Report Appendix: Guitar Buddy

ECE 445, Fall 2018
Project Number 15
TA: Channing Philbrick

Austin Born and Chris Horn
December 13, 2018

A. Additional Diagrams

A.1. Block Diagram

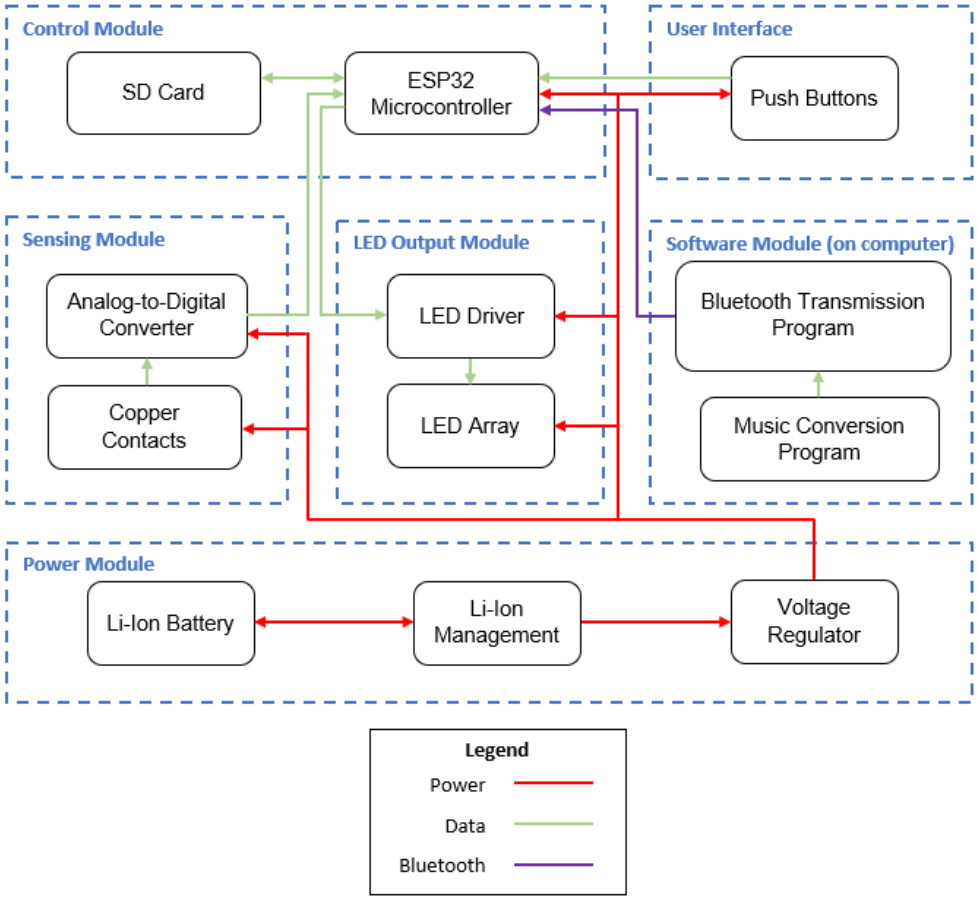


Figure A.1: Guitar Buddy System Block Diagram from the original design document.

A.2. Guitar Body Picture

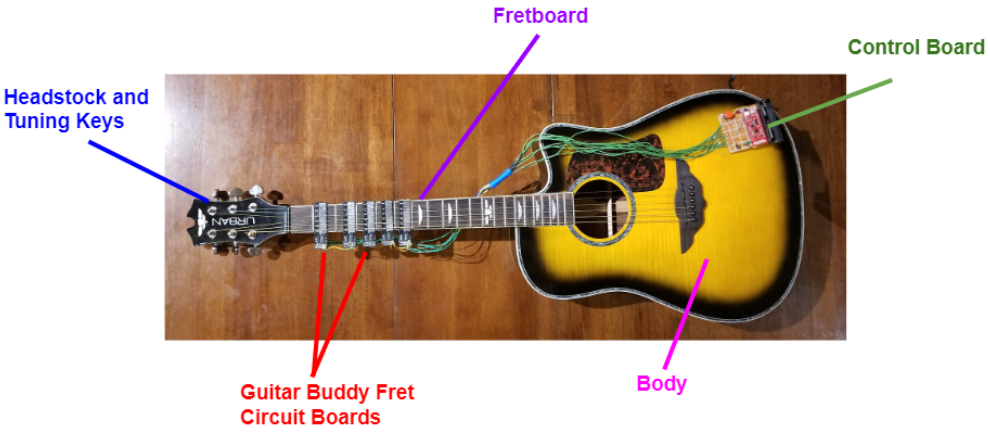


Figure A.2: Diagram of Guitar Buddy with control board and fret PCBs attached.

A.3. Model Picture

Figure A.3: Model of Austin Born holding the guitar and integrated Guitar Buddy system.

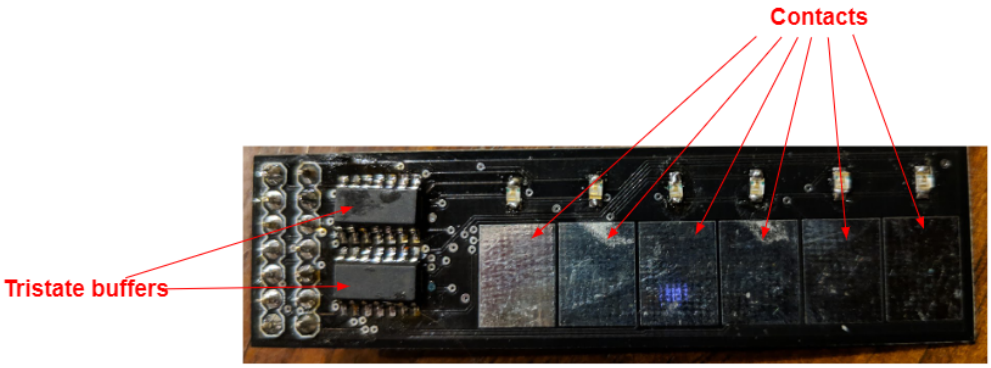


Figure A.4: Diagram of the PCB for each fret with core components labeled.

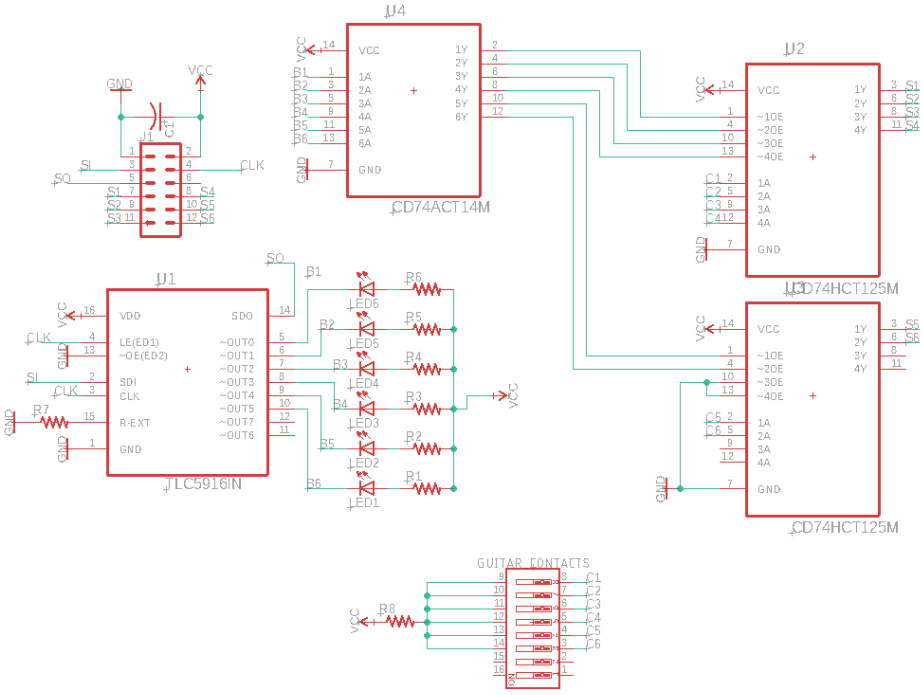


Figure A.5: Schematic of LED driver and array (one per fret).

