Abi Lopez and Feliciano Cortes

EE108: Final Project
Professor Subhasish Mitra
Winter 2022

Our project implements the extensions Chords for 4 points, Harmonics for 3 points, Stereo Effects for 1 point, and Enhanced waveform display for 1 point.

Specification:

To handle the Stereo Effects extension, I created a signal stereo to tell final_top.v which headphone to use to play the song. To handle the Chords and Harmonics extensions, we created a chord_player.v file and harmonics_player.v file which are logically similar to note_player and replaced note_player.v . To handle the Enhanced waveform display extension, we added a wave_module.v file which is logically similar to wave_display_top.v and calls wave_display_top.v multiple times to display different notes. Our note memory is {advance, note[5:0], duration[5:0], stereo_left, stereo_right, harmony} a total of 16 bits. The stereo_left and stereo_right metadata specify what headphone to play on, and harmony tells Harmony Player to harmonize the note.

High_level design & Key implementation details:

The block diagram for **music_player.v** Music Player instantiates chord_player in the place of note_player and sends the chord_sample to codec. Music Player outputs the samples for display and the new stereo signal. To see if chords and harmonize were functioning correctly, we slightly modified music_player_tb by increasing the delay to see Song 0 longer. We looked at display_sample and added the Chord Player samples. We made sure display_sample included the correct samples from Chord Player (sample_note1, sample_note2, sample_note3, and sample_chord). Visually, we also checked that sample_chord looked like a Sawtooth wave when harmonize was high. Also, see the timing diagram between song_reader, chord_player, and note_player.

The finite state machine for **song_reader.v** Song Reader receives chord_done signal in the place of note_done signal, and the WAITCHORDDONE state replaces the WAITNOTEDONE state. Functionally, the two signals and states are the same. In addition to the new_note, note, and duration signals, Song Reader also sends advance, harmonize, and stereo signals it reads from the Song Rom to Music_Player. We slightly modified song_reader_tb to test for chord_done instead of note_done. We looked at the state and made sure the state was INCREMENTNOTE 3'd4 after WAITCHORDDONE.

The block diagram for **chord_player.v** Chord Player receives notes from Song reader and schedules them in harmony_players. When Chord Player receives the advance signal, it counts the advance duration_to_load, and then sends chord_done when the count reaches 0. To schedule the notes, Chord Player looks at which note_player (1, 2, or 3) is free to receive a note. Chord Player sends the harmonize signal and note_samples to harmony_players. Harmony_Players each play a note_sample to form the chord_sample. Chord Player sends all the note_samples and the chord_sample inside display_sample to Music_Player. We tested Chord Player using similar logic to note_player_tb and sent one note followed by an advance or three notes followed by an advance. We looked at the signal sample_chord and made sure it was the sum of the three note samples (sample_out1, sample_out2, sample_out3).

The block diagram for **harmony_player.v** Harmony Player sends the note to frequency rom and gets the step_size1 for the fundamental frequency. Harmony Player receives the harmonize signal and if harmonize is high, then it increases the step_size for the 3 other harmonics. Harmony Player then sends step_size1, step_size2, step_size3, and step_size4 to 4 sine_readers to get 4 samples. The 4 samples are then shifted and added to create the harmonized sample. We tested Harmony Player using similar logic to note_player_tb and set the harmonize signal high and low. We looked at the sample_outs (sample_out1, sample_out2, sample_out3, sample_out4) of the sine_readers in Harmony Player and made sure sine_readers2-4 were only called when harmonize was high. We also checked that the resulting sample_out should be the sum of all the sample_outs (sample_out1, sample_out2, sample_out3, sample_out4) when harmonize is high.

The block diagram for **final_top.v** Final Top sends the play, reset, next, and new_frame signals to music_player and then receives the new_sample_generated signal, and the samples to display, and stereo signal. Final Top then flops the signals, sends the flopped chord sample (display_sample[15:0]) and the stereo signal to Adau1761 to tell it which headphone to play on. Final Top also sends the flopped_new_sample and flopped_display_sample to Wave_Module. We synthesized our design to make sure everything was working as expected.

The block diagram for **wave_module.v** Wave Module gets the samples to display and sends each of the four samples to a Wave_Display_Top. The Wave_Display_Top will get a valid_pixel to display the sample. Note 1 is going to be a red wave, Note 2 a blue wave, Note 3 a green wave, and Chord a white wave, so we send each valid_pixel to r, g, or b accordingly. In this way, we are displaying four waves in the same display.

The block diagram for **wave_display_top.v** Wave Display Top sends a sample to Wave_Capture, which “records” the sample in the RAM. Wave_Display reads the samples to display from the RAM and sends valid_pixel to Wave_Module.

The finite state machine for **wave_capture.v** Wave_Capture saves the note samples in the RAM.

The block diagram for **wave_display.v** Wave_Display checks the prev sample and current sample to see if the pixel should be displayed and sends valid_pixel accordingly. We modified the previous wave_display_tb and made sure valid_pixel was sent when expected. Also, see the timing diagram for Wave Display.

Problems that we encountered and handled.

Lowering WNS and TNS:
WNS was -2.826 and TNS was -894.343, so we added a flip-flop to flop new_frame, new_sample, and display_sample. We also modified Adau1761 to receive the flopped chord sample from the flopped display sample, and modified Wave Module to receive the flopped_new_sample and flopped_display_sample. These changes helped drop the WNS to around -2.2ns and the TNS to around -80ns.

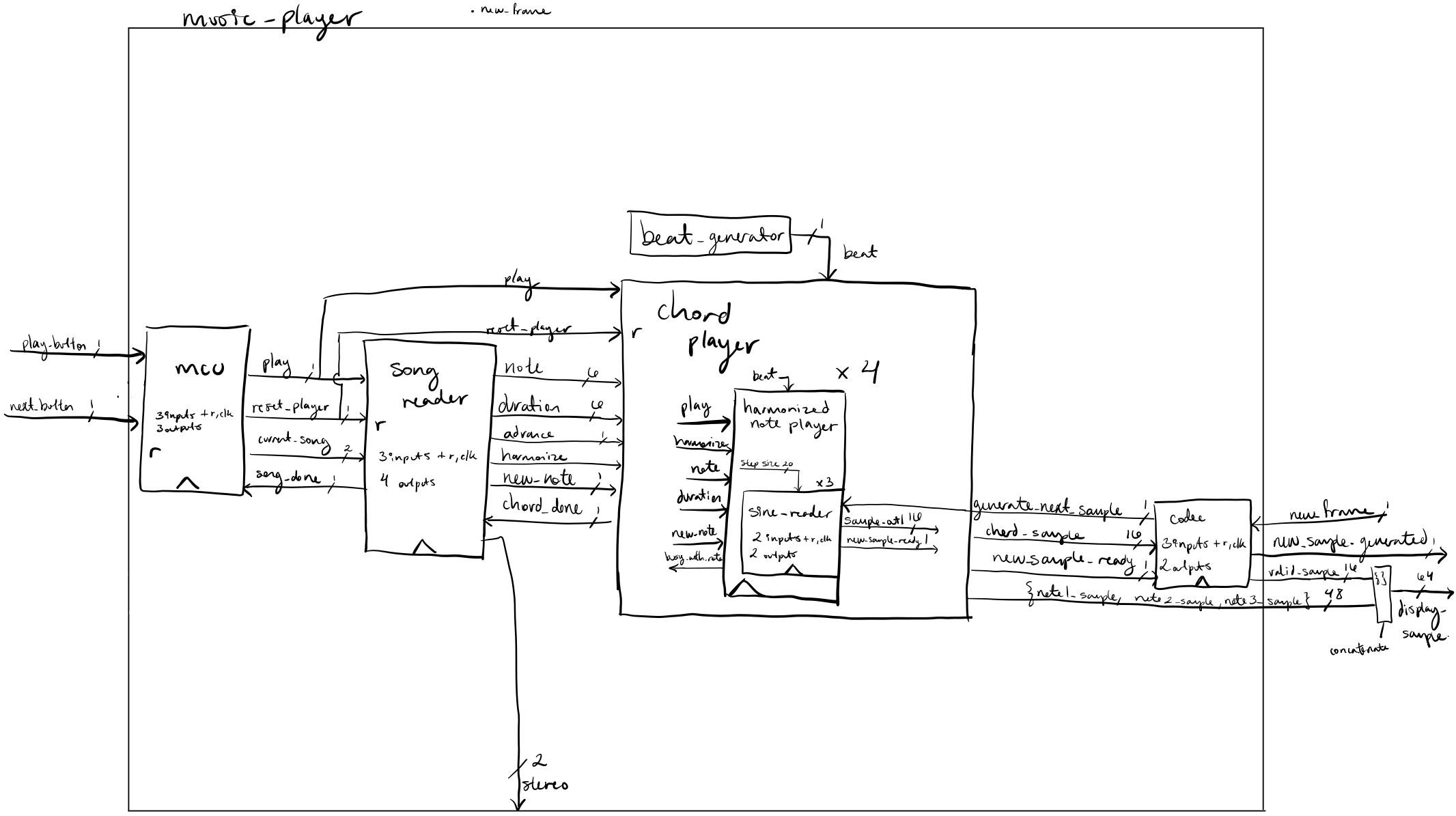
Display showing Black after Wave_Module:
Everything was wired correctly, wire sizes consistent, but we used a boolean OR operator instead of a bitwise OR operator in the assign r,g,b code. After correcting the error, the waves showed up as expected. Unfortunately, this tiny syntax error took more time than expected to catch.

