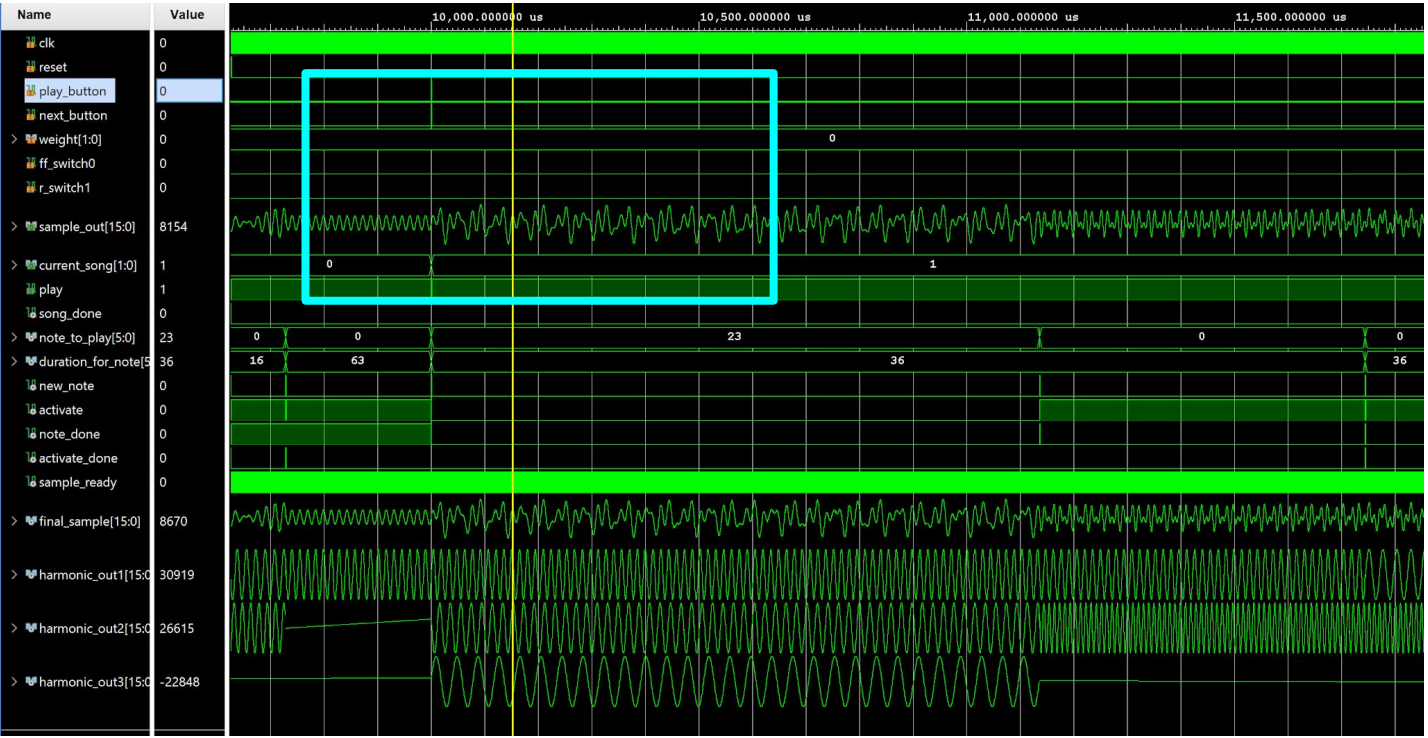
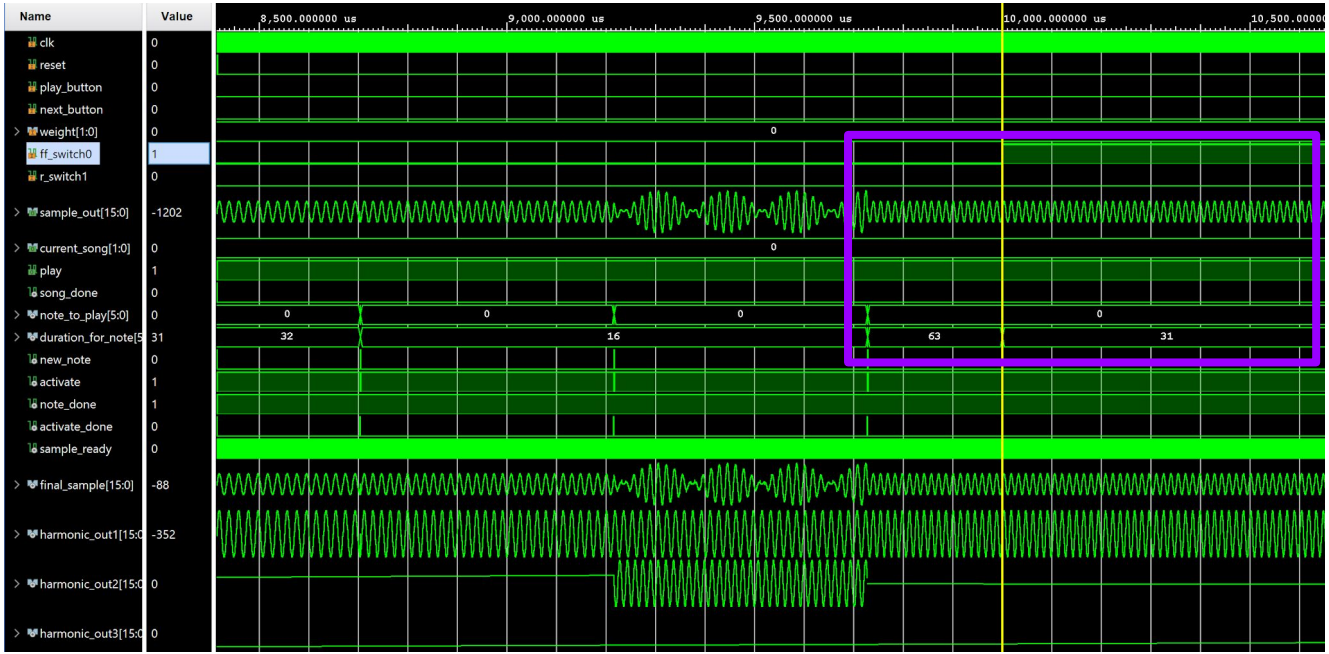
# Music Player



