Yi Ching Liao
SID: 2377483
EE371
Jun, 4, 2024
Lab 6 report

Design Procedure

My design for the final lab is a device that plays the music from a score while, on the VGA display, the corresponding piano key to each note will be shown. The user can choose to play two kinds of scores, a random or a pre-saved score. In the current design, there's only one pre-saved score, but with further increments of those, the users will have the ability to choose between saved songs. On the contrary, users can always generate a new random score.

Task#1 Generating and saving music scores

First of all, we have to identify the high-level idea of the processes to change a score notated with the musical alphabet to audio as well as video. Lancelot provided the idea of how MIDI works during my conversation with the TAs, which is storing musical contents merely by the notes and their duration. Instead of a musical alphabet, I use decimal numbers to represent both the note and duration.

Note: 0 = C4, 1 = C#4, 2 = D4, …., 23 = B5.
Duration: 0 = ½ beat, 1 = 1 beat, 2 = 2 beat, 3 = 4 beat

For the generated random score, these two numbers are stored in note_memory and lenght_memory in score_mem.sv module. Each pre-saved score contains 1 ROM for note and one for note, as well as stored in score_mem.sv.

The numbers in the random score are generating from 2 LSFR which ae modified to generate different random values. (These LSFR modules are modified from the LSFR I created in EE271) I made the generation ends when 64 notes were generated.

Task#2 Choose between random and saved score

Next, once the generation of the random score is finished, the machine enters a “ready” state, waiting for the user to select whether to play a random or pre-saved score. (KEY2 = Play random, KEY3 = Play saved)
Depending on the selection, the machine enters the state S_randnote_audio_video or S_savenote_audio_video and starts transferring numbers in the score memory into audio and video.

The transferring process involves an up-counter named length_counter. length_counter computes a max value with regard to each note’s length and the user input speed, and with the help of CLOCK_50, the up-counter can count until max and then generate a signal to notify the end of this note, which also triggers the next note.

Task#3 Transferring numbers to audio and video

I have two modules that received this note-end-notification signal(finish_len), note_to_audio.sv and note_to_video.sv. To read from the score memory, these modules output the address bits to score_mem.sv, and the address will increase by 1 every when finish_len is asserted for one clock cycle. This changes the note being played.

Moreover, we have to discuss how the numbers are used to define the notes. I create 24 ROMs and audio mif files for the notes spanning from C4 to B5. Then, case statements are used to match the note with the ROM and choosing which audio to output.

Finally, when the address reached 64(random score) or the end of the pre-saved score, the machine goes to the “done” state. In this state, the user can decide whether to replay the audio or generate and play a new random score.

ASM Chart of the control circuit

The machine’s state transitions are controlled by a control circuit, control.sv module. The following is the ASM chart of it.