Codec and r,g,b Errors:
When developing wave_module, I received an error saying I was driving the Adau codec and r,g,b in Final_Top multiple times. The problem was I used a codec for every note, and then realized I only needed to send the chord_sample to the codec. Initially I was sending every r,g,b from Wave_Display_Top directly to the r,g,b in Final_Top, but then realized I needed to combine the r,g,b's so I created the valid_pixel signal to tell us what r,g,b to color.

Chord Player Timing:
Chord Player was not sending out the samples at the expected time. To solve this issue, in Song Reader we decided to send continuous notes until an advance note was seen. At which point, we enter the WAITCHORDDONE state. In Chord Player we schedule the notes and play for the advance time and then send chord_done. Once Song Reader received chord_done, it entered the INCREMENTNOTE state to continue sending notes. With this logic, we were able to coordinate the timing in Chord Player and receive notes and send samples appropriately.

input
- clk
- reset
- play_button
- next_button
- new_frame

output
- new_sample_generated
- sample_display
- stereo



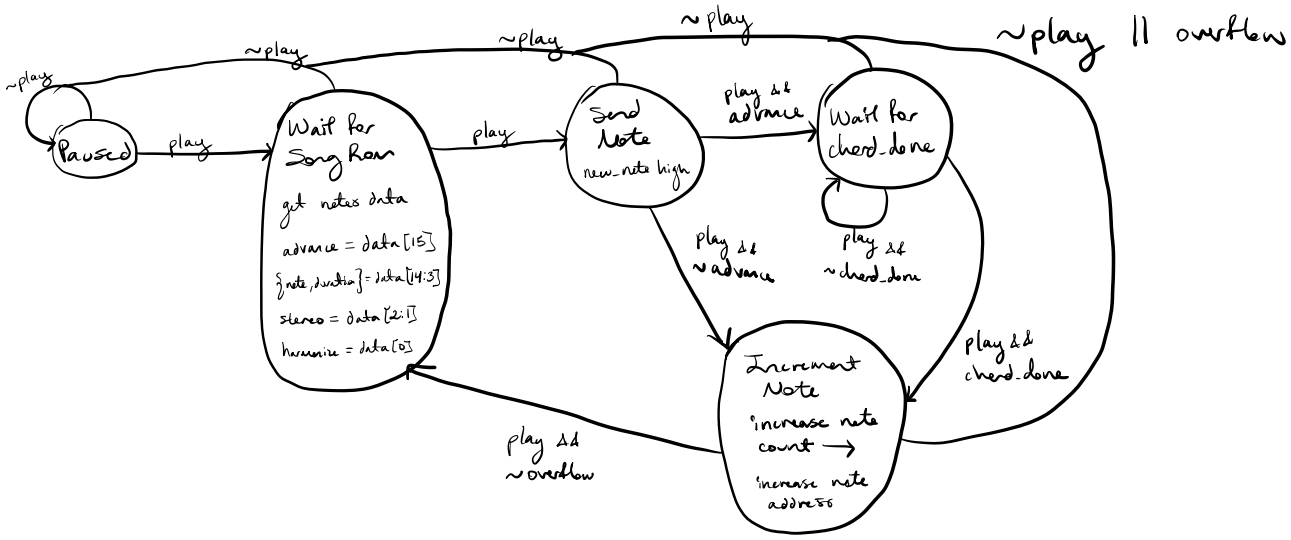
Song Reader

input

- clk
- reset
- play
- song [1:0]
- chord_done

output

- song_done
- advance
- harmonize
- note [5:0]
- duration [5:0]
- new_note
- stereo [1:0]



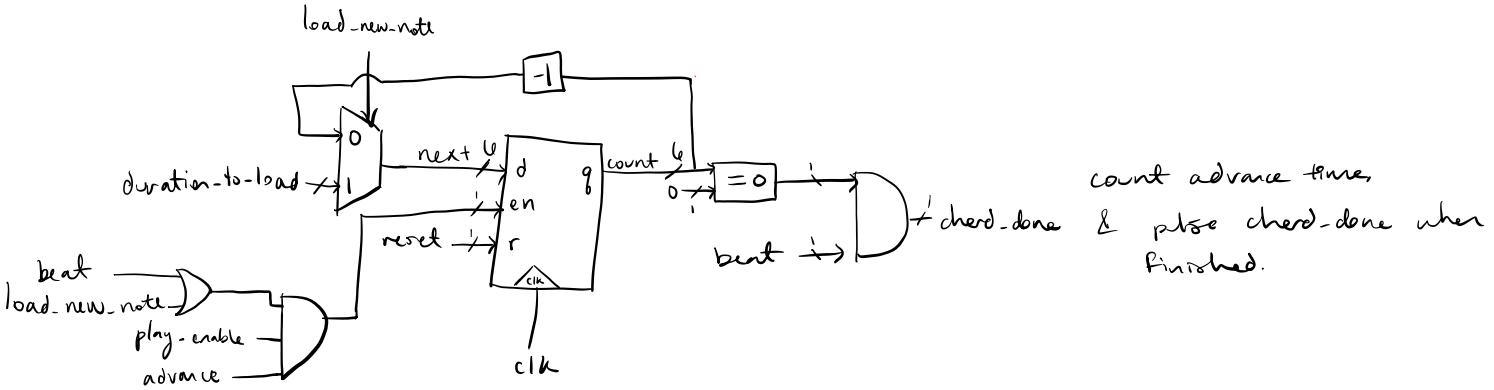
Chords Player

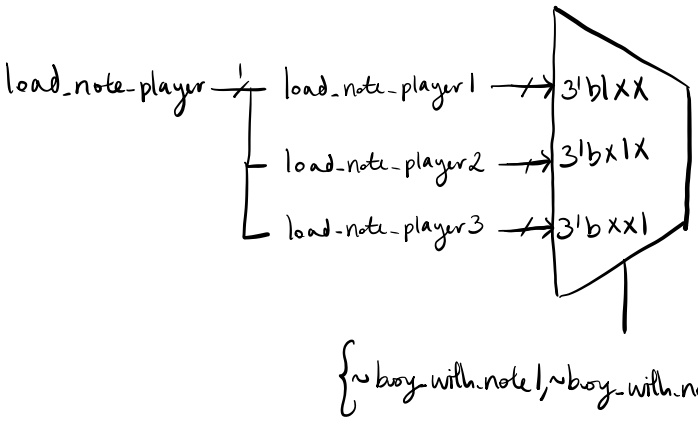
input

- clk
- reset
- play
- advance * new
- harmonize * new
- note_to_load [5:0]
- duration_to_load [5:0]
- load_new_note
- beat
- generate_next_sample

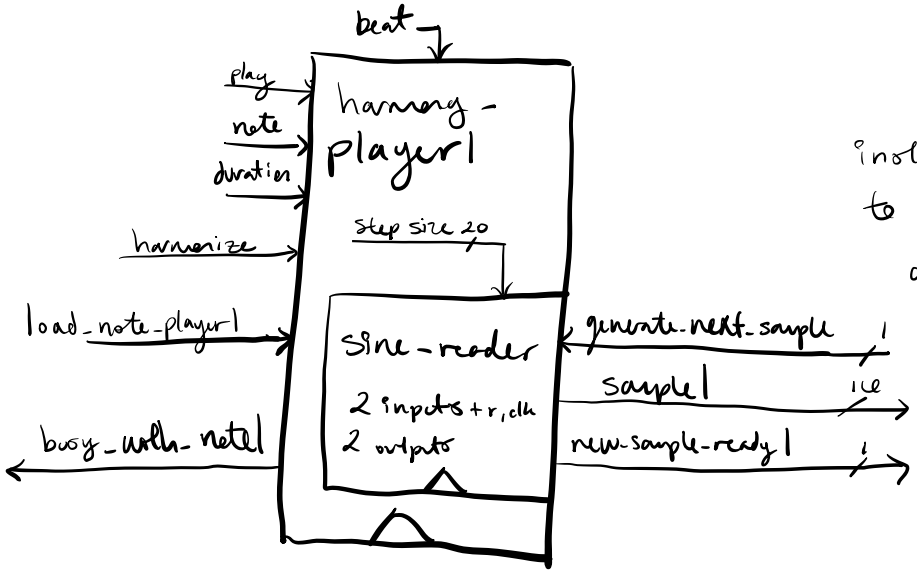
output

- chord_done * new
- display_sample * new
- new_sample_ready.