A.4. Software Module

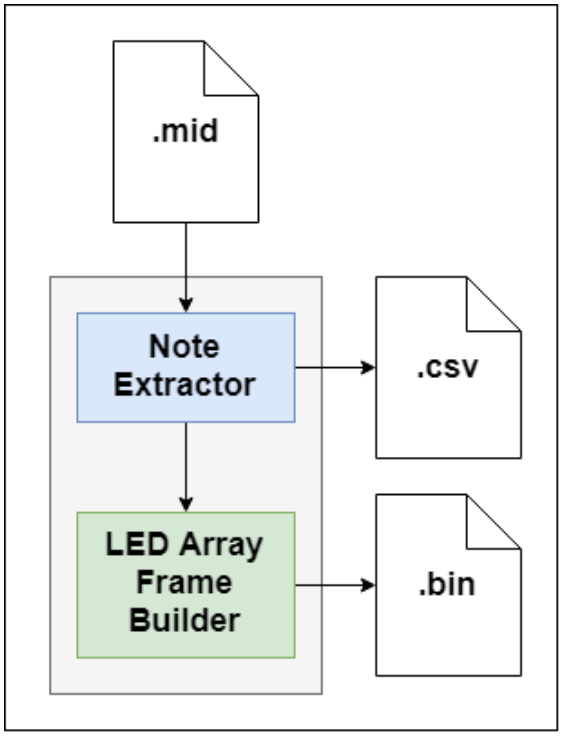


Figure A.6: Diagram illustrating the main mechanism of the software module.

A.5. Guitar Fret Notes map

	Strings						
Frets		E2	A2	D3	G3	B3	E4
	1	F2	B^b2	E^b3	A^b3	C4	F4
	2	G^b2	B2	E3	A3	D^b4	G^b4
	3	G2	C3	F3	B^b3	D4	G4
	4	A^b2	D^b3	G^b3	B3	E^b4	A^b4
	5	A2	D3	G3	C4	E4	A4
	6	B^b2	E^b3	A^b3	D^b4	F4	B^b4
	7	B2	E3	A3	D4	G^b4	B4
	8	C3	F3	B^b3	E^b4	G4	C5
	9	D^b3	G^b3	B3	E4	A^b4	D^b5
	10	D3	G3	C4	F4	A4	D5

Figure A.7: Map of notes on the fret board PCBs.

A.6. ESP32 Firmware Flowchart

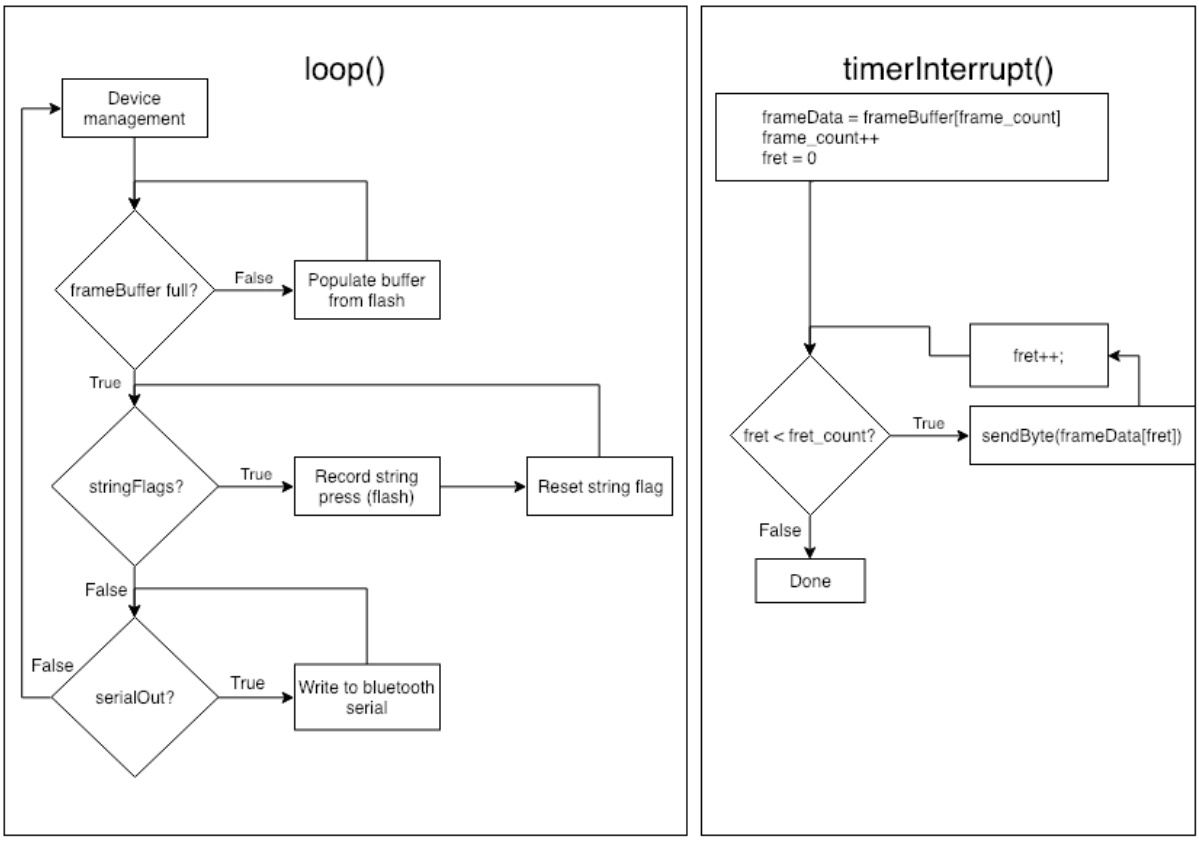


Figure A.8: High-level flowchart for firmware running on the control module (some features excluded).

A.7. Control Board

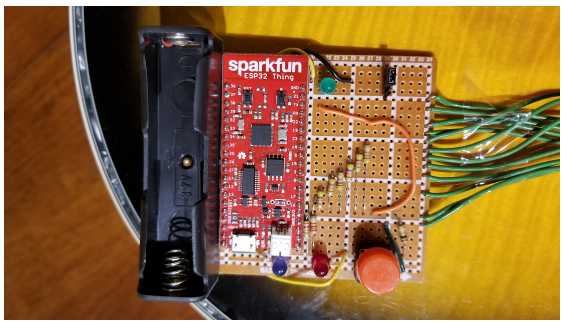
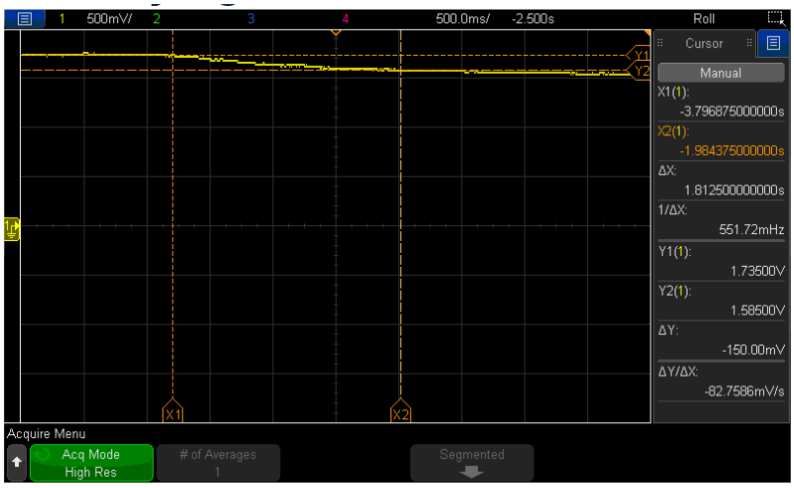


Figure A.9: Picture of the control board used on the guitar.



$$R_{therm_{RT}} = 10512\ \Omega$$
$$R_{cutoff} = 8850\ \Omega$$
$$R_{pullup} = 9754\ \Omega$$
$$V_{cutoff} = \frac{R_{cutoff}}{R_{pullup}+R_{cutoff}} \approx 1.58V$$

Figure A.10: Diagram of the verification of voltage ranges for the thermistor.