/

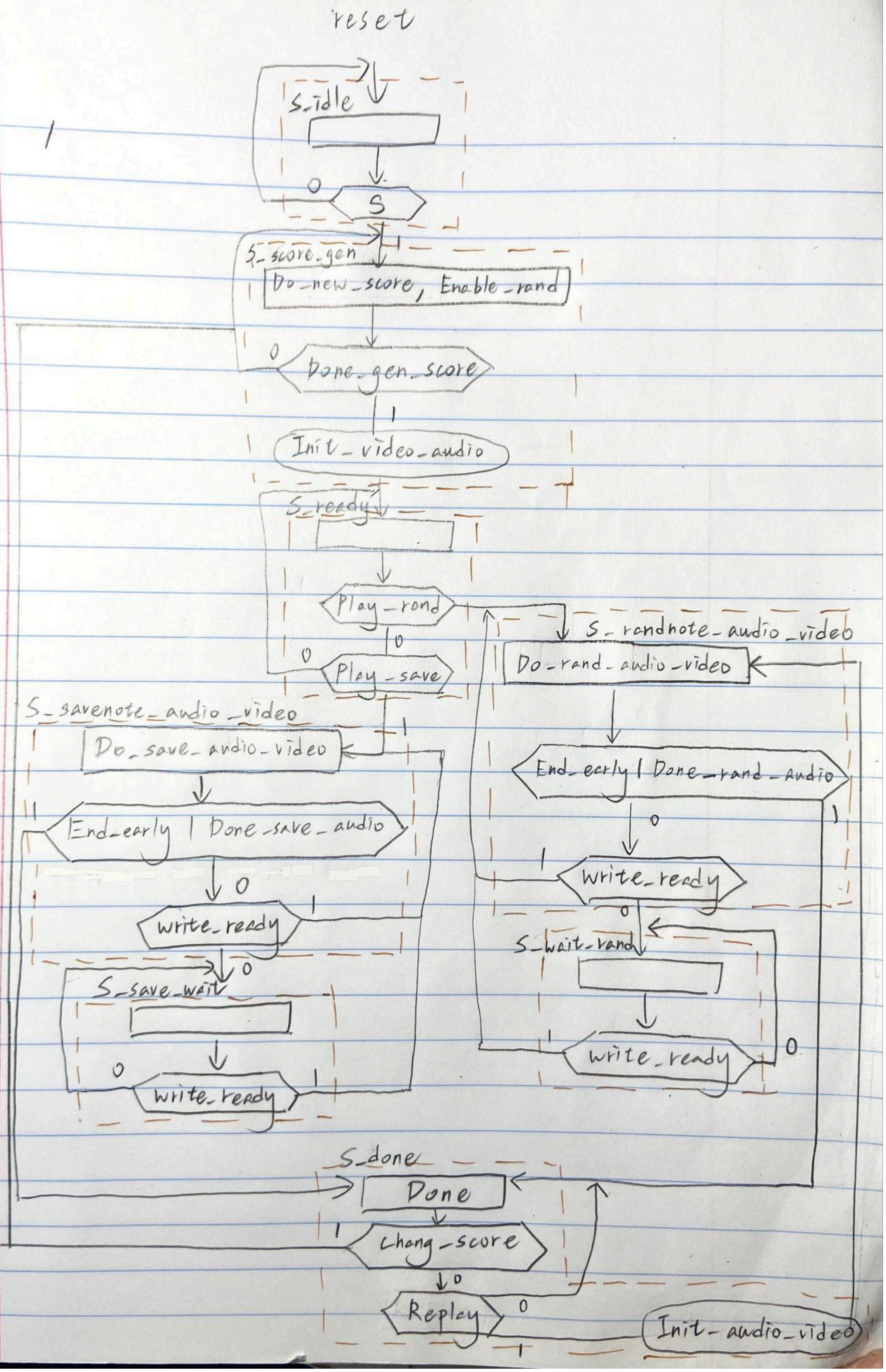

reset
S_idle
S
0
1
S_score_gen
Do_new_score, Enable_rand
Done_gen_score
0
1
Init_video_audio
S_ready
Play_rand
0
Play_save
0
1
S_randnote_audio_video
Do_rand_audio_video
End_early | Done_rand_audio
1
0
write_ready
1
0
S_wait_rand
write_ready
1
0
S_savenote_audio_video
Do_save_audio_video
End_early | Done_save_audio
1
0
write_ready
1
0
S_save_wait
write_ready
0
1
S_done
Done
Chang_score
1
0
Replay
0
1
Init_audio_video

Figure 1: ASM Chart of the control circuit of music generation machine

Top-level block diagram

All the signals are declared with the same name in all modules, to increase readability, I put the signal name on top of the wire instead of within the module blocks. The arrows help determine the input and output ports of the signal.

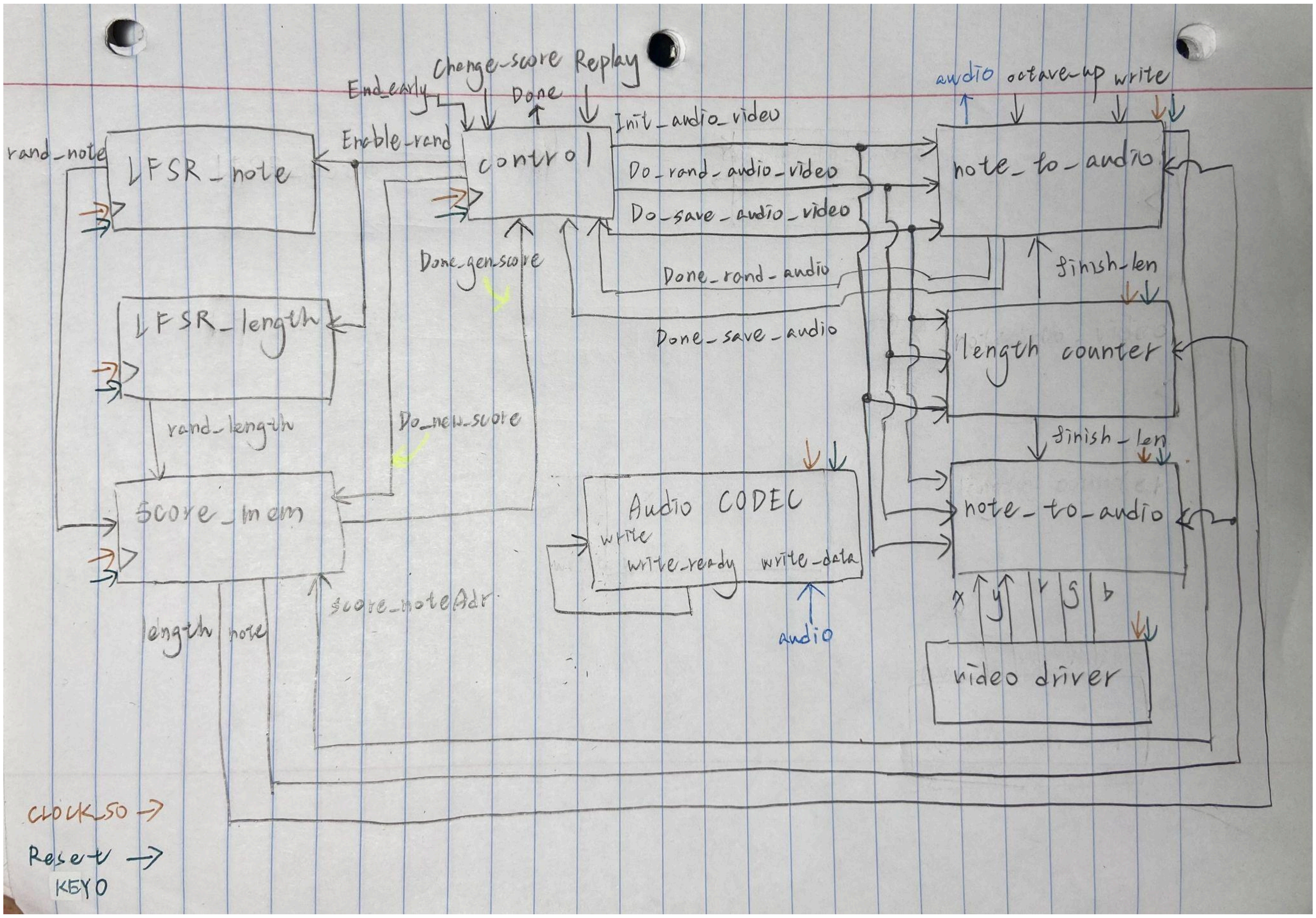


Figure 2: Top-level block diagram of music generation machine

Result

The final result differed from the proposal in two things. First, to synchronize different parts in the system, I added the length_counter.sv module that keeps track of the duration of notes and produces a signal pulse to inform the note change. Secondly, I discard the idea of using a FIFO buffer in between note_to_audio.sv and the audio codec. The FIFO buffer will increase the probability of unsynchronized audio and video.

Task 1 note_to_audio.sv

In the simulation, we wanna check if, by specifying the note value, note_to_audio.sv can output the correct 'audio' signals. Figure 3 shows that the note values matched the audio with q1~q5(highlighted), where q1~q5 is the output of ROMs storing C4~E4's audio signal.

