E155 Final Report: Stepper Motor Music

Members: Andrew Q. Pham & Sam Ting

Abstract

The team constructed a system that plays music on four stepper motors. Multiple brands of stepper motors were tested for volume and sound quality, and motor mounts were constructed to improve volume and sound quality. The system uses an ATSAM and ESP8266 WiFi module to allow users to request songs through a hosted webpage. The ATSAM parses the request, determines the notes to play from a lookup table, encodes the notes, and sends the encoded notes to the FPGA over SPI. The FPGA decodes the notes and steps the motors at the desired frequency to produce the notes. The final system allows the user to choose from nine distinct songs to play. Additionally, a Python script was developed by the team to help parse MIDI files and generate songs in the team's custom encoding scheme. Future work includes improving the song encoding scheme to produce better sound quality, making the MIDI parsing script more robust and incorporating it into the microcontroller, so a user could input any MIDI file of choice to play.

Introduction

Inspired by E155 Lab 5 and various online videos [1] [2], the team constructed a system that plays music on four stepper motors using the FPGA, ATSAM, and ESP8266 WiFi module.

The system allows users to select a song to play from a webpage accessed through a computer. The webpage is hosted on the ATSAM, and the computer and the ATSAM communicate through the ESP8266. Song requests are sent via UART from the ESP8266 to the ATSAM, which parses the request and sends the appropriate notes to the FPGA. The FPGA drives I/O pins, which are inputs to the stepper motor h-bridge drivers that drive the motors to play a note.

An overall block diagram of the system and the fully constructed system are shown in Figure 1 and Figure 2 respectively. The schematic for the full system is shown in Appendix B.

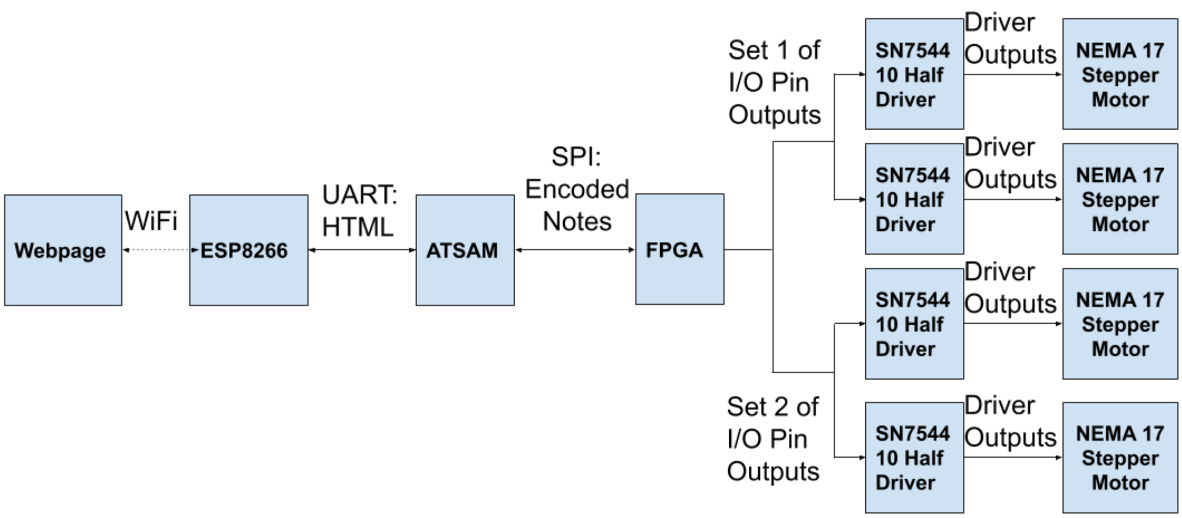


Figure 1. Overall block diagram of full system to play music based on user input from HTML page

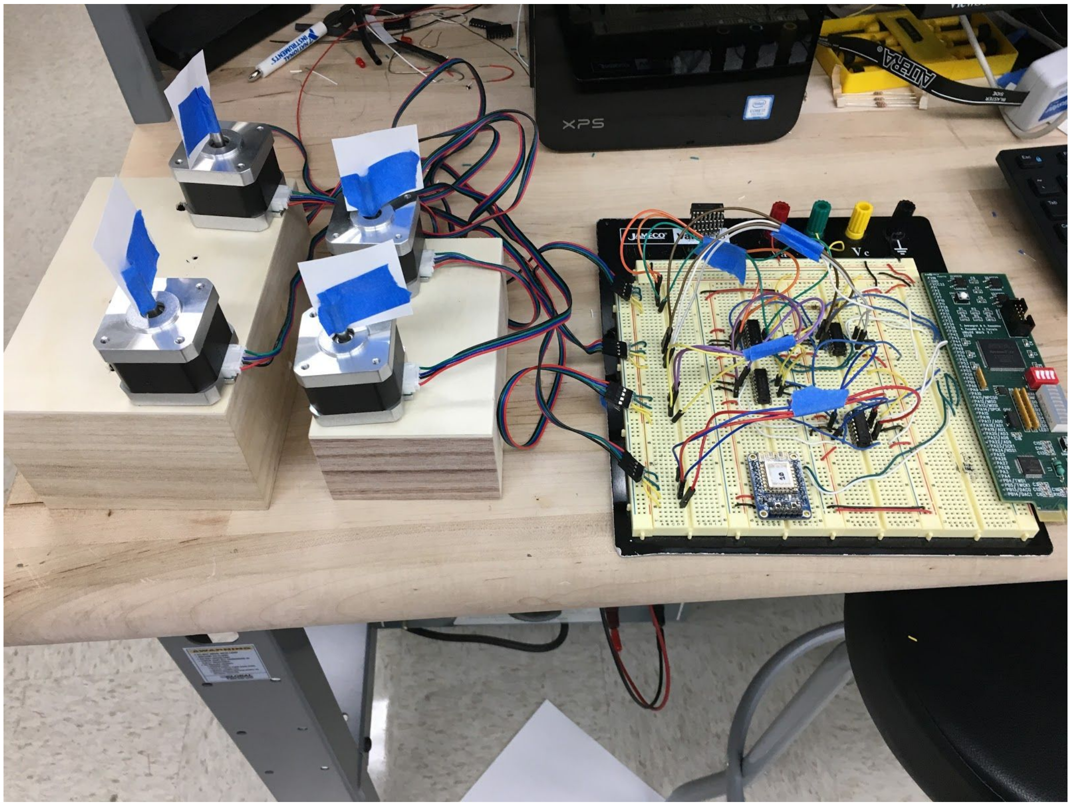


Figure 2. Full system hardware implementation

Design

Hardware

Motors & Drivers

The team scavenged for stepper motors across Mudd's campus and found five different brands/sizes of motors. The team evaluated each stepper motor for sound quality and required amperage. Two additional types of NEMA17 stepper motors were purchased from Amazon and tested for sound quality. The Usongshine NEMA17 stepper motors [3] were chosen, due to their superior sound quality in comparison to the other motors.

To spin the motors, the team constructed a motor driver circuit for the stepper motors using the SN754410 h-bridge driver. The maximum current rating of the SN754410 is 1 A, while the NEMA17 stepper motors are rated for up to 2 A. To compensate for this, two of the drivers were soldered together in parallel to drive each motor. The circuit schematic of a single motor and driver is shown in Appendix B (the second parallel SN754410 is omitted for diagram clarity).

Given that four motors are being driven, each at a maximum of 1 A continuous current and greater than or equal to 5 V of voltage, a larger power supply was necessary to power the setup. A 50 V, 25 A unregulated power supply was checked out from the stockroom to power the motor drivers. This new power supply is able to output significantly more current than the ones in the Digital Lab. The increase in current increased the output torque of the motors, which increased vibration and volume. The motors were run at ~5 V because at higher voltages the amount of constant current would cause the motor drivers to reach their thermal shutdown. The thermal shutdown would prevent the motors from stepping properly, causing them to vibrate at an undesired frequency. Moreover, the stacked motor drivers could only take a maximum of 2 amps of constant current, so a larger current would cause the drivers to explode.

The stepper motors' frequency ranges were tested by adjusting the slow clock addend value manually in Quartus. The result showed that note volume is significantly lower for notes lower than an E1. For notes higher than a D#4, the stepper motor was unable to step quickly enough to produce the note. Thus, all notes between an E1 and a D#4 were encoded on the ATSAM and were used to create songs. Unfortunately, the motors ordered off Amazon did not have an associated datasheet, so the team was unable to confirm whether this stepping frequency was on par with the expected maximum RPM of the motors.

Acoustic Hardware Improvements

During the problem presentation, the team presented motor volume as their primary technical problem. This problem was largely absolved by purchasing new motors. However, the motors had very irregular and choppy sound quality, depending on the surface they were fixed to. To solve this problem, two thin wooden boxes were purchased to mount the motors to. This gave the motors a stiff yet thin surface to reverberate against, which produced a much cleaner tone quality. Two boxes were purchased: one 4" x 6" x 3" and one 3" x 5" x 2".

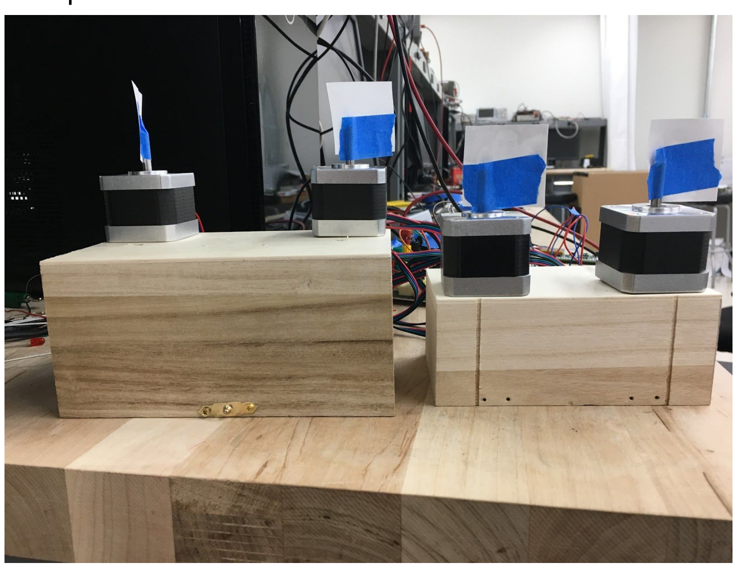


Figure 3. Photo of stepper motors mounted on large and small boxes

The motors were tested/mounted on several different locations across the top of the boxes to test which part of the box would cause the motor to be the loudest. The two box sizes were also tested to see which one produced a louder overall sound

Ultimately, experimental tests showed that the corners of each box caused the loudest sound, so each motor was mounted on a corner of their respective boxes. Moreover, the larger box produced a louder sound. Thus, the motors playing the melody of the song were mounted on the big box, while the background/bass motors were mounted on the smaller box.

FPGA

The FPGA performs two main tasks: 1) reading in encoded notes to play over SPI from the ATSAM and 2) decoding each encoded note to drive the stepper motors at the correct frequencies. This is depicted in the high-level block diagram in Figure 4.

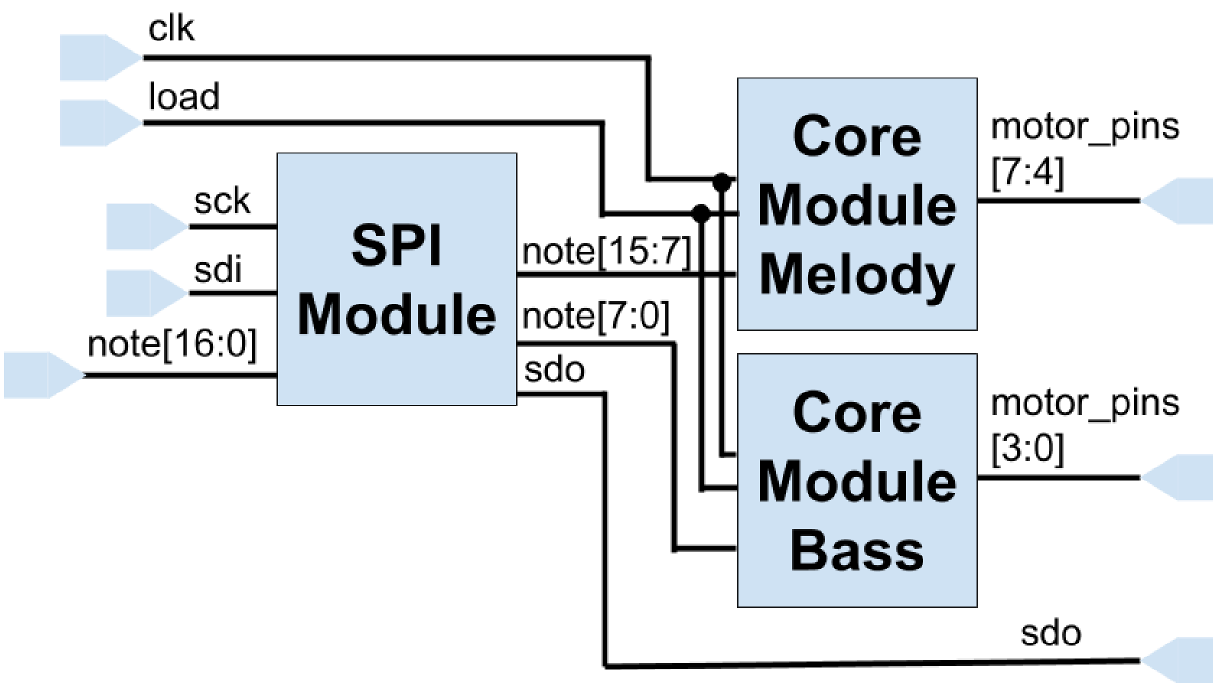


Figure 4. High-level block diagram of the digital hardware on the FPGA

SPI Communication Note Encoding

For each SPI interaction, the FPGA reads in an 8-bit encoded note. Bits 0-3 encode which note in an octave to play (ex. C), and bits 5-7 encode which octave the note is in. An example SPI interaction is shown in Figure 5. Two SPI interactions occur for each note. The first specifies the note to play on the first and second motors (the melody), and the second specifies the note to play on the third and fourth motor (the bass part). The octave decoding is discussed further in the description of the decode note module.