B. Requirements and Verification Table

Microcontroller Requirements and Verification	
Requirement	**Verification**
1. Powered by input voltage of 3.3 V ± 0.1 V.	1. a) Power controller with variable voltage source, starting at V_{in} = 3.2 V. b) Upload code setting all output pins to high output. c) Probe each output voltage, ensuring V_{out} = 3.3 V ± 0.1 V. d) Upload song data through Bluetooth connection and ensure the microcontroller doesn't crash. e) Set input voltage to 3.4 V and repeat b) through d).
2. Operating current $I_{max} \leq$ 500 mA at 3.3 V ± 0.1 V during radio transmission.	2. a) Power controller with 3.3 V ± 0.1 V, attaching an ammeter in series with supply. b) Connect variable voltage source to input pin 1 on controller at 0 V and ground to the GND pin. c) Execute program with constant wireless transmission outputting pin 1 to terminal. d) Alternate variable voltage source between digital low and digital high, and confirm that controller is sending wireless signal. e) Ensure $I_{max} \leq$ 500 mA.

Continued on next page

Microcontroller Requirements and Verification (Continued)	
Requirement	**Verification**
3. Operating current $I_{typical}$ ≤ 200 mA at 3.3 V ± 0.1 V during non-radio operation.	3. a) Power controller with 3.3 V ± 0.1 V, attaching an ammeter in series with supply. b) Execute regular, non-communication program. c) Ensure I_{max} ≤ 200 mA.
4. R/W compatibility with external SD storage.	4. a) Power controller with 3.3 V ± 0.1 V. b) Insert SD card into the SD card reader. c) Execute SD card test program to write to and read from SD card. d) Output data from SD card to terminal to verify that data is stored on SD card.
5. At least 10 GPIO pins for communication with other modules.	5. a) Power controller with 3.3 V ± 0.1 V. b) Connect variable voltage source to first GPIO pin on controller at 0 V and ground to the GND pin. c) Alternate variable voltage source between 0 V and 3 V, and confirm that controller is receiving signal from pin. d) Repeat previous step for at least 9 other GPIO pins on the controller. e) Disconnect variable voltage source from pins and connect a voltmeter to pin 1 and the ground to GND pin. f) Run pin output program to verify that the pin outputs 0 V when low and 3.3 ± 0.1 V when high. g) Repeat previous step for the other 9(+) GPIO pins.
Continued on next page	

Microcontroller Requirements and Verification (Continued)	
Requirement	**Verification**
6. At least 112.5 kB of SDRAM.	6. a) Power controller with 3.3 V ± 0.1 V. b) Write data to at least 112.5 kB of on chip RAM c) Verify that all data was written and is accessible.

Flash Storage Requirements and Verification	
Requirement	**Verification**
1. 3 MB or larger storage capacity.	1. a) Connect the flash storage to the microcontroller and power the system with 3.3 V ± 0.1 V. b) Write data to at least a 3MB portion of the flash storage. c) Read the written data to ensure that all data is still accessible.
2. R/W speeds of at least 512 kB/s.	2. a) Connect the flash storage to the microcontroller and power the system with 3.3 V ± 0.1 V. b) Write data to at least a 1MB portion of the flash storage. Record the transfer time and ensure that the average write speed is at least 512 kB/s. c) Read the written data. Record the transfer time and ensure that the average read speed is at least 512 kB/s.

Push Button Requirements and Verification	
Requirement	**Verification**
1. Durable and reliable operation; >1000 click life span.	1. a) Mount the switch to a convenient platform (such as a breadboard). b) Time the duration it takes to press the button 100 times. c) Continually press the button for 15 times as long as it took to depress it 100 times. d) Repeat for two other switches. e) Connect all three switches to 5V on one input, and a pulldown resistor to ground on the other. f) Press each switch and measure the voltage of the output of the switch to ensure it is connecting to 5V.

LED Driver Requirements and Verification	
Requirement	**Verification**
1. At least 60 bits of storage	1. a) Connect the LED driver sub module to the microcontroller submodule. Power the system with 3.3 V ± 0.1 V. b) Shift in 60 bits of digital low through the shift registers. Verify that all outputs are digital low. c) Shift in 60 bits of digital high through the shift registers. Verify that all outputs are digital high.
Continued on next page	

LED Driver Requirements and Verification (Continued)	
Requirement	**Verification**
2. Minimum $f_{clock_{max}}$ of 3.2 kHz	2. a) Connect the LED driver sub module to the microcontroller submodule. Power the system with 3.3 V ± 0.1 V. b) Shift in 60 bits of alternating digital high and low signals at a frequency of at least 3.2 kHz. c) Verify that all outputs match the expected value.
3. Supply a minimum I_{max} of 50 mA ± 1 mA per channel while maintaining a temperature below 50°C	3. a) Connect the LED driver sub module to the microcontroller submodule. Connect a LED of the LED array to the first output of the LED driver. Power the system with 3.3 V ± 0.1 V. b) Set the current output of the chip to 50 mA ± 1 mA by using the current trim input on the constant current shift register (specific to IC) c) Verify with ammeter that output current is 50 mA ± 1 mA. d) Allow to run for at 5 minutes. e) Verify the output current is still 50 mA ± 1 mA. f) Verify the temperature of the IC is below 50°C.
Continued on next page	

LED Driver Requirements and Verification (Continued)	
Requirement	**Verification**
4. Minimum I_{max} of 400 mA ± 8 mA per chip	4. a) Connect the LED driver sub module to the microcontroller submodule. Connect one LED of the LED array to each output of the LED driver (for 8 total). Power the system with 3.3 V ± 0.1 V. b) Set the current output of the chip to 50 mA per channel by using the current trim input on the constant current shift register (specific to IC). c) Verify the total output current is at least 400 mA ± 4 mA. d) Allow to run for at 5 minutes. e) Verify the total output current is still 400 mA ± 4 mA. f) Verify the temperature of the IC is below 50°C.

LED Array Requirements and Verification	
Requirement	**Verification**
1. LEDs are easily visible from at least 1 m away without being uncomfortablely bright	1. a) Connect a LED to an output of the LED driver module. Power the system with 3.3 V ± 0.1 V. b) Power the LED with no more than 50 mA. c) Check the LED is visible from 1 m ± 10 cm.
2. LEDs are less than 2 mm tall (surface mount)	2. a) Measure the height of the LED with a caliper. Ensure that it is less than 2mm.

Lithium-Ion Battery Requirements and Verification	
Requirement	**Verification**
1. Peak I_{out} must be at least 1.5 A continuously for 1 minute while maintaining a temperature under 60°C	1. a) Connect the battery submodule to a load that draws at least 1.5 A. b) Verify that output current is at least 1.5 A. c) Allow to run for 1 minute. d) Verify that the output current is still at least 1.5 A, and that the battery temperature is under 60°C.
2. Minimum 1500 mAh capacity	2. a) Connect the module to a 25 Ω load and an ammeter b) Allow the battery to drain until battery module cuts power (due to low voltage). c) Calculate the capacity of the battery and verify that it is at least 1500 mAh
3. Weight must be under 250 g	3. a) Weigh the battery; confirm the battery weights under 250 g
4. Battery must be under 10 cm along its largest axis, and less than 2 cm thick.	4. a) Use a mechanical caliper to measure the longest axis. Ensure that it is less than 10 cm b) Use a mechanical caliper to measure the diameter of the battery. Ensure that it is less than 2 cm.

Lithium-Ion Management Requirements and Verification	
Requirement	**Verification**
1. Protect the battery from over discharging by disconnecting the battery when the battery voltage dips under 3.0 V ± 0.05V.	1. a) Fully charge the battery, and then connect it to the battery management system. b) Connect the battery to a voltmeter. c) Connect the module to a 10 Ω ± 1 Ω load. d) Allow to discharge until the battery management system disconnects the battery and stops outputting power. Ensure the battery voltage does not drop below 2.95 V for any period of time.
2. Battery management must not exceed 80 °C under maximum load (note: this temperature applies to the battery management IC and any components for the management submodule. The battery itself has a lower maximum temperature, specified in the battery submodule.)	2. a) Fully charge the battery, and connect it to the battery management submodule. b) Apply a load to the system that results in a 1.5 A ± 0.1 A current draw. c) Allow to run for 1 minute. d) Measure the temperature of the battery management system and ensure that it is under 80°C. e) Apply a new load to the system that results in a 0.5 A ± 0.05 A current draw. f) Allow to run for 30 minutes. g) Measure the temperature of the battery management system and ensure that it is under 80°C.
Continued on next page	

Lithium-Ion Management Requirements and Verification (Continued)	
Requirement	**Verification**
3. Battery management system must cut power if temperature of battery exceeds 60°C.	3. a) Remove the thermistor from the battery housing. b) Connect a charged battery to the battery management system. c) Apply a 100 Ω± 10 Ωload across the output of the battery management system. d) Use an ammeter to confirm that there is current flowing through the load. e) Using a heat gun and a thermometer, heat the thermistor to 60°C to simulate a warming battery. Do not do this while the thermistor is still attached/next to the battery to avoid unnecessarily putting the battery at risk. f) When the thermistor reaches 60°C ± 1°C, use the ammeter to confirm that the battery management system cut power from the battery.
4. Battery management system must cut power if current draw exceeds 2.5 A ± 0.1 A for 0.5 s ± 0.5 s.	4. a) Connect a charged battery to the battery management system. b) Probe the output of the battery management system with a voltmeter. c) Connect with outputs of the battery management system with a high power 1 Ω± 0.1 Ω load. Start a timer. d) Verify the battery management system cuts power to the output within 0.5 s ± 0.5 s.