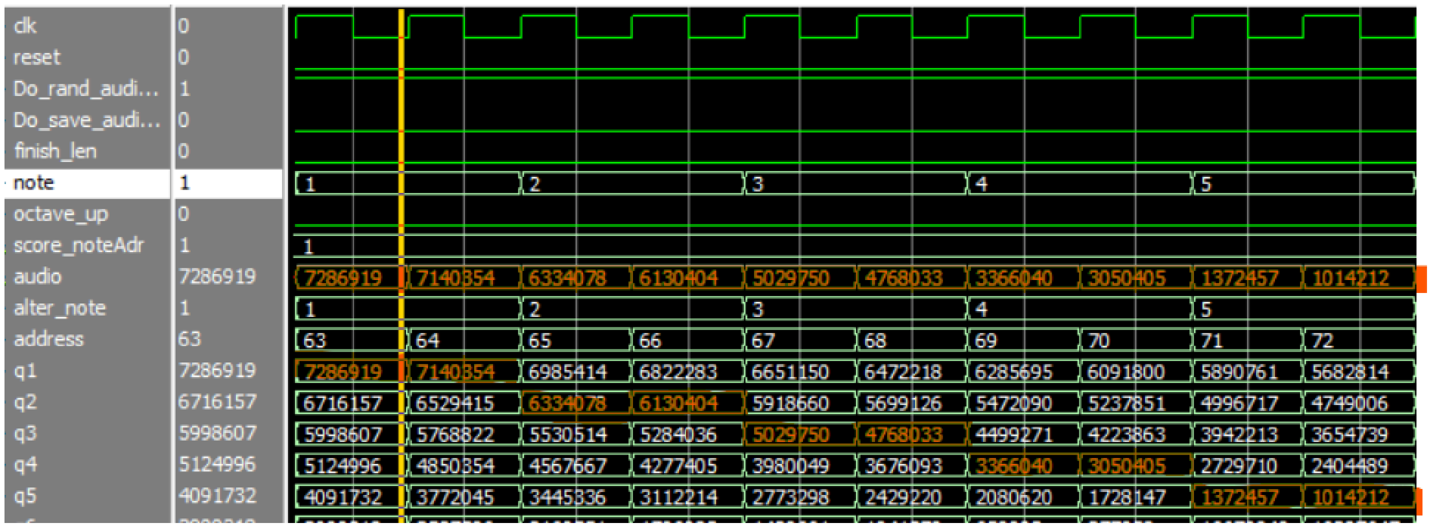


Figure 3: Modelsim simulation of testbench of note_to_audio.sv

Task 2 note_to_video.sv

The note is assigned to 0, and x, y are assigned with the following for loop.

```
for (i=0; i<=30; i=i+1) begin
                for (j=0; j<=180; j=j+1) begin
                        x <= i;
                        y <= j;
                        @(posedge CLOCK_50);
                end
        end
```

Whether or not the current location (x, y) is within the blue circle decides the RGB values. If the current location (x, y) is within the circle, the RGB values are defined as blue, otherwise they are determined by the keyboard color.