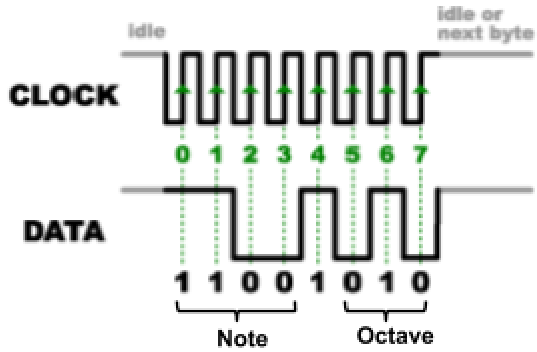


Figure 5. Example SPI Interaction

Project Core

The “Project Core” module in Verilog decodes the note and steps the motors at the desired frequency to play a note. The digital hardware of the core is shown in Figure 6. It is composed of three main parts: 1) a note decoding module, 2) a 2^N counter to reduce the clock frequency to the desired note frequency, and 3) a state machine to step the motors.

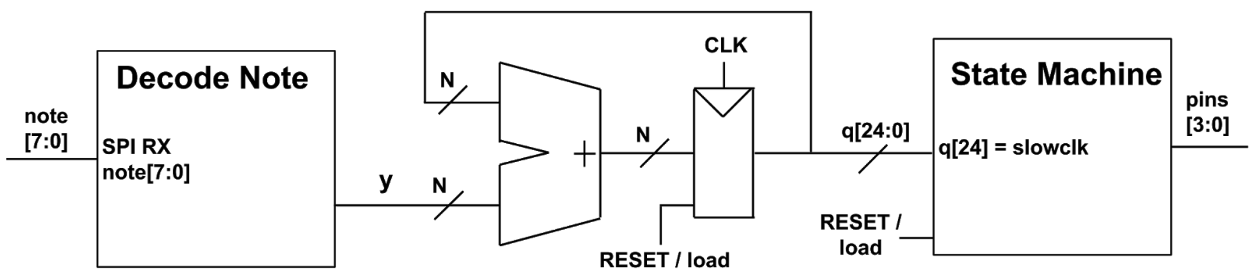


Figure 6. Digital hardware of the core module of the FPGA

Note Decoding

The decoder module assumes the note is in either octave 2 or 3, based on whether bit 7 of the note input is 0 or 1 respectively. The module then maps it to the number to be added to the 2^N counter to make the MSB of the counter oscillate at the desired note frequency. The spreadsheet of the mapping from note frequency to counter addend is given in Appendix C. The decode note module is implemented using a case statement, so the resulting digital hardware is a large chain of or gates and muxes to account for the 25 different input cases. The RTL viewer of the Decode Note Module is shown in Appendix D.

As described above, bit 7 determines which ‘base octave’ the note will be in; if bit 7 is zero, this indicates octave two -- otherwise the note is in octave three. These two octaves were not sufficient to cover the previously described range of the motor. Thus, bits 5 and 6 were utilized to either drop or raise the note an additional octave. After the decode note module, the addend for the counter is adjusted based on note[6] and note[5]; the former indicates that the addend should be multiplied by two, while the latter divides the addend by two. As a result, these flags either increase or drop the note by an octave respectively. This additional functionality allows any programmed songs to utilize the entire E1 to D#4 range of the motor.

Motor Stepping

A finite state machine (FSM) steps the motors. The state machine shown in Figure 7 is used to activate the different coils in the stepper motors in sequence to have the motor step/spin. The output logic, “pins,” drives GPIO output pins that facilitate the communication between the FPGA and the stepper motor drivers. The different states of the FSM correspond to the different inputs of the stepper motor drivers being activated/different coils in the motor being activated. The FSM transitions states on the MSB of the 2^N counter.

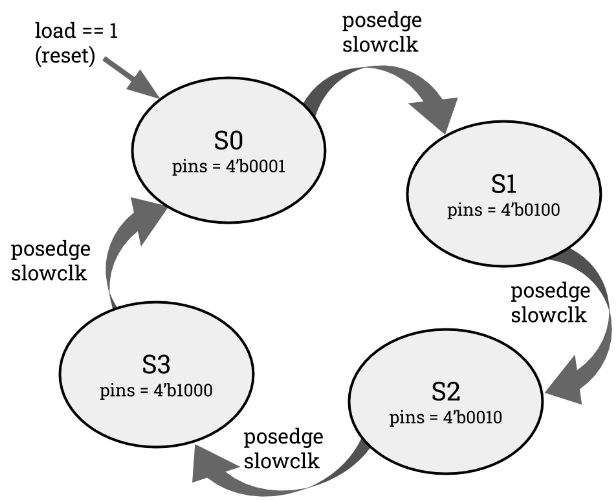


Figure 7. FSM to step the motors

As shown in Figure 7, this FSM only sets one output pin at a time high, based on its state. This is full-step driving, which drives the maximum current of all stepping methods. Half-stepping was also tested, but it was found that the motors were marginally quieter when half-stepping, so the team opted to use full-step operation.

In summary, the FPGA takes in an encoded note over SPI from the ATSAM, decodes the note to set the frequency at which the MSB of the 2^N counter oscillates, and then steps the stepper motors at that frequency using an FSM.

ATSAM

The ATSAM performs two main tasks: 1) hosting an HTML webpage for song requests and 2) transmitting the notes of the requested song to the FPGA via SPI. The high-level routine of the program is shown in Figure 8. The ATSAM hosts the webpage with an ESP8266, which generates a WiFi hotspot that a user computer can connect to to access the webpage. The

ATSAM parses the HTML song request to determine if a new song has been requested. More details on the HTML webpage and ESP8266 are given in the HTML Page/ESP8266 Section. After determining the song choice, the ATSAM reads the next note for each motor of the chosen song be played, encodes these notes, sends it over SPI to the FPGA, waits for the note's duration, and then repeats this process until all of the song's notes have been played or a different song is selected.

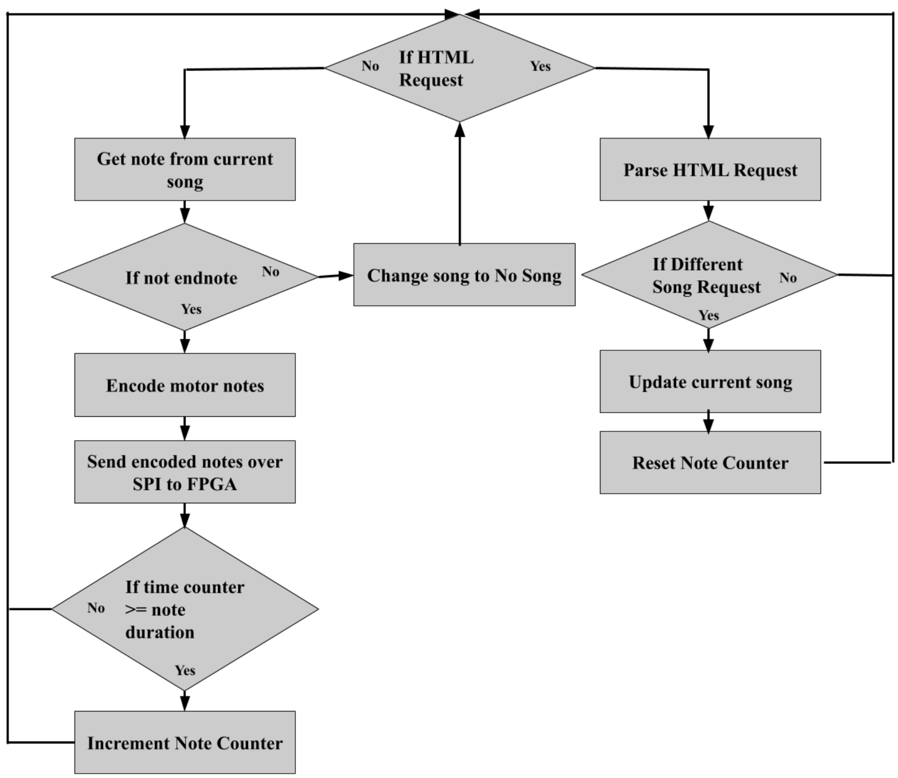


Figure 8. High Level Routines of ATSAM

Song Encoding

Nine songs were hard-coded in the C code, each of which could be played by pressing a button on the corresponding HTML web page. Songs were encoded with a custom data structure. This structure contains two data types: a float that stores the song's tempo and a two dimensional float array that stores the song's notes to be played on the motors. The array is an Nx2 array, where N is the number of notes in the song, and two is based on the number of distinct parts (melody and bass) of each song. Since the design contains four total motors, each part is played by two motors.

To ensure that the motors were in sync, the notes for each part of each song were divided into sixteenth note increments. For example, each half note in a given song was divided into eight consecutive sixteenth notes of the same frequency. By dividing longer notes into smaller repeated notes, note information for the motors was synchronously sent on every sixteenth note division. This prevented any timing issues associated with syncing the melody and bass parts of a song.

While this encoding method prevented timing issues, the motors would only play the equivalent number of sixteenth notes instead of a continuous quarter note or longer note. The song encoding could be improved in the future to achieve better sound quality on sustained notes.

HTML Page/ESP8266

An HTML webpage was designed to allow the user to request one of the encoded songs. An ESP8266 generates a WiFi hotspot that a user's computer can connect to. A user can then press a button on the web page to select a song. A screenshot of the webpage is shown in Figure 9.

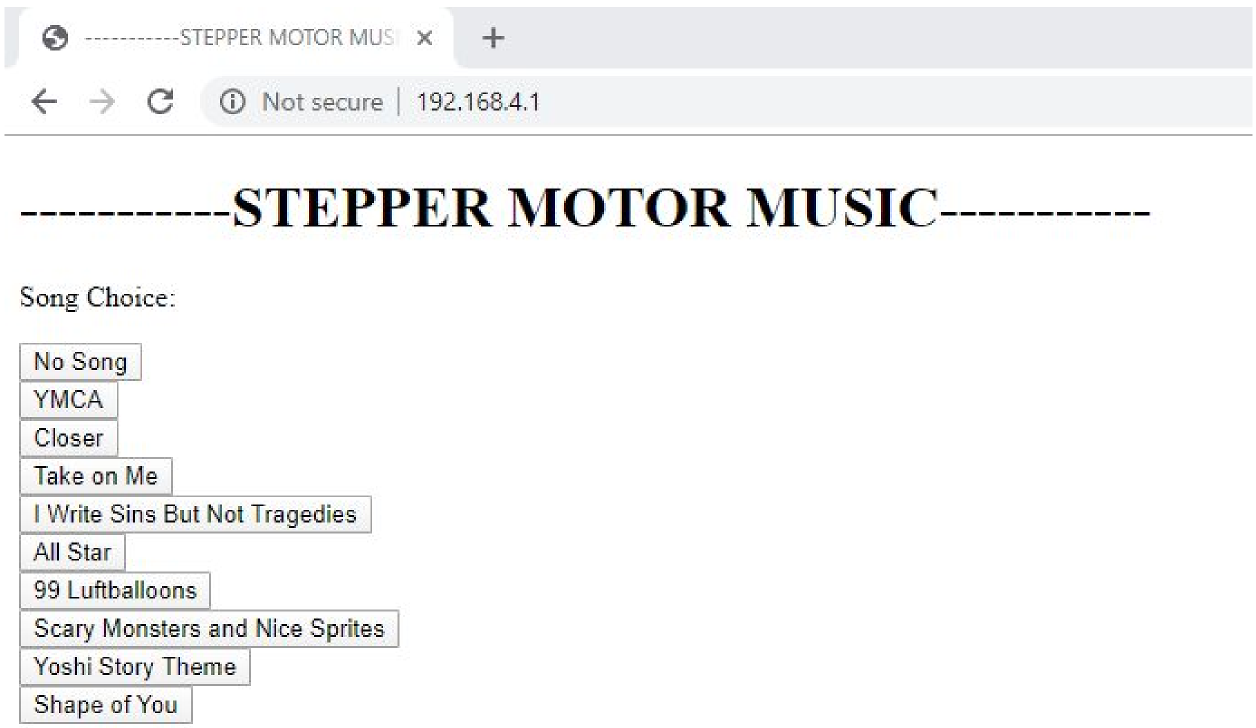


Figure 9. Screenshot of HTML webpage

The ESP8266 sends HTML requests to and from the ATSAM over UART. The implementation details of the UART communication and HTML webpage hosting can be found in the E155 Lab 6 Manual.

The user can choose one of the nine song options or no song to play. The user can also choose a different song to play in the middle of another song. If the user requests the same song again after the song has ended, the song will be repeated.

The HTML request communication had a default error-handling timeout. If the ATSAM was taking too long to respond to the ESP8266, then an error message would appear on the webpage. Due to this, note playing could not be blocking. In other words, the ATSAM could not wait for the duration of a note before parsing another UART response; otherwise, the ATSAM would timeout. The team solved this issue by making note playing non-blocking. The ATSAM samples the UART every one microsecond and counts the number of microseconds passed. If the number of microseconds passed if greater than or equal to the duration of the note, then the ATSAM increments the note counter and resets the microsecond count.

Results

Upon completion of the project, nine unique songs could be played on the stepper motors, based on user input from the HTML webpage. The songs were largely created by reading sheet music [4] [5] [6] [7] [8] and transcribing it into the previously described encoded format. Additional songs were created using the midi file parsing script in Appendix A [9].

A slideshow containing a sample selection of songs can be viewed at the following link: https://docs.google.com/presentation/d/1OD58IbyFZtqookncDBI-VVhnwSGlX3ZMpAzS-ybHats/edit?usp=sharing.

Conclusion/Future Work

The final system ultimately expanded upon the project proposal -- the final design could play nine distinct songs on four stepper motors instead of the originally proposed three and two, respectively. This arose due to additional time left in the last week to polish the project and create additional songs.

A Python script was developed by the team to help parse MIDI files and extract notes to create the note arrays for the song structures. The script was successful in parsing notes from a file; however, the team found that the MIDI file note layouts did not perfectly translate to a playable song on the system. MIDI files separate song melodies and basslines on different tracks within the file, so the parser had to be ran multiple times to extract the melody and bassline. From there, the two outputted files had to be manually edited and merged together to be playable. Future work could include making this script more robust and incorporating it to work on the microcontroller, so a user could input any midi file of choice to play.

Other future work includes making the hardware set up more robust. Alternative motor drivers could be purchased that could handle more current -- this would eliminate the need to solder multiple h-bridges together. Moreover, additional motors could be purchased to play additional harmonies in the programmed songs and increase the overall volume of the musical setup. Better wooden housing enclosures could also be built/tested to increase the volume of the motors.