Music Conversion Program Requirements and Verification	
Requirement	**Verification**
1. Generate the proper bytecode and parity bits	1. a) Using the conversion program, generate the bytecode for a sample song. b) Using a USB connection (or Bluetooth connection if it has been independently confirmed to work) transfer the bytecode the guitar. c) Visual compare the LED indicated chord pattern against tabs for the same sample song. d) Confirm that the displayed chords match.
2. Capable of converting 5 minutes worth of song notes within 30 seconds	2. a) Pick any 5 minute ± 15 s music video for YouTube. b) Use the conversion program to generate the bytecode. Time how long the software takes to run. c) Repeat steps a) and b) for 4 other songs.

Bluetooth Transmission Program Requirements and Verification	
Requirement	**Verification**
1. Verify and transmit bytecode song data to ESP32 with 95% accuracy, and that any failed transmissions are resent.	1. a) Use the conversion software to generate the bytecode for 30 five minute songs. b) Turn on the guitar and place it 2 m ± 20cm away. c) Transmit the bytecode data to the ESP32. d) Log any failed transmissions. e) Clear the flash storage on the ESP32, and repeat step a)-d) four more times. f) The total number of failed transmissions can not exceed 95%. Any failed transmissions must also be reattempted automatically.
Continued on next page	

Bluetooth Transmission Program Requirements and Verification (Continued)	
Requirement	**Verification**
2. Transmission rate of at least 500 kbps at 2 m distance	2. a) Turn on the guitar and place it 2 m ± 20 cm away. b) Transmit 2 MB worth of song data. Record the time it takes. c) Ensure the average data transmission rate is at least 500 kbps.
3. Adjust settings on the guitar with ≤ 1.5 s latency while laptop is 5 m away.	3. a) Turn on the guitar and place it 2 m ± 20 cm away. b) Connect the guitar via Bluetooth to the transmission program. c) Change a setting by using the transmission program. d) Use a software timer to measure the time between input the setting and receiving the confirmation packet from the guitar. e) Verify the latency is ≤ 1.5 s.

Copper Contacts Requirements and Verification	
Requirement	**Verification**
Continued on next page	

Copper Contacts Requirements and Verification (Continued)	
Requirement	**Verification**
1. The copper contacts must endure the equivalent of at least 1,000 hours of play time (without more than 20% increase in the resistance.	1. a) Connect one of the 6 guitar strings of different gauges and a copper contact to the input and output respectively of an ammeter. b) Sample and record the resistance of the string-contact circuit with at least 10 different locations of contact between string and copper contact. c) Simulate 1,000 hours of play time by rubbing the string against the contact rigorously for at least 15 minutes. d) Once again, sample and record the resistance of the string-contact circuit with at least 10 different locations of contact between string and copper contact. e) Confirm that the maximum resistance after the rubbing procedure is no more than 20% greater than the maximum resistance beforehand. f) Repeat this full procedure for each of the other 5 different gauges of guitar string.
2. Closed circuit current must be below 1 mA.	2. a) Connect a battery to the system and turn it on. b) Temporary insert an ammeter between the pull down/current limiting resistor between the guitar strings and ground. c) Close the circuit by depressing the guitar string onto any copper contact pad. d) Ensure that the current is less than 1 mA. e) Repeat for each of the 6 different guitar string gauges.
Continued on next page	

Copper Contacts Requirements and Verification (Continued)	
Requirement	**Verification**
3. Voltage of the copper contact must equalize to <0.2 V within 1 ms of firm contact with the guitar string.	3. a) Connect a battery to the system and turn it on. b) Attach on probe of an oscilloscope to the ground and another to the first bit data bus. c) Shift in a high bit to the shift register on the top fret's top E string. This will connect the top fret's left most copper contact to the data bus. d) Depress the string and measure the time it takes for the data bus voltage to drop to ≤ 0.2 V. Ensure that it is less than 1 ms. e) Repeat for each of the 6 different guitar string gauges.
4. Contact must be disconnected from string data bus through the tristate buffer when the corresponding LED is turned off.	4. a) Connect a battery to the system and turn it on. b) Turn on an LED on the top fret by shifting in a single high bit. c) Verify that the corresponding string bus's voltage is digital high. d) Depress the string corresponding with turned on LED. e) Verify that the corresponding string bus voltage is digital low. f) Repeat for each of the 6 different guitar string gauges.
Continued on next page	

Copper Contacts Requirements and Verification (Continued)	
Requirement	**Verification**
5. Resistance between copper contact and guitar strings contact point must contribute less than 10 Ω to resistance.	5. a) Connect a string and copper contact in series to an ammeter. b) Firmly depress the string against the contact. c) Ensure that the net resistance between 1 cm away from the contact point on the string and the opposite end of the copper contact is less than 10 Ω. d) Repeat for each of the 6 different guitar string gauges.

Analog-to-Digital Converter Requirements and Verification	
Requirement	**Verification**
1. Powered by 3.3 V ± 0.1 V	1. a) Connect the V_{cc} of the ADC submodule to a variable voltage source. b) Set the V_{cc} to 3.2 V. c) Connect an input on the ADC to ground, and verify the binary output from the ADC is 0 V ± 0.1 V. d) Connect an input on the ADC to 3.2 V, and verify the binary output from the ADC is 3.2 V ± 0.1 V. e) Set the V_{cc} to 3.4 V. f) Connect an input on the ADC to ground, and verify the binary output from the ADC is 0 V ± 0.1 V. g) Connect an input on the ADC to 3.4 V, and verify the binary output from the ADC is 3.4 V ± 0.1 V.

Continued on next page

Analog-to-Digital Converter Requirements and Verification (Continued)	
Requirement	**Verification**
2. ADC obtains precision of 0.1 V of the V_{in} within 1 ms.	2. a) Power ADC with a variable voltage source at V_{in} = 3.3 V. b) Connect ADC input to function generator with a square-wave function set to 250 Hz frequency with V_{low} = 0 V and V_{high} = 3 V, and PWM set to 50%. c) Determine the ADC binary output for 0.03 V and 3 V from the ADC. d) If the binary output for 0.03 V is n bits long, find the most significant bit less than the nth-least significant bit of the 3 V binary output which is 1. Determine the ADC pin that corresponds to this bit. e) Connect the function generator and the determined ADC pin to an oscilloscope. f) Confirm that the pin output switches high within 1 ms of the rising edge of the square wave.
3. Minimum 8 GPIO pins (including 6 allocated to ADC)	3. a) Power ADC with a variable voltage source at V_{in} = 3.3 V. b) Connect ADC input to function generator with a saw-tooth function set to 0.5 Hz frequency with V_{low} = 0 V and V_{high} = 3 V. c) Connect the function generator to an oscilloscope. d) For each of at least 8 GPIO pins, connect the pin to the oscilloscope and confirm that the pin switches between low and high at a regular interval. If the pin appears to remain high, steadily increase the function frequency and observe if the pin simply switches too frequently to be seen at 0.5 Hz.

