Lab 3 - Information README

1. Where (which directory path) is your SOF file located, and what it is called?

The SOF file name is simple_ipod_solution.sof and is located in the folder:
\Daniel_Li_51814762_Lab_3\rtl\lab3_template_de1soc

2. What is the status of the lab (what works, what doesn't)

All required functionality for this lab has been fully implemented and verified. This functionality is compatible with both the bonus parts from Lab 2. This includes the "R" key functionality and the audio playback at 44KHz with 8-bit samples. The functionalities have been integrated into my modules. In the case of the 44KHz, 8-bit sample, I have overridden the original 16-bit 22KHz audio sample.

I have included all program files and testbenches for all modules implemented in Lab 2 to confirm the correctness of the code through simulation. This includes the SignalTap simulation waveforms. The functionality includes the Flash FSM Controller, keyboard interface, frequency controller, frequency clock divider, Synchronizer, LED display, and SignalTap code.

3. Annotated simulation screenshots as required by the lab

No annotations were required for this Lab, but I have included annotations from Lab 2 in the Appendix.

4. Annotated SignalTap screenshots as required by the lab

No annotations were required for this Lab, but I have included annotations from Lab 2 in the Appendix.

5. Information on how to run the simulations (i.e. where the files are located, which program you used to run your simulation)

Simulations were not required for this lab, but I have included instructions on how to run them in the Appendix.

6. Any additional information that would be relevant for the TA marking your project.

I have included the pacoblaze directory containing the necessary files for running the program, the pracPICO.psm file containing the picoblaze code, and the PRACPICO.MEM file which contains the data for running the program.

I have also included the drawing of my Flash FSM from Lab 2 as a reference, although this is not required for this lab. The picture is located in the folder: Flash_FSM_State_Transitions_Drawing.

If there is any additional information required for the lab, please don't hesitate to reach out and let me know. Thank you for marking my lab!

Appendix

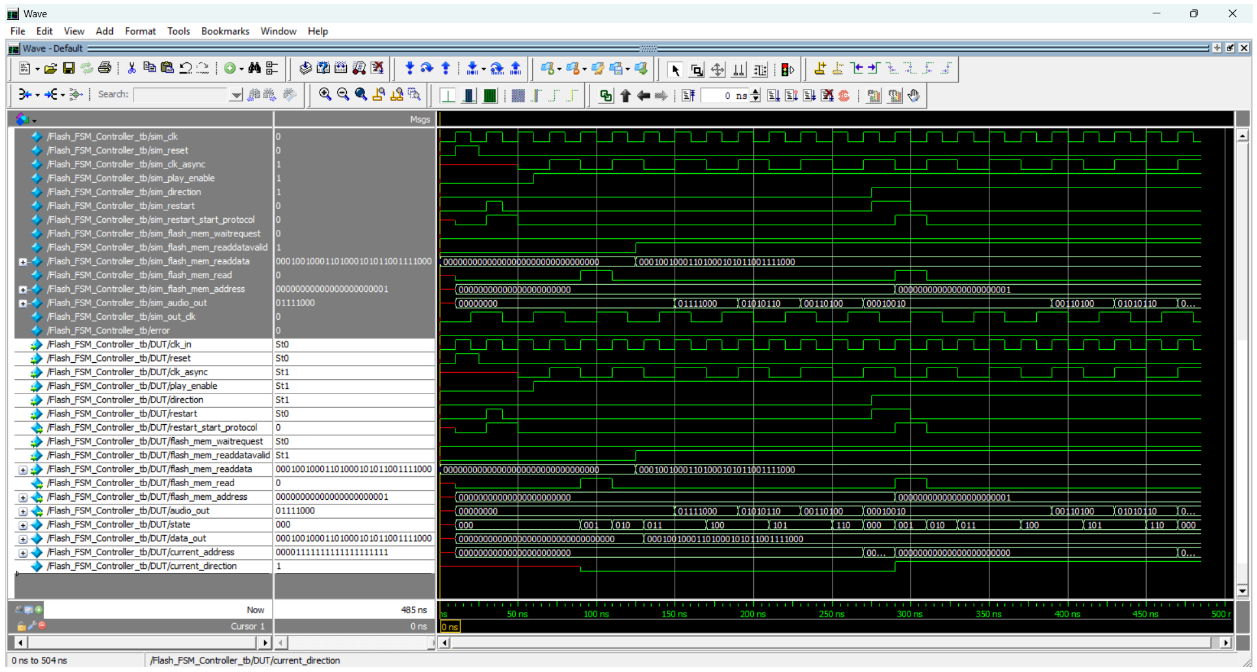
Instructions on how to run the simulations:

I have included the simulation testbenches and all the code required to run them in the folder: \Daniel_Li_51814762_Lab_3\sim. I have also included screenshots of the simulation results and .do files in subfolders within this folder. The program I used to run my simulations is ModelSim and I have included the .mpf file for the project in the folder.

Annotated simulations:

The annotated simulation screenshots are attached below for reference. I have included notes below illustrating the different aspects of the testbench and test cases I have aimed to simulate. They are attached in the following pages.

