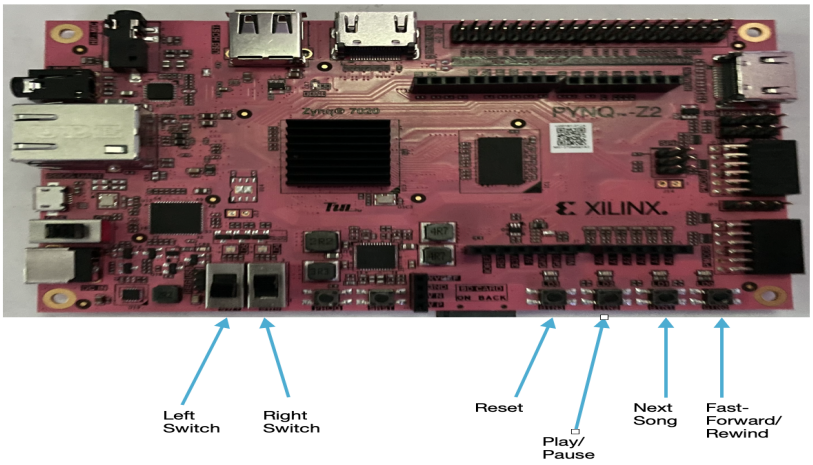
Dejon Jones
Daniel Garcia Lopez
Omar Ramos

EE108 Final Project

Music Player (New Features)



The left switch controls how much screen be used the (full width of the screen or the normal width)
The right switch adjusts the height of the signal

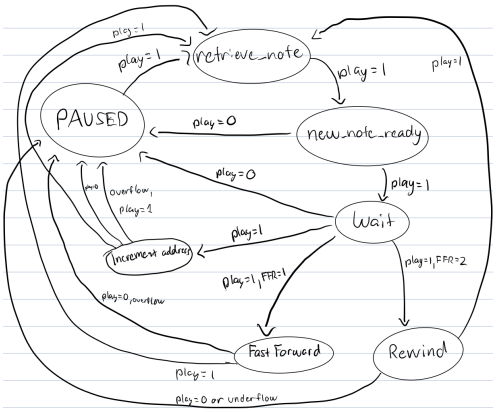
For our final project, our group decided to implement the following features: Playback control, Chords, Harmonics, and Adjustable waveform display.

Playback Control (4 points):

One of our first implementations was an interface that allowed our basic music player to now fast- forward and rewind the current playing song. To implement this feature, we decided that in order to play a song in fast forward mode we cut the duration of the note given by the song ROM in half. This would allow the note to play at 2x speed. When playing back at normal speed, we increment and use the note duration as usual. To play our song in rewind mode, we would decrement the note counter by -1 instead of +1. This would make sure that we are getting the address before next and not the address after next. Since checkpoint 1, we realized that

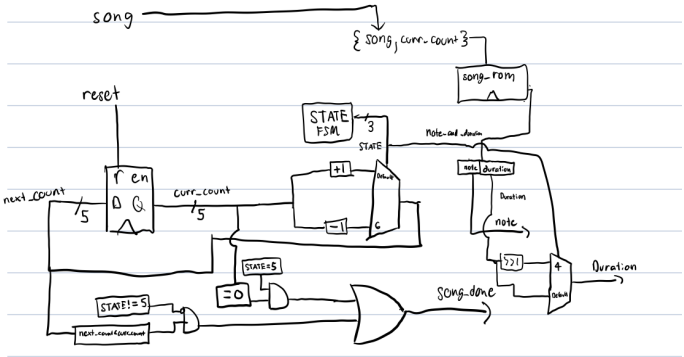
this does not allow for the perfect playback of songs with chords that have varying duration for their notes. To fix this we wrote our song ROM in a way that would allow us to play the song perfectly forwards or backwards. Each rest in the song can either be a normal rest or a rewind rest, neither of which are read when not in normal/FF playback mode or rewind mode respectively. These changes allow the songs to be played perfectly when rewinded. These are the FSM we used to map how pressing the FF button works and the block diagram for how the note duration changes depending on the playback_state.