References

[1] "Super Mario Theme - stepper motor music", *Youtube.com*, 2019. [Online]. Available: https://www.youtube.com/watch?v=0DBnGYMPaf4. [Accessed: 13- Dec- 2019].

[2] J. Kayne, ""Fireflies" (Owl City) - Played on 32 Stepper Motors", *Youtube.com*, 2019. [Online]. Available: https://www.youtube.com/watch?v=5YFajoHbdtM. [Accessed: 13- Dec- 2019].

[3 "Usongshine Nema 17 Stepper Motor", Amazon.com, 2019. [Online]. Available: https://www.amazon.com/Usongshine-nema-17-Stepper-Motor/dp/B07T8G9QQQ. [Accessed: 13- Dec- 209].

[4] “All Star Sheet Music,” *Musescore.com*. [Online]. Available: https://musescore.com/user/12898551/scores/2755271. [Accessed: 12-Dec-2019].

[5] “I Write Sins Not Tragedies String Quartet,” *Musescore.com*. [Online]. Available: https://musescore.com/user/178992/scores/171148. [Accessed: 12-Dec-2019].

[6] “Yoshi's Story Medley,” *Musescore.com*, 02-Apr-2018. [Online]. Available: https://musescore.com/cleverusernamedude/a-yoshis-story-medley. [Accessed: 12-Dec-2019].

[7] “Shape of You Sheet Music,” *Musescore.com*, 16-Jul-2018. [Online]. Available: https://musescore.com/user/3435661/scores/3798956. [Accessed: 12-Dec-2019].

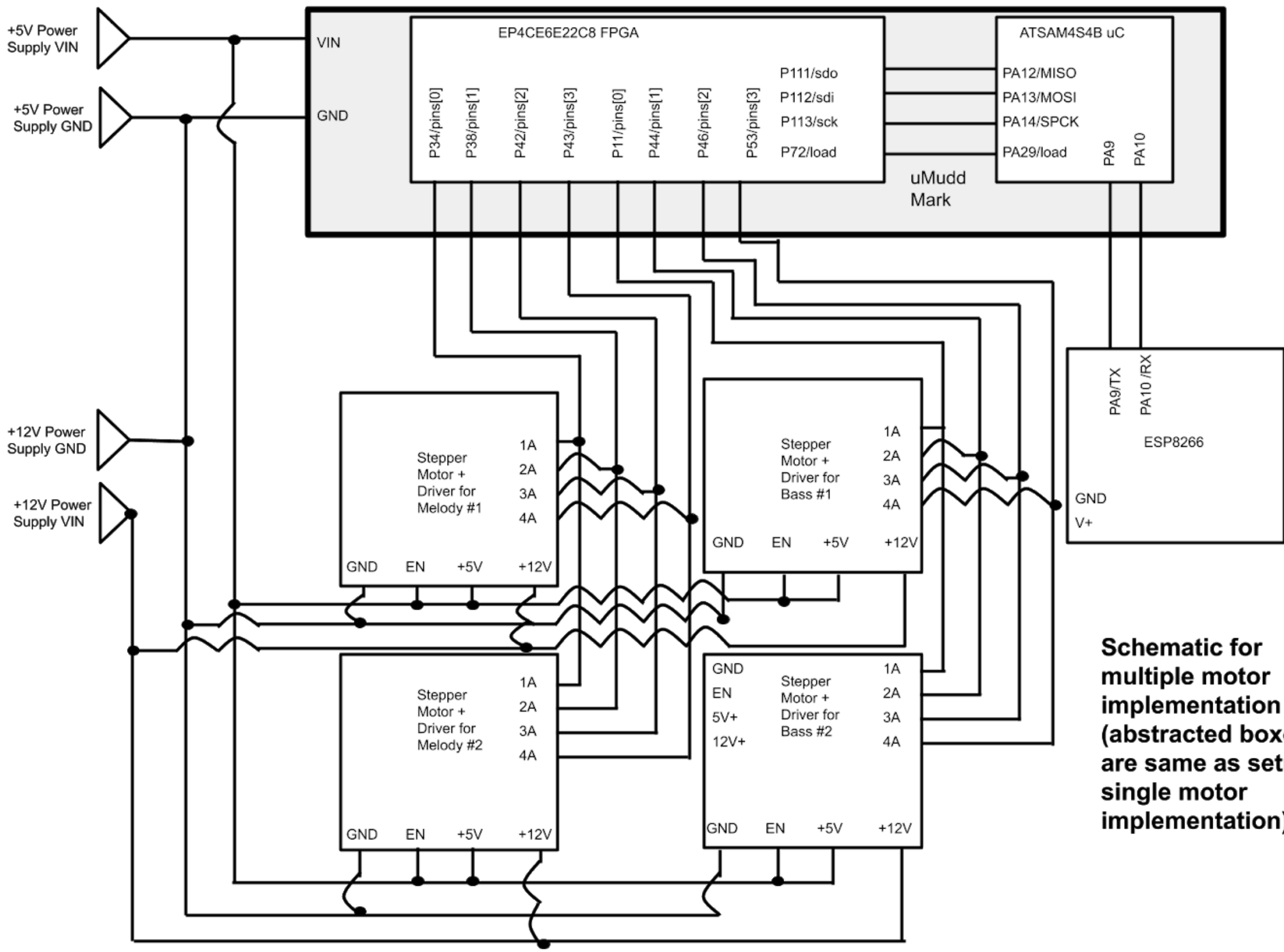
[8] “Take On Me Sheet Music,” *Sheetmusic-Free.com*. [Online]. Available: https://sheetmusic-free.com/take-on-me-sheet-music-a-ha-piano-sheet/. [Accessed: 12-Dec-2019].

[9] “Scary Monsters And Nice Sprites Lead.mid,” *BitMidi*, 2018. [Online]. Available: https://bitmidi.com/skrillex-scary-monsters-and-nice-sprites-lead-mid. [Accessed: 12-Dec-2019].

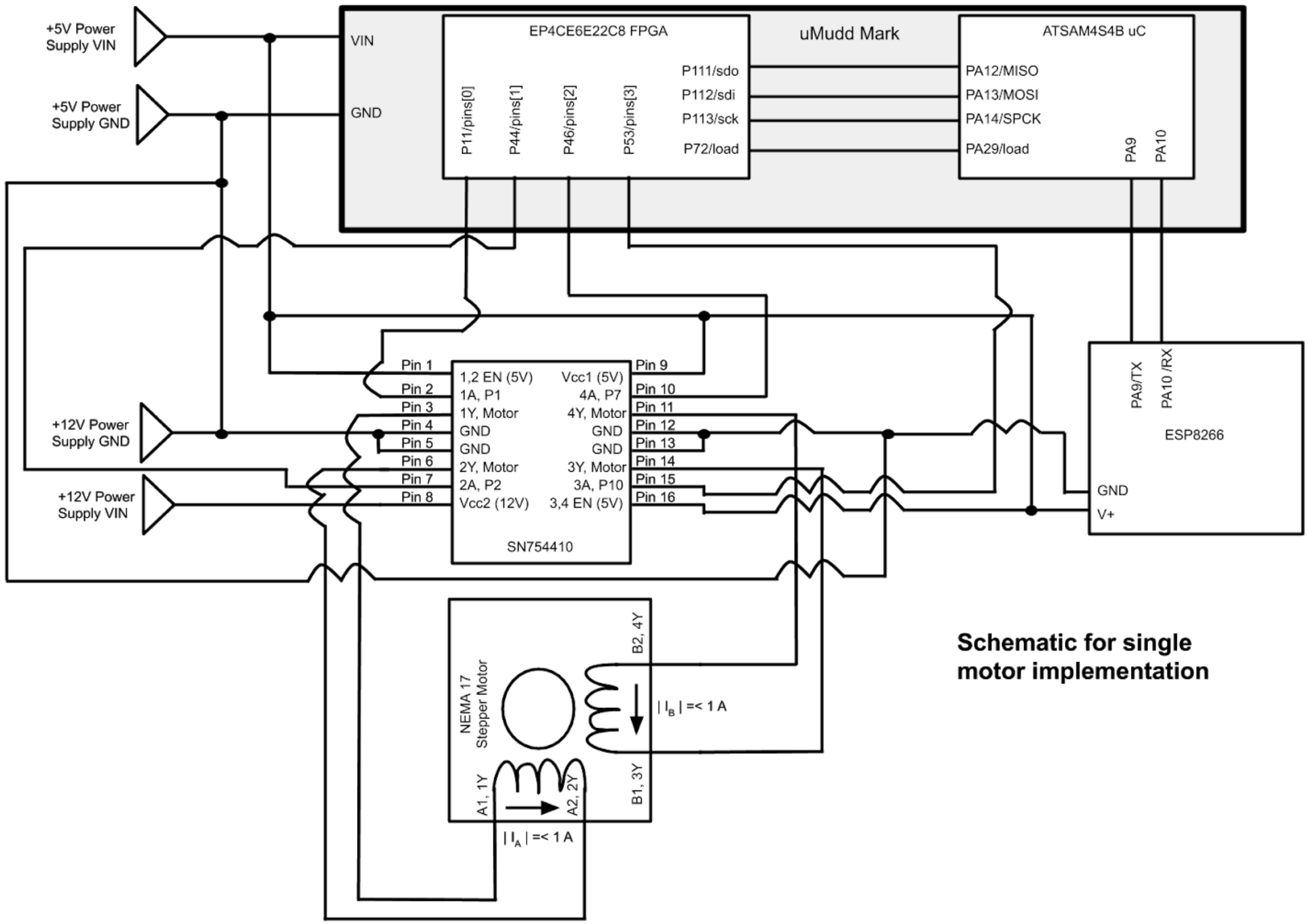
Appendix A. Code

Verilog Code: Refer to pages 19 - 21
C Code: Refer to pages 22 - 68
Midi Parsing Python Code: Refer to pages 69 - 70
File containing all encoded songs:

Appendix B. Full Size Schematics



Schematic for multiple motor implementation (abstracted boxes are same as setup in single motor implementation)



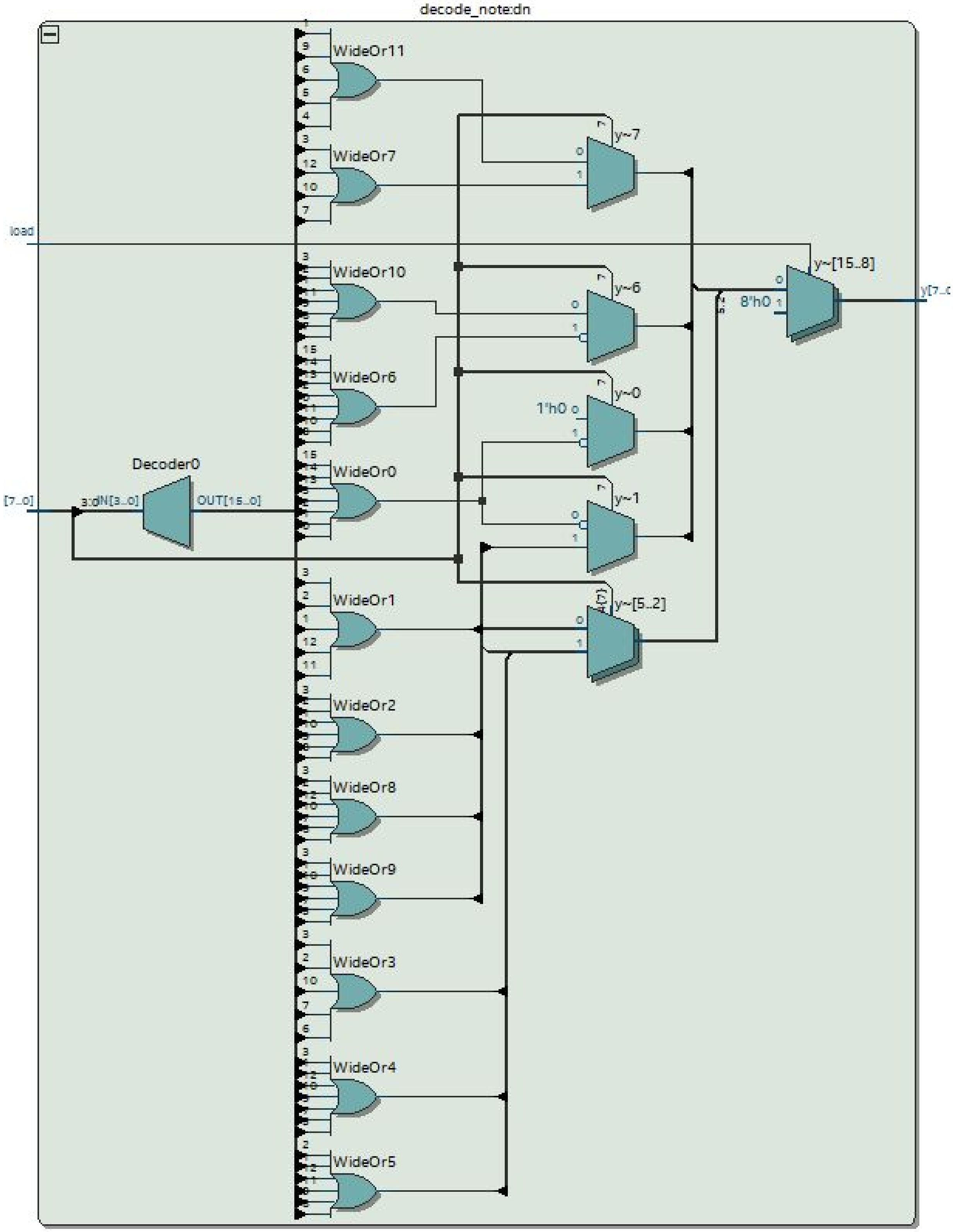
Schematic for single motor implementation

Appendix C. Note Frequency to Slow Clock Counter Addend Table

Note	Frequency	y value exact	y value (what to add to slowclk each cycle)
E1	-	-	34
F1	-	-	36
FS1	-	-	39
G1	-	-	41
GS1	-	-	43
A1	-	-	46
A#1	-	-	49
B1	-	-	52
C2	65.41	54.86988493	55
C#2	69.3	58.13305344	58
D2	73.42	61.58915994	62
D#2	77.78	65.24659302	65
E2	82.41	69.13051853	69
F2	87.31	73.24093645	73
F#2	92.5	77.594624	78
G2	98	82.2083584	82
G#2	103.83	87.09891686	87
A2	110	92.274688	92
A#2	116.54	97.76083763	98
B2	123.47	103.574143	104
C3	130.81	109.7313812	110
C#3	138.59	116.2577183	116
D3	146.83	123.1699313	123
D#3	155.56	130.493186	130
E3	164.81	138.2526484	138
F3	174.61	146.4734843	146