Below is a picture of the Flash_FSM_Controller testbench simulation:



```
/Flash_FSM_Controller_tb/sim_clk                           0
/Flash_FSM_Controller_tb/sim_reset                         0
/Flash_FSM_Controller_tb/sim_clk_async                     1
/Flash_FSM_Controller_tb/sim_play_enable                   1
/Flash_FSM_Controller_tb/sim_direction                     1
/Flash_FSM_Controller_tb/sim_restart                       0
/Flash_FSM_Controller_tb/sim_restart_start_protocol        0
/Flash_FSM_Controller_tb/sim_flash_mem_waitrequest         0
/Flash_FSM_Controller_tb/sim_flash_mem_readdatavalid       1
/Flash_FSM_Controller_tb/sim_flash_mem_readdata            00010010001101000101011001111000
/Flash_FSM_Controller_tb/sim_flash_mem_read                0
/Flash_FSM_Controller_tb/sim_flash_mem_address             00000000000000000000001
/Flash_FSM_Controller_tb/sim_audio_out                     01111000
/Flash_FSM_Controller_tb/sim_out_clk                       0
/Flash_FSM_Controller_tb/error                             0
/Flash_FSM_Controller_tb/DUT/clk_in                        St0
/Flash_FSM_Controller_tb/DUT/reset                         St0
/Flash_FSM_Controller_tb/DUT/clk_async                     St1
/Flash_FSM_Controller_tb/DUT/play_enable                   St1
/Flash_FSM_Controller_tb/DUT/direction                     St1
/Flash_FSM_Controller_tb/DUT/restart                       St0
/Flash_FSM_Controller_tb/DUT/restart_start_protocol        0
/Flash_FSM_Controller_tb/DUT/flash_mem_waitrequest         St0
/Flash_FSM_Controller_tb/DUT/flash_mem_readdatavalid       St1
/Flash_FSM_Controller_tb/DUT/flash_mem_readdata            00010010001101000101011001111000
/Flash_FSM_Controller_tb/DUT/flash_mem_read                0
/Flash_FSM_Controller_tb/DUT/flash_mem_address             00000000000000000000001
/Flash_FSM_Controller_tb/DUT/audio_out                     01111000
/Flash_FSM_Controller_tb/DUT/state                         000
/Flash_FSM_Controller_tb/DUT/data_out                      00010010001101000101011001111000
/Flash_FSM_Controller_tb/DUT/current_address               00001111111111111111111
/Flash_FSM_Controller_tb/DUT/current_direction             1

Now 485 ns
Cursor 1 0 ns
0 ns to 504 ns    /Flash_FSM_Controller_tb/DUT/current_direction
```

```
//Define States
`define IDLE 3'b000
`define ADDRESS 3'b001
`define WAIT_VALID 3'b010
`define OUTPUT1 3'b011
```

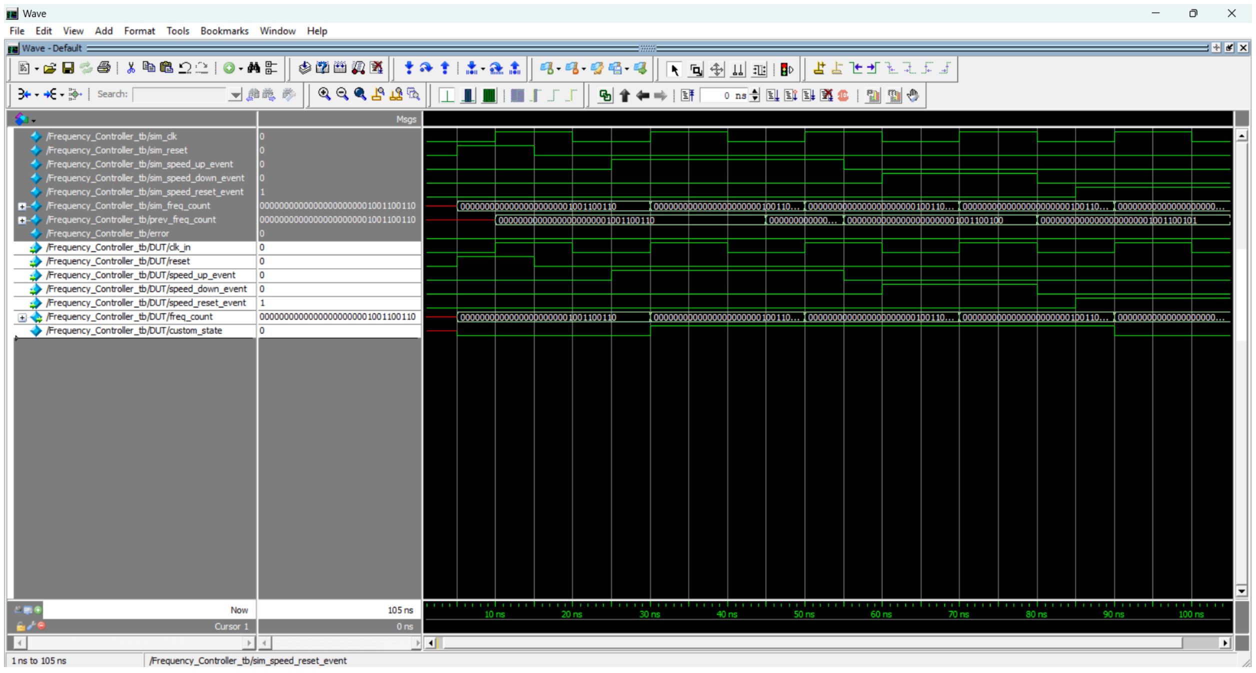
```
`define OUTPUT2 3'b100
`define OUTPUT3 3'b101
`define OUTPUT4 3'b110

//Stop and start music
`define STOP_MUSIC 1'b0
`define PLAY_MUSIC 1'b1

//Playback mode
`define FORWARD 1'b0
`define BACKWARD 1'b1

//Max address
`define MAX_ADDRESS 23'h7FFFF
```