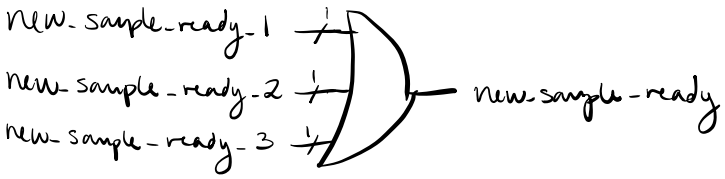




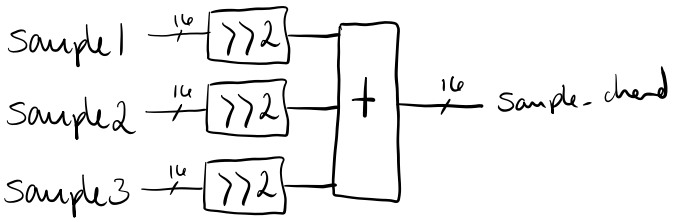
send new note to a free note player



instantiate 3 harmony_players to play note1, note2, and note3 and form the chord.



send samples to display.

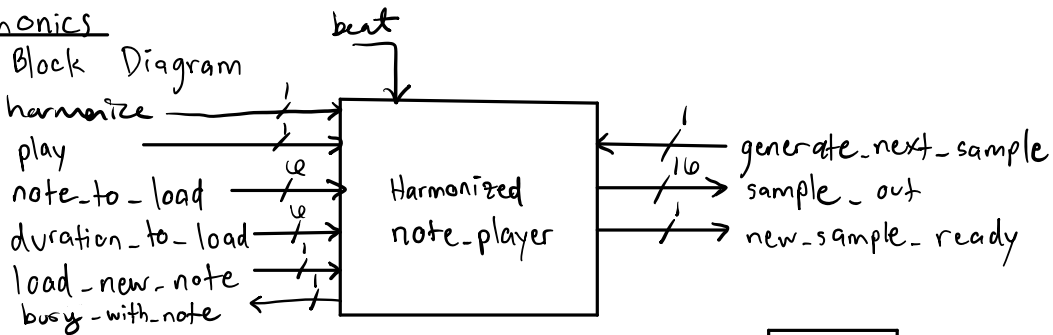


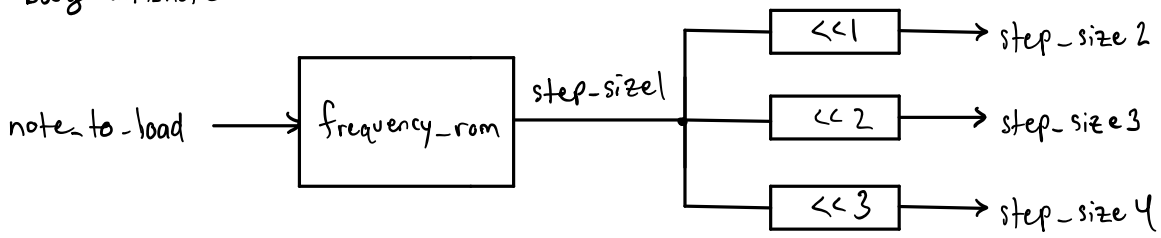
{ sample1, sample2, sample3, sample_ahead } —(64)→ display_sample