F#3	185	155.189248	155
G3	196	164.4167168	164
G#3	207.65	174.1894451	174
A3	220	184.549376	185
A#3	233.08	195.5216753	196
B3	246.94	207.148286	207
C4	-	-	220
C#4	-	-	232
D4	-	-	260
D#4	-	-	276

Appendix D. Decode Note RTL Viewer



decode_note:dn
WideOr11
WideOr7
load
WideOr10
WideOr6
Decoder0
[7..0]
dN[3..0]
OUT[15..0]
WideOr0
WideOr1
WideOr2
WideOr8
WideOr9
WideOr3
WideOr4
WideOr5
y~7
y~6
y~0
1'h0
y~1
y~[5..2]
y~[15..8]
8'h0

Appendix E. Parts List

Item	Source	Quantity
Usongshine Nema 17 Stepper Motor	Amazon	4
Large Wooden Box	Michaels	1
Small Wooden Box	Michaels	1
ESP8266 module	Sparkfun	1
SN754410 Quadruple Half-H Driver	Texas Instruments	9
MuddPi Mark V_1 Board	HMC	1

Appendix F. Final Budget Breakdown

Item	Quantity	Cost Per Unit	Total Cost
Usongshine Nema 17 Stepper Motor	4	10.36	$43.76
Large Wooden Box	1	$3.73	$3.73
Small Wooden Box	1	$1.79	$1.79
Wooden Box with Slits	1	$0.85	$0.85
TOTAL	-	-	**$50.13**

```
/////////////////////////////////////////////
// project.sv
// HMC E155 16 November 2019
// E155 Final Project: Stepper Motor Music
// Samantha Tign + Andrew Q. Pham
// apham@hmc.edu, sting@hmc.edu
/////////////////////////////////////////////

/////////////////////////////////////////////
// testbench
// tests SPI communication of stepper motors notes between FPGA and ATSAM
/////////////////////////////////////////////

module testbench();
    logic clk, load, sck, sdi, sdo;
    logic [7:0] note, note2;
    logic [31:0] i;

    // device under test
    project dut(clk, sck, sdi, sdo, load);

    // test case
    initial begin
        note       <= 8'h11;
        note2      <= 8'h91;
    end

    // generate clock and load signals
    initial
        forever begin
            clk = 1'b0; #5;
            clk = 1'b1; #5;
        end

    initial begin
      i = 0;
      load = 1'b1;
    end

    // shift in test vectors, wait until done
    always @(posedge clk) begin
      if (i == 0) load = 1'b1;
      else if (i == 8) load = 1'b0;
      else if (i<8) begin
        #1; sdi = note[7-i];
        #1; sck = 1; #5; sck = 0;
      end
      else if (i == 108) load = 1'b0;
      else if (i == 99) load = 1'b1;
      else if (i < 108 & i > 99) begin
        #1; sdi = note2[7-(i-100)];
        #1; sck = 1; #5; sck = 0;
      end
      else if (i > 10000000) begin
        $stop();
      end

       i = i + 1;

    end

endmodule


/////////////////////////////////////////////
// project
// Top level module with SPI interface and SPI core
/////////////////////////////////////////////

module project(input  logic clk,
               input  logic sck,
               input  logic sdi,
               output logic sdo,
               input  logic load,
               output logic [7:0] motor_pins);

```

```
    logic [15:0] notes; //create array for two notes that will be played at the same time

    project_spi spi(sck, sdi, sdo, notes);   //read in notes via SPI interaction with uC
    project_core core_m1(clk, load, notes[15:8], motor_pins[7:4]); //play notes for melody part
   project_core core_m2(clk, load, notes[7:0], motor_pins[3:0]); //play notes for bass part

endmodule

// SPI Module
module project_spi(input  logic sck,
                   input  logic sdi,
                   output logic sdo,
                   output logic [15:0] notes);

    always_ff @(posedge sck) begin
        {notes} = {notes[14:0], sdi};
    end
    // when done is first asserted, shift out msb before clock edge
    assign sdo = 0;
endmodule

// Motor Driving Module
module project_core(input  logic         clk,
                    input  logic         load,
                    input  logic [7:0]     note,
                    output logic [3:0]   motor_pins);

   logic [24:0] q; //bus used to create slowclk
   logic [7:0] y; //input note
   logic [7:0] newy; //y adjusted based on raise/lower octave flags
   logic slowclk; //slow clock used to control frequency of notes output from motor_pins
   logic highflag; //flag used to double note frequency and raise note by an octave
   logic lowflag; //flag used to halve note frequency and decrease note by an octave

   assign highflag = note[6];
   assign lowflag = note[5];

   //Decode input note
    decode_note dn(note, clk, load, y);
   //Compute slowclk addend based on flags
   assign newy = y*(1+highflag)/(1+lowflag);

   //Create slow clock
   counter counter(clk, load, newy, q);
   assign slowclk = q[24];

   //Drive motors with full step sequence
   playnote playnote(slowclk, load, newy, motor_pins);

endmodule

// Decode Note module: determine what frequency to drive each note at
// y values for each octave were determined using slowclk formula as seen in Appendix C of final report
module decode_note(input logic [7:0] note,
                   input logic clk,
                   input logic load,
                   output logic [7:0] y);
    always_comb
       if (load == 1) begin
           y = 0;
       end
       else begin
           if (note[7]) begin //note[7] high indicates third octave
               case (note[3:0])
                   4'b0000 : y <= 0;
                   4'b0001 : y <= 110;
                   4'b0010 : y <= 116;
                   4'b0011 : y <= 123;
                   4'b0100 : y <= 130;
                   4'b0101 : y <= 138;
                   4'b0110 : y <= 146;
                   4'b0111 : y <= 155;
                   4'b1000 : y <= 164;
                   4'b1001 : y <= 174;
```

```
                   4'b1010 : y <= 185;
                   4'b1011 : y <= 196;
                   4'b1100 : y <= 207;
                   default : y <= 0;

               endcase
           end
           else begin
               case (note[3:0]) //note[7] low indicates second octave
                   4'b0000 : y <= 0;
                   4'b0001 : y <= 55;
                   4'b0010 : y <= 58;
                   4'b0011 : y <= 62;
                   4'b0100 : y <= 65;
                   4'b0101 : y <= 69;
                   4'b0110 : y <= 73;
                   4'b0111 : y <= 78;
                   4'b1000 : y <= 82;
                   4'b1001 : y <= 87;
                   4'b1010 : y <= 92;
                   4'b1011 : y <= 98;
                   4'b1100 : y <= 104;
                   default : y <= 0;

               endcase
           end
       end

endmodule

//counter used to create slowclk from input 40 MHz clk
module counter(input logic clk,
               input logic load,
               input logic [7:0] y,
               output logic [24:0] q);
   always_ff @(posedge clk)
        if (load)    q <= 0;
        else         q <= q + y;

endmodule
```

```
module playnote(input logic slowclk,
                input logic load,
                input logic [7:0] y,
                output logic [3:0] pins);

   typedef enum logic [1:0] {S0, S1, S2, S3} statetype;
   statetype state, nextstate; // state register

   always_ff @(posedge slowclk, posedge load)
      if (load) state <= S0; //reset FSM if new notes are being loaded in
      else state <= nextstate;

   // next state logic -- always go to next state in loop regardless of inputs
   always_comb case (state)
      S0: nextstate = S1;
      S1: nextstate = S2;
      S2: nextstate = S3;
      S3: nextstate = S0;
      default: nextstate=S1;
   endcase

   // output logic for full step drive
   assign pins[0]= (state==S0) && (y != 0);
   assign pins[1]= (state==S2) && (y != 0);
   assign pins[2]= (state==S1) && (y != 0);
   assign pins[3]= (state==S3) && (y != 0);

endmodule
```

```c
// E155: Stepper Motor Music Project
// Authors: Andrew Q. Pham & Sam Ting
// Emails: apham@hmc.edu, sting@hmc.edu
// Date of Creation: 16 November 2019
//
// Description: Microcontroller code for Stepper Motor Music E155 Final Project Fall 2019
// Functionality: Send musical notes to hardware accelerator over SPI, create HTML webpage,
// where user can select a song, encode notes based on desired frequency

////////////////////////////////////////////////
// #includes
////////////////////////////////////////////////

#include "SAM4S4B.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "songs.h"
#include "notes.h"

////////////////////////////////////////////////
// Constants
////////////////////////////////////////////////
#define LOAD_PIN    29

#define BUFF_LEN 32
#define NUM_MOTORS 2

//Defining the web page in two chunks: everything before the current time, and everything after the current time
char* webpageStart = "<!DOCTYPE html><html><head><title>-----------STEPPER MOTOR MUSIC-----------</title>\
  <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">\
  </head>\
  <body><h1>-----------STEPPER MOTOR MUSIC-----------</h1>";
char* songStr = "<p>Song Choice:</p>"
                "<form action=\"0\"><input type=\"submit\" value=\"No Song\"></form>"
                "<form action=\"1\"><input type=\"submit\" value=\"YMCA\"></form>"
                "<form action=\"2\"><input type=\"submit\" value=\"Closer\"></form>"
                "<form action=\"3\"><input type=\"submit\" value=\"Take on Me\"></form>"
                "<form action=\"4\"><input type=\"submit\" value=\"I Write Sins But Not Tragedies\"></form>"
                "<form action=\"5\"><input type=\"submit\" value=\"All Star\"></form>"
                "<form action=\"6\"><input type=\"submit\" value=\"99 Luftballoons\"></form>"
                "<form action=\"7\"><input type=\"submit\" value=\"Scary Monsters and Nice Sprites\"></form>"
                "<form action=\"8\"><input type=\"submit\" value=\"Yoshi Story Theme\"></form>"
                "<form action=\"9\"><input type=\"submit\" value=\"Shape of You\"></form>";
char* webpageEnd   = "</body></html>";

// Create song struct that holds all the notes for a song and the tempo (based on either 16th or 8th notes)
typedef struct SONG{
    const float ms_dur_sixteenth;
    const float *notes;
}SONG;


// Create song structures to reference from website
const SONG NO_SONG = {0, &NO_SONG_NOTES[0][0]};
const SONG HOT_CROSS_BUNS = {25, &HOT_CROSS_BUNS_NOTES[0][0]};
const SONG LITTLE_LAMB = {20, &LITTLE_LAMB_NOTES[0][0]};
const SONG TAKE_ON_ME = {25, &TAKE_ON_ME_NOTES[0][0]};
const SONG I_WRITE_SINS = {32, &SINS_NOTES[0][0]};
const SONG ALL_STAR = {20, &ALL_STAR_NOTES[0][0]};
const SONG BALLOONS = {26, &BALLOONS_NOTES[0][0]};
const SONG MONSTERS = {35, &MONSTERS_NOTES[0][0]};
const SONG YOSHI = {20, &YOSHI_NOTES[0][0]};
const SONG SHAPEOFYOU = {23, &SHAPE_NOTES[0][0]};
const SONG YMCA = {22, &YMCA_NOTES[0][0]};
const SONG CLOSER = {41, &CLOSER_NOTES[0][0]};