- The Flash_FSM_Controller is responsible for interacting with the flash device and outputting data samples to the audio
- It is composed of 7 states: the first 3 are for interacting with the flash device and the last 4 are for outputting the audio
- In this testbench, my goal was to step through the different stages and complete an entire cycle of operation
- Overall, I went through 2 cycles of my FSM, one where the direction of audio was forwards and one where the direction of audio was backwards
- I had 2 clocks, one which was the 50MHz clock and another at 25MHz (just for simulation purposes, I chose this value), I used my synchronizer to test the synchronization of these 2 clock in my FSM
- I first set the reset signal of the FSM which brought me to the IDLE state of my FSM and all internal and external signals reset to 0
- Then I tested my restart signal (starts song from the beginning generated by pressing the “R” key), this brought me to the DUT.current_address of the data of 0. In addition, the start-finish protocol for my “R” key was set to 1 to indicate the action corresponding to the “R” key had taken affect
- Next, I set the play_enable to `PLAY_MUSIC and this along with the rising edge of the 25MHz (synchronized) brought me to the `ADDRESS stage which is the stage I pass the flash_mem_read signal and address data to the flash
- Following this, I waited until the waitrequest signal was deasserted which indicated the flash has my data to proceed to the next state `WAIT_VALID where I set the flash_mem_read signal back to 0
- Now this was the stage where my FSM waits for the readdatavalid signal from the flash to indicate the data provided by the flash was valid. Since I did not hook

it onto the flash in my testbench, I generated the signal myself and provided my own data which was 32’h12345678 for simplicity

- With the synchronized rise of the 25MHz and the 50MHz in addition with the PLAY_MUSIC value in the play_enable signal, the FSM progressed through the 4 different output stages which each output a 8 bit signal to the audio
- Although in this testbench I used a 25MHz clock, in my real module I used a 44KHz clock for the bonus
- I repeated the above steps for another cycle to test the audio playing in the Backwards direction. The only difference mainly between this and the Forward direction was the order in which the audio data was playing (reversed as indicated by the simulation waveform) and the current_address signal which was decrementing instead of incrementing

Below is a picture of the Frequency_Controller testbench simulation:



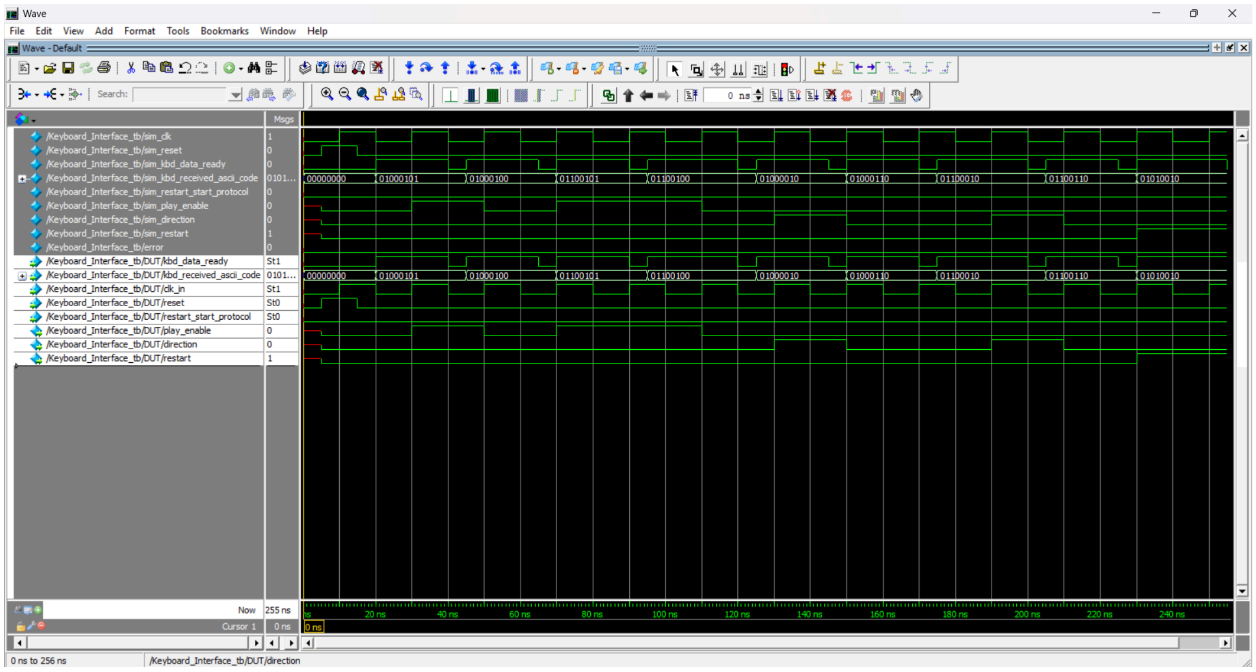
```
/Frequency_Controller_tb/sim_clk                 0
/Frequency_Controller_tb/sim_reset               0
/Frequency_Controller_tb/sim_speed_up_event      0
/Frequency_Controller_tb/sim_speed_down_event    0
/Frequency_Controller_tb/sim_speed_reset_event   1
/Frequency_Controller_tb/sim_freq_count          00000000000000000000001001100110
/Frequency_Controller_tb/prev_freq_count         00000000000000000000001001100110
/Frequency_Controller_tb/error                   0
/Frequency_Controller_tb/DUT/clk_in              0
/Frequency_Controller_tb/DUT/reset               0
/Frequency_Controller_tb/DUT/speed_up_event      0
/Frequency_Controller_tb/DUT/speed_down_event    0
/Frequency_Controller_tb/DUT/speed_reset_event   1
/Frequency_Controller_tb/DUT/freq_count          00000000000000000000001001100110
/Frequency_Controller_tb/DUT/custom_state        0
Now 105 ns
Cursor 1 0 ns
1 ns to 105 ns    /Frequency_Controller_tb/sim_speed_reset_event
```

```
//Music frequency
// `define MUSIC_FREQUENCY_DEFAULT 32'd1228 //for 16 bit samples at 22KHz
`define MUSIC_FREQUENCY_DEFAULT 32'd614 //Bonus: for 8 bit samples at 44KHz
`define MIN_FREQUENCY_COUNT 32'd1 //fastest frequency
`define MAX_FREQUENCY_COUNT 32'd4000 //slowest frequency
`define INCREMENT 32'd1 //Not specified in lab, value I chose to illustrate how the frequency changes dramatically
```

```
//Define default state
`define DEFAULT 1'b0 //default frequency is played
`define CUSTOM 1'b1 //custom frequency is played
```