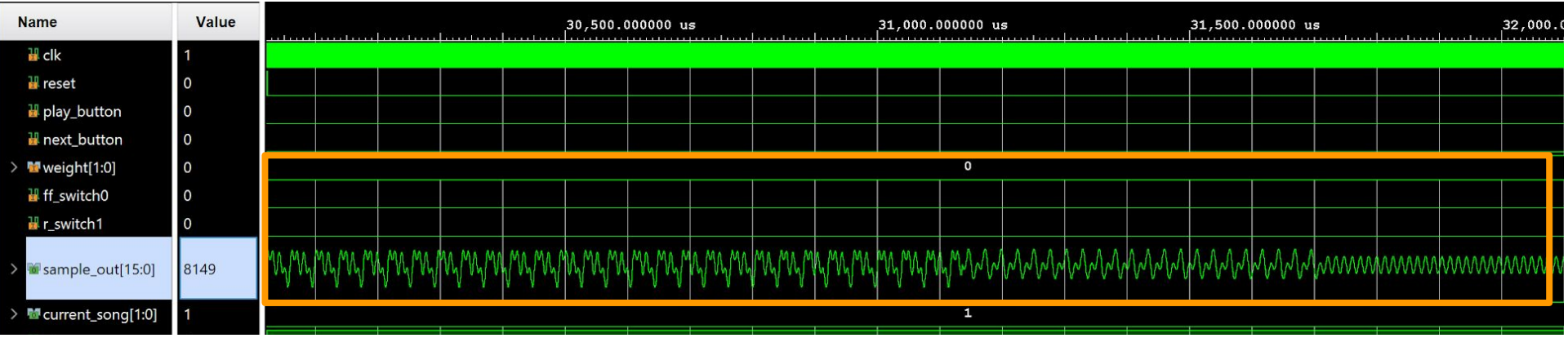

During song 0 we press the next button. It goes into the pause state, and a few clock cycles we press play to start song 1. Note: since we are zoomed out so much, it may appear that pressing play occurs on the same clk cycle as pressing next. In reality there are several clk cycles between the two.

# Music Player

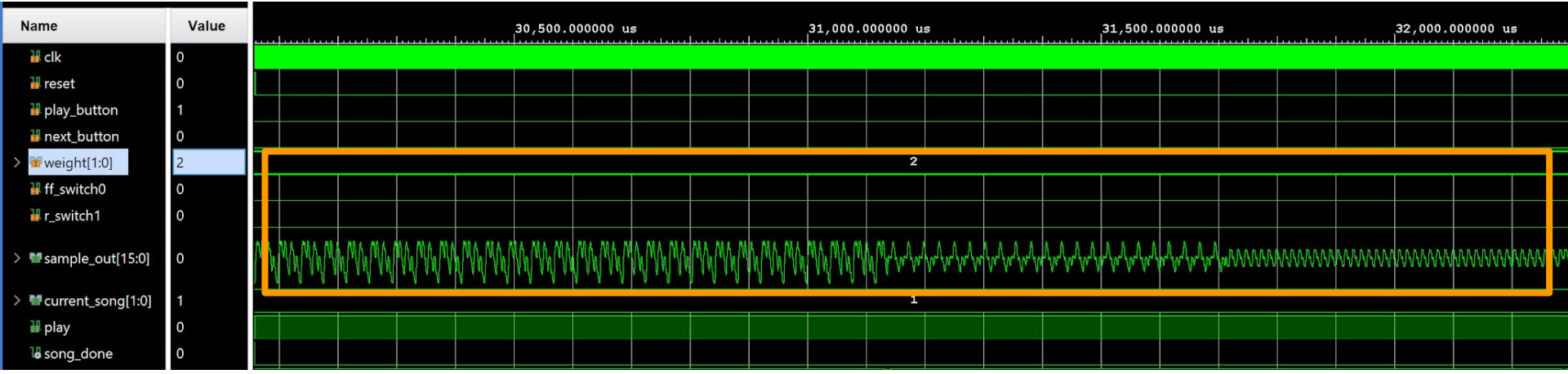



Note how once ff_switch0 (fastforward) goes HIGH, the duration is cut in half

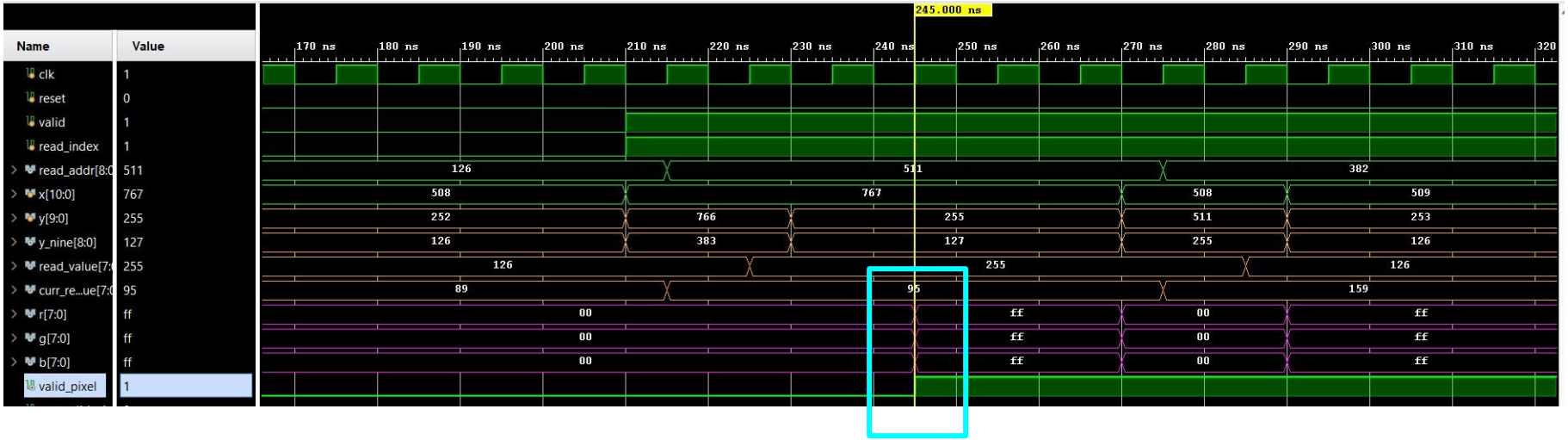
# Music Player




Note how final out is affected by the change in weight which adds harmonics.