Requirements and Verification Point Assignments		
Module	**High-Level Requirement**	**Points**
Control Module	• Microcontroller must manage wireless communication, read song data from flash storage, and generate control signals. • The microcontroller must be able to play 5-minute songs over the course of at least 2 hours of play time.	15
User Interface Module	• Must provide reliable interface buttons for the user to maneuver the device's settings.	5
LED Output Module	• LED array must be able to hold 60 bits of lighting information and refresh within 25 ms.	10
Power Supply Module	• Power supply must be powerful enough to keep the device running continuously for at least 2 hours. • Battery management system must keep battery within safe operating range.	5
Software Module	• Music conversion program must take online input (for example as MIDI files) and convert it to a format to be transmitted to the device. • Transmission program must send packets of data to device while ensuring data is not lost in transmission.	5

Continued on next page

Requirements and Verification Point Assignments (Continued)		
Module	**High-Level Requirement**	**Points**
Sensing Module	• Copper contacts should provide robust sensing points throughout the duration of the device's lifespan, without significant degradation. • Analog-to-Digital Converter should be precise enough to read the proper voltage changes.	10

C. Core Programs

C.1. MIDIToBytes.cpp for MIDI Conversion

```
/* MIDI to CSV and Streamable Binary format
 *
 * By Austin Born, Fall 2018
 *
 * C++ program to convert notes in MIDI files to a readable CSV and a
 * compressed byte map format to send to the ESP32.
 * The byte map will contain basic header information on song name,
 * tempo, and then a sequence of frames for the entire LED array.
 * Each byte in a frame represents one fret of the guitar, so a
 * single frame will have n bytes of data where n is the number of frets.
 * If there are roughly 32 frames per second, then the byte map for a
 * 5-minute song will be ~100 kB.
 */

/* About the MIDI Format:
 * Format:
 * <Header Chunk> = <MThd><length><format><ntrks><division>
 * <Track Chunk> = <MTrk><length><MTrk event>+...
 * <MTrk event> = <delta-time><event>
 * <event> = <MIDI event> | <sysex event> | <meta-event>
*/

//Include external libraries
#include <unistd.h>
#include <Windows.h>
#include <fstream>
#include <iostream>
#include <queue>
#include <string>
#include <ctime>
#include <cmath>
#include <sstream>
#include <map>
#include <iomanip>
#include "MIDIToBytes.h"

using namespace std;

//Initialize note and octave lists
string notes [12] = {"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"};
string octaves [11] = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};

//Initialize note map for binary format
map<string, int> note_map = {//Notes below lowest fret remapped
                             {"C2", 25}, {"C3", 25},
                             {"C#2", 33}, {"C#3", 33},
                             {"D2", 41}, {"D3", 41},
                             {"D#2", 49}, {"D#3", 49},
                             {"E2", 18}, {"E3", 18},
```

```
                            {"F2", 8}, {"F3", 26},
                            {"F#2", 16},
                            {"G2", 24},
                            {"G#2", 32},
                            {"A2", 40},
                            {"A#2", 48},
                            {"B2", 17},

                            //Frets 2-7
                            {"F#3", 16}, {"B3", 17}, {"E4", 18}, {"A4", 19}, {"C#5", 20}, {"F#5", 21},
                            {"G3", 24}, {"C4", 25}, {"F4", 26}, {"A#4", 27}, {"D5", 28}, {"G5", 29},
                            {"G#3", 32}, {"C#4", 33}, {"F#4", 34}, {"B4", 35}, {"D#5", 36}, {"G#5", 37},
                            {"A3", 40}, {"D4", 41}, {"G4", 42}, {"C5", 43}, {"E5", 44}, {"A5", 45},
                            {"A#3", 48}, {"D#4", 49}, {"G#4", 50},/*C#5 above*/{"F5", 52}, {"A#5", 53},

                            //Notes above highest fret remapped
                            {"B5", 35},
                            {"C6", 43},
                            {"C#6", 20}};

//Other constants
const int BUFFER_SIZE = 4;
const int byte_frame_freq = 32;

int main(int argc, char** argv){

    //Input checking
    if(argc != 3){
        cout << "Incorrect number of arguments. Requires 2 arguments: <Song name> <note channel>" << endl;
        return 0;
    }

    int channel_num = strtol(argv[2], NULL, 10);
    if(channel_num < 0 || channel_num > 16){
        cout << "Given note channel is outside total number of channels." << endl;
        return 0;
    }

    //Initializations
    char buf [BUFFER_SIZE];
    int i = 0;

    //Start clock
    clock_t begin = clock();

    //Open MIDI file
    fstream infile;
    string infile_name = "MIDI_Files/" + string(argv[1]) + ".mid";
```

```
98      if (FILE *file = fopen(infile_name.c_str(), "r")) {
99          fclose(file);
100     } else {
101         cout << "File does not exist" << endl;
102         return false;
103     }
104     infile.open(infile_name, fstream::in | ios::binary);
105 
106     //Create output file
107     fstream outfile;
108     string outfile_name = "CSV_Files/" + string(argv[1]) + ".csv";
109     remove(outfile_name.c_str());
110     outfile.open(outfile_name, fstream::in | fstream::out | fstream::trunc);
111 
112     //Get MIDI file length
113     infile.seekg(0, infile.end);
114     long file_length = infile.tellg();
115     long bytes_left = file_length;
116     infile.seekg(0, infile.beg);
117 
118     //Get "MThd", but do nothing with it
119     readFromFile(infile, buf, 4, bytes_left);
120 
121     //Get length of Header file
122     readFromFile(infile, buf, 4, bytes_left);
123     long length = buf[0];
124     for (i = 0; i < 4; i++)
125         length = (length << 8) + (unsigned char)buf[i];
126 
127     //Get format of Header file
128     readFromFile(infile, buf, 2, bytes_left);
129     short format;
130     for (i = 0; i < 2; i++)
131         format = (format << 8) + (unsigned char)buf[i];
132     outfile << "File format - " << (unsigned int)format << endl;
133 
134     //Get number of tracks
135     readFromFile(infile, buf, 2, bytes_left);
136     short ntrks;
137     for (i = 0; i < 2; i++)
138         ntrks = (ntrks << 8) + (unsigned char)buf[i];
139     outfile << "# of Tracks - " << ntrks << endl;
140 
141 
142     //Initialize Time Variables
143     bool tpq_timing = false;
144     long fps, tpf, tpq, tempo;
145     double time_multiplier = 0;
146     short division;
147 
148     //Get time division
149     readFromFile(infile, buf, 2, bytes_left);
150     for (i = 0; i < 2; i++)
151         division = (division << 8) + (unsigned char)buf[i];
```

```
152
153     //If division bit 15 = 1, division[14:8] is negative fps, division[7:0] is ticks per
         frame
154     if(division & 0x8000){
155         fps = -(int)(division & 0x7F00);
156         tpf = division & 0x00FF;
157         outfile << "Frames/second - " << fps << endl;
158         outfile << "Ticks/frame - " << tpf << endl;
159     }
160     //Else, division[14:0] is ticks/quarter note
161     else{
162         tpq_timing = true;
163         tpq = division & 0x7FFF;
164         outfile << "Ticks/quarter note - " << tpq << endl;
165     }
166
167     //Initialize track variables
168     long trk_length;
169     long trk_bytes_left;
170     float total_time;
171     int round_time;
172     int track_num = 0;
173     unsigned char prev_status;
174
175     //Loop through each track
176     while(bytes_left > 0){
177
178         //Increment track number
179         track_num += 1;
180         outfile << endl << "Start of track " <<track_num << " at Byte " << file_length -
          bytes_left << endl;
181
182         //Get "MTrk" but do nothing with it
183         readFromFile(infile, buf, 4, bytes_left);
184
185         //Get length of track
186         trk_length = 0;
187         readFromFile(infile, buf, 4, bytes_left);
188         for (int i = 0; i < 4; i++)
189             trk_length = (trk_length << 8) + (buf[i] & 0x00FF);
190         outfile << "Track length: " << trk_length << endl;
191
192         //Initialize MIDI Event variables
193         std::queue<long> delta_time_q;
194         bool vlq_left;
195         long long delta_time = 0;
196         long vlq_byte;
197
198         //Loop through each MIDI Event
199         trk_bytes_left = trk_length;
200         total_time = 0;
201         round_time = 0;
202         unsigned char status;
203         bool using_previous;
```

```
204
205         while(trk_bytes_left > 0){
206
207             //Get delta time of MIDI event
208             vlq_left = true;
209
210             //Push variable length quantity to queue
211             while(vlq_left){
212                 readFromFile(infile, buf, 1, bytes_left);
213                 trk_bytes_left -= 1;
214                 delta_time_q.push(buf[0]);
215                 if (!(buf[0] & 0x80))
216                     vlq_left = false;
217             }
218
219             //Initialize delta_time to 0
220             delta_time = 0;
221
222             //For byte in variable length quantity, add to total delta_time value
223             while(!delta_time_q.empty()){
224                 vlq_byte = delta_time_q.front();
225                 delta_time_q.pop();
226                 delta_time = (delta_time << 7) | (vlq_byte & 0x7F);
227             }
228             double delta_float = (double)delta_time;
229             delta_float *= time_multiplier;
230
231             total_time += delta_float;
232             round_time = round(total_time);
233
234             //Peek status bytes for MIDI event
235             status = peekFromFile(infile);
236
237             //Prepare status loop boolean
238             using_previous = false;
239
240             do{
241                 //Parse Status
242                 if ((status >> 4) == 0x8) { //Note off event
243                     if(!using_previous){
244                         readFromFile(infile, buf, 1, bytes_left);
245                         trk_bytes_left -= 1;
246                     }
247
248                     //Get Note number (and skip velocity)
249                     readFromFile(infile, buf, 2, bytes_left);
250                     trk_bytes_left -= 2;