```

```c

//////////////////////////////////////////////////
// Function Prototypes
//////////////////////////////////////////////////

void playNote(double notes[]);
void sendString(char* str);
int inString(char request[], char des[]);
int updateSongRequest(char request[]);
void updateSongChoice(int song_id, const SONG **song);

//////////////////////////////////////////////////
// Main
//////////////////////////////////////////////////

/**
 * Plays a hardcoded song by encoding the notes in the song, sending
 * them over SPI to the FPGA, and waiting for the note duration
**/
int main(void) {
    // Init peripherials on ATSAM
    samInit();
    pioInit();
    spiInit(MCK_FREQ/244000, 0, 1);
    tcInit();
    tcDelayInit();
    pioPinMode(LOAD_PIN, PIO_OUTPUT);
    uartInit(4,20);
    int song_id = 0;
    int note_count = 0;
    double ms_count  = 0;
    const SONG *song = &NO_SONG;

    while(1){
        // Wait for ESP8266 to send a request.
        // Receive web request from the ESP
        char request[BUFF_LEN] = "                  "; // initialize to known value
        int charIndex = 0;

        while(inString(request, "\n") == -1) {
            // Index to track note of song playing
            double notes[2] = {*(song->notes + note_count*NUM_MOTORS), *(song->notes + note_count*NUM_MOTORS + 1)};

            // Wait for a complete request to be transmitted before processing
            while(!uartRxReady()){

                if (ms_count == 0) {
                    playNote(notes);
                }

                tcDelayMicroseconds(1);
                ms_count += 0.001;

                if (ms_count >= song->ms_dur_sixteenth){
                    ms_count = 0;
                    if (notes[0] != END){
                        note_count++;
                        notes[0] = *(song->notes + note_count*NUM_MOTORS);
                        notes[1] = *(song->notes + note_count*NUM_MOTORS + 1);
                    }
                    else{
                      song_id = 0;
                      note_count = 0;
                      ms_count = 0;
                      updateSongChoice(0, &song); //begin playing newly selelcted song
                    }
                }

            }

            request[charIndex++] = uartRx();
```

```c
        }

        int new_song_id = updateSongRequest(request);

        if (new_song_id != song_id){ //if a different song is requested
            note_count = 0;
            ms_count = 0;
            song_id = new_song_id; //change song ID
            updateSongChoice(song_id, &song); //begin playing newly selelcted song
        }

        // finally, transmit the webpage over UART
        sendString(webpageStart); // webpage header code
        sendString(songStr); // button for controlling song choice
        sendString(webpageEnd);
    }

}

////////////////////////////////////////////////
// Functions
////////////////////////////////////////////////
/**
 * Change which song is currently being played based on webpage input
 * @param song_id   the new song id based on HTML input
 * @param **song    the pointer that should be updated with the new song request
 * @return 8-bit signal that encodes what note to play on which motor
**/
void updateSongChoice(int song_id, const SONG **song){
    switch (song_id){
        case 0:
            *song = &NO_SONG;
            break;
        case 1:
            *song = &YMCA;
            break;
        case 2:
            *song = &CLOSER;
            break;
        case 3:
            *song = &TAKE_ON_ME;
            break;
        case 4:
            *song = &I_WRITE_SINS;
            break;
        case 5:
            *song = &ALL_STAR;
            break;
        case 6:
            *song = &BALLOONS;
            break;
        case 7:
            *song = &MONSTERS;
            break;
        case 8:
            *song = &YOSHI;
            break;
        case 9:
            *song = &SHAPEOFYOU;
            break;


        default:
            *song = &NO_SONG;
    }
}

/**
 * Encodes note and which motor to play it into 8-bit signal
 * encoded_note[7:5] = octave of note
 * encoded_note[3:0] = note in octave
 * We encoded for four octaves
```

```c
 *      encoded_note[7:5] = 000 = octave 2
 *      encoded_note[7:5] = 100 = octave 3
 *      encoded_note[7:5] = 001 = divide note frequencies in octave 2 by 2 --> octave 1
 *      encoded_note[7:5] = 110 = multiply note frequencies in octave 3 by 2 --> octave 4
 * @param motor The number of the motor to play the note
 * @param id    The id of the note
 * @return 8-bit signal that encodes what note to play on which motor
**/
char encodeNote(int motor, double id){
    char encoded_note = 0x00;
    switch ((int)id){
        case 0:
            encoded_note |= 0x00;
            break;
        case C2:
            encoded_note |= 0x01;
            break;
        case CS2:
            encoded_note |= 0x02;
            break;
        case D2:
            encoded_note |= 0x03;
            break;
        case DS2:
            encoded_note |= 0x04;
            break;
        case E2:
            encoded_note |= 0x05;
            break;
        case F2:
            encoded_note |= 0x06;
            break;
        case FS2:
            encoded_note |= 0x07;
            break;
        case G2:
            encoded_note |= 0x08;
            break;
        case GS2:
            encoded_note |= 0x09;
            break;
        case A2:
            encoded_note |= 0x0a;
            break;
        case AS2:
            encoded_note |= 0x0b;
            break;
        case B2:
            encoded_note |= 0x0c;
            break;
        case C3:
            encoded_note |= 0x81;
            break;
        case CS3:
            encoded_note |= 0x82;
            break;
        case D3:
            encoded_note |= 0x83;
            break;
        case DS3:
            encoded_note |= 0x84;
            break;
        case E3:
            encoded_note |= 0x85;
            break;
        case F3:
            encoded_note |= 0x86;
            break;
        case FS3:
            encoded_note |= 0x87;
            break;
        case G3:
```

```c
                encoded_note |= 0x88;
                break;
            case GS3:
                encoded_note |= 0x89;
                break;
            case A3:
                encoded_note |= 0x8a;
                break;
            case AS3:
                encoded_note |= 0x8b;
                break;
            case B3:
                encoded_note |= 0x8c;
                break;
            case C4:
                encoded_note |= 0xc1;
                break;
            case CS4:
                encoded_note |= 0xc2;
                break;
            case D4:
                encoded_note |= 0xc3;
                break;
            case DS4:
                encoded_note |= 0xc4;
                break;
            case E4:
                encoded_note |= 0xc5;
                break;

            case E1:
                encoded_note |= 0x28;
                break;
            case F1:
                encoded_note |= 0x29;
                break;

            case FS1:
                encoded_note |= 0x27;
                break;

            case G1:
                encoded_note |= 0x28;
                break;
            case GS1:
                encoded_note |= 0x29;
                break;

            case A1:
                encoded_note |= 0x2a;
                break;
            case AS1:
                encoded_note |= 0x2b;
                break;
            case B1:
                encoded_note |= 0x2c;
                break;

            default:
                encoded_note |= 0x00;
                break;
        }

        encoded_note |= (motor << 4);

        return encoded_note;
    }

    /**
     * Plays a note by encoding it and sending it over SPI to the FPGA
     *
     * @param note_id ID number of note defined in constants section above
```

```c
  **/
void playNote(double notes[]) {
    pioDigitalWrite(LOAD_PIN, 1);        // Set LOAD Pin high to initiate SPI interation
    for (int mtr = 0; mtr < NUM_MOTORS; mtr++){
        char note = encodeNote(mtr, notes[mtr]);
        spiSendReceive(note);
    }
    pioDigitalWrite(LOAD_PIN, 0);
}
// Sends a null terminated string of arbitrary length
void sendString(char* str) {
  char* ptr = str;
  while (*ptr) uartTx(*ptr++);
}

//determines whether a given character sequence is in a char array request, returning 1 if present, -1 if not present
int inString(char request[], char des[]) {
  if (strstr(request, des) != NULL) {return 1;}
  return -1;
}

int updateSongRequest(char request[])
{
  int song_id;

  // The request has been received. now process to determine whether to turn the LED on or off
  if (inString(request, "0")==1) {
    song_id = 0;
  } else if (inString(request, "1")==1) {
    song_id= 1;
  } else if (inString(request, "2")==1) {
    song_id= 2;
  }
  else if (inString(request, "3")==1) {
    song_id= 3;
  }
  else if (inString(request, "4")==1) {
    song_id= 4;
  }
  else if (inString(request, "5")==1) {
    song_id= 5;
  }
  else if (inString(request, "6")==1) {
    song_id= 6;
  }
  else if (inString(request, "7")==1) {
    song_id= 7;
  }
  else if (inString(request, "8")==1) {
    song_id= 8;
  }
  else if (inString(request, "9")==1) {
    song_id= 9;
  }
  else if (inString(request, "10")==1) {
    song_id= 10;
  }
  else if (inString(request, "11")==1) {
    song_id= 11;
  }
  else if (inString(request, "12")==1) {
    song_id= 12;
  }

  else {
    song_id = 0;
    }

  return song_id;
}

```

```c

//Note Definitions
#define REST 0
#define C2 1
#define CS2 2
#define D2 3
#define DS2 4
#define E2 5
#define F2 6
#define FS2 7
#define G2 8
#define GS2 9
#define A2 10
#define AS2 11
#define B2 12
#define C3 13
#define CS3 14
#define D3 15
#define DS3 16
#define E3 17
#define F3 18
#define FS3 19
#define G3 20
#define GS3 21
#define A3 22
#define AS3 23
#define B3 24
#define END 25
#define C4 26
#define CS4 27
#define D4 28
#define E4 29
#define E1 30
#define F1 31
#define FS1 32
#define G1 33
#define GS1 34
#define A1 35
#define AS1 36
#define B1 37
#define DS4 38