# Wave_display pt1 (sine wave)

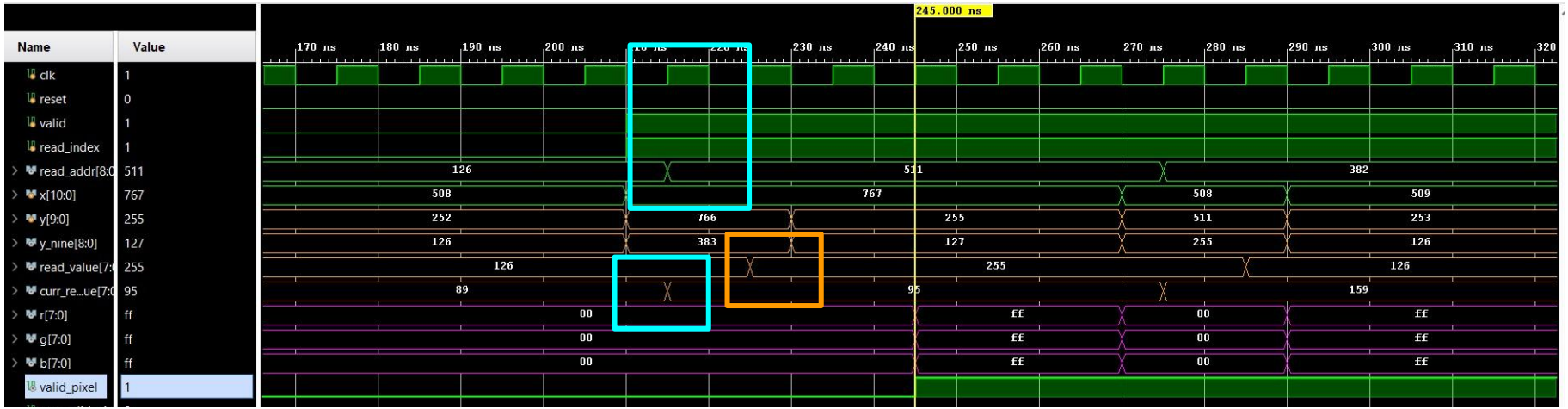



x: 00000000000, y: 1111111111 = valid_pixel: 0 -> TRUE
x: 00100000000, y: 1111111111 = valid_pixel: 0 -> TRUE
x: 01000000000, y: 1111111111 = valid_pixel: 0 -> TRUE
x: 00000000000, y: 0000000000 = valid_pixel: 0 -> TRUE
x: 01000000000, y: 0111111111 = valid_pixel: 1 -> TRUE
x: 00100011000, y: 0111111111 = valid_pixel: 1 -> TRUE

Valid_pixel goes high at the same time when rbg pixels equal white as expected.

Text to the right outuds TRUE when the our expected_valid_pixel matches valid_expel

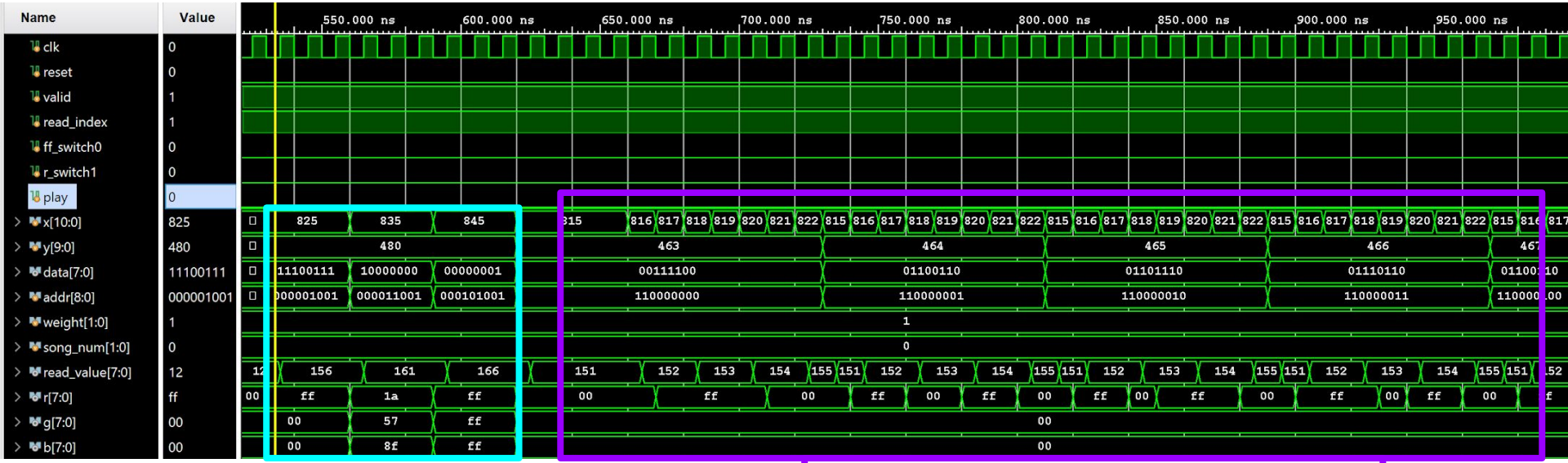
# wave_display pt2 (sine wave)