- In this testbench I aimed to simulate how the freq_count changed based on the inputs to the system
- First, I reset the module, this ensured the custom_state was at the `DEFAULT value and the freq_count would be at the value of `MUSIC_FREQUENCY_DEFAULT which was the default value of my audio frequency (44KHz)
- Then I proceeded to give 2 speed_up_event signals to the Frequency_Controller to simulate how it changes the freq_count counter
- As indicated by the waveform, this set the value of the custom_state to `CUSTOM and the freq_count to be decremented by 2 (increasing frequency, decreases the count)
- I then proceeded to give a speed_down_event signal to the Frequency_Controller, this did not change the value of the custom_state (`CUSTOM) but it did increment the value of the freq_count by 1 (decreasing frequency, increases the count)
- Finally, since the value of freq_count was 1 above that of the `MUSIC_FREQUENCY_DEFAULT value, I decided to give a speed_reset_event to test if the freq_count resetted to the default value and if the custom_state reset to `DEFAULT. As indicated by the waveform, the results did reset as expected

Below is a picture of the Keyboard_Interface testbench simulation:

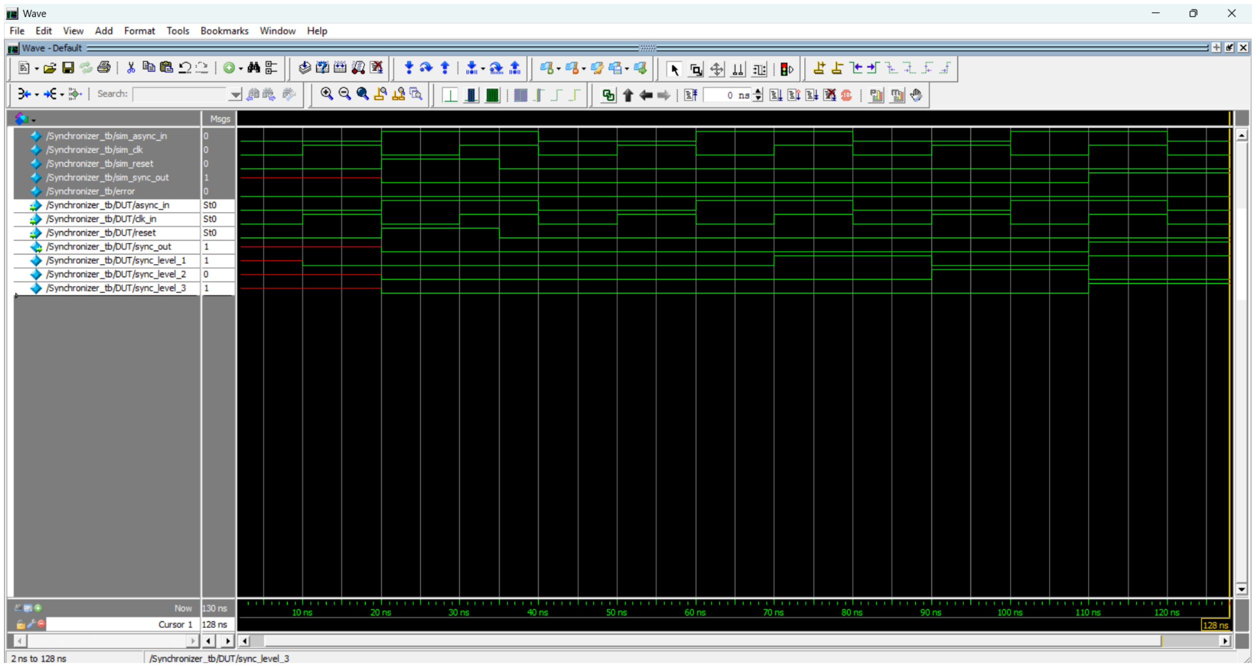

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/Keyboard_Interface_tb/sim_clk 1
/Keyboard_Interface_tb/sim_reset 0
/Keyboard_Interface_tb/sim_kbd_data_ready 0
/Keyboard_Interface_tb/sim_kbd_received_ascii_code 0101...
/Keyboard_Interface_tb/sim_restart_start_protocol 0
/Keyboard_Interface_tb/sim_play_enable 0
/Keyboard_Interface_tb/sim_direction 0
/Keyboard_Interface_tb/sim_restart 1
/Keyboard_Interface_tb/error 0
/Keyboard_Interface_tb/DUT/kbd_data_ready St1
/Keyboard_Interface_tb/DUT/kbd_received_ascii_code 0101...
/Keyboard_Interface_tb/DUT/clk_in St1
/Keyboard_Interface_tb/DUT/reset St0
/Keyboard_Interface_tb/DUT/restart_start_protocol St0
/Keyboard_Interface_tb/DUT/play_enable 0
/Keyboard_Interface_tb/DUT/direction 0
/Keyboard_Interface_tb/DUT/restart 1
00000000
01000101
01000100
01100101
01100100
01000010
01000110
01100010
01100110
01010010
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