251                     unsigned char note_num = buf[0];
252
253                     //Record in CSV
254                     outfile << round_time << ",Off," << noteFinder(note_num) << "," << (
    status & 0xF) + 1 << endl;
255                     break;
256                 }
```

```
257                 else if ((status >> 4) == 0x9) { //Note on event
258                     if(!using_previous){
259                         readFromFile(infile, buf, 1, bytes_left);
260                         trk_bytes_left -= 1;
261                     }
262 
263                     //Get Note number, use velocity to tell if on or off
264                     readFromFile(infile, buf, 2, bytes_left);
265                     trk_bytes_left -= 2;
266                     unsigned char note_num = buf[0];
267 
268                     //Record in CSV
269                     if (buf[1] != 0x00)
270                         outfile << round_time << ",On," << noteFinder(note_num) << ","
    << (status & 0xF) + 1 << endl;
271                     else
272                         outfile << round_time << ",Off," << noteFinder(note_num) << ","
    << (status & 0xF) + 1 << endl;
273                     break;
274                 }
275 
276                 //Other unimportant MIDI events
277                 else if ((status >> 4) == 0xA){ //Polyphonic key pressure
278                     if(!using_previous){
279                         readFromFile(infile, buf, 1, bytes_left);
280                         trk_bytes_left -= 1;
281                     }
282                     readFromFile(infile, buf, 2, bytes_left);
283                     trk_bytes_left -= 2;
284                     outfile << round_time << ", Polyphonic key pressure event" << ",
    Channel: " << (status & 0xF) + 1 << endl;
285                     break;
286                 }
287                 else if ((status >> 4) == 0xB){ //Control Change
288                     if(!using_previous){
289                         readFromFile(infile, buf, 1, bytes_left);
290                         trk_bytes_left -= 1;
291                     }
292                     readFromFile(infile, buf, 2, bytes_left);
293                     trk_bytes_left -= 2;
294                     outfile << round_time << ", Control change event" << ", Channel: "
    << (status & 0xF) + 1 << endl;
295                     break;
296                 }
297                 else if ((status >> 4) == 0xC){ //Program Change
298                     if(!using_previous){
299                         readFromFile(infile, buf, 1, bytes_left);
300                         trk_bytes_left -= 1;
301                     }
302                     readFromFile(infile, buf, 1, bytes_left);
303                     trk_bytes_left -= 1;
304                     outfile << round_time << ", Program change event" << ", Channel: "
    << (status & 0xF) + 1 << endl;
305                     break;
```

```
306                 }
307                 else if ((status >> 4) == 0xD){ //Channel Pressure
308                     if(!using_previous){
309                         readFromFile(infile, buf, 1, bytes_left);
310                         trk_bytes_left -= 1;
311                     }
312                     readFromFile(infile, buf, 1, bytes_left);
313                     trk_bytes_left -= 1;
314                     outfile << round_time << ", Channel Pressure event" << ", Channel: "
    << (status & 0xF) + 1 << endl;
315                     break;
316                 }
317                 else if ((status >> 4) == 0xE){ //Pitch Wheel Change
318                     if(!using_previous){
319                         readFromFile(infile, buf, 1, bytes_left);
320                         trk_bytes_left -= 1;
321                     }
322                     readFromFile(infile, buf, 2, bytes_left);
323                     trk_bytes_left -= 2;
324                     outfile << round_time << ", Pitch wheel event" << ", Channel: " << (
    status & 0xF) + 1 << endl;
325                     //outfile << "status:" << (status & 0xFF) << endl;
326                     break;
327                 }
328                 else if (status == 0xF0){ //System Exclusive
329                     if(!using_previous){
330                         readFromFile(infile, buf, 1, bytes_left);
331                         trk_bytes_left -= 1;
332                     }
333                     readFromFile(infile, buf, 1, bytes_left);
334                     trk_bytes_left -= 1;
335                     outfile << round_time << ", System Exclusive event" << endl;
336                     break;
337                 }
338                 else if (status == 0xF2){ //Song Position Pointer
339                     if(!using_previous){
340                         readFromFile(infile, buf, 1, bytes_left);
341                         trk_bytes_left -= 1;
342                     }
343                     readFromFile(infile, buf, 2, bytes_left);
344                     trk_bytes_left -= 2;
345                     outfile << round_time << ", Song position pointer" << endl;
346                     break;
347                 }
348                 else if (status == 0xF3){ //Song Select
349                     if(!using_previous){
350                         readFromFile(infile, buf, 1, bytes_left);
351                         trk_bytes_left -= 1;
352                     }
353                     readFromFile(infile, buf, 1, bytes_left);
354                     trk_bytes_left -= 1;
355                     outfile << round_time << ", Song select event" << endl;
356                     break;
357                 }
```

```
358                 else if (status == 0xF6){ //Tune Request
359                     if(!using_previous){
360                         readFromFile(infile, buf, 1, bytes_left);
361                         trk_bytes_left -= 1;
362                     }
363                     outfile << round_time << ", Tune request" << endl;
364                     break;
365                 }
366                 else if (status == 0xF7){ //End of Exclusive
367                     if(!using_previous){
368                         readFromFile(infile, buf, 1, bytes_left);
369                         trk_bytes_left -= 1;
370                     }
371                     outfile << round_time << ", End of exclusive" << endl;
372                     break;
373                 }
374                 else if (status == 0xF8){ //Timing Clock
375                     if(!using_previous){
376                         readFromFile(infile, buf, 1, bytes_left);
377                         trk_bytes_left -= 1;
378                     }
379                     outfile << round_time << ", Timing clock" << endl;
380                     break;
381                 }
382                 else if (status == 0xFA){ //Start
383                     if(!using_previous){
384                         readFromFile(infile, buf, 1, bytes_left);
385                         trk_bytes_left -= 1;
386                     }
387                     outfile << round_time << ", Start" << endl;
388                     break;
389                 }
390                 else if (status == 0xFB){ //Continue
391                     if(!using_previous){
392                         readFromFile(infile, buf, 1, bytes_left);
393                         trk_bytes_left -= 1;
394                     }
395                     outfile << round_time << ", Continue" << endl;
396                     break;
397                 }
398                 else if (status == 0xFC){ //Stop
399                     if(!using_previous){
400                         readFromFile(infile, buf, 1, bytes_left);
401                         trk_bytes_left -= 1;
402                     }
403                     outfile << round_time << ", Stop" << endl;
404                     break;
405                 }
406                 else if (status == 0xFE){ //Active sensing
407                     if(!using_previous){
408                         readFromFile(infile, buf, 1, bytes_left);
409                         trk_bytes_left -= 1;
410                     }
411                     outfile << round_time << ", Active sensing" << endl;
```

```
412                     break;
413                 }
414                 else if (status == 0xFF){ //Reset (escape for meta events)
415                     if(!using_previous){
416                         readFromFile(infile, buf, 1, bytes_left);
417                         trk_bytes_left -= 1;
418                     }
419 
420                     //Get meta event type
421                     readFromFile(infile, buf, 1, bytes_left);
422                     trk_bytes_left -= 1;
423                     char meta_event_type = buf[0];
424 
425                     //Get meta event length
426                     readFromFile(infile, buf, 1, bytes_left);
427                     trk_bytes_left -= 1;
428                     char length = buf[0];
429 
430                     if (meta_event_type == 0x00){ //Sequence Number
431                         readFromFile(infile, buf, length, bytes_left);
432                         trk_bytes_left -= length;
433                         outfile << round_time << ", Sequence number event" << endl;
434                     }
435                     else if (meta_event_type == 0x01){ //Text Event
436                         readFromFile(infile, buf, length, bytes_left);
437                         trk_bytes_left -= length;
438                         outfile << round_time << ", Text event" << endl;
439                         outfile << "Length of text: " << (int)length << endl;
440                     }
441                     else if (meta_event_type == 0x02){ //Copyright Notice
442                         readFromFile(infile, buf, length, bytes_left);
443                         trk_bytes_left -= length;
444                         outfile << round_time << ", Copyright event" << endl;
445                     }
446                     else if (meta_event_type == 0x03){ //Sequence/Track Name
447                         readFromFile(infile, buf, length, bytes_left);
448                         trk_bytes_left -= length;
449                         outfile << round_time << ", Sequence/Track Name event" << endl;
450                     }
451                     else if (meta_event_type == 0x04){ //Instrument Name