```

```c

//Song Arrays
#include "notes.h"
#include <stdio.h>

//No Song
const float NO_SONG_NOTES[][2] = {{END, END}};

// Hot Cross Buns, Hard-Coded
const float HOT_CROSS_BUNS_NOTES[][2] = {



{B2, C2},
{B2, REST},
{B2, C2},
{REST, REST},
{A2, C2},
{A2, REST},
{A2, C2},
{REST, REST},
{G2, C2},
{G2, REST},
{G2, C2},
{G2, REST},
{G2, C2},
{G2, REST},
{REST, REST},
{REST, REST},

{B2, C2},
{B2, REST},
{B2, C2},
{REST, REST},
{A2, C2},
{A2, REST},
{A2, C2},
{REST, REST},
{G2, C2},
{G2, REST},
{G2, C2},
{G2, REST},
{G2, C2},
{G2, REST},
{REST, REST},
{REST, REST},

{G2, REST},
{REST, C2},
{G2, REST},
{REST, C2},
{G2, REST},
{REST, C2},
{G2, REST},
{REST, C2},
{G2, REST},
{REST, C2},
{A2, REST},
{REST, C2},
{A2, REST},
{REST, C2},
{A2, REST},
{REST, C2},
{A2, REST},
{REST, C2},

{B2, C2},
{B2, REST},
{B2, C2},
{REST, REST},
{A2, C2},
{A2, REST},
```

```
{A2, C2},
{REST, REST},
{G2, C2},
{G2, REST},
{G2, C2},
{G2, REST},
{G2, C2},
{G2, REST},
{END, END}};


// Take On Me by a-ha, Hard-Coded
const float TAKE_ON_ME_NOTES[][2] = {
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},

{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},
{CS4, FS3},

{CS3, CS3},
{CS3, CS3},
{CS3, CS3},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},

{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},

{REST, C3},
{REST, C3},
{REST, REST},
{REST, C2},
{REST, REST},
{REST, REST},
{REST, C2},
{REST, C2},

{REST, C3},
{REST, C3},
{REST, REST},
{REST, C2},
{REST, REST},
{REST, REST},
{REST, C2},
{REST, C2},

{REST, C3},
{REST, C3},
{REST, REST},
{REST, C2},
{REST, REST},
```

```
{REST, REST},
{REST, C2},
{REST, C2},

{REST, C3},
{REST, C3},
{REST, REST},
{REST, C2},
{REST, REST},
{REST, REST},
{REST, C2},
{REST, C2},

{FS3, B1},
{FS3, B1},
{D3, B1},
{B2, B1},
{REST, B1},
{B2, B1},
{REST, B1},
{E3, B1},

{REST, REST},
{E3, },
{REST, E2},
{E3, E2},
{E3, E2},
{GS3, E2},
{GS3, E2},
{A3, E2},

{B3, A1},
{A3, A1},
{A3, A1},
{A3, A1},
{E3, A1},
{REST, REST},
{D3, C2},
{REST, C2},

{FS3, REST},
{REST, D2},
{FS3, D2},
{REST, D2},
{FS3, D2},
{E3, CS2},
{E3, CS2},
{FS3, CS2},
{E3, CS2},

{FS3, B1},
{FS3, B1},
{D3, B1},
{B2, B1},
{REST, B1},
{B2, B1},
{REST, B1},
{E3, B1},

{REST, REST},
{E3, },
{REST, E2},
{E3, E2},
{E3, E2},
{GS3, E2},
{GS3, E2},
{A3, E2},

{B3, A1},
{A3, A1},
{A3, A1},
{A3, A1},
```

```
{E3, A1},
{REST, REST},
{D3, C2},
{REST, C2},

{FS3, REST},
{REST, D2},
{FS3, D2},
{REST, D2},
{FS3, D2},
{E3, CS2},
{E3, CS2},
{FS3, CS2},
{E3, CS2},

{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},

{D3, B1},
{D3, B1},
{D3, B1},
{D3, B1},
{D3, B1},
{D3, B1},
{CS3, B1},
{B2, B1},
{B2, B1},

{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{E2, REST},

{CS3, A1},
{CS3, A1},
{CS3, A1},
{CS3, A1},
{CS3, A1},
{A2, A1},
{REST, A1},
{REST, A1},

{REST, D2},
{FS3, D2},
{REST, D2},
{FS3, D2},
{FS3, CS2},
{FS3, CS2},
{E3, CS2},
{E3, CS2},

{D3, B1},
{D3, B1},
{D3, B1},
{D3, B1},
{D3, B1},
{CS3, B1},
{CS3, B1},
{B2, B1},

{B2, A1},
{B2, A1},
{REST, A1},
{REST, A1},
{REST, A1},
```

```
{REST, A1},
{REST, A1},
{E2, A1},

{CS3, D2},
{CS3, D2},
{CS3, D2},
{CS3, D2},
{CS3, CS2},
{B2, CS2},
{A2, CS2},
{A2, CS2},

{A2, REST},
{B2, REST},
{B2, REST},
{CS3, REST},
{CS3, REST},
{B2, REST},
{A2, REST},
{A2, REST},

{REST, REST},
{REST, REST},
{D3, REST},
{D3, REST},
{D3, REST},
{D3, REST},
{D3, REST},
{D3, REST},

{E3, REST},
{E3, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},

{REST, REST},
{REST, REST},
{FS2, REST},
{A2, REST},
{A2, REST},
{A2, REST},
{A2, REST},
{A2, REST},

{A2, REST},
{GS2, REST},
{GS2, REST},
{GS2, REST},
{GS2, REST},
{FS2, REST},
{FS2, REST},
{FS2, REST},

{A1, A1},
{A1, A1},
{A1, A1},
{A1, A1},
{A1, A1},
{A1, A1},
{A1, A1},
{A1, A1},

{GS2, GS2},
{GS2, GS2},
{GS2, GS2},
{GS2, GS2},
{GS2, GS2},
```

```
{GS2, GS2},
{GS2, GS2},
{GS2, GS2},

{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},

{E3, REST},
{E3, REST},
{REST, REST},
{FS3, REST},
{FS3, REST},
{FS3, REST},
{E3, REST},
{E3, REST},

{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},
{A2, A2},

{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},

{FS3, FS3},
{FS3, FS3},
{FS3, FS3},
{FS3, FS3},
{FS3, FS3},
{FS3, FS3},
{FS3, FS3},
{FS3, FS3},

{E3, REST},
{E3, REST},
{REST, REST},
{FS3, REST},
{FS3, REST},
{FS3, REST},
{E3, REST},
{E3, REST},

{CS3, CS3},
{CS3, CS3},
{CS3, CS3},
{CS3, CS3},
{CS3, CS3},
{CS3, CS3},
{CS3, CS3},
{CS3, CS3},

{GS3, GS3},
{GS3, GS3},
{GS3, GS3},
{GS3, GS3},
{GS3, GS3},
```

```
{GS3, GS3},
{GS3, GS3},
{GS3, GS3},

{A3, A3},
{A3, A3},
{A3, A3},
{A3, A3},
{A3, A3},
{A3, A3},
{A3, A3},
{A3, A3},

{REST, REST},
{REST, REST},
{B3, B3},
{CS3, CS3},
{REST, REST},
{B3, B3},
{A3, A3},
{A3, A3},

{A3, A3},
{A3, A3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},

{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},

{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},

{END, END}};

// All Star by Smash Mouth, Hard-Coded
const float ALL_STAR_NOTES[][2] = {
{DS3, REST},
{DS3, REST},

{DS3, REST},
{DS3, REST},

{AS3, DS2},
{AS3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},

{F3, AS2},
{F3, AS2},
{DS3, AS2},
{DS3, AS2},
```

```
{DS3, AS2},
{DS3, AS2},
{GS3, AS2},
{GS3, AS2},

{GS3, G2},
{GS3, G2},
{G3, G2},
{G3, G2},
{G3, G2},
{G3, G2},
{F3, G2},
{F3, G2},

{F3, GS2},
{F3, GS2},
{DS3, GS2},
{DS3, GS2},
{REST, GS2},
{REST, GS2},
{DS3, GS2},
{DS3, GS2},

{AS3, DS2},
{AS3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{F3, DS2},
{F3, DS2},

{F3, AS2},
{F3, AS2},
{DS3, AS2},
{DS3, AS2},
{DS3, AS2},
{DS3, AS2},
{C3, AS2},
{C3, AS2},

{C3, G2},
{C3, G2},
{AS2, G2},
{AS2, G2},
{AS2, G2},
{AS2, G2},
{AS2, G2},
{AS2, G2},

{REST, GS2},
{REST, GS2},
{REST, GS2},
{REST, GS2},
{DS2, GS2},
{DS2, GS2},
{DS2, GS2},
{DS2, GS2},


{AS3, DS2},
{AS3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{F3, DS2},
{F3, DS2},

{F3, AS2},
{F3, AS2},
{DS3, AS2},
```

```
{DS3, AS2},
{DS3, AS2},
{DS3, AS2},
{GS3, AS2},
{GS3, AS2},

{GS3, G2},
{GS3, G2},
{G3, G2},
{G3, G2},
{G3, G2},
{G3, G2},
{F3, G2},
{F3, G2},

{F3, GS2},
{F3, GS2},
{DS3, GS2},
{DS3, GS2},
{DS3, GS2},
{DS3, GS2},
{AS3, GS2},
{AS3, GS2},

{AS3, DS2},
{AS3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{G3, DS2},
{F3, DS2},
{F3, DS2},

{F3, AS2},
{F3, AS2},
{DS3, AS2},
{DS3, AS2},
{DS3, AS2},
{DS3, AS2},
{F3, AS2},
{F3, AS2},

{F3, G2},
{F3, G2},
{C3, G2},
{C3, G2},
{C3, G2},
{C3, G2},
{C3, G2},
{C3, G2},

{REST, GS2},
{REST, GS2},
{REST, GS2},
{REST, GS2},
{DS3, GS2},
{DS3, GS2},
{DS3, GS2},
{C3, GS2},

{DS3, REST},
{DS3, REST},
{DS3, DS2},
{DS3, DS2},
{DS3, G2},
{DS3, G2},
{DS3, REST},
{REST, REST},

{DS3, REST},
{DS3, REST},
{DS3, AS2},
```

```
{REST, AS2},
{DS3, D3},
{DS3, D3},
{REST, REST},
{REST, REST},

{DS3, REST},
{REST, REST},
{DS3, G2},
{DS3, G2},
{DS3, AS2},
{REST, AS2},
{DS3, REST},
{DS3, REST},

{DS3, REST},
{DS3, REST},
{DS3, GS2},
{REST, GS2},
{G3, C3},
{DS3, C3},
{REST, REST},
{REST, REST},


{G3, REST},
{G3, REST},
{AS3, DS2},
{AS3, DS2},
{GS3, G2},
{GS3, G2},
{G3, REST},
{G3, REST},

{F3, REST},
{F3, REST},
{F3, AS2},
{F3, AS2},
{F3, D3},
{F3, D3},
{F3, REST},
{DS3, REST},


{G3, REST},
{G3, REST},
{DS3, G2},
{REST, G2},
{DS3, AS2},
{DS3, AS2},
{C3, REST},
{AS2, REST},

{DS3, REST},
{DS3, REST},
{DS3, GS2},
{DS3, GS2},
{C3, C3},
{AS2, C3},
{REST, REST},
{REST, REST},

{G3, REST},
{AS3, REST},
{AS3, DS2},
{G3, DS2},
{C4, G2},
{C4, G2},
{G3, REST},
{AS3, REST},

{AS3, REST},
```

```
{G3, REST},
{C4, AS2},
{C4, AS2},
{G3, D3},
{AS3, D3},
{AS3, REST},
{GS3, REST},

{GS3, REST},
{G3, REST},
{G3, G2},
{F3, G2},
{REST, AS2},
{F3, AS2},
{F3, REST},
{DS3, REST},

{F3, REST},
{F3, REST},
{DS3, GS2},
{DS3, GS2},
{REST, C3},
{REST, C3},
{REST, REST},
{AS2, REST},

{DS3, REST},
{DS3, REST},
{DS3, DS2},
{REST, DS2},
{DS3, G2},
{DS3, G2},
{DS3, REST},
{REST, REST},

{DS3, REST},
{DS3, REST},
{DS3, AS2},
{DS3, AS2},
{REST, D3},
{REST, D3},
{REST, REST},
{AS2, REST},

{G3, REST},
{F3, REST},
{DS3, G2},
{REST, G2},
{F3, AS2},
{DS3, AS2},
{C3, REST},
{REST, REST},

{AS2, REST},
{AS2, REST},
{AS2, GS2},
{AS2, GS2},
{REST, C3},
{REST, C3},
{REST, REST},
{REST, REST},

{G3, G3},
{G3, G3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{REST, REST},
{DS3, DS3},
{C3, C3},

{DS3, DS3},
```

```
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{REST, REST},
{DS3, DS3},
{C3, C3},

{DS3, DS3},
{REST, REST},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},

{DS3, DS3},
{REST, REST},
{G3, G3},
{G3, G3},
{G3, G3},
{G3, G3},
{G3, G3},
{G3, G3},

{G3, G3},
{G3, G3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{REST, REST},
{DS3, DS3},
{C3, C3},

{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{REST, REST},
{DS3, DS3},
{C3, C3},

{DS3, DS3},
{REST, REST},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},

{DS3, DS3},
{REST, REST},
{G3, G3},
{G3, G3},
{G3, G3},
{G3, G3},
{DS3, DS3},
{DS3, DS3},

{G3, G3},
{G3, G3},
{G3, G3},
{G3, G3},
{AS3, AS3},
{AS3, AS3},
{AS3, AS3},
{AS3, AS3},

{GS3, GS3},
```

```
{GS3, GS3},
{G3, G3},
{G3, G3},
{F3, F3},
{REST, REST},
{F3, F3},
{F3, F3},

{F3, F3},
{F3, F3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},

{DS3, DS3},
{REST, REST},
{DS3, DS3},
{REST, REST},
{F3, F3},
{REST, REST},
{DS3, DS3},
{REST, REST},

{G3, G3},
{G3, G3},
{F3, F3},
{F3, F3},
{F3, F3},
{F3, F3},
{F3, F3},
{REST, REST},

{F3, F3},
{F3, F3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{F3, F3},
{F3, F3},

{G3, G3},
{G3, G3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},

{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{DS3, DS3},
{END, END}};

// I Write Sins But Not Tragedies by Panic! At The Disco, Hard-Coded
const float SINS_NOTES[][2] = {

{REST, A1},

{REST, E2},
{REST, C3},
{REST, C3},
{REST, REST},
{REST, E2},
{REST, B2},
```

```
{REST, B2},

{REST, A1},
{REST, E2},
{REST, C3},
{REST, C3},
{REST, REST},
{REST, E2},
{REST, B2},
{REST, B2},

{REST, A1},
{REST, E2},
{REST, C3},
{REST, C3},
{REST, REST},
{REST, E2},
{REST, B2},
{REST, B2},

{REST, A1},
{REST, E2},
{REST, C3},
{REST, C3},
{REST, REST},
{REST, E2},
{REST, B2},
{REST, B2},

{REST, D2},
{REST, C3},
{REST, D3},
{REST, D3},
{REST, REST},
{REST, C3},
{REST, D3},
{REST, D3},

{REST, D2},
{REST, C3},
{REST, D3},
{REST, D3},
{REST, REST},
{REST, C3},
{REST, D3},
{REST, D3},

{REST, F2},
{REST, B2},
{REST, C3},
{REST, C3},
{REST, REST},
{REST, B2},
{REST, C3},
{REST, C3},

{REST, F2},
{REST, B2},
{REST, C3},
{REST, C3},
{REST, REST},
{REST, B2},
{REST, C3},
{REST, C3},

{REST, A1},
{REST, E2},
{REST, C3},
{REST, C3},
{REST, REST},
{A2, E2},
{C3, C3},
```

```
    {B2, C3},

    {B2, A1},
    {A2, E2},
    {A2, C3},
    {REST, C3},
    {REST, REST},
    {A2, E2},
    {C3, B2},
    {B2, B2},

    {B2, A1},
    {A2, E2},
    {C3, C3},
    {B2, C3},
    {B2, REST},
    {A2, E2},
    {C3, B2},
    {B2, B2},

    {B2, A1},
    {A2, E2},
    {C3, C3},
    {B2, C3},
    {B2, REST},
    {A2, E2},
    {A2, B2},
    {A2, B2},

    {D3, D2},
    {REST, C3},
    {D3, D3},
    {REST, D3},
    {D3, REST},
    {D3, C3},
    {E3, D3},
    {C3, D3},

    {REST, D2},
    {REST, E3},
    {A2, D3},
    {A2, D3},
    {A2, REST},
    {A2, C3},
    {G2, D3},
    {G2, D3},

    {D3, F2},
    {D3, B2},
    {C3, D3},