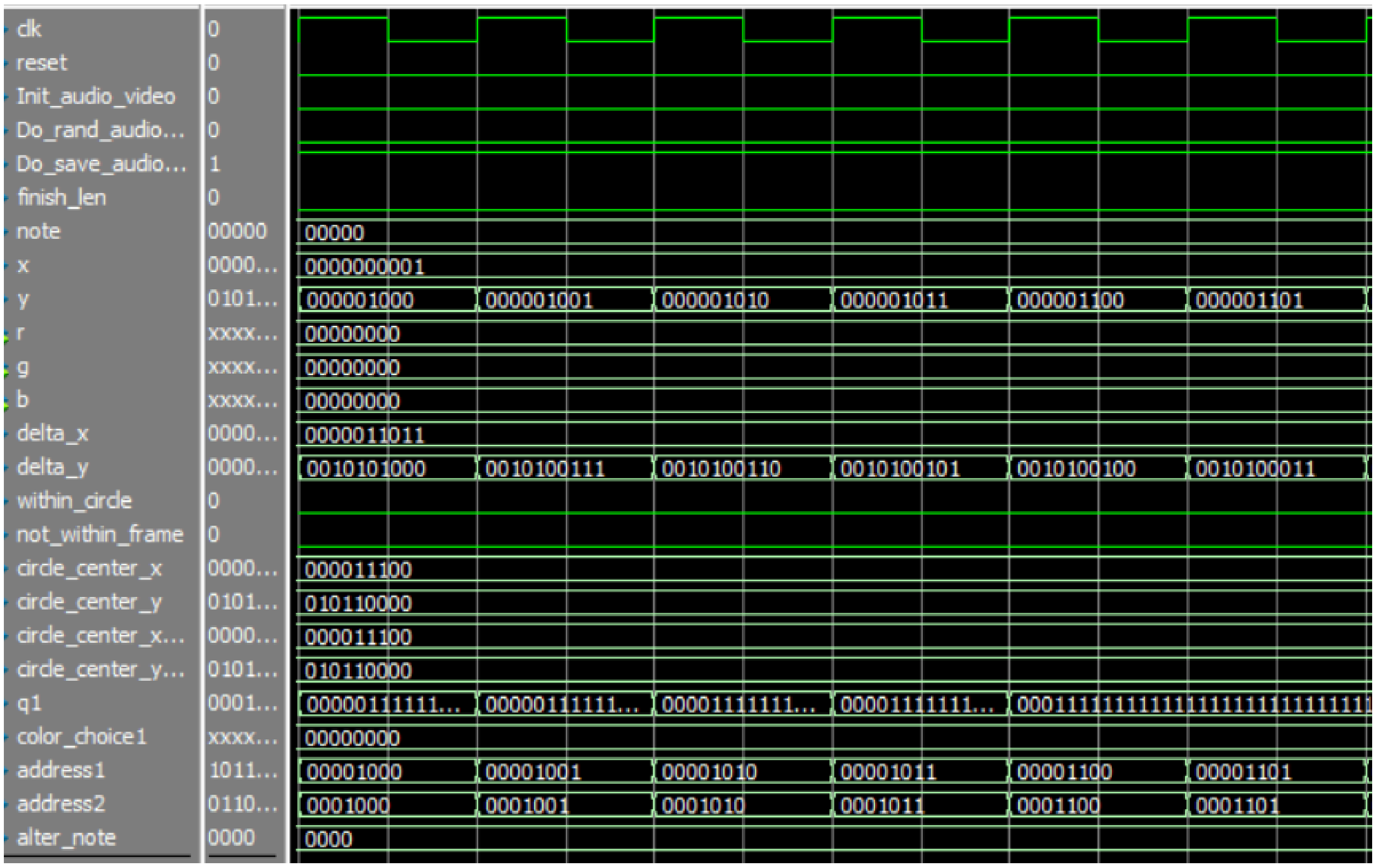


Figure 4: Modelsim simulation of testbench of note_to_video.sv

Task 3 length_counter.sv

Originally under CLOCK_50, the length_counter.sv had to count 50M clock cycles for each 60BPM beat. But for the testbench, I reduce the cycles in the following manners:

speed =>	0	1	2	3
numbers of clock cycles	3	6	9	12

The expected behavior of the simulation is to see finish_len being asserted one clock cycle when the count reaches the max value. In Figure 5, the length_counter.sv behaves as expected for 4 combinations of length and speed.

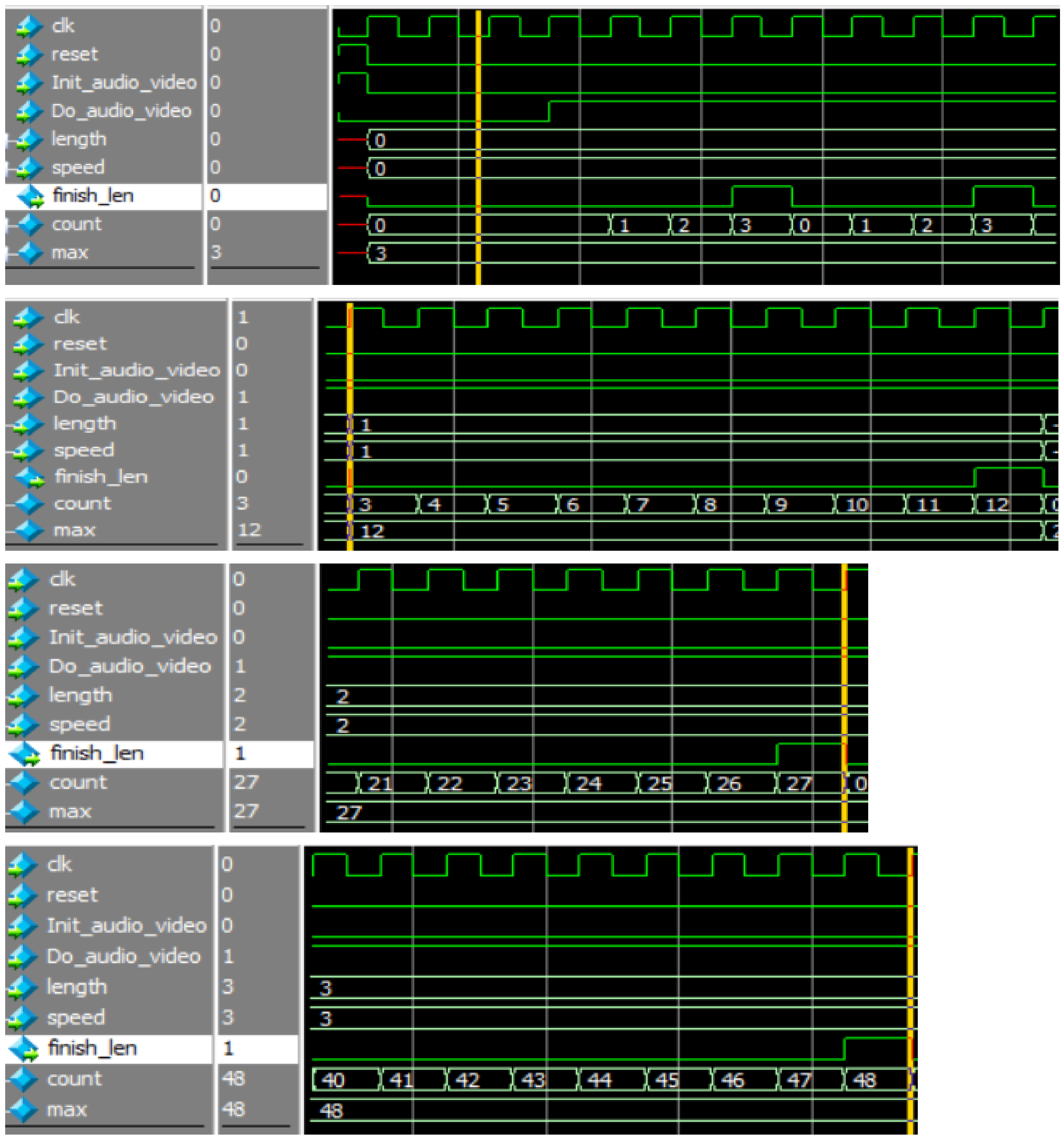


Figure 5: Modelsim simulation of testbench of length_counter.sv

Task 4 score_mem.sv

While Do_new_score is asserted, score_mem.sv module stores rand_note into note_memory and rand_length into length_memory. So at the beginning of the testbench simulation, we store values in the memories until they're full. Figure 6 shows the assigned sequence of rand_note and rand_length. Next, we deassert Do_new_score and read from the memories to check if the values are properly stored. From Figure 7 we can verify that values are stored in the correct manner.

