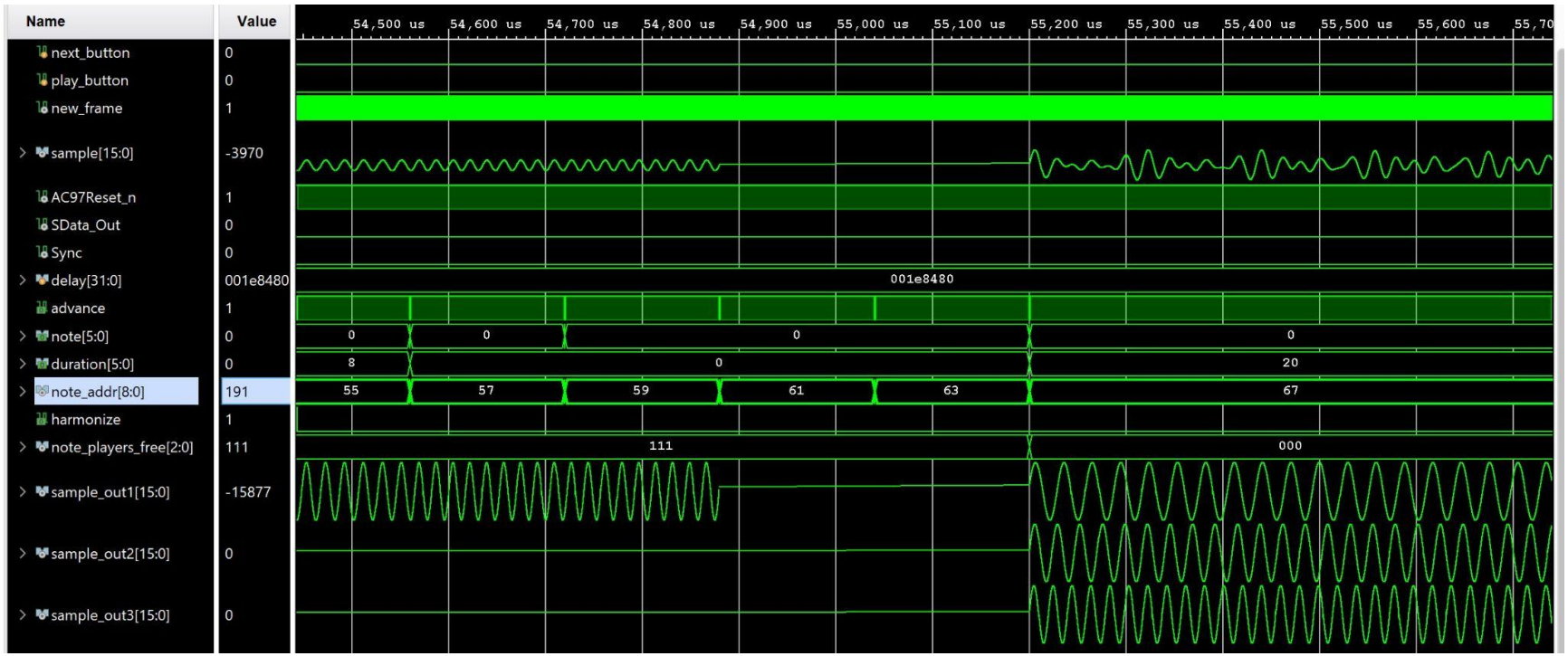