452                         readFromFile(infile, buf, length, bytes_left);
453                         trk_bytes_left -= length;
454                         outfile << round_time << ", Instrument name event" << endl;
455                     }
456                     else if (meta_event_type == 0x05){ //Lyric
457                         readFromFile(infile, buf, length, bytes_left);
458                         trk_bytes_left -= length;
459                         outfile << round_time << ", Lyrics event" << endl;
460                     }
461                     else if (meta_event_type == 0x06){ //Text  Marker
462                         readFromFile(infile, buf, length, bytes_left);
463                         trk_bytes_left -= length;
464                         outfile << round_time << ", Text Marker" << endl;
465                     }
```

```
466                     else if (meta_event_type == 0x07){ //Cue Point
467                         readFromFile(infile, buf, length, bytes_left);
468                         trk_bytes_left -= length;
469                         outfile << round_time << ", Cue point" << endl;
470                     }
471                     else if (meta_event_type == 0x20){ //MIDI Channel Prefix
472                         readFromFile(infile, buf, length, bytes_left);
473                         trk_bytes_left -= length;
474                         outfile << round_time << ", MIDI Channel Prefix" << endl;
475                     }
476                     else if (meta_event_type == 0x21){ //MIDI Channel Prefix
477                         readFromFile(infile, buf, length, bytes_left);
478                         trk_bytes_left -= length;
479                         outfile << round_time << ", MIDI Port" << endl;
480                     }
481                     else if (meta_event_type == 0x2F){ //End of Track
482                         readFromFile(infile, buf, length, bytes_left);
483                         trk_bytes_left -= length;
484                         outfile << round_time << ",End of track" << endl;
485                     }
486                     else if (meta_event_type == 0x51){ //Set Tempo
487                         readFromFile(infile, buf, length, bytes_left);
488                         trk_bytes_left -= length;
489                         tempo = 0;
490                         for (i = 0; i < length; i++){
491                             tempo = (tempo << 8) | (0x00FF & buf[i]);
492                         }
493                         time_multiplier = tempo*byte_frame_freq*0.000001/tpq;
494                         outfile << round_time << ", Tempo to " << (long)tempo << " usec/
    quarter note" << endl;
495                     }
496                     else if (meta_event_type == 0x54){ //SMPTE Offset
497                         readFromFile(infile, buf, length, bytes_left);
498                         trk_bytes_left -= length;
499                         outfile << round_time << ", SMPTE event" << endl;
500                     }
501                     else if (meta_event_type == 0x58){ //Time Signature
502                         readFromFile(infile, buf, length, bytes_left);
503                         trk_bytes_left -= length;
504                         outfile << round_time << ", Time Signature event" << endl;
505                     }
506                     else if (meta_event_type == 0x59){ //Key Signature
507                         readFromFile(infile, buf, length, bytes_left);
508                         trk_bytes_left -= length;
509                         outfile << round_time << ", Key signature event" << endl;
510                     }
511                     else if (meta_event_type == 0x7F){ //Sequencer Specific Meta-Event
512                         readFromFile(infile, buf, length, bytes_left);
513                         trk_bytes_left -= length;
514                         outfile << round_time << ", Sequencer-specific event" << endl;
515                     }
516                     break;
517                 }
518                 else{
```

```
519                 //Use previous status byte as status
520                 status = prev_status;
521                 using_previous = true;
522                 }
523 
524             }while(using_previous);
525 
526             //Update previous status bytes
527             prev_status = status;
528         }
529     }
530 
531     //Open new binary file
532     string binfile_name = "BIN_Files/" + string(argv[1]) + ".bin";
533     remove(binfile_name.c_str());
534     ofstream binfile(binfile_name, ios::binary);
535 
536     //C array frame number for Chris' use
537     int final_frame_num = 0;
538 
539     //Prepare byte_map
540     int MAP_BYTES = 5;
541     char byte_map[MAP_BYTES];
542     for(int i = 0; i < MAP_BYTES; i++)
543         byte_map[i] = 0x00;
544 
545     //Convert CSV to Binary file
546     string str_in;
547     char * pch;
548     int cur_frame, last_frame = 0;
549     bool found_channel = false;
550     vector<string> str_vec;
551 
552     //Open CSV file and start from beginning
553     outfile.clear();
554     outfile.seekg(0, ios::beg);
555 
556     //Loop through lines in CSV
557     while(getline(outfile, str_in)){
558         char cstr[str_in.size()+1];
559         strcpy(cstr, str_in.c_str());
560         pch = strtok(cstr,",");
561         while(pch != NULL){
562             str_vec.push_back(pch);
563             pch = strtok(NULL, ",");
564         }
565 
566         //If line has 4 comma-separated values, it's a note
567         if(str_vec.size() == 4){
568             if(str_vec[3] == argv[2]){
569                 found_channel = true;
570                 stringstream frame_num(str_vec[0]);
571                 frame_num >> cur_frame;
572                 if(cur_frame != last_frame){
```

```
573
574                     //Print byte_map (cur_frame - last_frame) times
575                     for(int a = 0; a < (cur_frame - last_frame); a++)
576                         for(int i = 0; i < MAP_BYTES; i++)
577                             binfile.write(byte_map + i, 1);
578                     last_frame = cur_frame;
579                 }
580                 //Adjust byte_map based on str_vec[2]
581                 int byte_map_num = note_map[str_vec[2]];
582                 unsigned char fret_bits = 0x80 >> (byte_map_num % 8);
583                 int fret = (byte_map_num / 8) - 2;
584                 if(str_vec[1] == "On")
585                     byte_map[fret] |= fret_bits;
586                 else
587                     byte_map[fret] &= ~fret_bits;
588             }
589         }
590         //If end of track, exit
591         else if(found_channel)
592             if(str_vec.size() == 2)
593                 if(str_vec[1] == "End of track")
594                     break;
595         while(!str_vec.empty())
596             str_vec.pop_back();
597     }
598
599     //Close files
600     infile.close();
601     outfile.close();
602     binfile.close();
603
604     //C array of file for Chris' use
605     char file_bytes[(last_frame)*MAP_BYTES];
606     fstream binfile2;
607     string binfile2_name = "BIN_Files/" + string(argv[1]) + ".bin";
608     binfile2.open(binfile2_name, fstream::in | ios::binary);
609     for(int i = 0; i < (last_frame)*MAP_BYTES; i++)
610         binfile2.read(&(file_bytes[i]), 1);
611     unsigned char file_bytes_2d[MAP_BYTES][last_frame];
612     for(int i = 0; i < (last_frame)*MAP_BYTES; i++){
613         file_bytes_2d[i%5][i/5] = (const char)file_bytes[i];
614         //cout << std::hex << (int)file_bytes_2d[i/5][i%5] << " ";
615     }
616     binfile2.close();
617
618     //Copy frame array to .txt for debugging
619     fstream songfile;
620     string songfile_name = string(argv[1]) + ".txt";
621     songfile.open(songfile_name,  fstream::in | fstream::out | fstream::trunc);
622     songfile << "{{" << charToString(file_bytes_2d[0][0]);
623     for(int j = 0; j < last_frame; j++)
624         songfile << "," << charToString(file_bytes_2d[0][j]);
625     songfile << "}";
626     for(int i = 0; i < MAP_BYTES; i++){
```

```
627         songfile << ",{" << charToString(file_bytes_2d[i][0]);
628         for(int j = 0; j < last_frame; j++)
629             songfile << "," << charToString(file_bytes_2d[i][j]);
630         songfile << "}";
631         }
632     songfile << "}";
633     songfile.close();
634 
635 
636     //Stop clock
637     clock_t end = clock();
638     cout << std::fixed << "Total elapsed time: " <<  int(end - begin) << " ms" << endl;
639     return 0;
640 }
641 
642 //Helper function to read a number of bytes from MIDI file
643 void readFromFile(std::fstream& infile, char * bytebuf, int length, long &bytes_left){
644     for(int i = 0; i < length; i++)
645         infile.read(&(bytebuf[i % BUFFER_SIZE]), 1);
646     bytes_left -= length;
647 }
648 
649 //Helper function to peek next 2 bytes from MIDI file (only used for peeking status
        bytes)
650 unsigned char peekFromFile(std::fstream& infile){
651         return infile.peek();
652 }
653 
654 //Helper function to map proper note and octave
655 std::string noteFinder(int note_num){
656     string this_note = notes[note_num % 12];
657     this_note += octaves[(note_num / 12)];
658     return this_note;
659 }
660 
661 //Used to debugging array copied to .txt
662 std::string charToString(unsigned char chari){
663     int charint = (int)chari;
664     std::string hex;
665     int big = chari/16;
666     if(big < 10)
667         hex += "0x" + to_string(big);
668     else if(big == 10)
669         hex += "0xa";
670     else if(big == 11)
671         hex += "0xb";
672     else if(big == 12)
673         hex += "0xc";
674     else if(big == 13)
675         hex += "0xd";
676     else if(big == 14)
677         hex += "0xe";
678     else if(big == 15)
679         hex += "0xf";
```

```
680
681     int little = chari%16;
682     if(little < 10)
683         hex += to_string(little);
684     else if(little == 10)
685         hex += "a";
686     else if(little == 11)
687         hex += "b";
688     else if(little == 12)
689         hex += "c";
690     else if(little == 13)
691         hex += "d";
692     else if(little == 14)
693         hex += "e";
694     else if(little == 15)
695         hex += "f";
696     return hex;
697 }
```