When the read_addr changes (in blue) the curr_read_value changes at the same time as expected since read_addr changing is the enable to the flipflop that pushes a new read_value

When read_addr changes, read_value changes one clock cycle later because the RAM takes one clock cycle as expected

# Wave_display pt3 (icons)

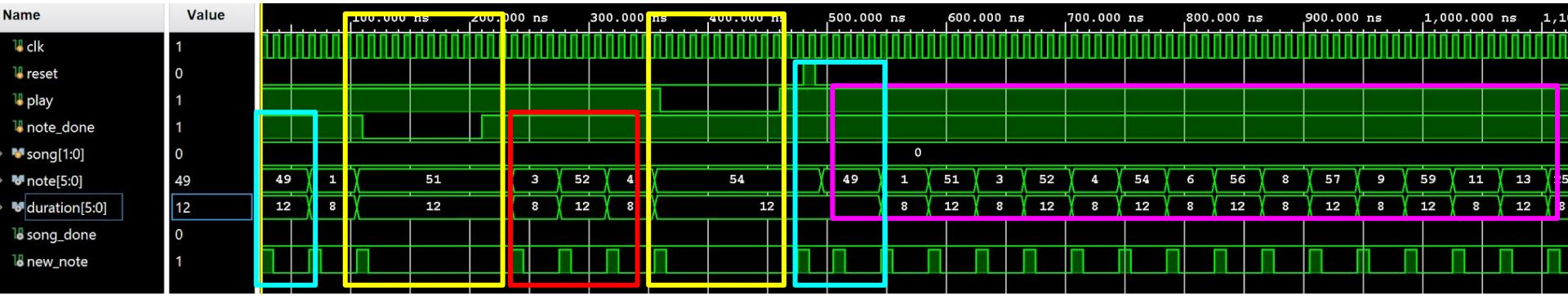



Here x=825 is the first position for pause button icon, 835 is first for the fastforward and 845 is the first position is for rewind. Play is false so pause is drawn in red (FF0000) while the rest is drawn in blue then white. Data's first position is a 1 so pause and rewind is in color and it is red since pause is the current state. Fastforward is blue since ff_switch is low and data first pos is 1. Rewind is in white since data first pos is 0.

Looping through and drawing song number 0:

| Data | x | color |
|---|---|---|
| 0 | 815 | black |
| 0 | 816 | black |
| 1 | 817 | red |
| 1 | 818 | red |
| 1 | 819 | red |
| 1 | 820 | red |
| 0 | 821 | black |
| 0 | 822 | red |

# Song_Reader



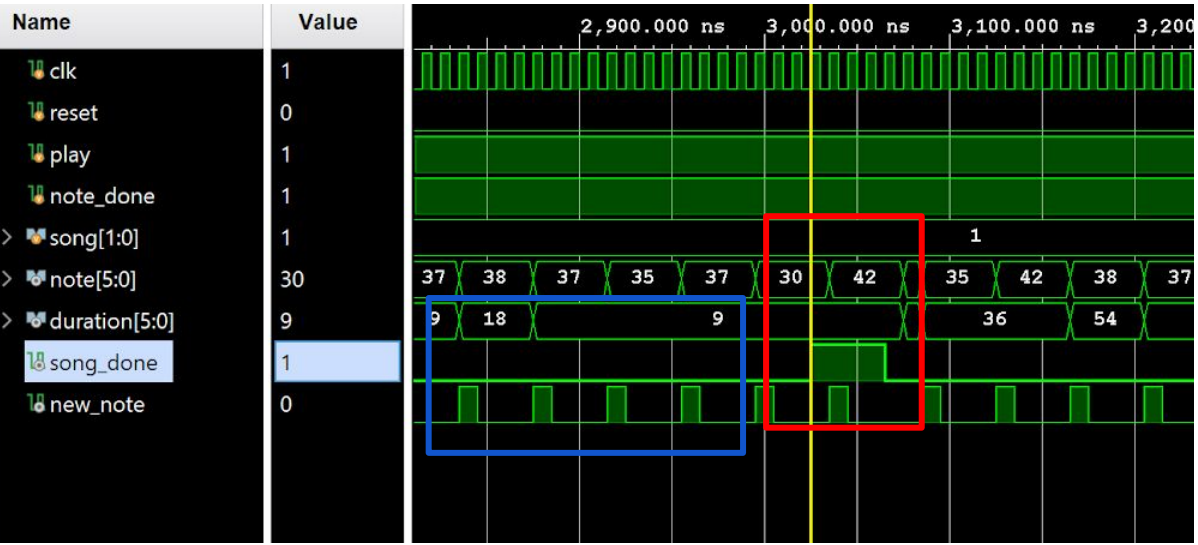

While play is LOW, song_reader holds the note it is on and doesn't move on.

Each time new_note goes HIGH, the note will change, and when it is LOW it holds on to the current note.

The first note in song0 is 49 as demonstrated in the first blue box, so when reset goes HIGH it starts over with note 49 as demonstrated in the second blue box.

While play is HIGH, song_reader is able to move on and process each note.

