Software Overview

Year: 2017 **Semester:** Spring **Team:** 12 **Project:** Guitutar
Creation Date: January 27, 2017 **Last Modified:** January 31, 2017
Author: Cole Giannotti **Email:** agiannot@purdue.edu

Assignment Evaluation:

Item	Score (0-5)	Weight	Points	Notes
Assignment-Specific Items				
Software Overview		x2		
Description of Algorithms		x2		
Description of Data Structures		x2		
Program Flowcharts		x3		
State Machine Diagrams		x3		
Writing-Specific Items				
Spelling and Grammar		x2		
Formatting and Citations		x1		
Figures and Graphs		x2		
Technical Writing Style		x3		
Total Score				

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:
Relevant overall comments about the paper will be included here

1.0 Software Overview

The Guitutar calls for a primarily firmware oriented implantation. Firmware on a microcontroller will handle the reading of songs, powering up LEDs, calculating user accuracy, and communicating with the remote user control via Bluetooth. The controls will be implemented in a state machine manner, performing different tasks depending on the state of the information given and the user's interactions. An array of songs will be stored, in which the user can select from to play. Once a song is selected, the user will be prompted to select and play mode. The learning mode will repeatedly check for the correct note to be played before moving on to the next note. The tempo mode will display each note to the beat of the song and will not wait for the user to play any notes. These modes will be controlled through if statements keeping track of one mode in an on/off manner.

The microcontroller will power the LEDs for a specific chord when read. Depending on the mode of operation, the firmware will either specify a delay in time before the next note is to be shown or wait for the user to play the current note. These modes will be defined by a singular Boolean variable where false depicts learning mode is active and true depicts tempo mode is active.

2.0 Description of Algorithms

Flow charts for all algorithms can be found in the Appendix 1 of this document.

2.1 Running Songs

A function purely devoted to running operations within the duration of a song. It will read in the notes to be played/displayed and check for accuracy of the user. Depending on the mode of Guitutar, this function will either wait for the user to properly play each note or move along the notes of a song within the beat of the song keeping track of how well the user is performing.

2.2 Reading Notes

Notes will be read in as an array of frets followed by a delay value. Bytes will be taken and processed one at a time. Each byte with be decoded into a string and fret combination (the encoding process is described in Section 3.1 of this document) by masking with an AND operator and bit shifts. The fret/delay information will be taken by applying a binary 00011111 mask to the byte, which isolates the last 5 bits. Then a binary bit right shift of 5 will make the 3 most significant bits now the least significant bits and bring in leading zeros [1]. This will get the value of the string/delay indicator/end indicator. The bytes will be read in and stored in a 7-element array. The first 6 elements are assigned to the 6 strings of the guitar and the last element will be used to store how much time should be delayed before playing this chord. Zeros in the array indicate that the string should not be played for this chord while frets out of bound will indicate that the string should be played open (or without any frets being pressed down).

2.3 Checking What Notes Were Played

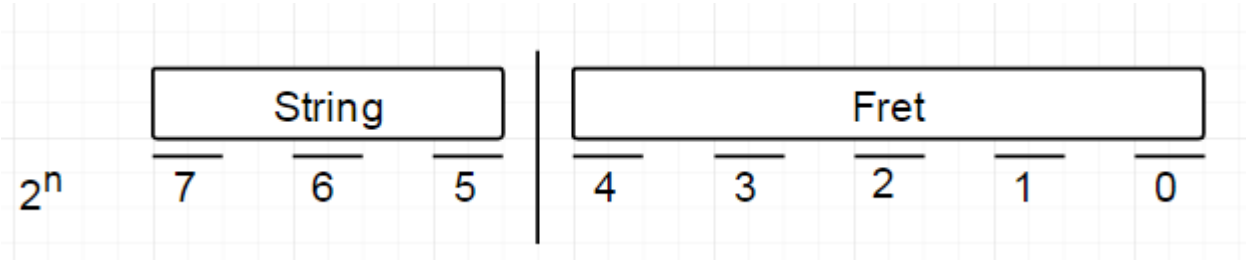
An algorithm will be written to see which strings are pressed against which frets when a note is played. The algorithm uses the knowledge that the hardware implements a grid matrix of rows (strings) and columns (frets). The hardware knowledge will be used to send out a signal through a guitar string and check to see which fret returns the signal. Because the microcontroller can run

at extremely high speeds compared to a human, strings will not be tested in parallel but instead chained one after another. This will guarantee uniqueness of all inputs therefore remove aliasing of similar or reversed chords. An example: (String 1: Fret 1) and (String 2: Fret 2) would be indistinguishable from (String 1: Fret 2) and (String 2: Fret 1) if signals were sent down the strings in parallel.