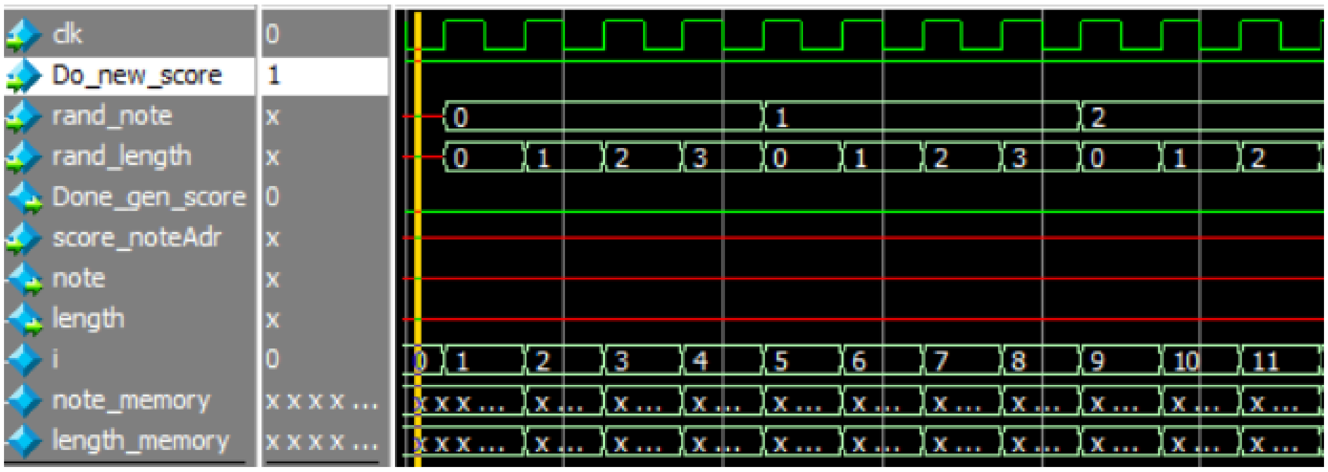


Figure 6: Modelsim simulation of testbench of score_mem.sv (storing memories)

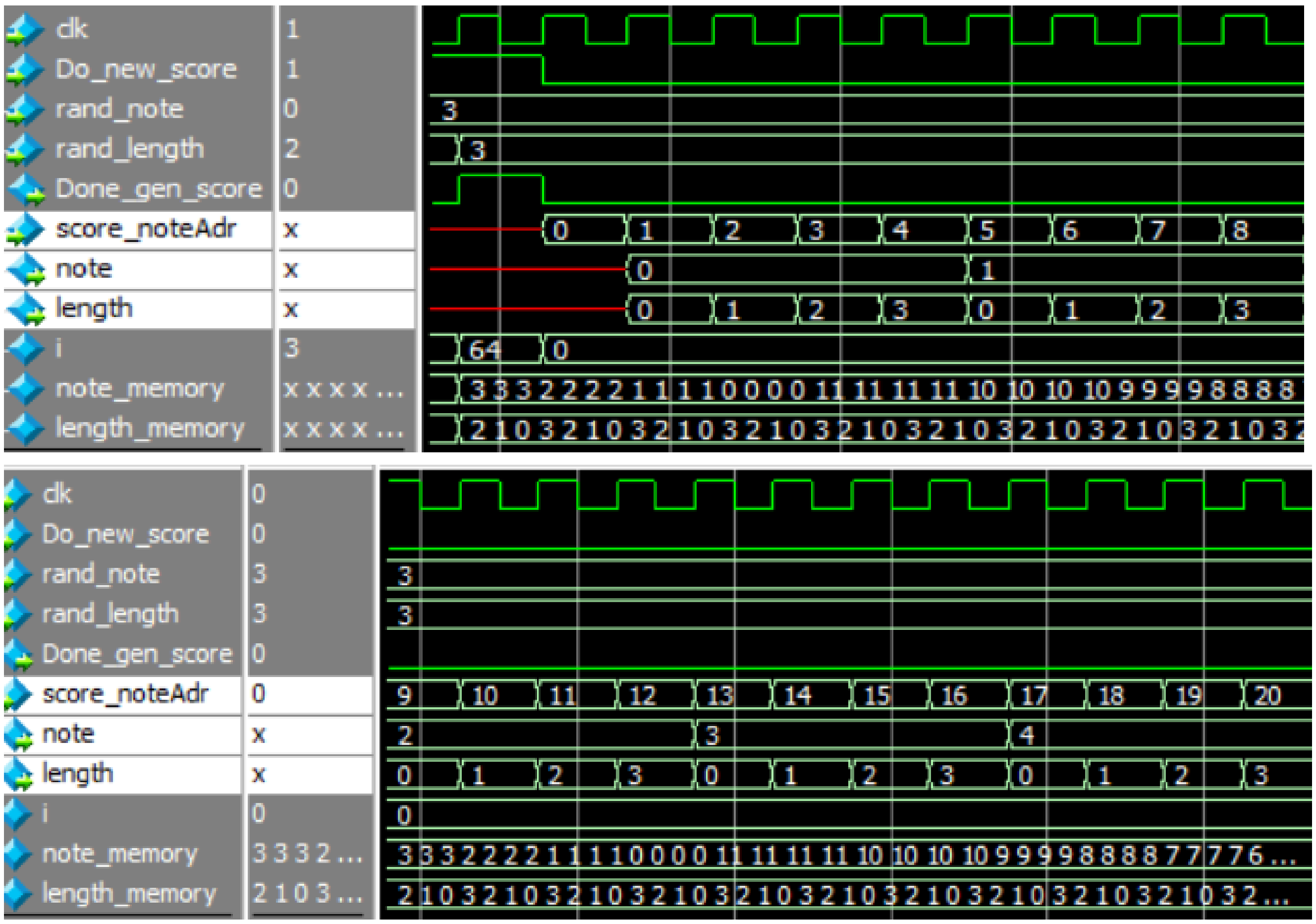


Figure 7: Modelsim simulation of testbench of score_mem.sv (reading memories)

Task 5 DE1_SoC_test_purpose.sv

As setting up a testbench involving audio and VGA signals is complicated, I created an additional top-level module, DE1_SoC_test_purpose, especially for simulation. DE1_SoC_test_purpose is obtained from the original DE1_SoC eliminating the initiation of audio codec and video driver. However, by verifying that the signals transferred to the audio codec and video driver are properly produced, we can still effectively check our top-level module's behavior. In the following section, we checked 1. The states transferred correctly. 2. In each state, the desired outputs from different modules are gained. To increase readability, the signals are rearranged in ModelSim and important signals are cropped from the image of the top-level simulation for each state.