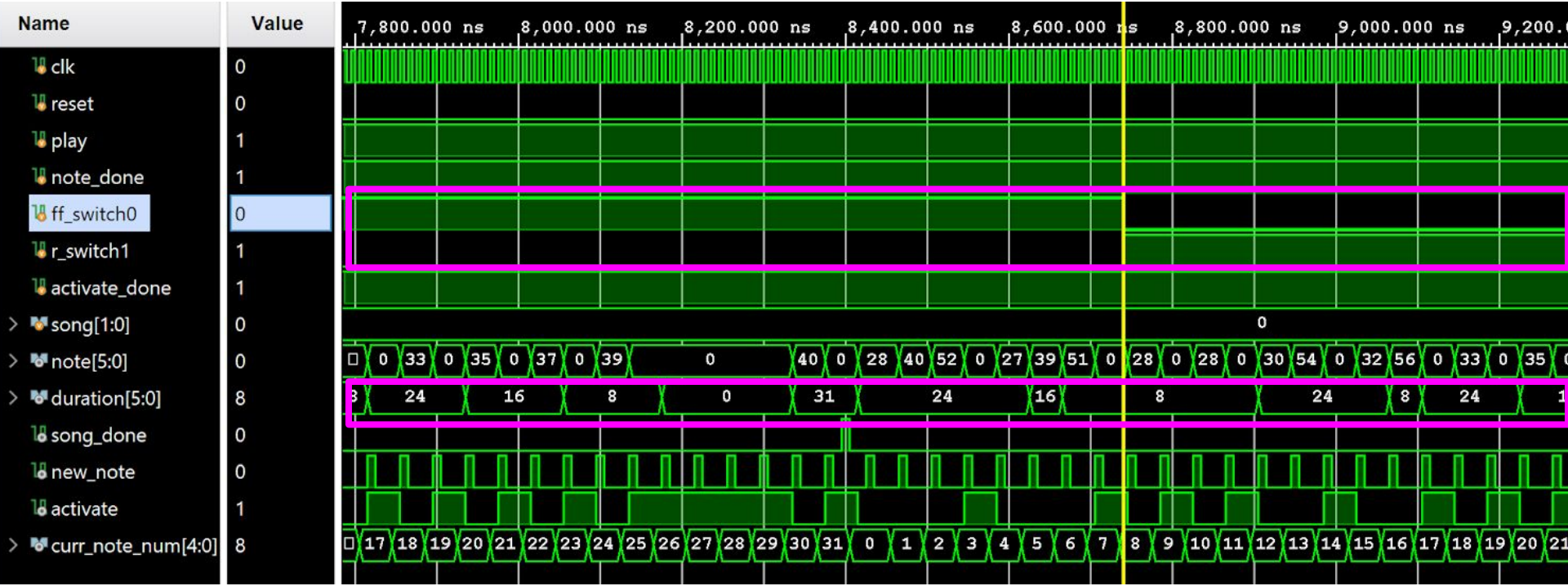
# Song_Reader pt. 2




When the final note in song1 is played, song_done goes high. Note that the song does not change because song is an input and it is hardcoded to be 1 here. In the system itself you would normally see it transition to the paused state associated to song2.

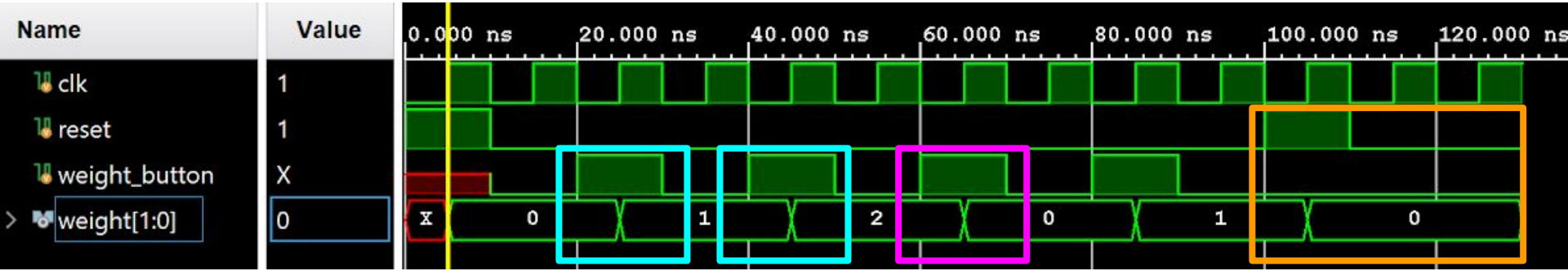
Although the durations are different, new note still happens at the same rate and that is because in normally it would be note_player who notifies song_reader note_done must go HIGH. For testbench purposes, we just keep note_done HIGH.

# Song Reader




While ff_switch0 or r_switch1 are HIGH, duration gets cut in half.