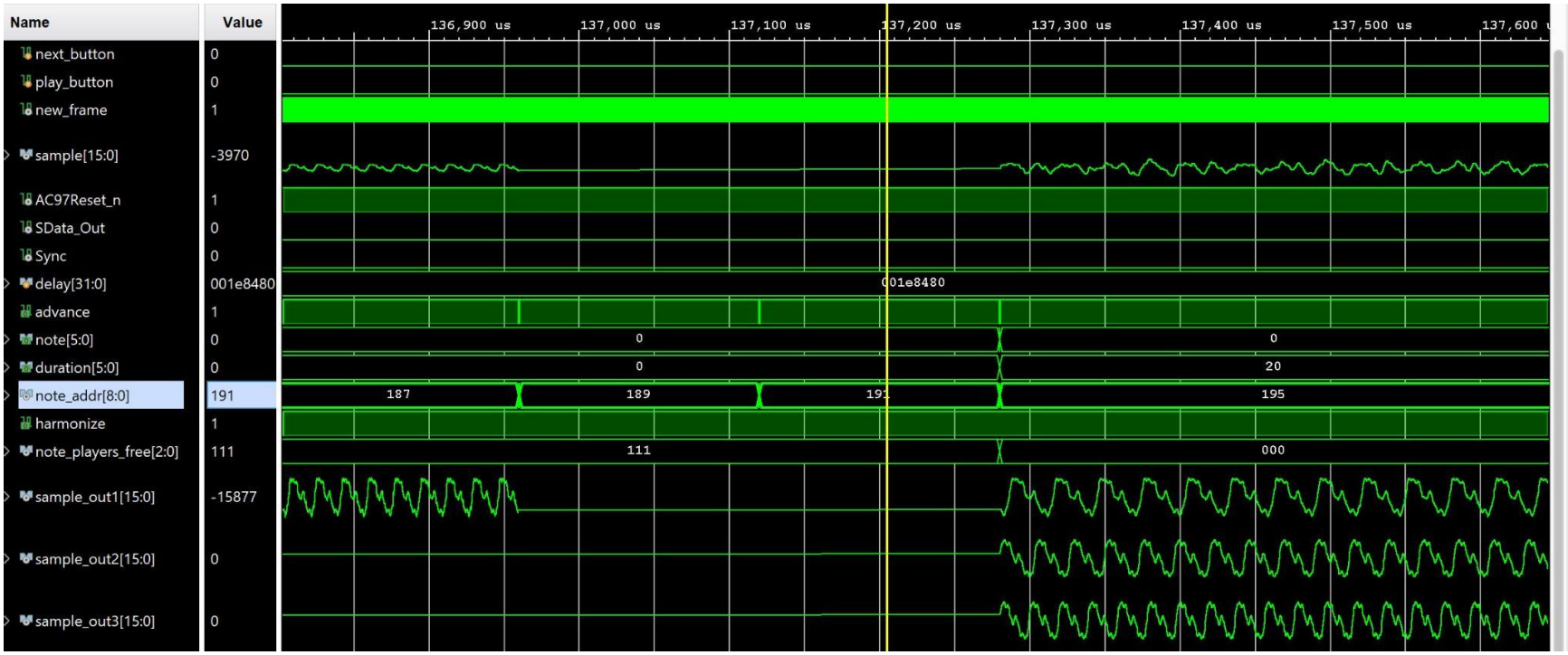
Final Project Simulations