Listing 1: MIDI to CSV, and CSV to Binary converter.

```
#Import Modules
import sys
import serial
import os
import time
from array import array

 # Serial Preparations
ser = serial.Serial()
ser.baudrate = 57600
ser.port = 'COM7'#COM10 no work
ser.open()

# Initialize song array
song = array('B')

# Open binary file
file_name = 'BIN_Files/'+str(sys.argv[1])+'.bin'
file_size = os.path.getsize(file_name)

# Send file size over Bluetooth
count = 0
size_1 = file_size & int('0xff00',16)
size_1 >>= 8
size_1 = size_1.to_bytes(1, 'big')
size_2 = file_size & int('0x00ff',16)
size_2 = size_2.to_bytes(1, 'big')
print(size_1)
print(size_2)
ser.write(size_1)
ser.write(size_2)
time.sleep(0.5)

# Send song bytes one at a time
with open(file_name, 'rb') as file:
    for i in range(file_size):
        count += 1
        byte = file.read(1)
        if byte != "":
            ser.write(byte)

# Print bytes sent
print("sent "+str(count)+" bytes")
```

Listing 2: Bluetooth transmission of binary file to ESP32 (laptop end).

```
/*
 * ESP32 Bluetooth Controller Program
 *
 * By Austin Born, Fall 2018
 *
 * Boilerplate code is in the Public Domain, by Evandro Copercini, 2018
 *
 * The code creates a bridge betweeon Serial and Classical Bluetooth (SPP),
 * and shows the functionality of SerialBT.
 */

//Libraries and variable definitions
#include "BluetoothSerial.h"
#include <stdio.h>

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

#define BOARDCOUNT 5
#define BUFFERDEPTH 15000

#define CLK 21
#define SDI 22
#define LE 19
#define EN 14

//Variable initializations
int i = 0;
long frame = 0;
int board = 0;
int unavail = 0;
int line = 0;
unsigned char hex[4];
int bytes_left = 0;
long file_size = 0;
int file_size_2 = 0;
char frameBuffer[BUFFERDEPTH][BOARDCOUNT];
long frame_count = 0;
bool song_loaded = false;
BluetoothSerial SerialBT;

// updateFrame() helper function to update current LED frame
void updateFrame(){
      for(int fret = 0; fret < BOARDCOUNT; fret++){
      sendByte(frameBuffer[frame_count][fret]);
    }
    load();
    frame_count++;
}

//Shift-high helper
```

```
53 void sh() {
54   digitalWrite(SDI, HIGH);
55   shift();
56   digitalWrite(SDI, LOW);
57
58 }
59
60 //Shift-low helper
61 void sl() {
62   digitalWrite(SDI, LOW);
63   shift();
64
65 }
66
67 // Function to send a byte of data to frets
68 void sendByte(byte data) {
69
70   //mask last two bits  since bits 7 and 8 are not used
71   char mask = 0xFC; // 0011 1111
72   data = mask & data;
73   //send data serially over SDI
74   //reverse order since first bit in is last bit
75   for (int i = 0; i < 8; i++) {
76     //shift 1000 0000 to right to change bit
77     mask = 0x01 << i;
78
79     //send data unmasked bit
80     if (data & mask) {
81       sh();
82     }
83     else {
84       sl();
85     }
86   }
87 }
88
89 // Shift helper
90 void shift() {
91   digitalWrite(CLK, HIGH);
92   digitalWrite(CLK, LOW);
93 }
94
95 // Load helper
96 void load() {
97   digitalWrite(LE, HIGH);
98
99   digitalWrite(LE, LOW);
100 }
101
102 // Testboards function for debugging
103 void testBoards(int numOfBoards) {
104   for (char i = 0x00; i != 0x40; i++) {
105     for (int j = 0; j < numOfBoards; j++) {
106       sendByte(i);
```

```
    }
    load();
    delay(50);
  }
  for (int k = 0; k < 4; k++) {
    for (char i = 0x01; i != 0x40; i = i << 1) {
      for (int j = 0; j < numOfBoards; j++) {
        sendByte(i);
      }
      load();
      delay(50);
    }
    for (char i = 0x40; i != 0x00; i = i >> 1) {
      for (int j = 0; j < numOfBoards; j++) {
        sendByte(i);
      }
      load();
      delay(50);
    }
  }
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < numOfBoards; j++) {
      sendByte(0xFF);
    }
    load();
    delay(300);
    for (int j = 0; j < numOfBoards; j++) {
      sendByte(0x00);
    }
    load();
    delay(300);
  }
}


// Initial setup
void setup() {

  // Prepare serial i/o
  SerialBT.begin("ESP32-GuitarBuddy"); //Bluetooth device name
  Serial.flush();
  Serial.begin(57600);

  // Initialize pins
  pinMode(CLK, OUTPUT);
  pinMode(SDI, OUTPUT);
  pinMode(LE, OUTPUT);
  pinMode(EN, OUTPUT);
  digitalWrite(EN, HIGH);

  // Wait to receive file size bytes over Bluetooth
  while(1){
    if(SerialBT.available()){
      file_size = (int)SerialBT.read();
```

```
      if(file_size == 0){
        delay(100);
        continue;
      }
      file_size <<= 8;
      delay(200);
      file_size_2 = (int)SerialBT.read();
      file_size += file_size_2;
      Serial.print(file_size);
      break;
      }
  }

  // After file size is received, input rest of byte data to frameBuffer
  bytes_left = file_size;
    while(bytes_left > 0){
      if(SerialBT.available()){
        frameBuffer[frame][board] = SerialBT.read();
        if(board == 4){
          frame++;
          board = 0;
        }
        else
          board++;
        bytes_left -= 1;
      }
    }

  // Print frameBuffer for debugging
  for(int f = 0; f < frame; f ++){
    Serial.print("[");
    for(int fr = 0; fr < BOARDCOUNT; fr++){
      Serial.print(frameBuffer[f][fr], BIN);
      Serial.print(", ");
    }
    Serial.println("]");
  }
}

// Loop through frameBuffer, sending each byte with a set delay
void loop() {
//  while(1)
//    testBoards(1);
  board = 0;
  frame = 0;
  bytes_left = file_size;
  frame_count = 0;
  while(bytes_left > 0){
    updateFrame();
    bytes_left -= BOARDCOUNT;
    delay(30);
  }
```

```
213 }
```

Listing 3: Program flashed to ESP32 for binary receiver, storage, and playback.