Harmony Player

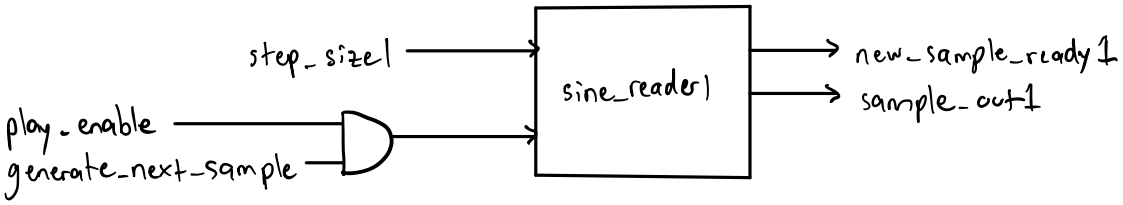
Harmonics

- Block Diagram

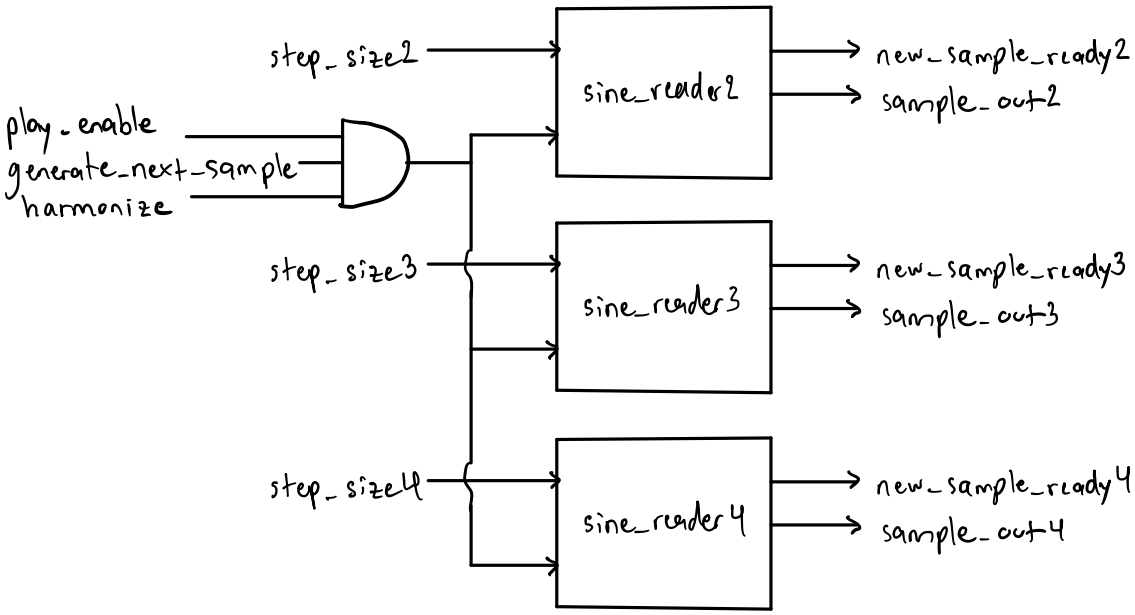


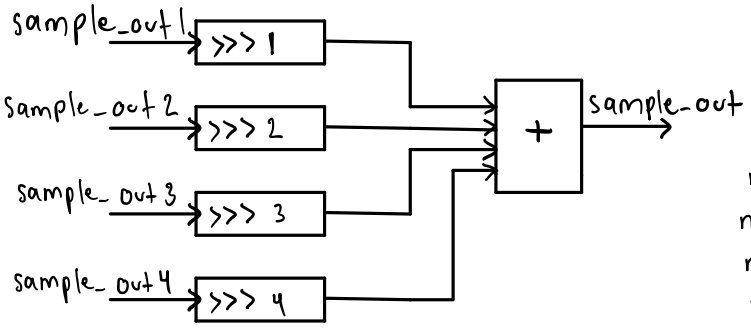


get step-size for harmonics

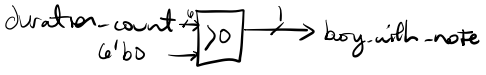


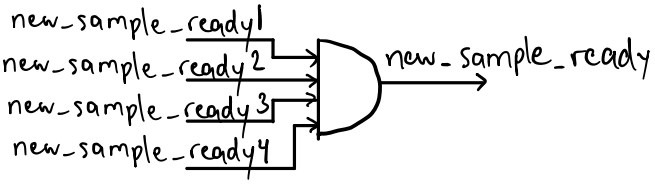
get samples for notes





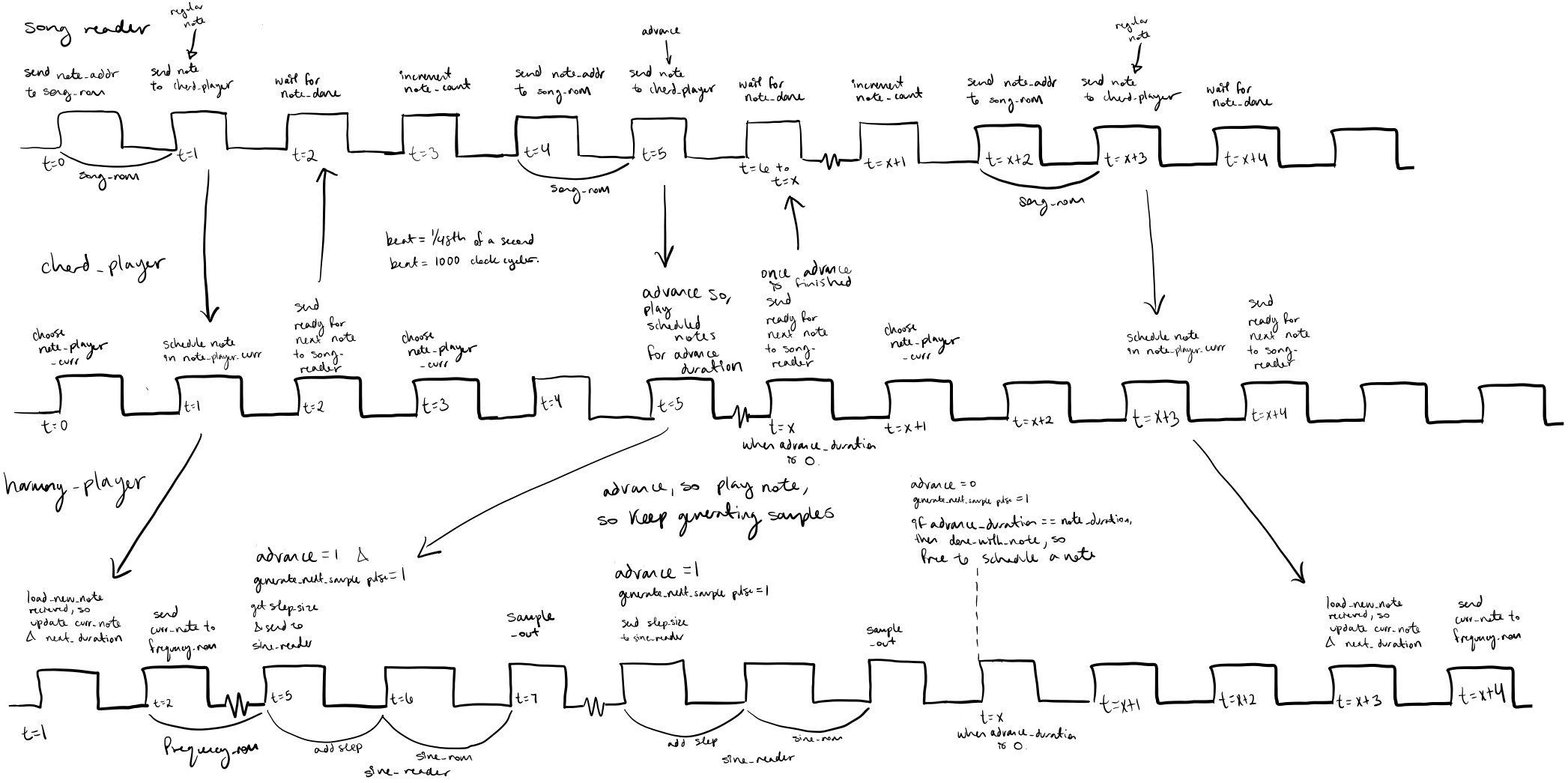
* Duration logic same as note-player





send sample

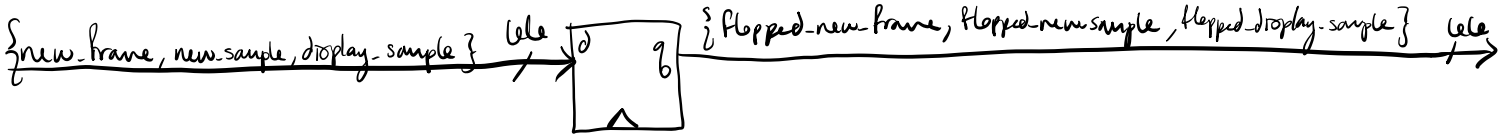
Timing b/w song reader, chord player, note player

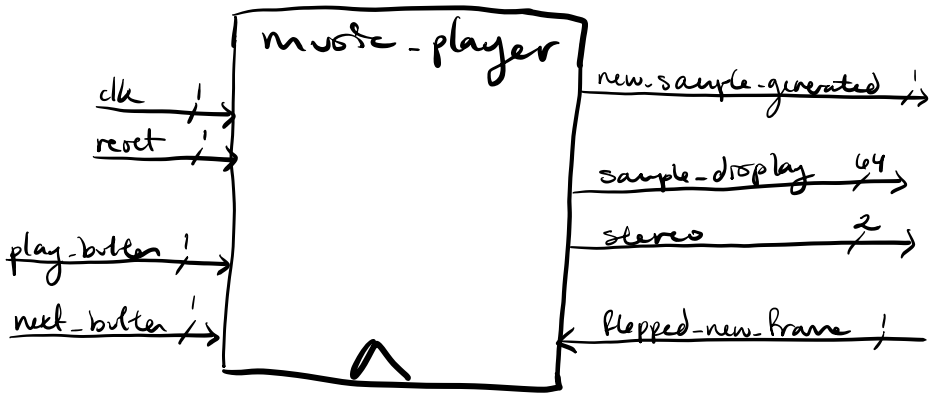


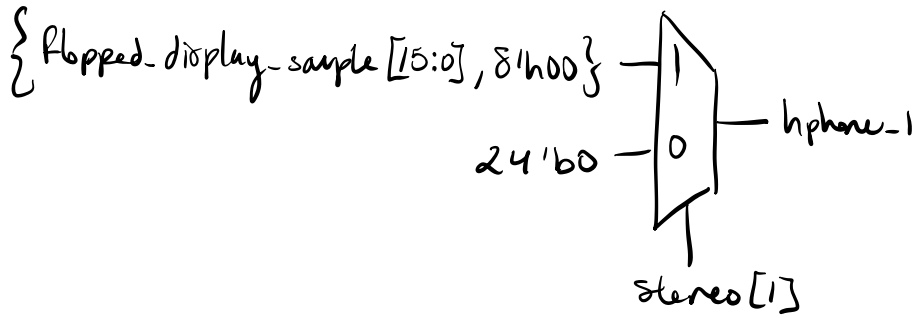
Final_top.v

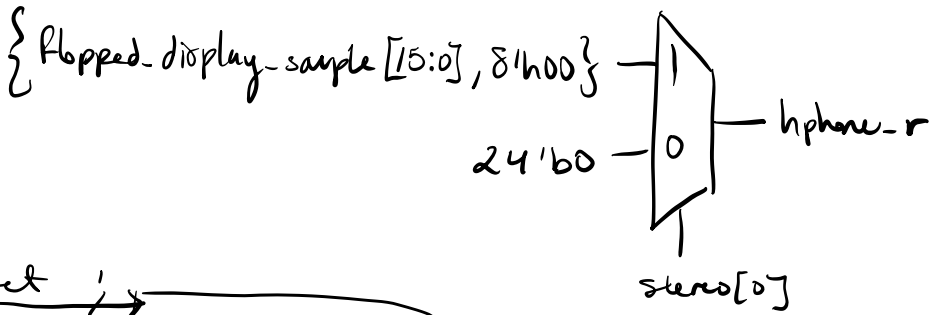
input
- sys_clk
- btn
- adau1761 interface

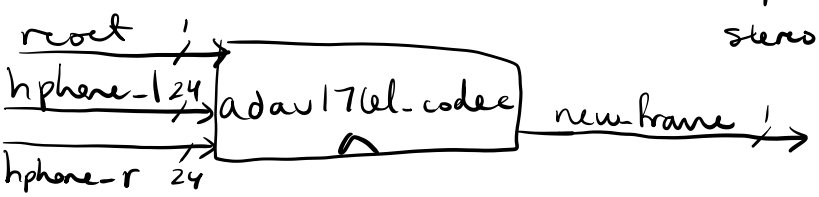
output
- adau1761 interface
- LEDs
- hdmi output

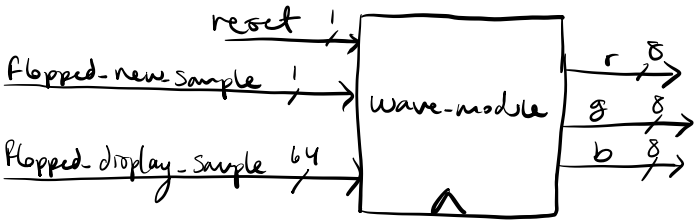










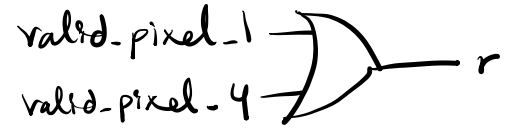


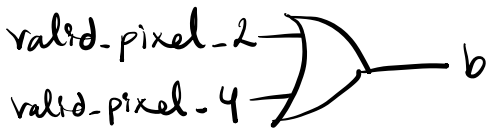
Wave_module

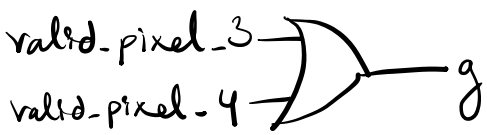
input	output
• clk	• r [7:0]
• reset	• g [7:0]
• new_sample	• b [7:0]
• display_sample [63:0]	
• X [10:0]	
• y [9:0]	
• valid	
• vsync	

display_sample $\xrightarrow{64}$ { sample_note1, sample_note2, sample_note3, sample_chord }