Now 255 ns
Cursor 1 0 ns
0 ns
0 ns to 256 ns
/Keyboard_Interface_tb/DUT/direction

```
//Stop and start music
`define STOP_MUSIC 1'b0
`define PLAY_MUSIC 1'b1

//Playback mode
`define FORWARD 1'b0
`define BACKWARD 1'b1
```

- My goal for this testbench was to test if the controls responded correctly to the keyboard inputs regardless if the character sent in by the keyboard was lower or uppercase
- I first reset the module which resulted in a sim_play_enable of `STOP_MUSIC, sim_direction of `FORWARD, and a sim_restart of 0
- I then created a signal for capital E which resulted in sim_play_enable of `START_MUSIC and not affecting the other signals as indicated by the waveform
- I then created a signal for capital D which resulted in sim_play_enable of `STOP_MUSIC which did not affect the other signals
- I repeated the above step for lowercase E and lowercase D and as indicated by the waveform this behaved as expected
- Then I repeated the above tests but instead of using E or D, I tested the music direction using the capital B which set the sim_direction as `BACKWARD while not affecting the other signals

- I then used the capital F which set the sim_direction as `FORWARD while not affecting the other signals
- I repeated the tests with the lowercase versions of these characters as well and it behaved as expected
- Finally I tested the restart signal which was for the bonus of the “R” key which set the sim_restart signal to 1 while not affecting the other keys

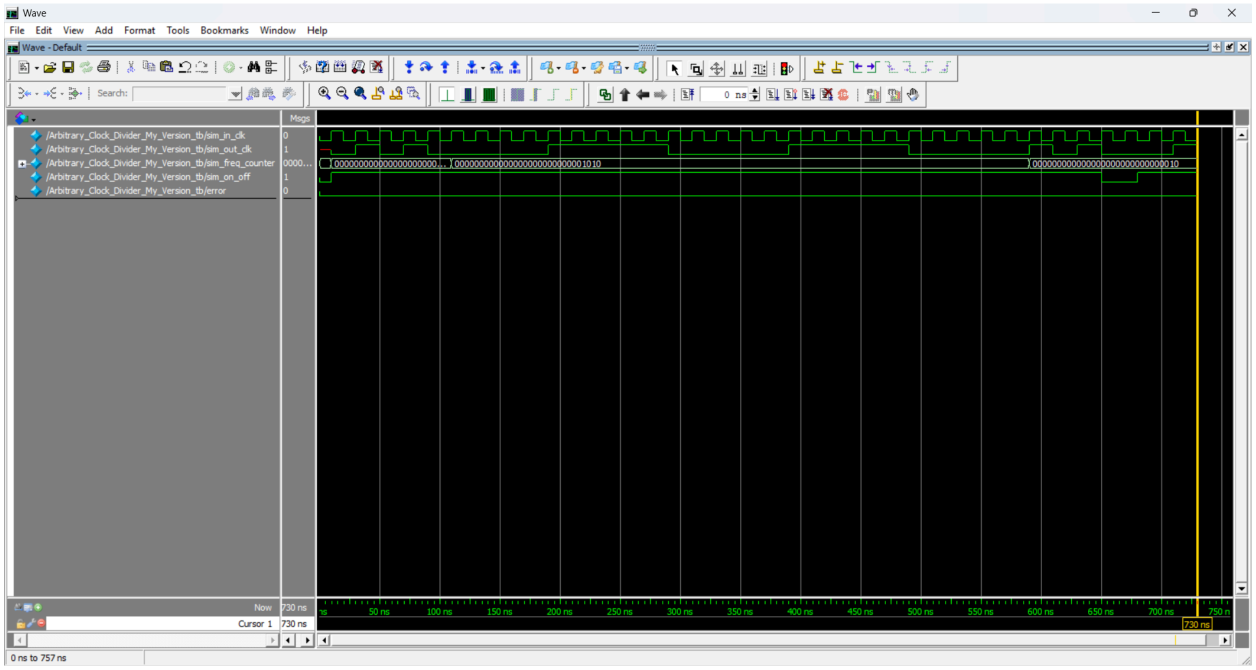
Below is a picture of the Synchronizer testbench simulation:



```
/Synchronizer_tb/sim_async_in      0
/Synchronizer_tb/sim_clk           0
/Synchronizer_tb/sim_reset         0
/Synchronizer_tb/sim_sync_out      1
/Synchronizer_tb/error             0
/Synchronizer_tb/DUT/async_in      St0
/Synchronizer_tb/DUT/clk_in        St0
/Synchronizer_tb/DUT/reset         St0
/Synchronizer_tb/DUT/sync_out      1
/Synchronizer_tb/DUT/sync_level_1  1
/Synchronizer_tb/DUT/sync_level_2  0
/Synchronizer_tb/DUT/sync_level_3  1
Now       130 ns
Cursor 1  128 ns    128 ns
2 ns to 128 ns    /Synchronizer_tb/DUT/sync_level_3
```