Finite State Machine



Our default state would be a paused song until the play button is pressed (goes high). Our song will then play at normal speed. The song will pause if the play button goes high again. If the fast forward button is pressed, the song will then decrease all wait times (rests) by a factor of 2. If the fast forward button is pressed again, the machine will move into the rewind state where the note counter is decremented as opposed to incremented every beat. In order to exit the fast forward state or the rewind state, the song must either rewind to the beginning (underflow of counter), be reset, reach the end of the song (overflow), or the user must play the pause button. All of these scenarios put the machine in the pause state.

Block Diagram

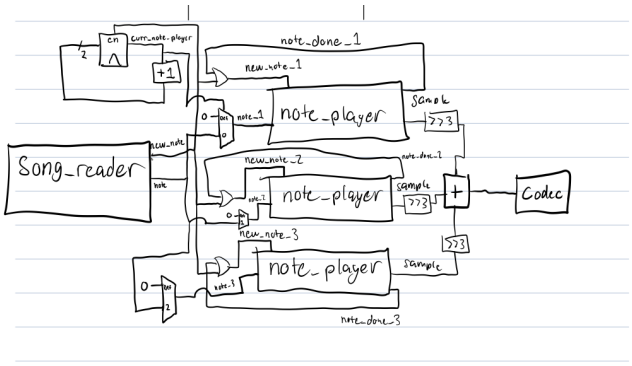


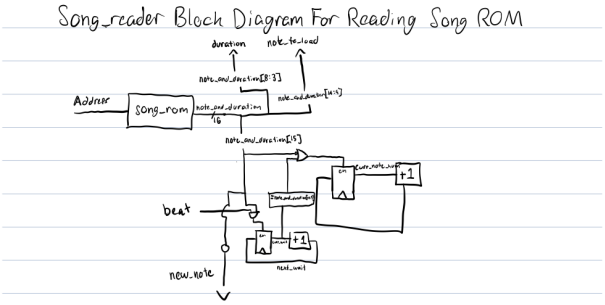
Chords(4 points):

Our next feature added to our basic music player was allowing our music player to play chords instead of only playing notes. Originally, our music player would only allow our song to play if one note is being played at a

time. Now, our music player is capable of playing three notes at the same time, resulting in a chord. We used the following block diagrams to implement this feature:

Block Diagrams

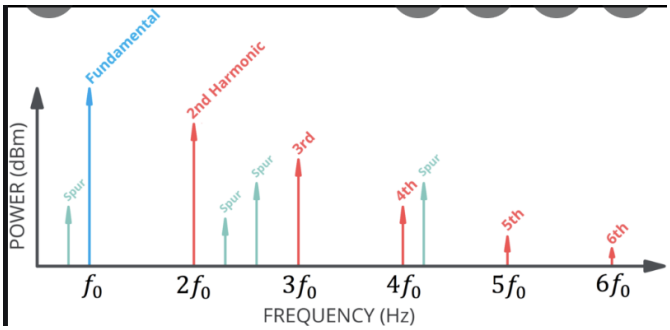




The block diagram on the left describes the changes we made to the music player. The diagram shows the instantiation of multiple note players (in order to play multiple notes at the same time) and the sum of their sample outs in order to produce the combined waveform. The diagram also describes the logic used to assign each note player its note_to_load so that the song is played properly without hiccups. The curr_note_player_num describes which note player will receive which note, and it is enabled by the new_note signal outputted by the song_reader. The diagram on the right shows the changes we made to song_reader in order to support the advancing of the song. We wired the beat signal to the song_reader so that when prompted to rest (advance the song), the song_reader would be able to rest without outputting any notes to any of the note_players. Once the given duration has passed, the song_reader will increment note_count_num to the next note it should enqueue.