• instantiate 4 wave_display_top
for note1, note2, note3, & chord







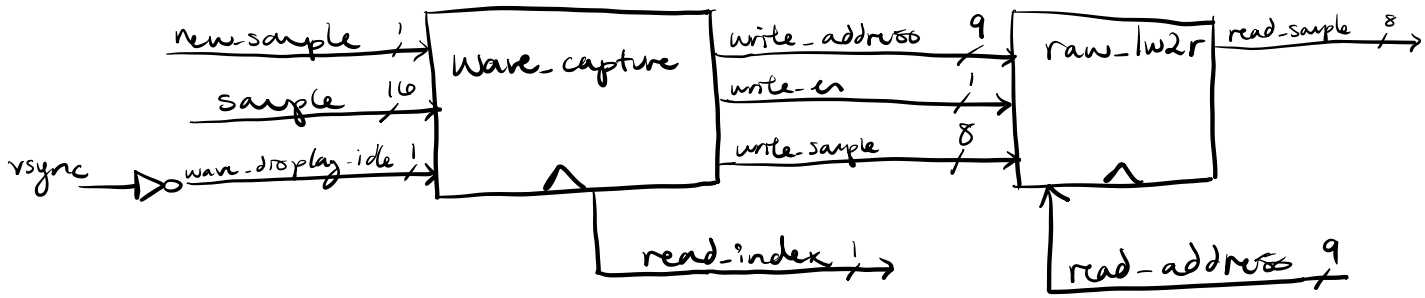
Wave Display Top

input

- clk
- reset
- new_sample
- sample [15:0]
- x [10:0]
- y [9:0]
- valid
- vsync

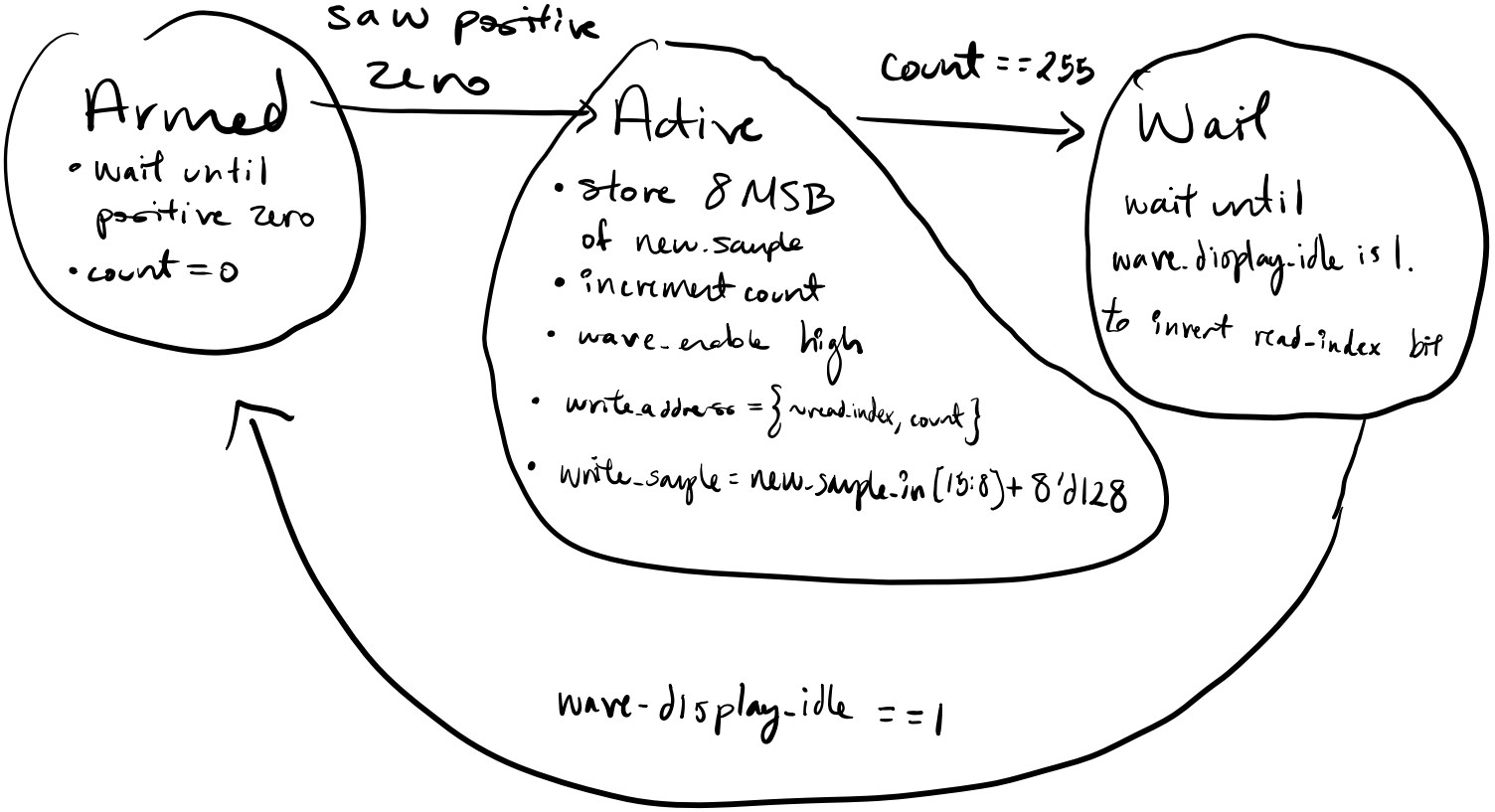
output

valid_pixel [7:0]



FSM

wave capture



wave_display

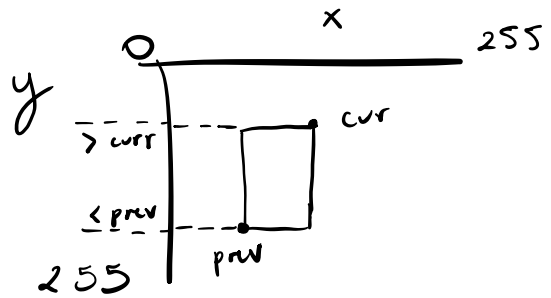
input

- clk
- reset
- x [10:0]
- y [9:0]
- valid
- read_index (RAM)
- read_value [7:0] (RAM)

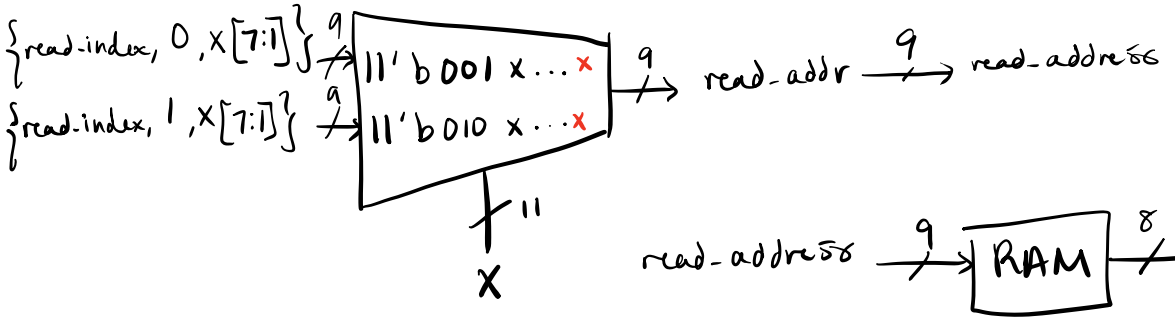
output

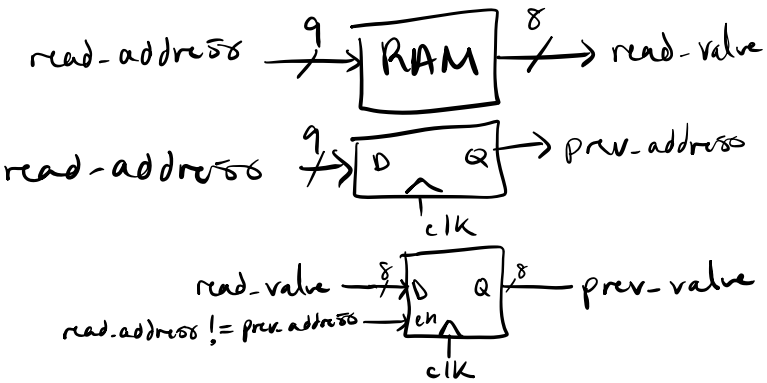
read_address [8:0] (RAM)

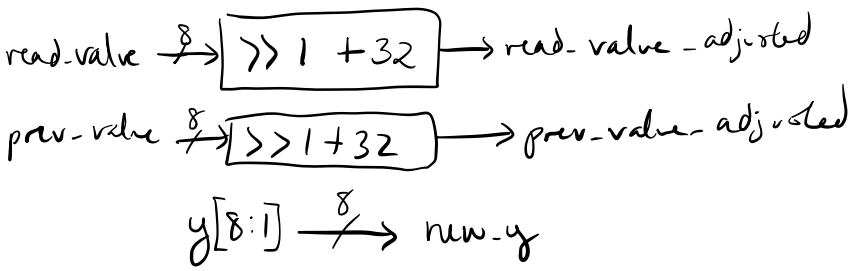
valid_pixel [7:0]