    {B2, D3},
    {B2, REST},
    {C3, C3},
    {B2, D3},
    {B2, D3},

    {A2, F2},
    {G2, B2},
    {G2, C3},
    {G2, C3},
    {G2, REST},
    {G2, B2},
    {REST, C3},
    {REST, C3},

    {REST, A1},
    {G3, E2},
    {G3, C2},
    {G3, C2},
    {F3, REST},
    {F3, E2},
    {E3, D2},
```

```
{E3, D2},

{D3, A1},
{C2, E2},
{REST, C3},
{REST, C3},
{REST, REST},
{A2, E2},
{C3, B2},
{B2, B2},

{B2, A1},
{A2, E2},
{C3, C3},
{B2, C3},
{B2, REST},
{A2, E2},
{C3, B2},
{A2, B2},

{D3, A1},
{REST, E2},
{D3, C3},
{REST, C3},
{D3, REST},
{REST, E2},
{E3, B2},
{C3, B2},

{D3, D2},
{REST, C3},
{D3, D3},
{REST, D3},
{D3, REST},
{REST, C3},
{E3, D3},
{C3, D3},

{REST, D2},
{REST, C3},
{A2, D3},
{A2, D3},
{A2, REST},
{G2, C3},
{G2, D3},
{G2, D3},

{D3, F2},
{D3, B2},
{D3, C3},
{REST, C3},
{D3, REST},
{REST, B2},
{E3, C3},
{REST, C3},

{C3, A2},
{C3, A2},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},

{G3, G3},
{G3, G3},
{E3, E3},
{E3, E3},
{D3, D3},
{REST, REST},
{D3, D3},
```

```
{C3, C3},

{G3, G3},
{F3, F3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{F3, F3},
{E3, E3},

{E3, E3},
{D3, D3},
{C3, C3},
{C3, C3},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},

{D3, D3},
{D3, D3},
{C3, C3},
{G3, G3},
{G3, G3},
{E3, E3},
{E3, E3},
{D3, D3},

{D3, D3},
{E3, E3},
{E3, E3},
{REST, REST},
{REST, REST},
{C3, C3},
{D3, D3},
{D3, D3},

{D3, D3},
{D3, D3},
{C3, C3},
{D3, D3},
{REST, REST},
{C3, C3},
{REST, REST},
{D3, D3},

{D3, D3},
{E3, E3},
{C3, C3},
{C3, C3},
{REST, REST},
{C3, C3},
{D3, D3},
{C3, C3},

{D3, D3},
{G3, G3},
{F3, F3},
{E3, E3},
{D3, D3},
{D3, D3},
{D3, D3},
{C3, C3},

{G3, G3},
{G3, G3},
{G3, G3},
{E3, E3},
{E3, E3},
{E3, E3},
{D3, D3},
```

```
{D3, D3},

{REST, REST},
{G3, G3},
{F3, F3},
{E3, E3},
{E3, E3},
{E3, E3},
{E3, E3},
{A3, A3},

{C4, C4},
{C4, C4},
{B3, B3},
{A3, A3},
{REST, REST},
{REST, REST},
{REST, REST},
{REST, REST},

{D3, D3},
{D3, D3},
{C3, C3},
{G3, G3},
{G3, G3},
{E3, E3},
{E3, E3},
{D3, D3},

{D3, D3},
{E3, E3},
{E3, E3},
{REST, REST},
{REST, REST},
{C3, C3},
{D3, D3},
{D3, D3},

{D3, D3},
{D3, D3},
{C3, C3},
{D3, D3},
{REST, REST},
{C3, C3},
{REST, REST},
{D3, D3},

{D3, D3},
{E3, E3},
{C3, C3},
{C3, C3},
{REST, REST},
{REST, REST},
{REST, REST},
{A2, A2},

{A2, A2},
{B2, B2},
{B2, B2},
{C3, C3},
{C3, C3},
{D3, D3},
{D3, D3},
{D3, D3},

{A2, A1},
{A2, A1},
{A2, A1},
{A2, A1},

{END, END}};
```

```
// Mary Had A Little Lamb, Hard-Coded
const float LITTLE_LAMB_NOTES[][2] = {


{B3, C2},
{B3, REST},

{A3, C2},
{A3, REST},

{G3, C2},
{G3, REST},

{A3, C2},
{A3, REST},

{B3, C2},
{B3, B2},

{B3, C2},
{B3, B2},

{B3, B2},
{B3, B2},
{B3, B2},

{REST, REST},

{A3, A2},
{A3, A2},

{A3, A2},
{A3, A2},

{A3, A2},
{A3, A2},
{A3, A2},

{REST, REST},

{B3, REST},
{B3, REST},

{D3, REST},
{D3, REST},

{D3, D2},
{D3, D2},
{D3, D2},
{REST, REST},

{B3, C2},
{B3, REST},

{A3, C2},
{A3, REST},

{G3, C2},
{G3, REST},

{A3, 125},
{A3, REST},

{B3, B2},
{B3, B2},

{B3, B2},
{B3, B2},

{B3, B2},
{B3, B2},
```

```
{B3, B2},

{REST, REST},

{G3, C2},
{G3, REST},

{G3, C2},
{G3, REST},

{B3, B2},
{B3, B2},

{A3, A2},
{A3, A2},
{G3, G2},
{G3, G2},
{END, END}};


const float MONSTERS_NOTES[][2] = {
  {REST, C2},
  {G2, G2},
  {F2, F2},
  {G2, G2},
  {AS2, AS2},
  {AS2, AS2},
  {G2, G2},
  {D3, D3},
  {C3, C3},
  {AS2, AS2},
  {REST, F1},
  {REST, F1},
  {AS1, AS1},
  {A1, A1},
  {F2, F2},
  {DS2, DS2},
  {D2, D2},
  {C2, C2},
  {AS2, AS1},
  {F1, F1},
  {AS2, AS2},
  {G1, G1},
  {AS1, AS1},
  {D2, D2},

  {G3, G3},
  {D3, D3},
  {AS2, AS2},
  {G2, G2},

  {F3, F2},
  {AS2, AS2},
  {G2, G2},
  {F2, F2},

    {REST, C2},
  {G2, G2},
  {F2, F2},
  {G2, G2},
  {AS2, AS2},
  {AS2, AS2},
  {G2, G2},
  {D3, D3},
  {C3, C3},
  {AS2, AS2},
  {REST, F1},
  {REST, F1},
  {AS1, AS1},
  {A1, A1},
  {F2, F2},
  {DS2, DS2},
```

```
    {D2, D2},
    {C2, C2},
    {AS1, AS2},
    {F1, F1},
    {AS2, AS2},
    {G1, G1},
    {AS1, AS1},
    {D2, D2},

    {G3, G3},
    {D3, D3},
    {AS2, AS2},
    {G2, G2},

    {F3, F2},
    {AS2, AS2},
    {G2, G2},
    {F2, F2},

    {END, END}};


const float YOSHI_NOTES[][2] = {

{REST ,AS2},
{REST ,AS2},
{REST ,AS2},
{REST ,F3},
{REST ,F3},
{REST ,F3},
{REST ,F2},
{REST ,F2},

{REST ,AS2},
{REST ,AS2},
{REST ,F3},
{REST ,F3},
{REST ,F3},
{REST ,F3},
{REST ,F2},
{REST ,F2},

{REST ,AS2},
{REST ,AS2},
{REST ,AS2},
{REST ,F3},
{REST ,F3},
{REST ,F3},
{REST ,F2},
{REST ,F2},

{REST ,AS2},
{REST ,AS2},
{REST ,F3},
{REST ,F3},
{REST ,F3},
{REST ,F3},
{REST ,F2},
{REST ,F2},


{E3 ,AS2},
{E3 ,AS2},
{E3 ,AS2},
{E3 ,F3},
{E3 ,F3},
{E3 ,F3},
{REST ,F2},
{REST ,F2},

{FS3 ,AS2},
{FS3 ,AS2},
```

```
{E3 ,F3},
{E3 ,F3},
{FS3 ,F3},
{FS3 ,F3},
{REST ,F2},
{REST ,F2},

{E3 ,AS2},
{E3 ,AS2},
{E3 ,AS2},
{E3 ,F3},
{E3 ,F3},
{E3 ,F3},
{E3 ,F2},
{E3 ,F2},

{E3 ,AS2},
{E3 ,AS2},
{E3 ,F3},
{E3 ,F3},
{E3 ,F3},
{E3 ,F3},
{E3 ,F2},
{E3 ,F2},

{CS3 ,AS2},
{CS3 ,AS2},
{CS3 ,AS2},
{CS3 ,F3},
{CS3 ,F3},
{CS3 ,F3},
{REST ,F2},
{REST ,F2},

{D3 ,AS2},
{D3 ,AS2},
{CS3 ,F3},
{CS3 ,F3},
{D3 ,F3},
{D3 ,F3},
{REST ,F2},
{REST ,F2},

{CS3 ,AS2},
{CS3 ,AS2},
{CS3 ,AS2},
{CS3 ,F3},
{CS3 ,F3},
{CS3 ,F3},
{CS3 ,F2},
{CS3 ,F2},

{CS3 ,AS2},
{CS3 ,AS2},
{CS3 ,F3},
{CS3 ,F3},
{CS3 ,F3},
{CS3 ,F3},
{CS3 ,F2},
{CS3 ,F2},

{E3 ,AS2},
{E3 ,AS2},
{E3 ,AS2},
{E3 ,F3},
{E3 ,F3},
{E3 ,F3},
{REST ,F2},
{REST ,F2},

{FS3 ,AS2},
{FS3 ,AS2},
```

```
{E3 ,F3},
{E3 ,F3},
{FS3 ,F3},
{FS3 ,F3},
{REST ,F2},
{REST ,F2},


{E3 ,AS2},
{E3 ,AS2},
{E3 ,AS2},
{E3 ,F3},
{E3 ,F3},
{E3 ,F3},
{REST ,F2},
{REST ,F2},

{D3 ,AS2},
{D3 ,AS2},
{CS3 ,F3},
{CS3 ,F3},
{D3 ,F3},
{D3 ,F3},
{REST ,F2},
{REST ,F2},

{CS3 ,AS2},
{CS3 ,AS2},
{CS3 ,AS2},
{CS3 ,F3},
{CS3 ,F3},
{CS3 ,F3},
{REST ,F2},
{REST ,F2},

{B2 ,AS2},
{B2 ,AS2},
{A2 ,F3},
{A2 ,F3},
{B2 ,F3},
{B2 ,F3},
{REST ,F2},
{REST ,F2},

{A2 ,AS2},
{A2 ,AS2},
{A2 ,AS2},
{A2 ,F3},
{A2 ,F3},
{A2 ,F3},
{A2 ,F2},
{A2 ,F2},

{A2 ,AS2},
{A2 ,AS2},
{A2 ,F3},
{A2 ,F3},
{A2 ,F3},
{A2 ,F3},
{A2 ,F2},
{A2 ,F2},
{END, END}};



const float SHAPE_NOTES[][2] = {
{CS2, CS2},
{CS2, CS2},
{CS2, CS2},
{E2, E2},
{E2, E2},
{E2, E2},
```

```
    {CS2, CS2},
    {CS2, CS2},

    {CS2, CS2},
    {CS2, CS2},
    {CS2, CS2},
    {E2, E2},
    {E2, E2},
    {E2, E2},
    {CS2, CS2},
    {CS2, CS2},

    {CS2, CS2},
    {CS2, CS2},
    {CS2, CS2},
    {E2, E2},
    {E2, E2},
    {E2, E2},
    {CS2, CS2},
    {CS2, CS2},

    {DS2, DS2},
    {DS2, DS2},
    {DS2, DS2},
    {CS2, CS2},
    {CS2, CS2},
    {CS2, CS2},
    {B1, B1},
    {B1, B1},

    {CS2, CS2},
    {CS2, CS2},
    {CS2, CS2},
    {E2, E2},
    {E2, E2},
    {E2, E2},
    {CS2, CS2},
    {CS2, CS2},

    {CS2, CS2},
    {CS2, CS2},
    {CS2, CS2},
    {E2, E2},
    {E2, E2},
    {E2, E2},
    {CS2, CS2},
    {CS2, CS2},

    {CS2, CS2},
    {CS2, CS2},
    {CS2, CS2},
    {E2, E2},
    {E2, E2},
    {E2, E2},
    {CS2, CS2},
    {CS2, CS2},

    {DS2, DS2},
    {DS2, DS2},
    {DS2, DS2},
    {CS2, CS2},
    {CS2, CS2},
    {CS2, CS2},
    {B1, REST},
    {B1, REST},

    {E2, CS2},
    {REST, REST},
    {E2, REST},
    {E2, CS2},
    {E2, REST},
    {E2, REST},
```

```
{REST, CS2},
{E2, REST},

{E2, FS1},
{E2, REST},
{E2, REST},
{REST, FS1},
{E2, REST},
{REST, REST},
{E2, FS1},
{E2, REST},

{E2, AS1},
{E2, REST},
{E2, REST},
{REST, AS1},
{FS2, REST},
{GS2, REST},
{GS2, AS1},
{GS2, REST},

{GS2, B2},
{GS2, REST},
{GS2, REST},
{GS2, B2},
{REST, REST},
{REST, REST},
{REST, B2},
{REST, REST},

{GS2, CS2},
{FS2, REST},
{GS2, REST},
{GS2, CS2},
{REST, REST},
{GS2, REST},
{REST, CS2},
{GS2, REST},

{FS2, FS1},
{FS2, REST},
{FS2, REST},
{GS2, FS1},
{FS2, REST},
{REST, REST},
{FS2, FS1},
{E2, REST},

{FS2, AS1},
{REST, REST},
{FS2, REST},
{FS2, AS1},
{FS2, REST},
{E2, REST},
{E2, AS1},
{CS2, REST},

{CS2, B1},
{CS2, REST},
{CS2, REST},
{CS2, B1},
{REST, REST},
{REST, REST},
{REST, B1},
{CS2, REST},

{GS2, CS2},
{GS2, REST},
{GS2, REST},
{REST, CS2},
{GS2, REST},
{GS2, REST},
```

```
    {GS2, CS2},
    {GS2, REST},

    {GS2, FS1},
    {GS2, REST},
    {GS2, REST},
    {GS2, FS1},
    {REST, REST},
    {GS2, REST},
    {REST, FS1},
    {GS2, REST},

    {B2, AS1},
    {B2, REST},
    {GS2, REST},
    {GS2, AS1},
    {FS2, REST},
    {FS2, REST},
    {E2, AS1},
    {GS2, REST},

    {GS2, B1},
    {GS2, REST},
    {GS2, REST},
    {GS2, B1},
    {FS2, REST},
    {E2, REST},
    {E2, B1},
    {E2, REST},

    {E2, CS2},
    {E2, REST},
    {E2, REST},
    {B2, CS2},
    {B2, REST},
    {GS2, REST},
    {GS2, CS2},
    {GS2, REST},

    {FS2, FS1},
    {FS2, REST},
    {FS2, REST},
    {FS2, FS1},
    {REST, REST},
    {FS2, REST},
    {REST, FS1},
    {FS2, REST},

    {FS2, AS1},
    {REST, REST},
    {FS2, REST},
    {REST, AS1},
    {FS2, REST},
    {REST, REST},
    {E2, AS1},
    {CS2, E2},

    {CS2, B1},
    {GS2, REST},
    {GS2, REST},
    {GS2, B1},
    {B2, REST},
    {CS3, REST},
    {CS3, B1},
    {CS3, REST},

    {CS3, CS2},
    {REST, REST},
    {CS3, REST},
    {REST, CS2},
    {CS3, REST},
    {REST, REST},
```

```
{B2, CS2},
{REST, REST},

{CS3, FS1},
{CS3, REST},
{GS3, REST},
{GS3, FS1},
{FS3, REST},
{FS3, REST},
{FS3, FS1},
{FS3, REST},

{E3, AS1},
{FS3, REST},
{FS3, REST},
{GS3, AS1},
{FS3, REST},
{FS3, REST},
{E3, AS1},
{E3, E2},

{CS3, B1},
{CS3, REST},
{E3, REST},
{E3, B1},
{GS3, REST},
{FS3, REST},
{FS3, B1},
{FS3, REST},


{CS3, CS2},
{CS3, REST},
{CS3, REST},
{CS3, CS2},
{B3, REST},
{FS3, REST},
{GS3, CS2},
{GS3, REST},

{FS3, FS1},
{E3, REST},
{E3, REST},
{REST, FS1},
{E3, REST},
{E3, REST},
{E3, FS1},
{E3, REST},

{E3, AS1},
{FS3, REST},
{FS3, REST},
{GS3, AS1},
{FS3, REST},
{FS3, REST},
{E3, AS1},
{E3, E2},

{FS3, B1},
{FS3, REST},
{E3, REST},
{E3, B1},
{CS3, REST},
{REST, REST},
{CS3, B1},
{REST, REST},

{CS3, CS2},
{CS3, REST},
{CS3, REST},
{REST, CS2},
{CS3, REST},
```

```
{CS3, REST},
{B3, CS2},
{B3, REST},

{CS3, FS1},
{CS3, REST},
{GS3, REST},
{GS3, FS1},
{FS3, REST},
{FS3, REST},
{FS3, FS1},
{FS3, REST},

{E3, AS1},
{FS3, REST},
{FS3, REST},
{GS3, AS1},
{FS3, REST},
{FS3, REST},
{E3, AS1},
{E3, E2},

{CS3, B1},
{CS3, REST},
{E3, REST},
{E3, B1},
{GS3, REST},
{FS3, REST},
{FS3, B1},
{REST, REST},


{CS3, CS2},
{CS3, REST},
{CS3, REST},
{CS3, CS2},
{B3, REST},
{FS3, REST},
{GS3, CS2},
{GS3, REST},

{FS3, FS1},
{E3, REST},
{E3, REST},
{REST, FS1},
{CS3, REST},
{REST, REST},
{CS3, FS1},
{REST, REST},

{B3, AS1},
{FS3, REST},
{GS3, REST},
{GS3, AS1},
{FS3, REST},
{E3, REST},
{E3, AS1},
{E3, E2},

{CS3, B1},
{CS3, REST},
{GS2, REST},
{GS2, REST},
{B2, REST},
{B2, REST},
{CS3, B1},
{CS3, REST},


{CS3, CS2},
{CS3, REST},
{REST, REST},
```

```
{REST, CS2},
{REST, REST},
{REST, REST},
{E3, CS2},
{FS3, REST},

{GS3, FS1},
{GS3, REST},
{FS3, REST},
{E3, FS1},
{E3, REST},
{REST, REST},
{FS3, FS1},
{REST, REST},

{FS3, AS1},
{FS3, REST},
{FS3, REST},
{REST, AS1},
{REST, REST},
{CS3, REST},
{E3, AS1},
{FS3, E2},

{GS3, B1},
{GS3, REST},
{FS3, REST},
{E3, B1},
{E3, REST},
{E3, REST},
{FS3, B1},
{FS3, REST},

{CS3, CS2},
{CS3, REST},
{CS3, REST},
{CS3, CS2},
{REST, REST},
{CS3, REST},
{E3, CS2},
{FS3, REST},

{GS3, FS1},
{GS3, REST},
{CS3, REST},
{CS3, FS1},
{E3, REST},
{E3, REST},
{FS3, FS1},
{FS3, REST},

{FS3, AS1},
{FS3, REST},
{FS3, REST},
{FS3, AS1},
{REST, REST},
{REST, REST},
{E3, AS1},