1. S_score_gen
   After s is asserted, the machine should enter S_score_gen. Do_new_score and Enable_rand will be asserted until note_memory and length_memory are full. In Figure, we verified that S_score_gen is entered as rand_note/length starts generating. Then, Done_gen_score is asserted when the memories are filled with random numbers. At the same time, Now_S3 indicates that we have entered S_ready(is also S3).

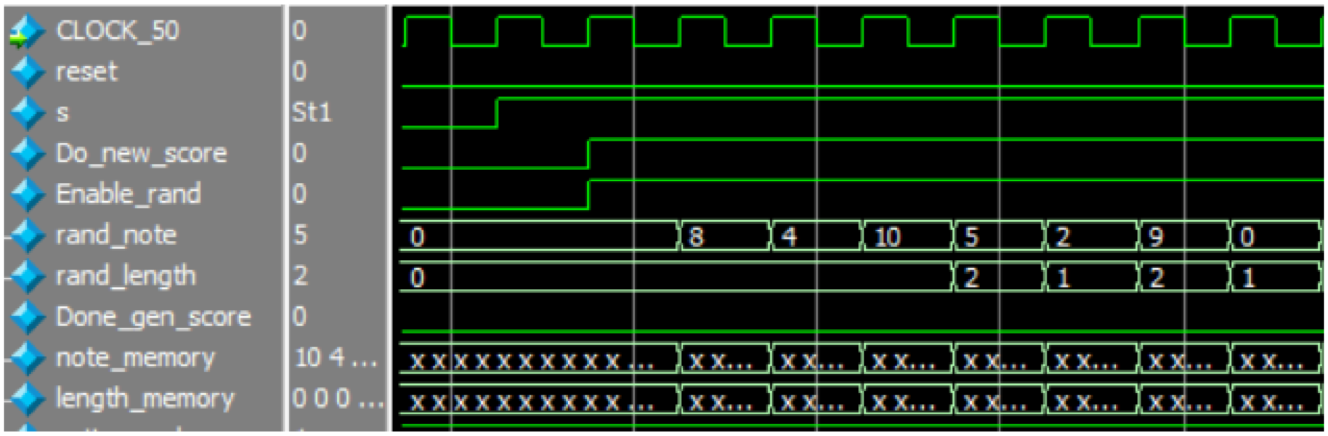


Figure 8: Modelsim simulation of testbench of DE1_SoC_test_purpose.sv (S_score_gen)

2. S_rand/savenote_audio_video
   There are two possible next states for S_ready. In Figure 9 and 10, we chose to enter S_randnote_audio_video and S_savenote_audio_video respectively. The two states share similar outcomes. We can see that, after every pulse of finish_len, the scorre_noteAdr(indicates which address to read from score memory) increases and consequently changes the note's and length's value. The 'audio' signal outputs the audio waveform's sampled value. Figure and show the start and end of states S_rand/savenote_audio_video.