Music Player

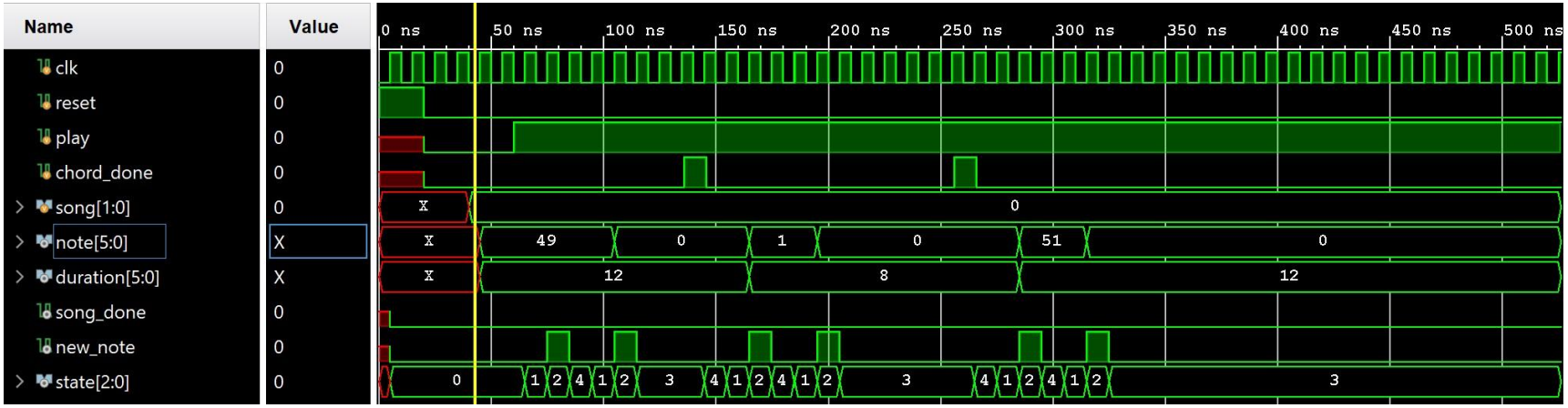


Checked to see that the sample, which is the sample_chord is the correct sum of the three notes playing, which we can see as expected at time 55,200 us.



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



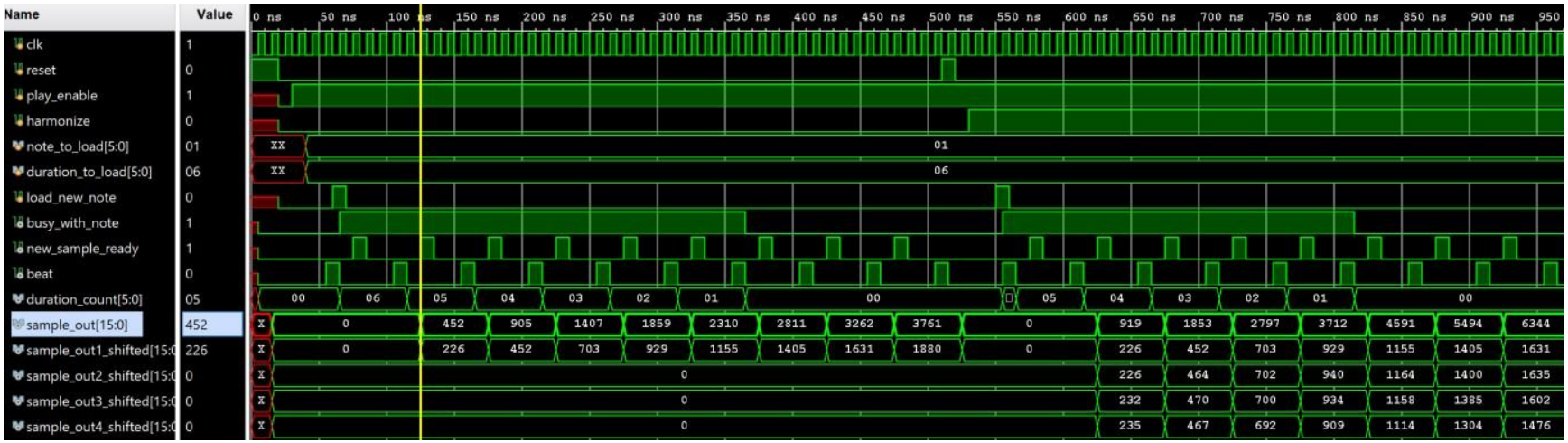
- At 60 ns, we set play to be high and Song Reader is in state 0, GETNOTE, and quickly advance to state 1 then 2, SENDNOTE
    - At 70ns, we grab the first note of song 00 {note = 49, duration = 12}
- Once we have a note, we move through states 3 and 4, where we increment our note counter
- at 130ns, 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