{FS3, E2},

{GS3, B1},
{GS3, REST},
{FS3, REST},
{E3, B1},
{FS3, REST},
{FS3, REST},
{FS3, B1},
{CS3, REST},

{CS3, CS2},
{CS3, REST},
{CS3, REST},
```

```
{CS3, CS2},
{REST, REST},
{REST, REST},
{E3, CS2},
{E3, REST},

{GS3, FS1},
{GS3, REST},
{B3, REST},
{GS3, FS1},
{FS3, REST},
{FS3, REST},
{E3, FS1},
{E3, REST},

{FS3, AS1},
{FS3, REST},
{FS3, REST},
{FS3, AS1},
{REST, REST},
{CS3, REST},
{E3, AS1},
{FS3, E2},

{GS3, B1},
{GS3, REST},
{FS3, REST},
{E3, B1},
{E3, REST},
{E3, REST},
{CS3, B1},
{FS3, REST},

{CS3, CS2},
{REST, REST},
{CS3, REST},
{CS3, CS2},
{CS3, REST},
{CS3, REST},
{E3, CS2},
{FS3, REST},

{GS3, FS1},
{GS3, REST},
{CS3, REST},
{CS3, FS1},
{E3, REST},
{E3, REST},
{FS3, FS1},
{FS3, REST},

{FS3, AS1},
{FS3, REST},
{FS3, REST},
{FS3, AS1},
{FS3, REST},
{FS3, REST},
{E3, AS1},
{FS3, E2},

{GS3, B1},
{GS3, REST},
{FS3, REST},
{E3, B1},
{FS3, REST},
{FS3, REST},
{FS3, B1},
{CS3, REST},

{CS3, CS2},
{REST, REST},
{CS3, REST},
```

```
{REST, CS2},
{E3, REST},
{REST, REST},
{E3, CS2},
{REST, REST},

{FS3, FS1},
{REST, REST},
{FS3, REST},
{REST, FS1},
{GS3, REST},
{REST, REST},
{GS3, FS1},
{REST, REST},

{GS3, AS1},
{GS3, REST},
{GS3, REST},
{GS3, AS1},
{GS3, REST},
{GS3, REST},
{E3, AS1},
{FS3, E2},

{GS3, B1},
{GS3, REST},
{FS3, REST},
{E3, B1},
{FS3, REST},
{FS3, REST},
{FS3, B1},
{CS3, REST},

{CS3, CS2},
{REST, REST},
{CS3, REST},
{REST, CS2},
{E3, REST},
{REST, REST},
{E3, CS2},
{REST, REST},

{FS3, FS1},
{REST, REST},
{FS3, REST},
{REST, FS1},
{GS3, REST},
{REST, REST},
{GS3, FS1},
{REST, REST},

{GS3, AS1},
{GS3, REST},
{GS3, REST},
{GS3, AS1},
{GS3, REST},
{GS3, REST},
{E3, AS1},
{FS3, E2},

{GS3, B1},
{GS3, REST},
{FS3, REST},
{E3, B1},
{FS3, REST},
{FS3, REST},
{FS3, B1},
{CS3, REST},
{CS3, REST},
{CS3, REST},
{CS3, REST},
{CS3, REST},
```

```
{CS3, REST},
{CS3, REST},
{CS3, REST},
{END, END}};


//99 Luftballoons

const float BALLOONS_NOTES[][2] = {

  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {REST,  E2},
  {FS2,   E2},
  {FS2,   E2},
  {GS2,   E2},
  {E2,  E2},
  {E2,  E2},
  {E2,  E2},
  {GS2,   E2},
  {GS2,   E2},
  {FS2,   CS2},
  {FS2,   CS2},
  {E2,  CS2},
  {CS2,   CS2},
  {CS2,   CS2},
  {CS2,   CS2},
  {CS2,   CS2},
  {CS2,   CS2},
  {A2,  E2},
  {A2,  E2},
  {A2,  E2},
  {A2,  E2},
  {A2,  E2},
  {A2,  E2},
  {A2,  E2},
  {A2,  E2},
  {A2,  FS2},
  {A2,  FS2},
  {GS2,   FS2},
  {FS2,   FS2},
  {FS2,   FS2},
  {E2,  FS2},
  {E2,  FS2},
  {E2,  FS2},
  {FS2,   E2},
  {FS2,   E2},
  {GS2,   E2},
  {E2,  E2},
  {E2,  E2},
  {E2,  E2},
  {GS2,   E2},
  {GS2,   E2},
  {FS2,   CS2},
  {FS2,   CS2},
  {E2,  CS2},
  {CS2,   CS2},
  {CS2,   CS2},
  {CS2,   CS2},
```

```
    {CS2,   CS2},
    {CS2,   CS2},
    {E2,  E2},
    {E2,  E2},
    {CS2,   E2},
    {E2,  E2},
    {E2,  E2},
    {E2,  E2},
    {CS2,   E2},
    {CS2,   E2},
    {E2,  FS2},
    {CS2,   FS2},
    {CS2,   FS2},
    {E2,  FS2},
    {E2,  FS2},
    {E2,  FS2},
    {E2,  FS2},
    {E2,  FS2},
    {FS2,   E2},
    {FS2,   E2},
    {FS2,   E2},
    {GS2,   E2},
    {E2,  E2},
    {E2,  E2},
    {E2,  E2},
    {E2,  E2},
    {FS2,   CS2},
    {FS2,   CS2},
    {E2,  CS2},
    {CS2,   CS2},
    {REST,  CS2},
    {REST,  CS2},
    {REST,  CS2},
    {CS2,   CS2},
    {A2,  E2},
    {A2,  E2},
    {A2,  E2},
    {A2,  E2},
    {A2,  E2},
    {A2,  E2},
    {A2,  E2},
    {A2,  E2},
    {A2,  FS2},
    {A2,  FS2},
    {GS2,   FS2},
    {FS2,   FS2},
    {FS2,   FS2},
    {FS2,   FS2},
    {E2,  FS2},
    {E2,  FS2},
    {FS2,   E2},
    {FS2,   E2},
    {GS2,   E2},
    {E2,  E2},
    {E2,  E2},
    {E2,  E2},
    {GS2,   E2},
    {GS2,   E2},
    {FS2,   CS2},
    {FS2,   CS2},
    {E2,  CS2},
    {CS2,   CS2},
    {CS2,   CS2},
    {CS2,   CS2},
    {CS2,   CS2},
    {CS2,   CS2},
    {E2,  E2},
    {E2,  E2},
    {CS2,   E2},
    {E2,  E2},
    {REST,  E2},
    {REST,  E2},
```

```
    {CS2,   E2},
    {CS2,   E2},
    {FS2,   FS2},
    {FS2,   FS2},
    {FS2,   FS2},
    {FS2,   FS2},
    {FS2,   FS2},
    {FS2,   FS2},
    {E2,  FS2},
    {E2,  FS2},
    {REST,  REST},
    {REST,  REST},
    {REST,  REST},
    {REST,  REST},
    {REST,  REST},
    {REST,  REST},
    {REST,  REST},
    {REST,  REST},
    {E2,  REST},
    {REST,  REST},
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {E2,  REST},
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {G2,  REST},
    {GS2,   REST},
    {FS2,   REST},
    {E2,  REST},
    {REST,  REST},
    {E2,  REST},
    {CS2,   REST},
    {E2,  REST},
    {REST,  REST},
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {E2,  REST},
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {E1,  REST},
    {E1,  REST},
    {E1,  REST},
    {E1,  REST},
    {REST,  REST},
    {E1,  REST},
    {CS2,   REST},
    {E2,  REST},
    {REST,  REST},
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {E2,  REST},
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {G2,  REST},
    {GS2,   REST},
    {FS2,   REST},
    {E2,  REST},
    {REST,  REST},
    {E2,  REST},
    {CS2,   REST},
    {E2,  REST},
    {REST,  REST},
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {E2,  REST},
```

```
    {E2,  REST},
    {CS2,   REST},
    {CS2,   REST},
    {E1,  REST},
    {E1,  REST},
    {E1,  REST},
    {E1,  REST},
    {REST,  REST},
    {B2,  REST},
    {CS3,   REST},
    {E3,  REST},
    {REST,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {E3,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {G3,  REST},
    {GS3,   REST},
    {FS3,   REST},
    {E3,  REST},
    {REST,  REST},
    {E3,  REST},
    {CS3,   REST},
    {E3,  REST},
    {REST,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {E3,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {B2,  REST},
    {B2,  REST},
    {B2,  REST},
    {B2,  REST},
    {REST,  REST},
    {B2,  REST},
    {CS3,   REST},
    {E3,  REST},
    {REST,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {E3,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {G3,  REST},
    {GS3,   REST},
    {FS3,   REST},
    {E3,  REST},
    {REST,  REST},
    {E3,  REST},
    {CS3,   REST},
    {E3,  REST},
    {REST,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {E3,  REST},
    {E3,  REST},
    {CS3,   REST},
    {CS3,   REST},
    {B2,  REST},
    {B2,  REST},
    {B2,  REST},
    {B2,  REST},
    {REST,  REST},
```

```
    {B2,  REST},
    {B2,  REST},
    {B2,  REST},
    {FS2,   E1},
    {FS2,   E1},
    {FS2,   E1},
    {FS2,   E1},
    {GS2,   E1},
    {GS2,   E1},
    {E2,  E1},
    {E2,  E1},
    {E2,  E1},
    {E2,  E1},
    {E2,  E1},
    {E2,  E1},
    {GS2,   E1},
    {GS2,   E1},
    {GS2,   E1},
    {GS2,   E1},
    {FS2,   FS1},
    {FS2,   FS1},
    {FS2,   FS1},
    {FS2,   FS1},
    {E2,  FS1},
    {E2,  FS1},
    {CS2,   FS1},
    {CS2,   FS1},
    {CS2,   FS1},
    {CS2,   FS1},
    {CS2,   FS1},
    {CS2,   FS1},
    {REST,  FS1},
    {REST,  FS1},
    {CS2,   FS1},
    {CS2,   FS1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {REST,  A1},
    {REST,  A1},
    {REST,  A1},
    {REST,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  A1},
    {A2,  E1},
    {A2,  E1},

    {END, END}};







const float CLOSER_NOTES[][2] = {
    {AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
```

```
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{C4,C4},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
```

```
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{C4,C4},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{C4,C4},
{C4,C4},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{AS3,AS3},
{GS3,GS3},
{GS3,GS3},
{AS3,AS3},
{END, END}};




const float YMCA_NOTES[][2] = {

{B2, C2 },
{A2, C2 },
{G2, C2 },
{A2, C2 },
{B2, C2 },
{D3, C2 },
{D3, C2 },
{REST, C2 },
{B2, C2 },
{D3, C2 },
{D3, C2 },
{REST, C2 },
{E3, C2 },
{E3, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
```

```
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{B2, C2 },
{A2, C2 },
{G2, C2 },
{A2, C2 },
{B2, C2 },
{D3, C2 },
{D3, C2 },
{REST, C2 },
{B2, C2 },
{D3, C2 },
{D3, C2 },
{REST, C2 },
{E3, C2 },
{E3, C2 },
{C3, C2 },
{C3, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{C3, C2 },
{B2, C2 },
{A2, C2 },
{B2, C2 },
{C3, C2 },
{E3, C2 },
{E3, C2 },
{REST, C2 },
{G3, C2 },
{E3, C2 },
{E3, C2 },
{REST, C2 },
{FS3, C2 },
{FS3, C2 },
{FS3, C2 },
{REST, C2 },
{REST, C2 },
{E3, C2 },
{E3, C2 },
{E3, C2 },
{REST, C2 },
{REST, C2 },
{D3, C2 },
{D3, C2 },
{D3, C2 },
{D3, C2 },
{REST, C2 },
{C3, C2 },
{C3, C2 },
{C3, C2 },
{C3, C2 },
{REST, C2 },
{REST, C2 },
{B2, C2 },
{B2, C2 },
{B2, C2 },
{A2, C2 },
{A2, C2 },
{REST, C2 },
{D3, C2 },
{B2, C2 },
{B2, C2 },
{REST, C2 },
{REST, C2 },
```

```
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{REST, C2 },
{B2, C2 },
{A2, C2 },
{G2, C2 },
{A2, C2 },
{B2, C2 },
{D3, C2 },
{D3, C2 },
{D3, C2 },
{B2, C2 },
{D3, C2 },
{D3, C2 },
{REST, C2 },
{E3, C2 },
{B2, C2 },
{END, END}};
```

```
from mido import MidiFile

note_encodings = {
        0: "REST",
        28: "E1",
        29: "F1",
        30: "FS1",
        31: "G1",
        32: "GS1",
        33: "A1",
        34: "AS1",
        35: "E1",
        36: "C2",
        37: "CS2",
        38: "D2",
        39: "DS2",
        40: "E2",
        41: "F2",
        42: "FS2",
        43: "G2",
        44: "GS2",
        45: "A2",
        46: "AS2",
        47: "B2",
        48: "C3",
        49: "CS3",
        50: "D3",
        51: "DS3",
        52: "E3",
        53: "F3",
        54: "FS3",
        55: "G3",
        56: "GS3",
        57: "A3",
        58: "AS3",
        59: "B3",
        60: "C4",
        61: "CS4",
        62: "D4",
        63: "DS4",
        64: "E4"
}

midifile_name = 'pirates.mid'
track_num = 1

mid = MidiFile(midifile_name)
f = open("temp.txt", "w+")

motor_notes = [0, 0]
```

```
motor_on = [0, 0]
deltat_skipped = 0
deltat_per_sixteenth = mid.ticks_per_beat/4;

for i, track in enumerate(mid.tracks):
    if i == track_num:
        print('Track {}: {}'.format(i, track.name))
        for msg in track:
            if (msg.type == 'note_on'):
                if not motor_on[0]:
                    motor_notes[0] = msg.note
                    motor_on[0] = 1
                elif not motor_on[1]:
                    motor_notes[1] = msg.note
                    motor_on[1] = 1
                else:
                    deltat_skipped += msg.time
            elif (msg.type == 'note_off'):
                if(msg.note == motor_notes[0] or msg.note == motor_notes[1]):
                    for i in range((msg.time+deltat_skipped)/deltat_per_sixteenth):
                        f.write("{\"" + note_encodings[motor_notes[0]] + "\", \"" +
                                note_encodings[motor_notes[1]] +  "\"},\n")

                    if(msg.note == motor_notes[0]):
                        motor_notes[0] = 0
                        motor_on[0] = 0
                    elif (msg.note == motor_notes[1]):
                        motor_notes[1] = 0
                        motor_on[1] = 0
                    deltat_skipped = 0
                else:
                    deltat_skipped += msg.time

            print(msg)

        f.write("{\"END\", \"END\"}\n")
        f.close()
```