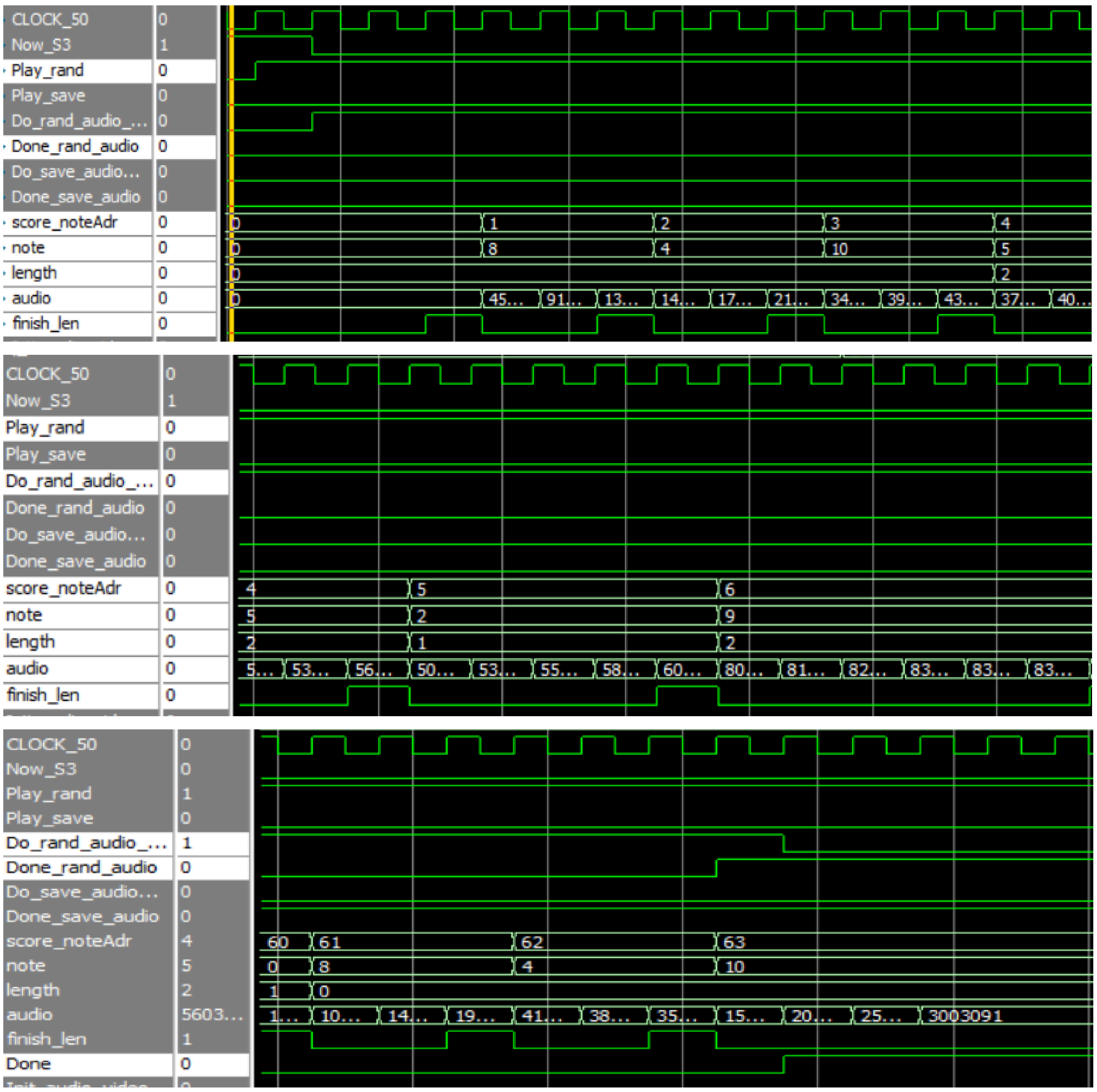


Figure 9: Modelsim simulation of testbench of DE1_SoC_test_purpose.sv (S_randnote_audio_video)

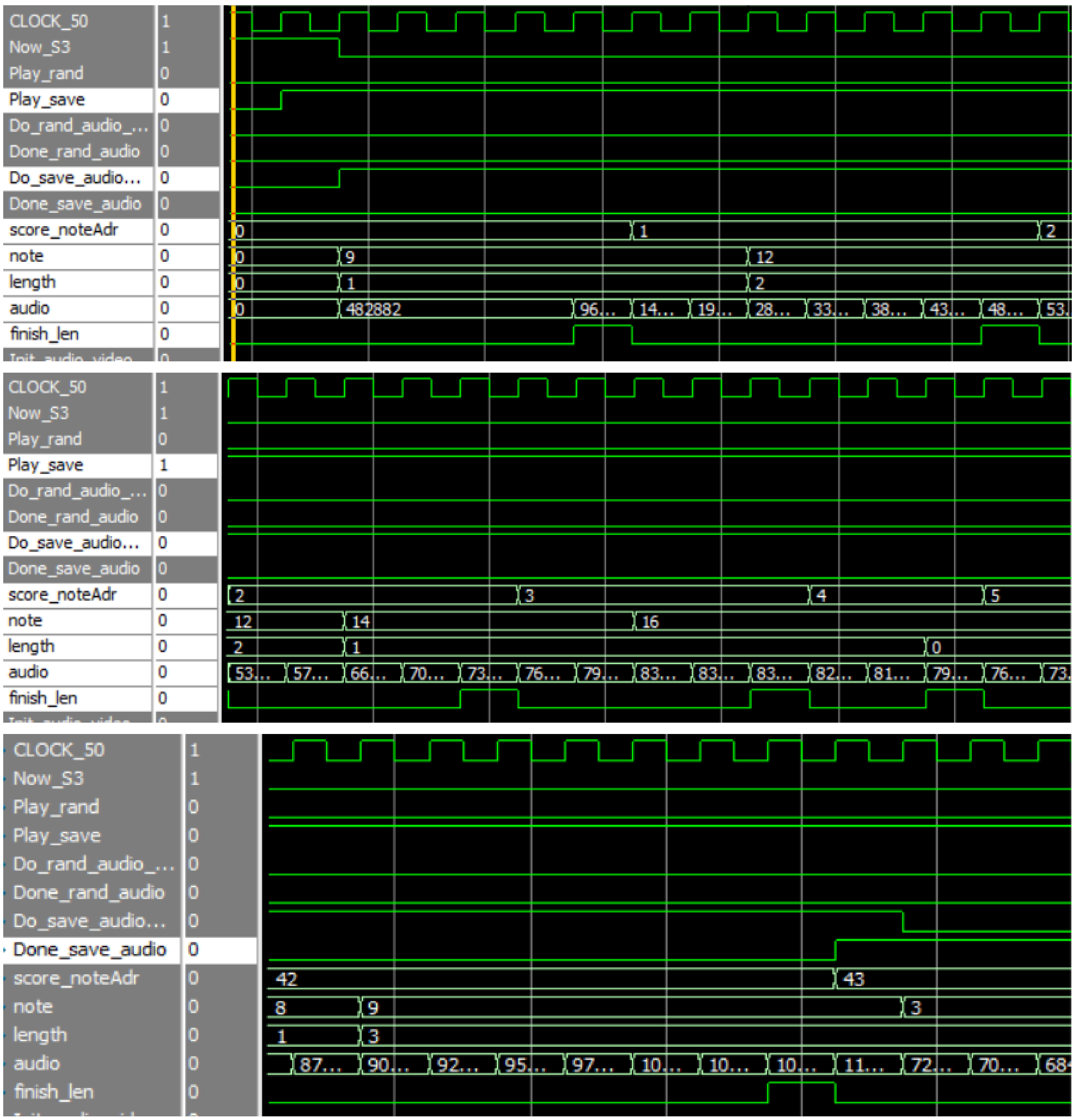
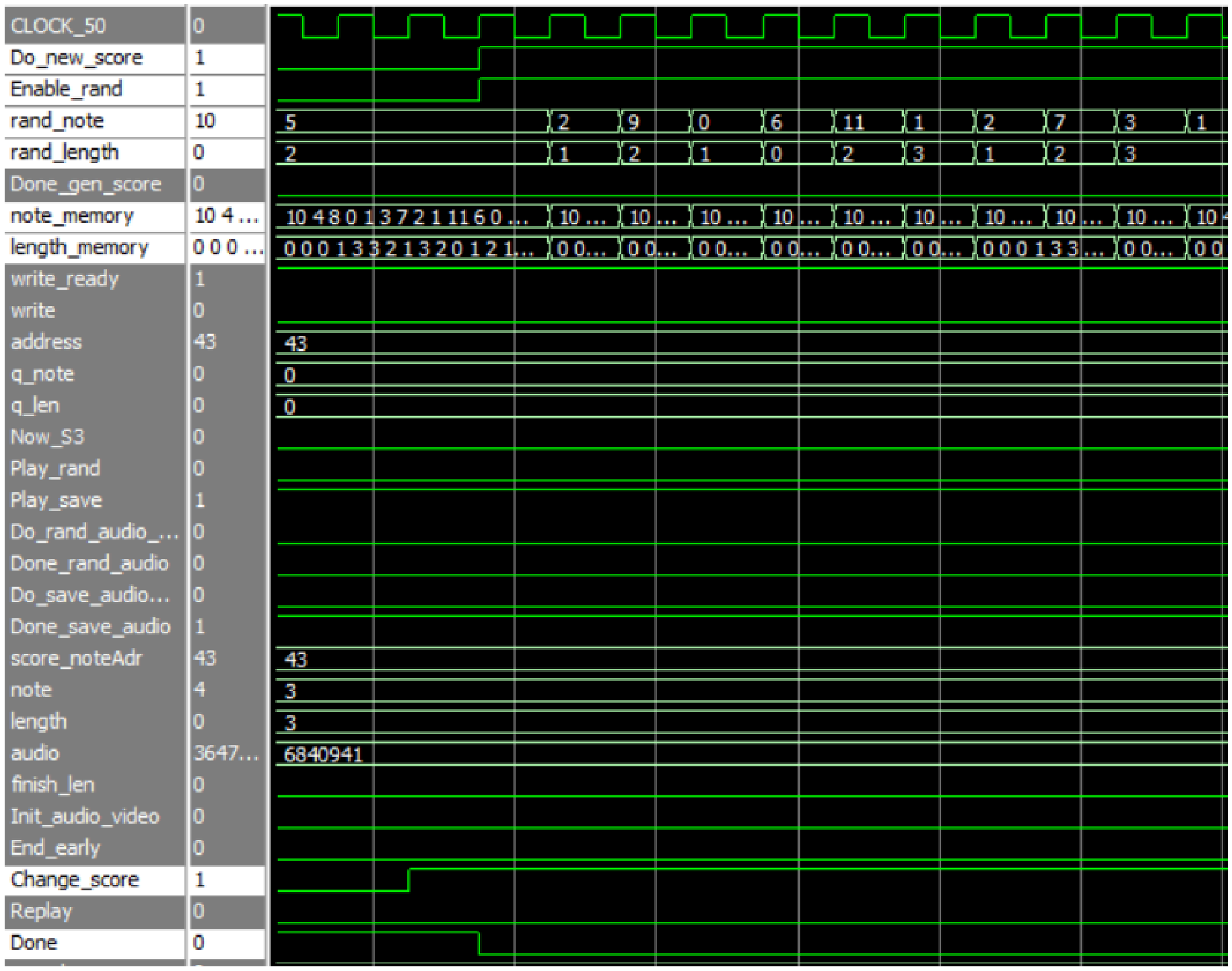