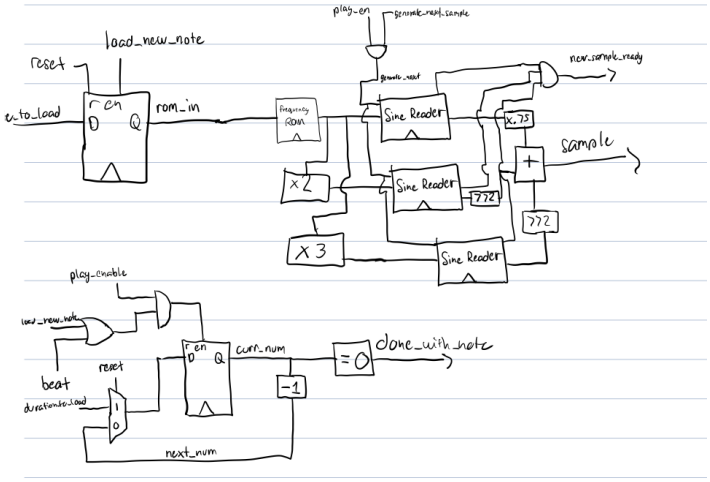
Harmonics (3 points):

The third feature that we decided to add to our basic music player was harmonics. Similar to chords, we are allowing our music player to now play more complex sounds in a song than just a single note at a time. Unlike chords, we are now adding weighted harmonics to the note being played which can possibly give different instrument sounds if done correctly. Overall, this feature is very similar to chords as we are combining different waveforms to achieve a harmonic sound. The main difference is that the waveforms that we are combining are all weighted multiples of the inputted base frequency (note_to_load). The picture below perfectly visualizes what is going on. Our base frequency is *f*o and the 2nd and 3rd harmonics are represented at 2*f*o and 3*f*o (multiples of the base frequency) and are being played at lower dBm (amplitude/volume).



The following block diagram was used to help us implement this feature:

Block Diagram



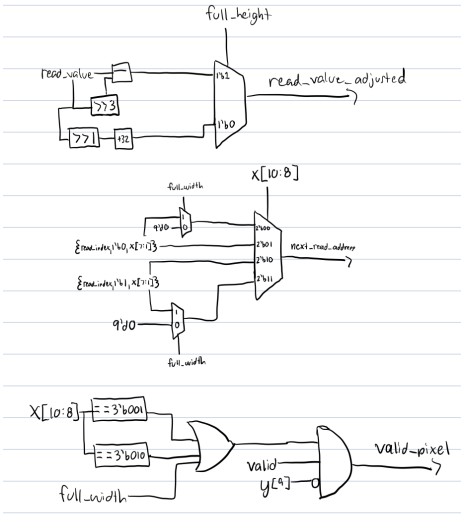
The block diagram shows the changes we made to note_player in order to support the playing of harmonics. The diagram shows how the step_size output of the frequency ROM is manipulated in order to create three multiples of the base frequency (x1, x2, x3). The step_sizes are then fed to their own respective sine_readers (we now have three instantiations to support parallel composition of waveforms). The output of those sine_readers are then summed together, but before that they are multiplied by their weights in order to compose the proper harmony. The new_smaple_ready signal that goes to the codec is the output of the AND gate that takes in new_sample_ready from each sine_reader.

Advanced waveform display (1 point):

The last new feature that we added to our basic music player was an advanced waveform controlled display. This feature allows us to adjust how the waveform of the song is going to be displayed while the song is playing. Our waveform will be adjusted based on the two input switches that we added. The left switch will

determine if the screen will use the full width of the screen or the normal width. When using the normal width, our waveform will only be allowed to write in the 2nd and 3rd quadrants of our screen, but will be able to also write in the 1st and 4th quadrants when using full width mode. The right switch works similarly to the left switch, but is responsible for adjusting the height of the signal. Originally, the signal will be displayed in the top half of the screen. When toggled, the signal will then be able to take almost the full height of the screen (about 90%). We used the following block diagram to implement this feature:

Block Diagram



In this diagram we describe the logic that is used to determine read_adjusted_value (when controlling the height of the waveform). When the switch is high, the reda_adjusted_value becomes 7/8 of its normal value, and when low it is 50% of its normal value. The width control works by allowing signals in the first and fourth quadrant to be recognized as valid pixels to the VGA, allowing the waveform to display across the entire width of the screen. When the width switch is low, the first and fourth quadrants are ignored as usual and only half the screen's width is used.