- The goal of this testbench was to visually see how when given two different clock signals, what the synchronizer does to
- I first reset the synchronizer resulting in all signals in the synchronizer having a value of 0
- The synchronizer was composed of 3 levels, async_level_1, async_level_2, and aysnc_level_3
- On a rising edge of the sim_async_in which was at 25MHz, the synchronizer picks up the value on the rising edge of the sim_clk at 50MHz, this value is sent to the async_level_1 as indicated in the waveform
- On the next clock cycle, async_level_1 passes this value to async_level_2 while async_level_3 still stores its previous level and async_level_1 picks up a new value from sim_async_in
- Next, the value in async_level_2 is propagated into async_level_3, async_level_2 picks up the value from async_level_1, and async_level_1 picks up a new value from async_in
- Eventually from async_level_3, this value is propagated into async_out which is the output of the synchronizer as seen in the waveform above
- Overall, this system creates a propagation of a signal based on the sim_clk ensuring that the final signal async_out is high when sim_clk is high thereby achieving a synchronized signal

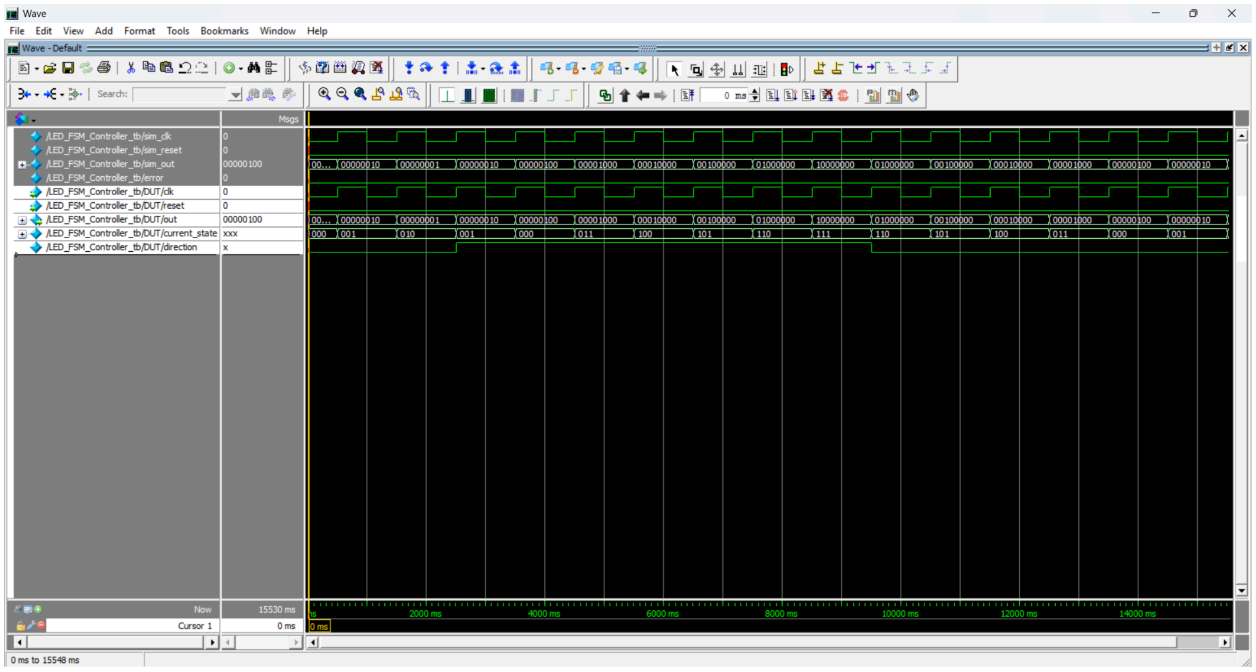
Below is a picture of the Arbitrary_Clock_Divider_My_Version (frequency divider) testbench simulation:



```
/Arbitrary_Clock_Divider_My_Version_tb/sim_in_clk        0
/Arbitrary_Clock_Divider_My_Version_tb/sim_out_clk       1
/Arbitrary_Clock_Divider_My_Version_tb/sim_freq_counter  0000...
/Arbitrary_Clock_Divider_My_Version_tb/sim_on_off        1
/Arbitrary_Clock_Divider_My_Version_tb/error             0
```