# Interactive Instrument Editing (iie)

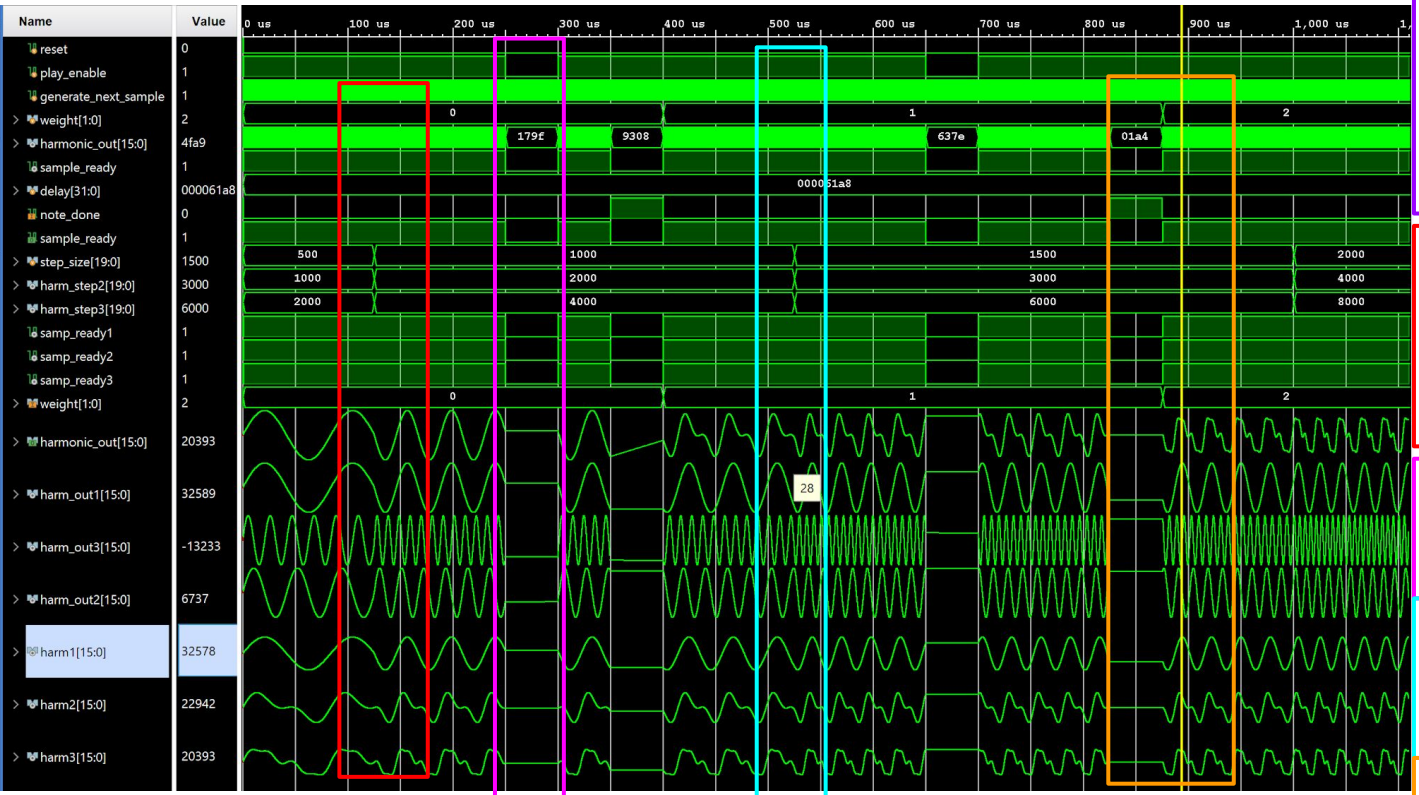



Everytime you press the weight button it increments weight.

When you press the weight button and weight is already at the max (2) it wraps around and goes to 0 next.

When reset goes HIGH, weight reverts back to 0.

# create_harmonic




This module produces three sine waves, harm_out1,2 and 3. These sine waves 2 and 3 are bit shifted versions from input “step_size.” These sign waves are then added together with different weights, forming harm1,2 and 3. Based on the “weight” variable, we will select our output “harmonic_out” to be either harm1,2 or 3.

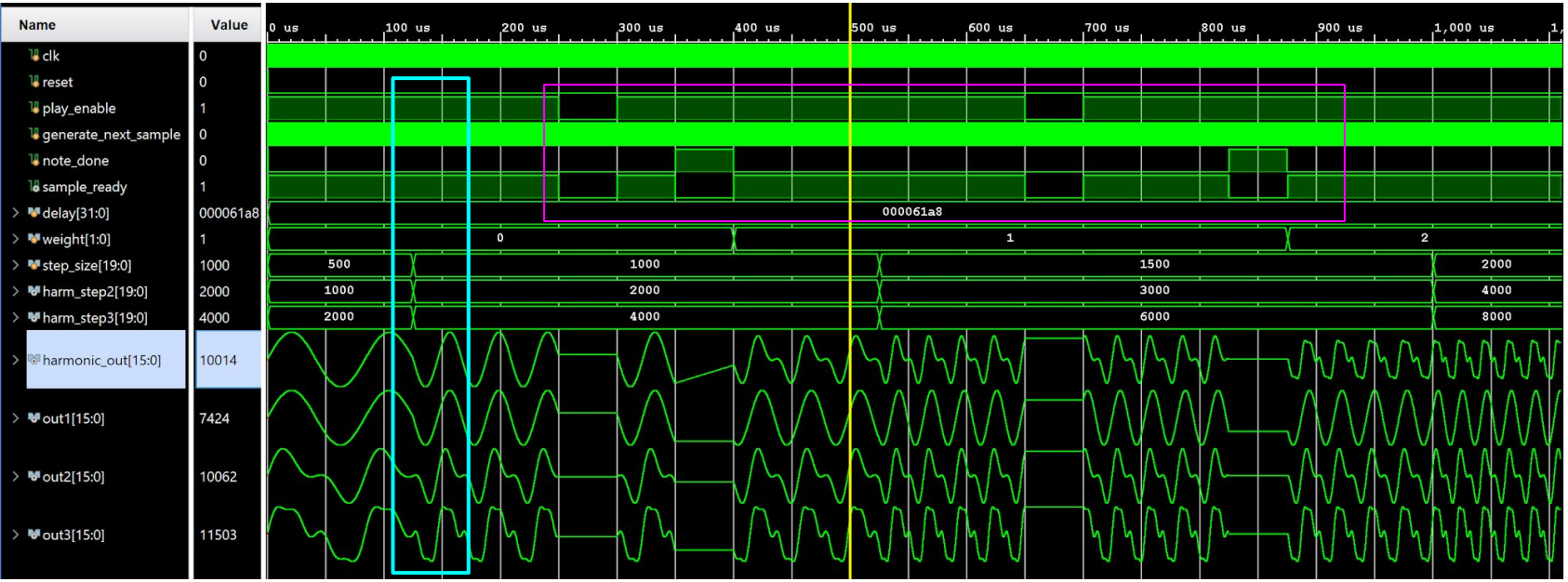
Since weight = 0, we are selecting the output harmonic_out to be harm1. In this block we can also see the transition from one note to the next and what they looks like.

Play enable is disabled therefore there the sine waves stop producing the next sample.

