < (SAMPLES / 2)) {
      storedWave[i] = 255;
      } else {
      storedWave[i] = 0;
      }
  }
}

// Generate triangle wave and place values in storedWave
void generateTriangle() {
  for (int i = 0; i < SAMPLES; i++) {
      if (i < (SAMPLES/2)) {
      storedWave[i] = (i * 256 * 2) / SAMPLES;
      } else {
      storedWave[i] = 255 - (((i - (SAMPLES / 2)) * 256 * 2) / SAMPLES);
      }
  }
}

void incrementedScan() {
   // Scan the keys once every 10 loops
  count++;
  if (count >= 10) {
      count = 0;
  }
  if (!count) {
      scanKeys();
  }
}

void loop() {

  if (Serial3.available()) { // if there is bluetooth data, update command
      while(Serial3.available()) { // While there is more to be read, keep
reading.
      command = Serial3.read();
      }
  }
  if (command == 'm') { //MIDI OPERATION
      MIDIOp();
  } else if (command == 'i')  { //INDEPENDENT OPERATION
      sineWave();
  } else if (command == 'q') {
      squareWave();
  } else if (command == 'a') {
      sawtoothWave();
  } else if (command == 't') {
      triangleWave();
  } else {
      sineWave();
  }

}
```

```
// Interrupt code for timer 2
ISR(TIMER2_COMPA_vect){
  PORTB = buf;
  t++;
}
```