- The sim_in_clk signal is an oscillating clock with a frequency of 50MHz, this acts as the main input clock driver for the circuit
- I first set the sim_on_off to be on to generate an output clock signal, my goal was to first test if the clock behaved correctly when the clock is on
- I first set the sim_out_clk to have a frequency value of 25MHz which is equivalent to the counter value of 30’d2. This leads to a period double that of the 50MHz input clock. From the simulation above we can see that this leads to a clock (sim_out_clk) that updates half as fast as the sim_in_clk signal
- I then set the sim_out_clk to have a frequency value of 5MHz which as indicated in the simulation above corresponds to the value of 30’b1010. This leads to a period 5 times as long as the input clock, this can be seen on the sim_out_clk signal above
- Finally, I tested the sim_on_off signal by turning the signal off (1’b0) for a clock period and then turning the signal back on (1’b1) for a clock period as seen in the last part of the simulation. As one can see, when I set the output clk with a frequency of 25MHz and the sim_on_off signal off, the output clock no longer generated until I turned the sim_on_off signal back on which is the expected behavior

Below is a picture of the LED_FSM_Controller testbench simulation:

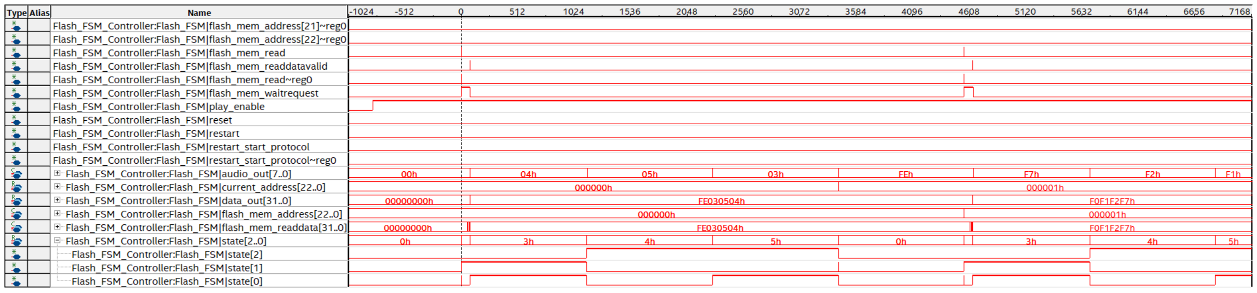


```
/LED_FSM_Controller_tb/sim_clk            0
/LED_FSM_Controller_tb/sim_reset          0
/LED_FSM_Controller_tb/sim_out            00000100   00... 00000010 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000 01000000 00100000 00010000 00001000 00000100 00000010
/LED_FSM_Controller_tb/error              0
/LED_FSM_Controller_tb/DUT/clk            0
/LED_FSM_Controller_tb/DUT/reset          0
/LED_FSM_Controller_tb/DUT/out            00000100   00... 00000010 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000 01000000 00100000 00010000 00001000 00000100 00000010
/LED_FSM_Controller_tb/DUT/current_state  xxx        000 001 010 001 000 011 100 101 110 111 110 101 100 011 000 001
/LED_FSM_Controller_tb/DUT/direction      x

Now      15530 ms
Cursor 1     0 ms
0 ms to 15548 ms
```

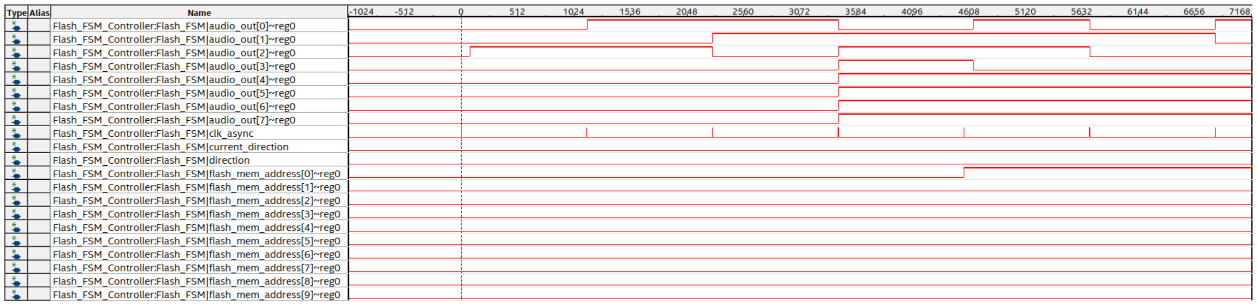
- My goal when testing this circuit was to step through the different states of the Finite State Machine (FSM) and check the current state, output, and direction of the circuit
- The output is in one hot code for the LEDs where the position of the 1 corresponds to the LED currently lit up
- As we can see the FSM starts at the state 3'b000, output 8'b00000100, and direction 1'b0. This corresponds to LED2 on and direction moving to the right which is the expected behavior of the given solution SOF file
- From there we can see the states move from LED2 to LED1 to LED0 as indicated by the output and direction
- Once we reach LED0, the next state is LED1. In this state transition, we can also see the direction change from moving right to left as indicated by the 1'b1 for the direction signal
- This continues until we reach LED7 where in this state transition, the direction then changes from 1'b1 to 1'b0 again indicating the direction is moving to the right now instead of the left
- The output of the circuit is very intuitive especially with the one hot code representation and we can clearly see how the LEDs light up based on the output