Weight changes to 1, therefore we select harm2 as our output

Note_done goes high so we stop producing a sine wave. Also weight = 2, so we select harm3 as output

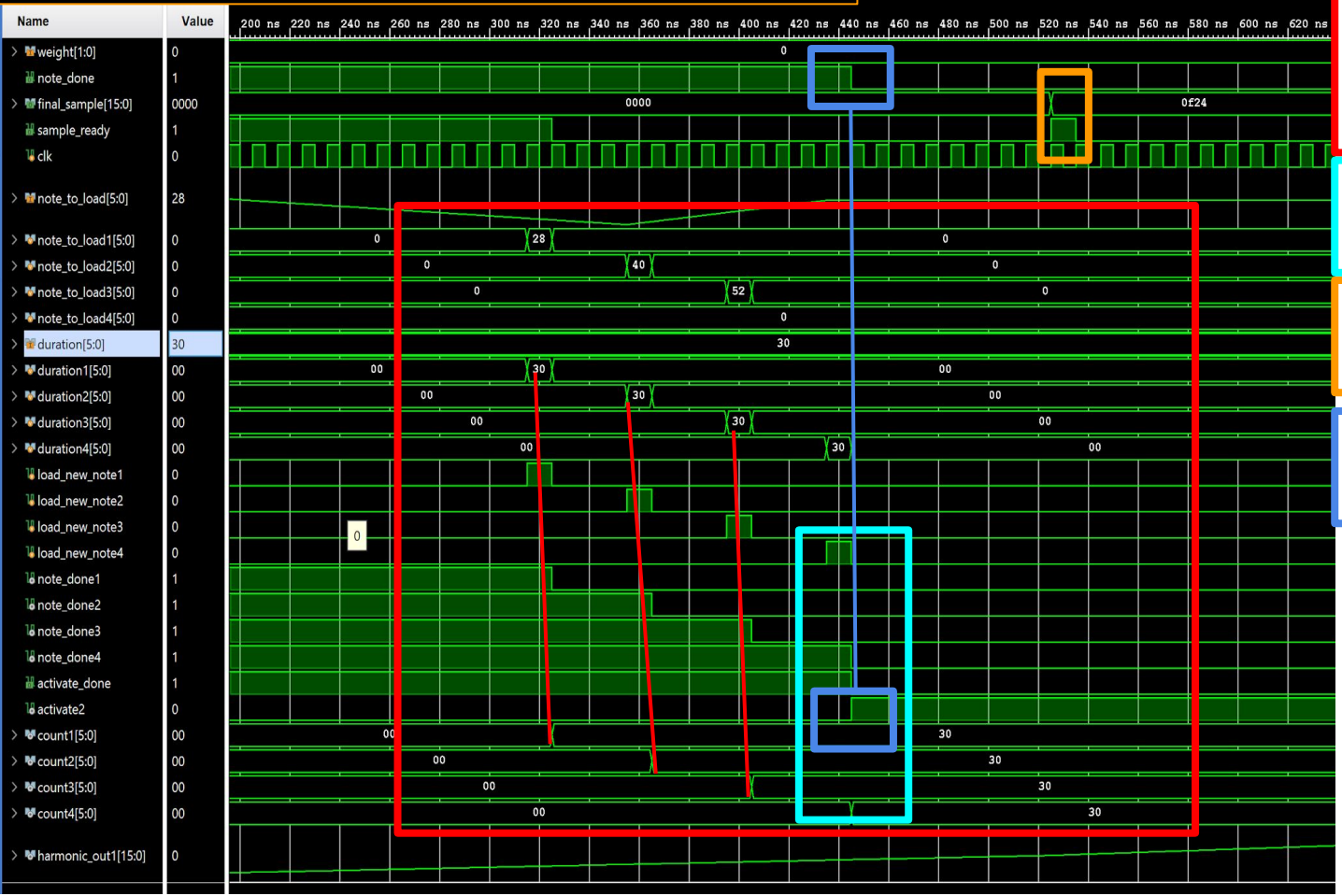
# note_player




This is the outer function that wraps create harmonics and has a similar functionality. Weight = 0, so we ask create harmonic for out1. When weight = 1, we ask for out2 and output it, and when weight = 3, we ask for out3 and output it,

Additionally, sample ready going low or note done going high will stop the sine waves from getting next sample.