Figure 10: Modelsim simulation of testbench of DE1_SoC_test_purpose.sv (S_savenote_audio_video)

3. S_done

   Once the Done_rand/save_audio is asserted, the machine enters the S_done state, waiting for the user to specify whether to replay or generate a new random score. Figure 11 demonstrates the case when the user hits Change_score. The machine exits S_done and enters S_score_gen to fill note_memory and length_memory with different values.



CLOCK_50	0
Do_new_score	1
Enable_rand	1
rand_note	10
rand_length	0
Done_gen_score	0
note_memory	10 4 ...
length_memory	0 0 0 ...
write_ready	1
write	0
address	43
q_note	0
q_len	0
Now_S3	0
Play_rand	0
Play_save	1
Do_rand_audio_...	0
Done_rand_audio	0
Do_save_audio...	0
Done_save_audio	1
score_noteAdr	43
note	4
length	0
audio	3647...
finish_len	0
Init_audio_video	0
End_early	0
Change_score	1
Replay	0
Done	0

Figure 11: Modelsim simulation of testbench of DE1_SoC_test_purpose.sv (S_done)

Experience Report

This lab is by far the most complicated one for me. However, I have learned a lot about how to 1. Identify a task with a suitable difficulty that is within my ability but is yet challenging. 2. Change abstract ideas into reality. 3. Utilize the ASM chart and the idea of control and data circuits. The spec for this lab provides enough information and resources for me to implement more advanced functions.

I have met lots of problems and the following includes some of those. First, I have trouble with the synchronization of the audio and video. Both note_to_audio.sv and note_to_video.sv have their own counter to count for the length of the notes. This resulted in a disaster as the video and audio were literally doing their own things. Thus, I made the module length_counter.sv which acts like a control signal for note transition. However, I still found that the audio is lagging the video. This is caused by the FIFO buffer for the audio as it increases the timing difference between the audio and video as the buffer size grows. Secondly, the VGA display is, at first, a little confusing. At first, I assign the RGB outputs in an always_ff block. In the VGA display, the figure was heavily distorted. I then realized that we should use the combinational logic circuit to interact with the video driver or else the output will be delayed and be misplaced on the screen. Lastly, I found my machine stuck at the S_wait state and froze at the first note. With the help of Professor Justin, I realized that the write_ready signal is asserted very shortly that a sequential logic circuit can't capture the signal and I should instead use a combinational logic circuit. Consequently, I assign write to write_ready and deleted two of my S_wait states.

This lab took me approximately 51 hours, broken down as follows:
• Reading – 30 minutes
• Planning – 2 hours
• Design – 2 hours
• Coding – 17 hours
• Testing – 5 hour
• Debugging – 17 hour
• Making Lab report and video– 8 hour