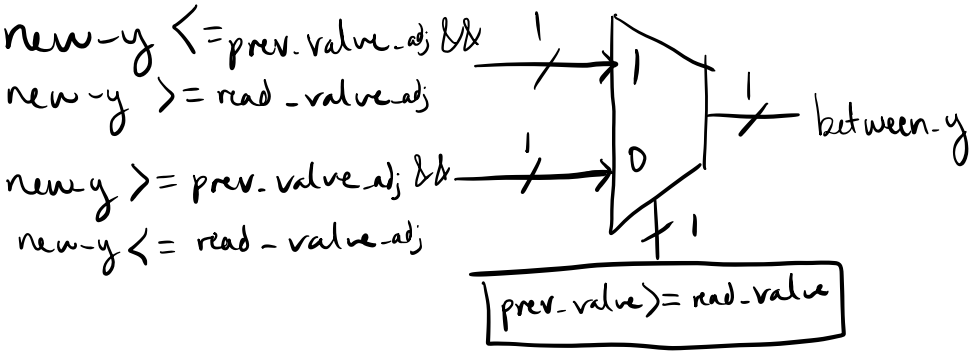


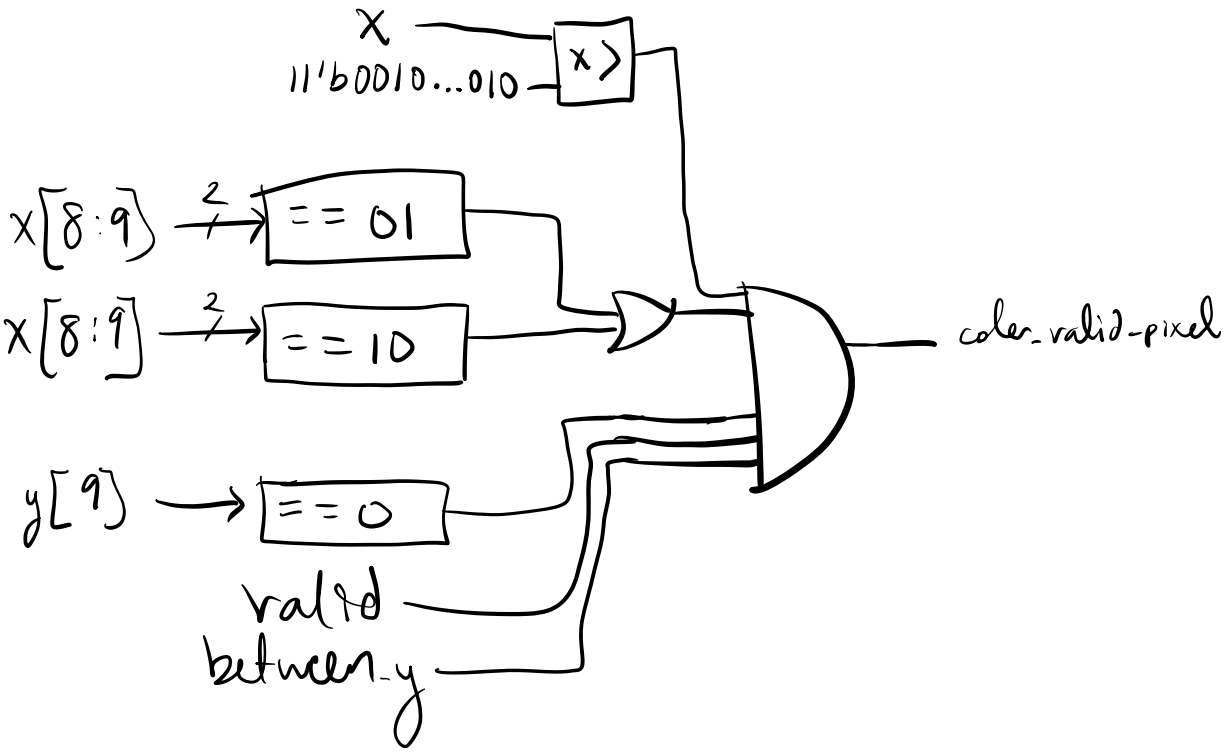
read_address = {read_index, x[9], x[7:1]}

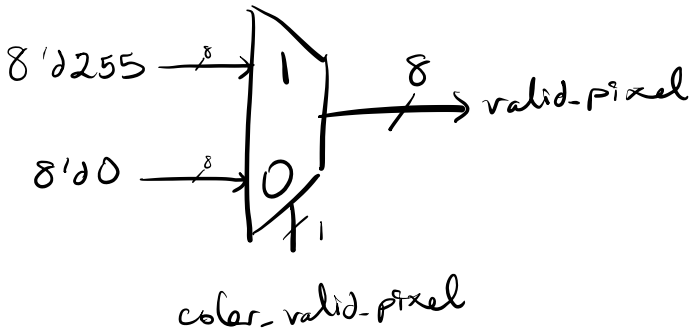




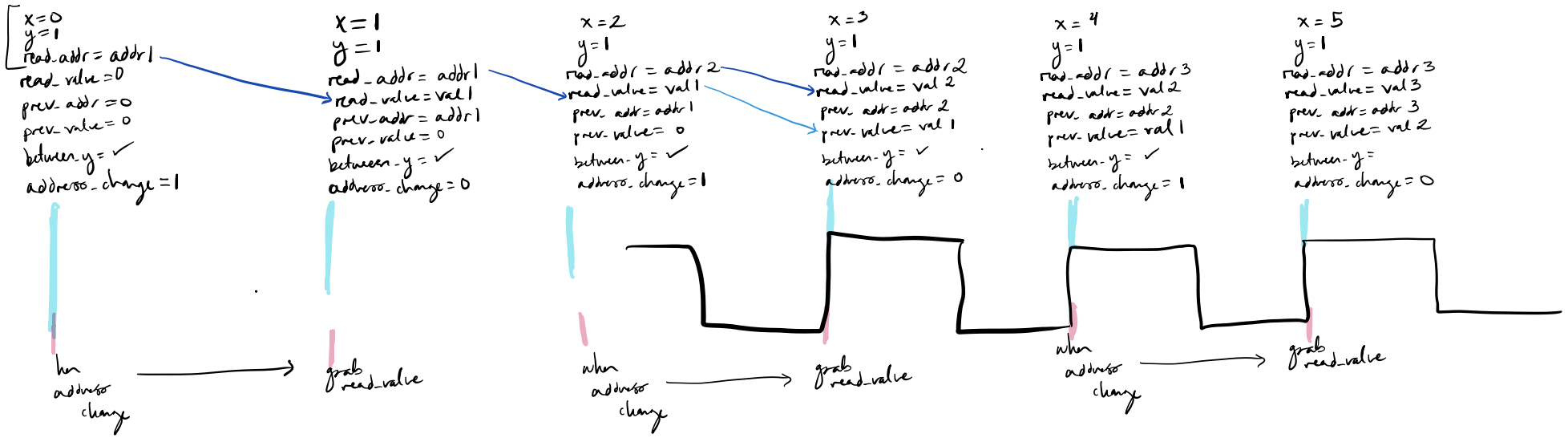





x
11'b0010...010
x>
x[8:9]
2
==01
x[8:9]
2
==10
y[9]
==0
valid
between_y
color_valid_pixel