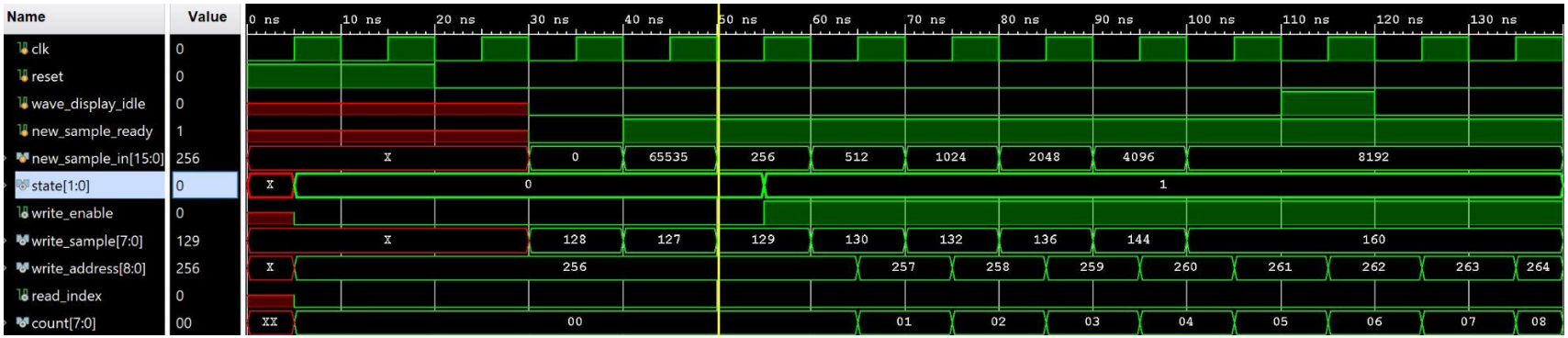
Harmonics



- 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

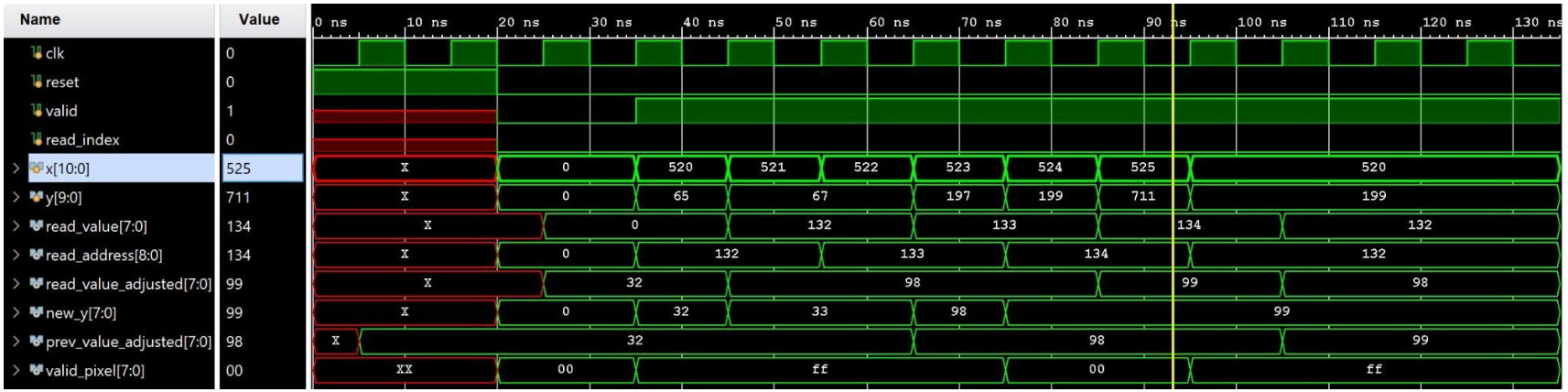
Wave Capture (did not change from our lab_5 to our final)



- 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.