3.0 Description of Data Structures

3.1 Note Encoding

The encoding algorithm being used is designed to reduce the space required to store notes of a song. Each string and fret combination will be reduced down to a single byte (8 bits) of data. By splitting the bits into groups of 3 and 5, the space needed to range all possible options of strings and of frets is covered. The first three bits of the byte will be dedicated to define the string the note is played on. With 3 bits of data, the range is from 0 to 7 allowing for 8 options. The values of 0 to 5 will be used to indicate the 6 strings of the guitar leaving 6 and 7 unused. The values of 6 and 7 will be assigned special meanings to indicate a delay (for beat purposes) and end character respectively. The last 5 bits range from 0 to 31 allowing for 32 options. Having 32 options covers the 22 frets on the guitar. This also allows for 32 steps of time that can be used to delay when the note should be played. This is more than enough to cover almost all common beats. However, this range will not be sufficient for songs with periods of rest that are 32 times longer than the smallest interval between notes.



Within the stored data, a sequence of bytes followed by an end character will represent a chord. If a string is not found within the grouping, then that string is not to be played for that chord. This omission method minimizes storage required again by removing space wasted by storing the lack of information.

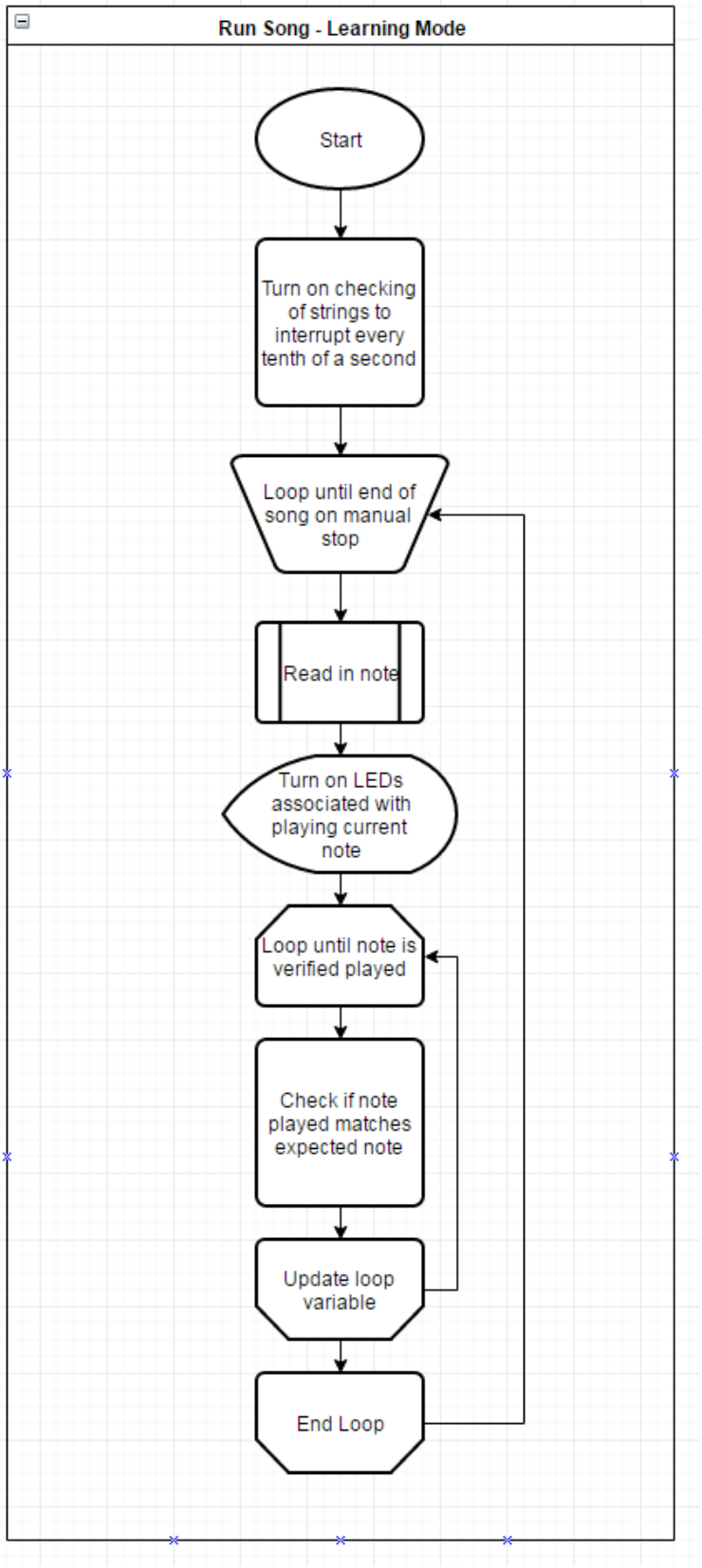
3.2 Song Structure

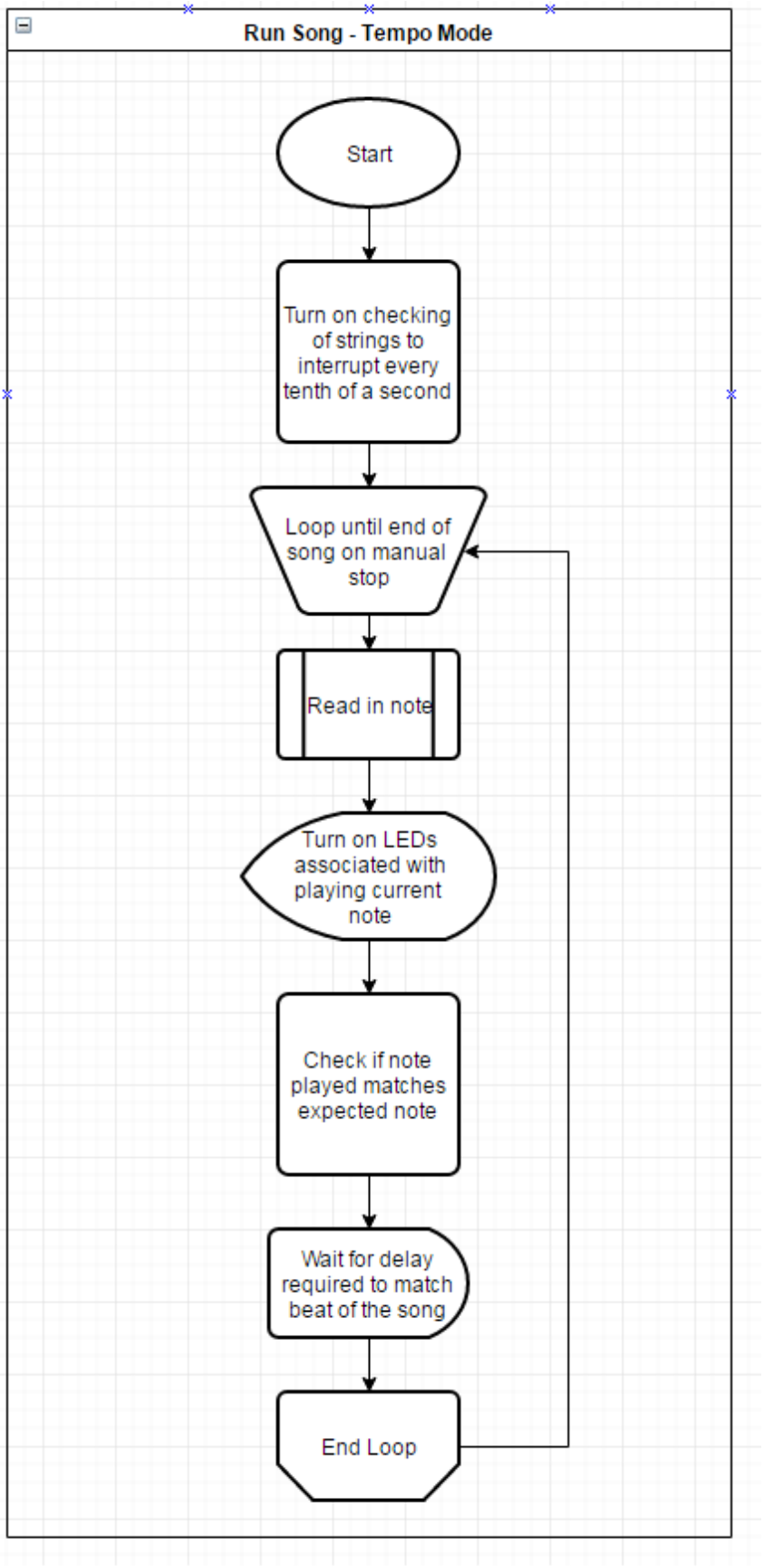
A defined data structure will be used to store song information. This structure will contain three elements: the song's name, the default tempo, and a pointer to the location where notes data is stored. There will be an array of these structures used to store all available songs.