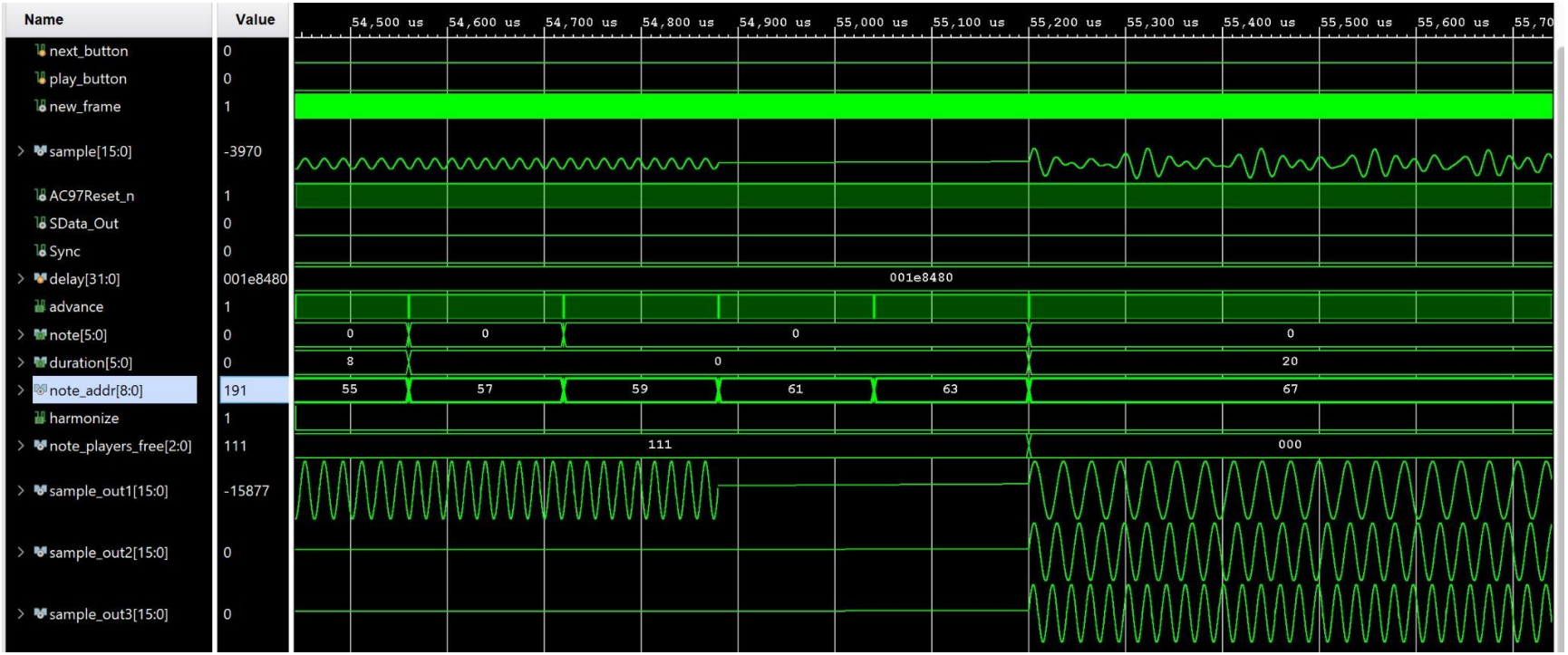

8'd255
8
1
8
valid_pixel
8'd0
8
0
1
color_valid_pixel

Timing Wave Display

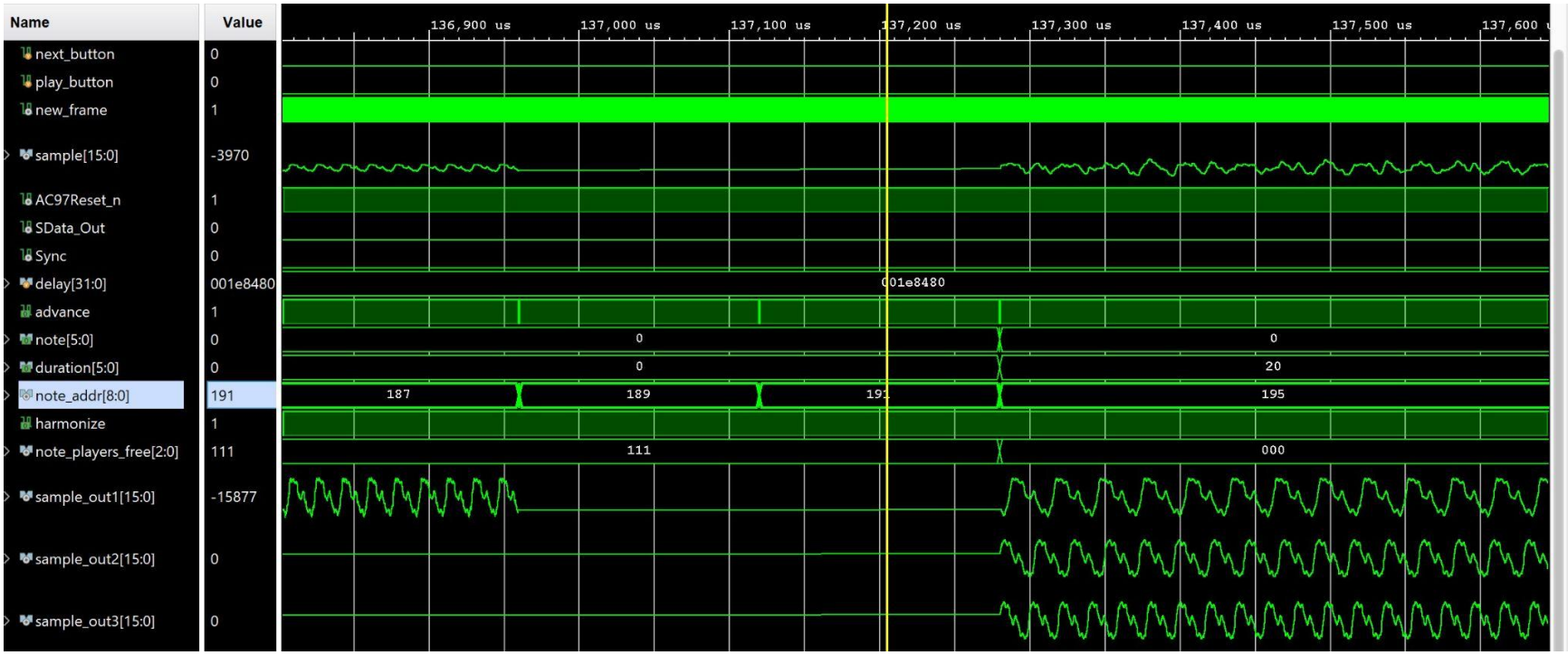


Final Project Simulations

Music Player

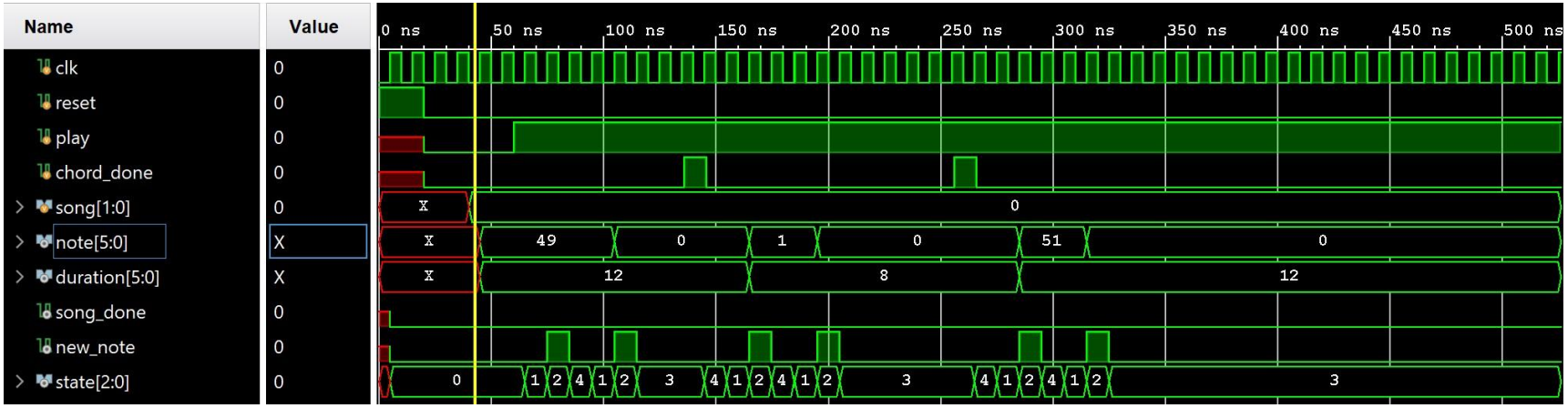


Checked to see that the sample, which is the sample_chord is the correct sum of the three notes playing, which we can see as expected at time 55,200 us.



With Harmonize high, all the sample_outs from the harmony_players should look like sawtooth waves, which we see as expected. We checked to see that the sample, which is the sample_chord is the correct sum of the three notes playing, which we can see as expected at time 137, 300 us.

Song Reader



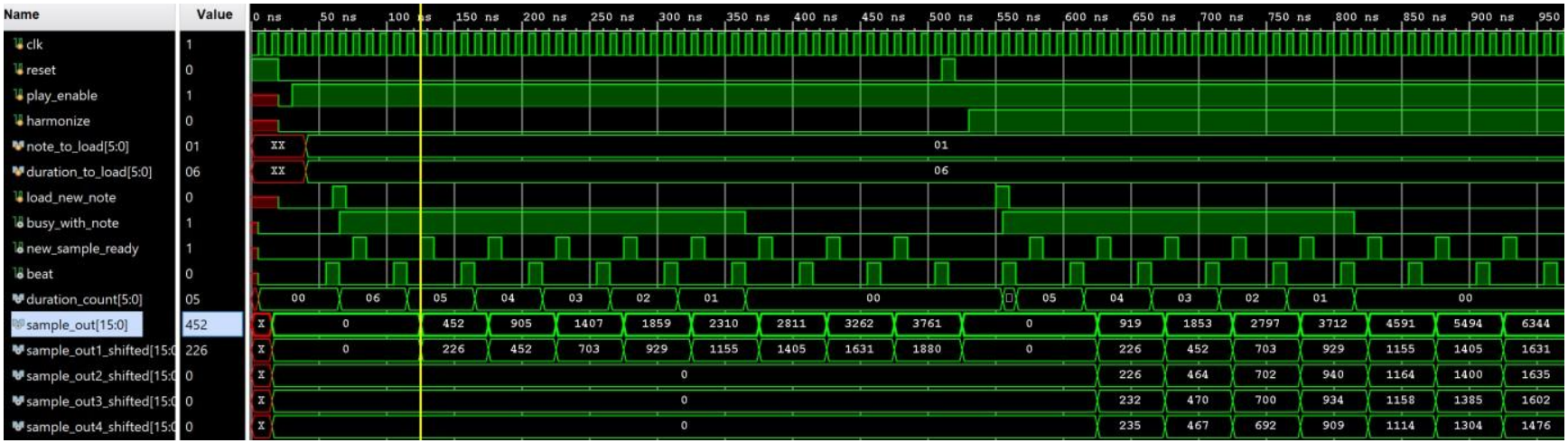
- At 60 ns, we set play to be high and Song Reader is in state 0, GETNOTE, and quickly advance to state 1 then 2, SENDNOTE
    - At 70ns, we grab the first note of song 00 {note = 49, duration = 12}
- Once we have a note, we move through states 3 and 4, where we increment our note counter
- at 130ns, we send in a “chord_done”, which correctly resets our note metadata
- After new_note at 170ns, we load a new note {note = 1, duration = 8}
- Finally, get a “new_note” at 320ns, putting us in state 3, WAITCHORDDONE