# Harmonic Chord Player



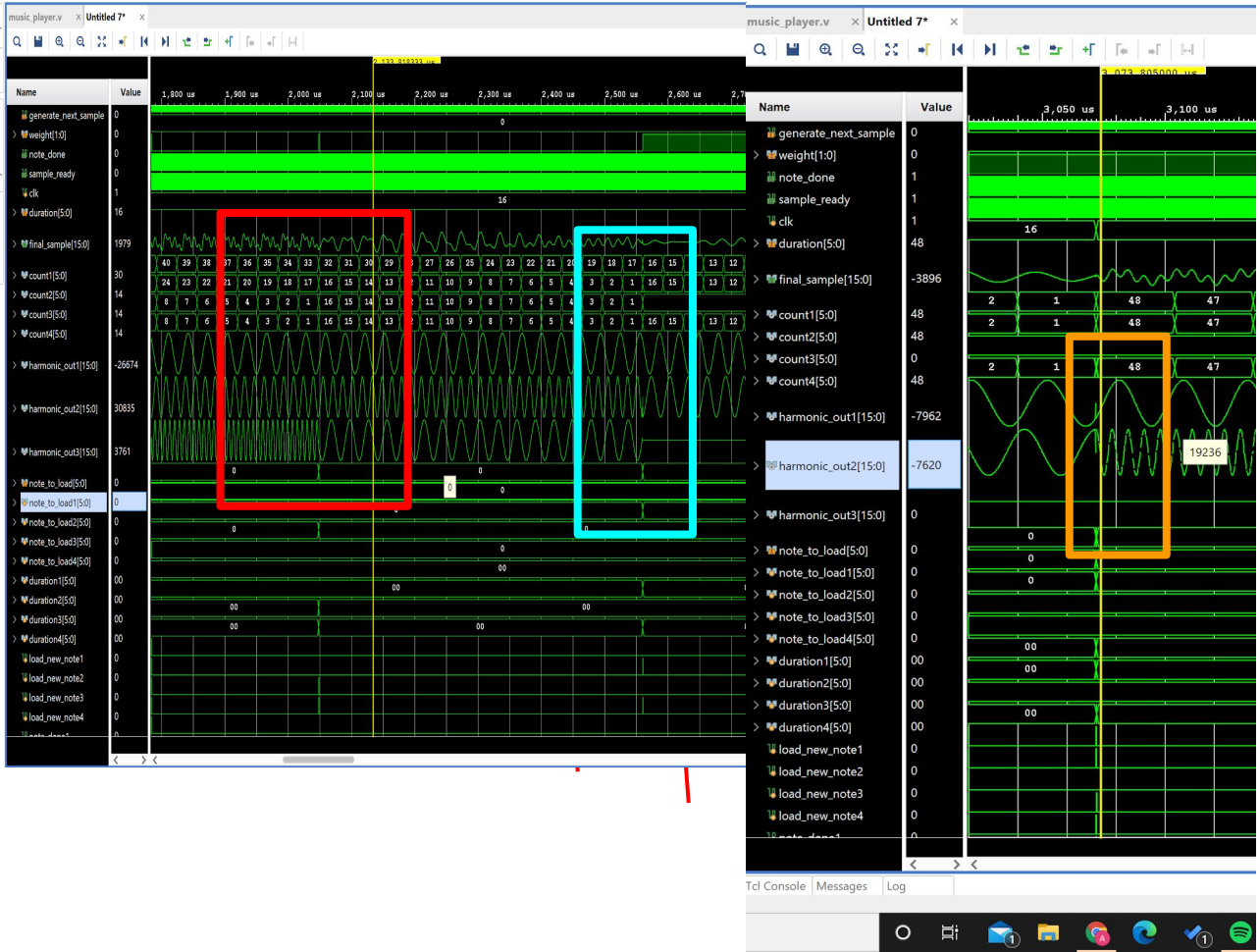

This module is made up of three note players and one additional counter that keeps track of “activate” which tells when to play the notes.This sequence demonstrates how information from song reader is loaded into the different note players.

Activate2 goes high, indicating the song should start playing

Several cycles later the first sample is ready to play

When all notes are loaded, note done turns off, signaling to song reader to stop sending information

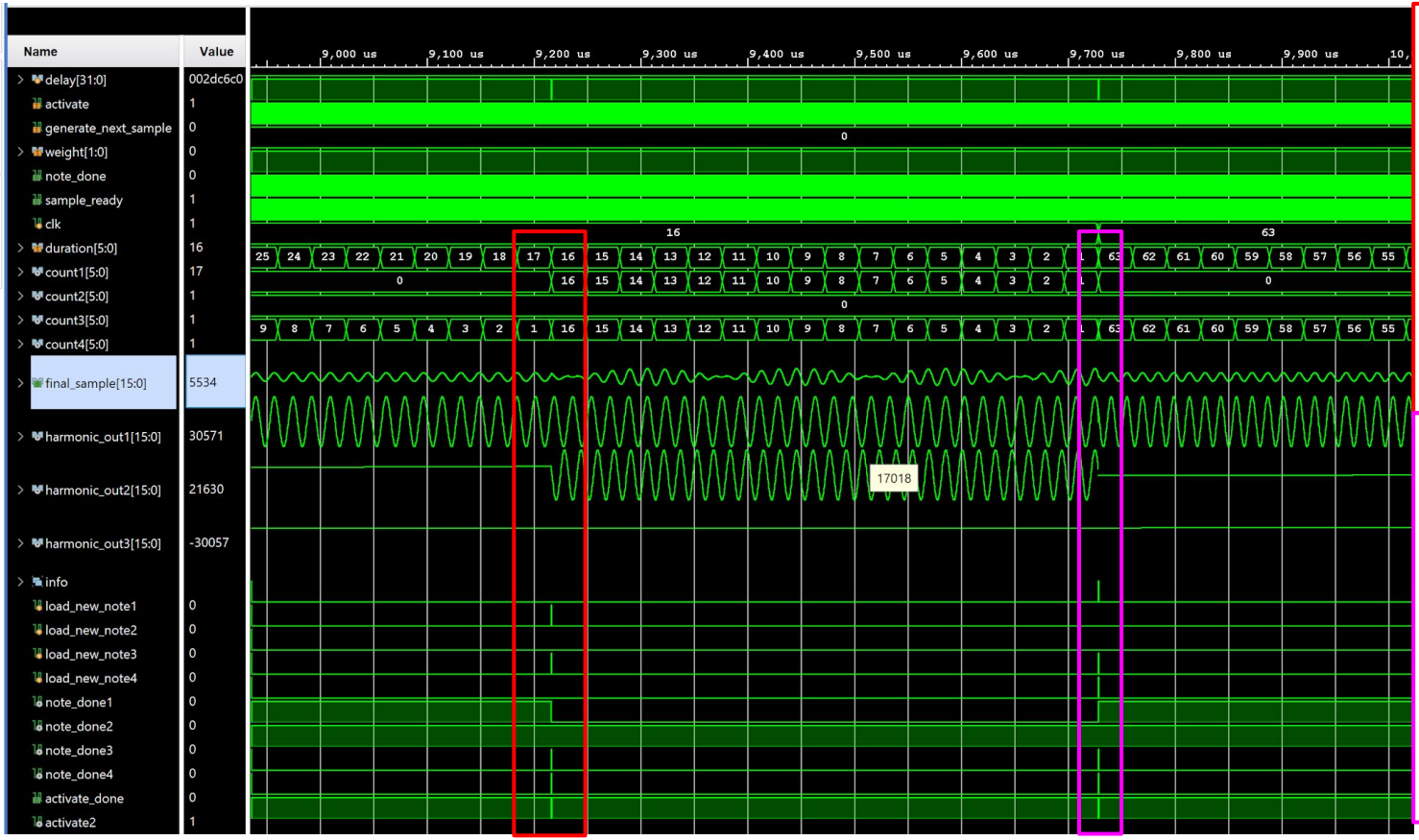
# Harmonic Chord Player




Finishing playing one note while continuing to play the other two notes.

Play only two notes at a time

The final sample is a combination of harmonic out1,2 and 3.
This part demonstrates that we are able to switch one note while continuing to play other notes.




Going from playing 1 note to two. Notice how note_done2 is high. Is the next free note player so the note loads into noteplayer2. Following that note is an activate which tells the two loaded notes to play.

Noteplayer 2 finishes its note and stops playing. Note playet 1 receives another note to play and continues playing it as long as activate2 is high.