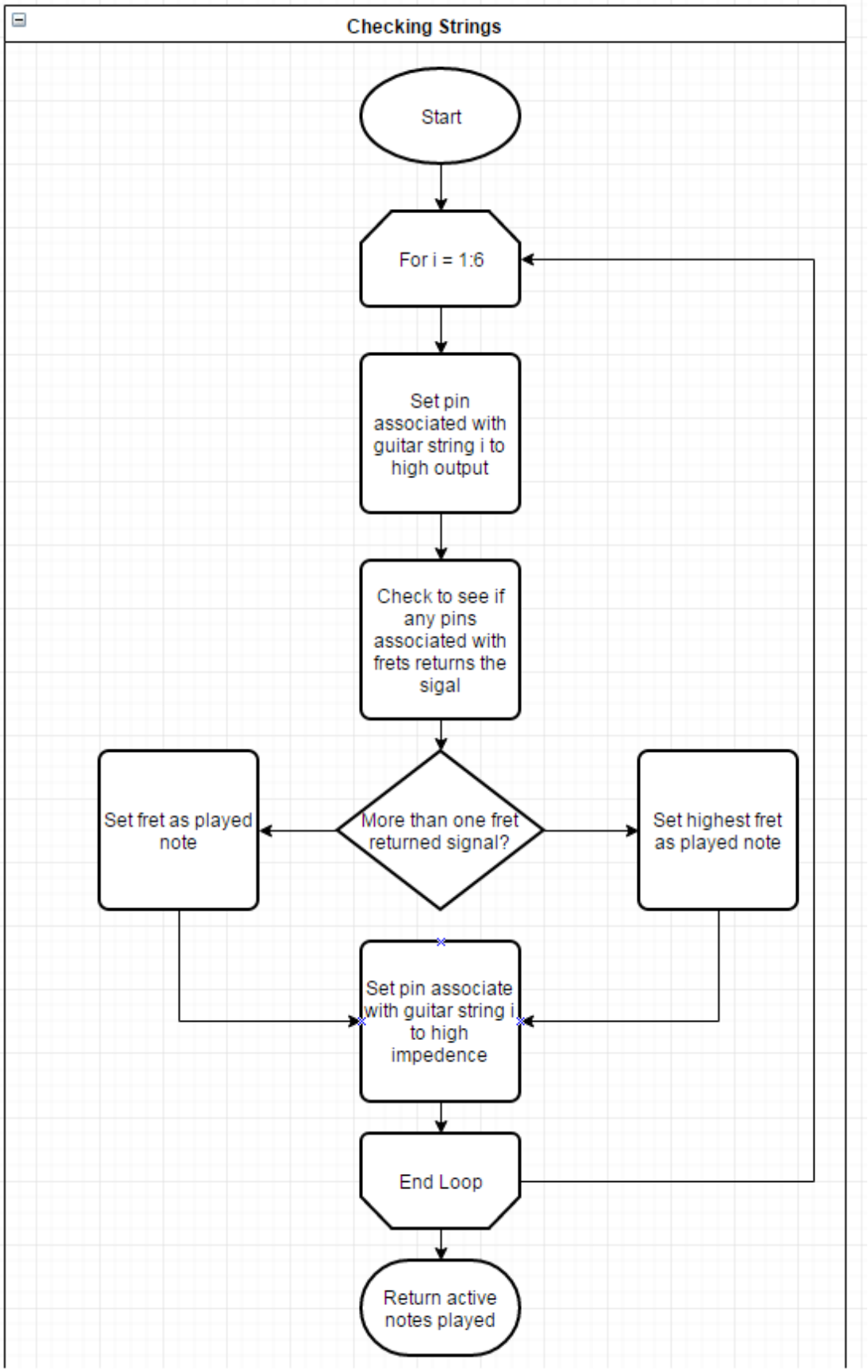
4.0 Sources Cited:

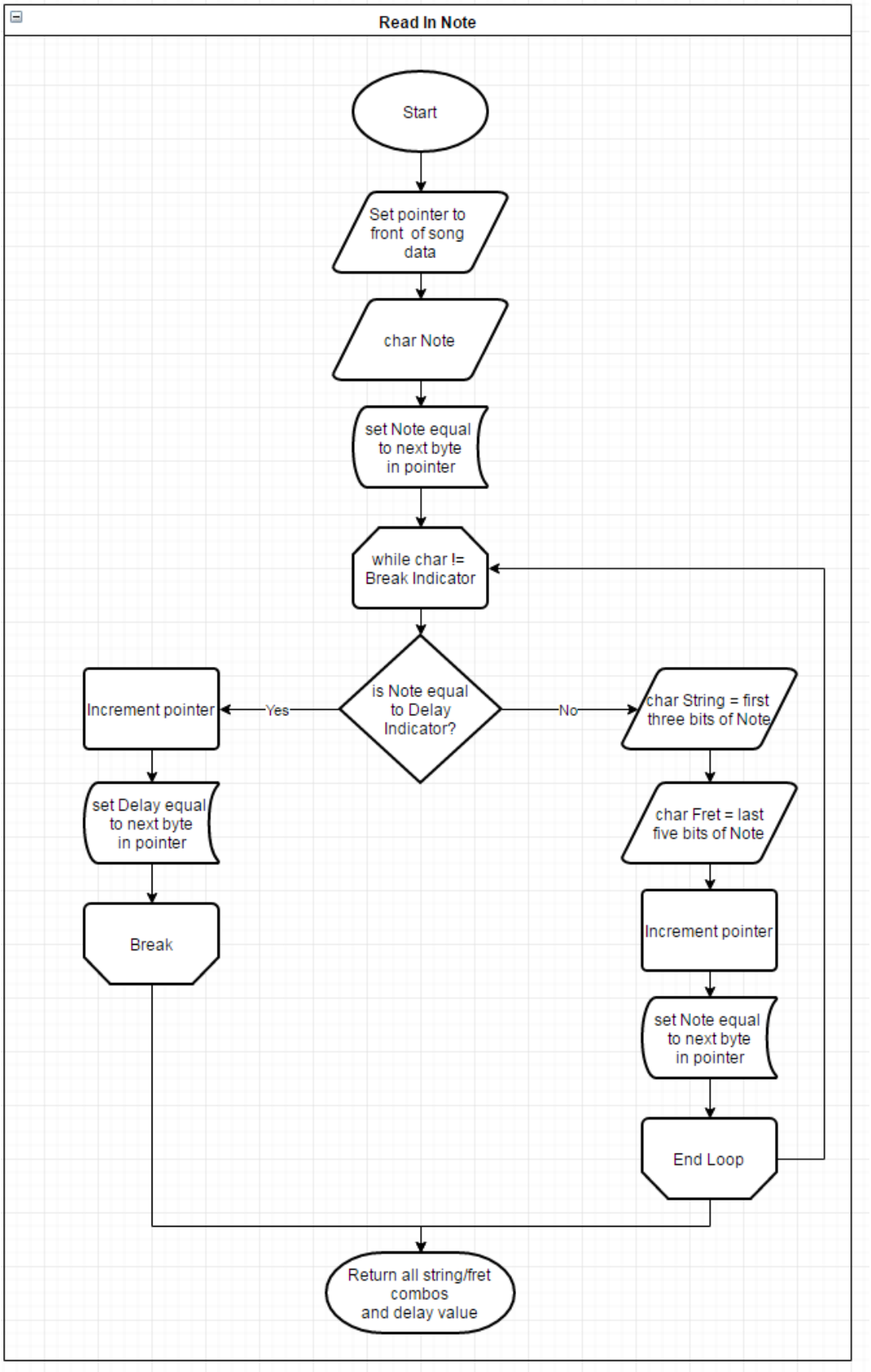
[1] C. Lin. (2003). *Bitshift Operators* [Online Lecture Notes]. Available: https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/BitOp/bitshift.html

Appendix 1: Program Flowcharts




Run Song - Tempo Mode
Start
Turn on checking of strings to interrupt every tenth of a second
Loop until end of song on manual stop
Read in note
Turn on LEDs associated with playing current note
Check if note played matches expected note
Wait for delay required to match beat of the song
End Loop


Checking Strings
Start
For i = 1:6
Set pin associated with guitar string i to high output
Check to see if any pins associated with frets returns the sigal
More than one fret returned signal?
Set fret as played note
Set highest fret as played note
Set pin associate with guitar string i to high impedence
End Loop
Return active notes played


Read In Note
Start
Set pointer to front of song data
char Note
set Note equal to next byte in pointer
while char != Break Indicator
is Note equal to Delay Indicator?
Yes
Increment pointer
set Delay equal to next byte in pointer
Break
No
char String = first three bits of Note
char Fret = last five bits of Note
Increment pointer
set Note equal to next byte in pointer
End Loop
Return all string/fret combos and delay value

Appendix 2: State Machine Diagrams