Chords


Name
Value
clk
reset
play_enable
note_to_load[5:0]
duration_to_load[5:0]
load_new_note
advance
harmonize
beat
new_sample_ready
display_sample[63:0]
chord_done
sample_out1[15:0]
sample_out2[15:0]
sample_out3[15:0]
sample_chord[15:0]
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
260 ns
280 ns
300 ns
320 ns
340 ns
360 ns
380 ns
400 ns
420 ns
440 ns
460 ns
480 ns
500 ns
520 ns
074301c400000241
090603bb01f703ac
0afb05b1041f0531
0cbe07a70616069d
0eb1

- Display_sample is the concatenation of {sample_out1, sample_out2, sample_out3, sample_out4}
- We test that our chord_player can still play one note before an advance note (before 100ns)
    - Each note's duration is 2
- At 55ns, we get an advance note, which has a duration of 2
- Note_player_busy only marks one harmony_player as busy
- We test that our chords are working by loading up three consecutive notes before an advance note (before 220ns)
    - Each note's duration is 6
- At 225ns, we get an advance note, which has a duration of 6
- Each "note_players" signals turn on as expected as they each get told to load a new note and are marked busy while they're playing
- We pause at 255ns which is handled correctly
- We see each "note_player" (now harmony_player) generate signals correctly

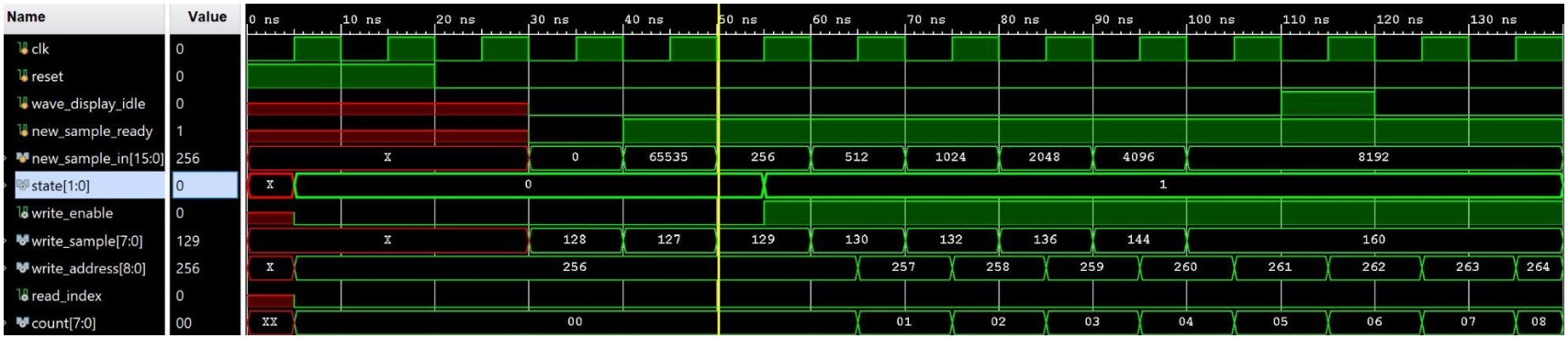
Harmonics



- This test shows that we can go between no harmonics, and then activating harmonics (525ns)
- At ~50ns, we load a new note without harmonics, and it plays out correctly for its duration
    - In this case, sample_out 2-4 don't need to do anything

- At 550ns, we load another note, this time with harmonics
    - we see that all 4 sample_outs start doing work simultaneously
    - Their sum gives the final sample_out, which works as expected

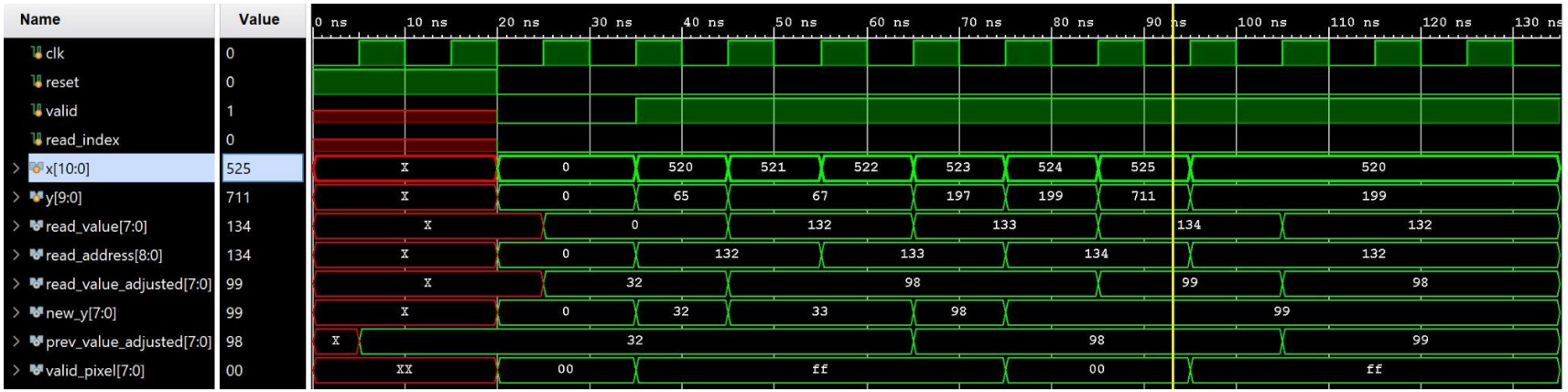
Wave Capture (did not change from our lab_5 to our final)



- At 40ns, we set new_sample_in to be a negative number and set new_sample_ready to be 1.
- At 50ns, we set new_sample_in to be a positive number (16'd256) and set new_sample_ready to be 1.
    - Here we should see the state change from Armed (0) to Active (1), which we see at 55ns on the clock.
    - In Active, we should save the first positive number in write_address {~read_index, count}. For the very first sample, count should be 0, which we see and since we set read_index to 0, we're going to write to address {1, 0}. In decimal, this address is 256.
    - Here we set new_sample_in to be 16'd256, so the top 8 MSB would be 8'd1. To accurately store the number, we add 8'd128, so write_sample should be 8'd129, as expected.
    - Write_enable is set to high since we are currently writing in new samples and are in the Active state.
- For the rest of the new_sample_in's, we set new_sample_ready to be 1 and double the quantity, every 10ns. So the next new_sample_in is 16'd512, then 16'd1024, and so on. We did this so we could notice a change in the write_address.
    - We stay in the Active state since we still haven't saved 256 new samples.
    - The count increases by 1 for each new sample.
    - The write_address also increases by 1 for each new sample.
    - The write_sample also increases by 1 since it is just taking into account the top 8 MSB bits.

    - Write enable is high as expected.
- As a test, we set wave_display_idle to be 1, but since we are in the Active state, and not the Wait state, it is ignored, as expected.

Wave Display



- For visibility, we set the values to be in decimal.
- At 35ns, we set X to be 11’d520 ({3'b010, 7'b0000100, 1'b0}) and Y to be 10’d65 (y = 10'b0001000001), read_index is 0, and valid is 1.
    - As a result, read_address is {read_index -> 0, 3’b010-> 1, 7'b0000100} which in decimal is 9’d132, as we can see.
    - Then, read_value and prev_value are adjusted to display correctly. At this time (35 ns) both are 0, so read_value_adjusted and prev_value_adjusted are 8’d32.
    - Y is translated, removing the MSB and the LSB so new_y is 8’b00100000, which is 8’d32, as we can see.
    - Since new_y is between prev_value_adjusted and new_value_adjusted, it is a valid pixel so valid_pixel is high.
- At 45ns, we increase X by 1, and increase Y by 2.
    - The read_address would remain the same since X’s LSB is cut off.
    - We defined RAM to return the 8 LSBs of X. The RAM returns read_value a cycle later, so now we have the read_value 8’d132 from the read_address 9’d132.
    - Then, read_value (8’d132) and prev_value (0) are adjusted to display correctly. At this time (35 ns) are 0, so read_value_adjusted is 8’d98 and prev_value_adjusted is 8’d32.

    - new_y is 8’d33 now and since it is between read_value_adjusted and prev_value_adjusted the pixel is valid and valid_pixel is high.
- At every clock cycle, we are incrementing X by 1, and see read_address change after two clock cycles, as expected. We modify Y to place it between read_value_adjusted and prev_read_value, it is between the range, so it is a valid_pixel (1), and the color is red.
- read_value is accepted one clock cycle after read_address changes, and since read_address changes every two clock cycles, read_value should also only change after two clock cycles.
- prev_value updates one clock cycle after read_value changes, and we can see that, as expected.
- At 75 ns, we set Y to not be between read_value_adjusted and prev_read_value, so valid_pixel should be 0 and color is black, which we see, as expected.
- At time 90ns, we set Y to be a high value with Y[9] is 1 and so the Y is no longer at the top of the display. Therefore, it is no longer a valid pixel and we see that valid_pixel at this point is 0, as expected.