SignalTap Annotations:

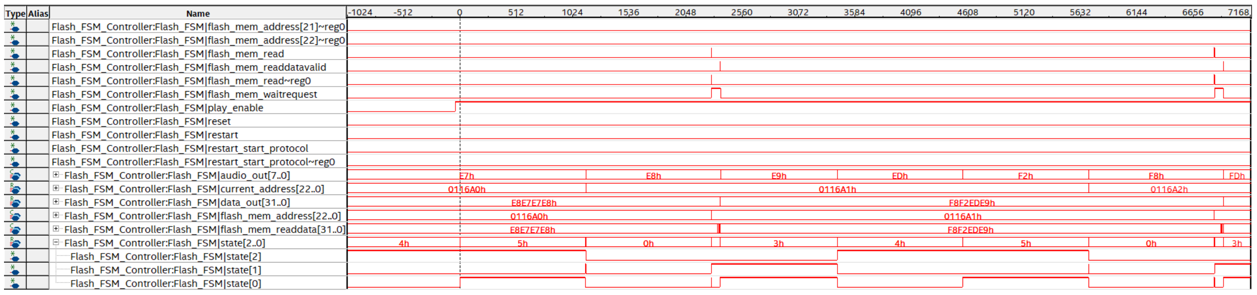
SignalTap Picture 1:


Type
Alias
Name
Flash_FSM_Controller:Flash_FSM|flash_mem_address[21]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[22]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_read
Flash_FSM_Controller:Flash_FSM|flash_mem_readdatavalid
Flash_FSM_Controller:Flash_FSM|flash_mem_read~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_waitrequest
Flash_FSM_Controller:Flash_FSM|play_enable
Flash_FSM_Controller:Flash_FSM|reset
Flash_FSM_Controller:Flash_FSM|restart
Flash_FSM_Controller:Flash_FSM|restart_start_protocol
Flash_FSM_Controller:Flash_FSM|restart_start_protocol~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[7..0]
Flash_FSM_Controller:Flash_FSM|current_address[22..0]
Flash_FSM_Controller:Flash_FSM|data_out[31..0]
Flash_FSM_Controller:Flash_FSM|flash_mem_address[22..0]
Flash_FSM_Controller:Flash_FSM|flash_mem_readdata[31..0]
Flash_FSM_Controller:Flash_FSM|state[2..0]
Flash_FSM_Controller:Flash_FSM|state[2]
Flash_FSM_Controller:Flash_FSM|state[1]
Flash_FSM_Controller:Flash_FSM|state[0]
-1024 -512 0 512 1024 1536 2048 2560 3072 3584 4096 4608 5120 5632 6144 6656 7168
00h 04h 05h 03h FEh F7h F2h F1h
000000h 000001h
00000000h FE030504h F0F1F2F7h
000000h 000001h
00000000h FE030504h F0F1F2F7h
0h 3h 4h 5h 0h 3h 4h 5h

SignalTap Picture 2:


Type
Alias
Name
Flash_FSM_Controller:Flash_FSM|audio_out[0]~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[1]~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[2]~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[3]~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[4]~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[5]~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[6]~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[7]~reg0
Flash_FSM_Controller:Flash_FSM|clk_async
Flash_FSM_Controller:Flash_FSM|current_direction
Flash_FSM_Controller:Flash_FSM|direction
Flash_FSM_Controller:Flash_FSM|flash_mem_address[0]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[1]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[2]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[3]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[4]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[5]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[6]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[7]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[8]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[9]~reg0
-1024 -512 0 512 1024 1536 2048 2560 3072 3584 4096 4608 5120 5632 6144 6656 7168

SignalTap Picture 3:


Type
Alias
Name
Flash_FSM_Controller:Flash_FSM|flash_mem_address[21]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_address[22]~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_read
Flash_FSM_Controller:Flash_FSM|flash_mem_readdatavalid
Flash_FSM_Controller:Flash_FSM|flash_mem_read~reg0
Flash_FSM_Controller:Flash_FSM|flash_mem_waitrequest
Flash_FSM_Controller:Flash_FSM|play_enable
Flash_FSM_Controller:Flash_FSM|reset
Flash_FSM_Controller:Flash_FSM|restart
Flash_FSM_Controller:Flash_FSM|restart_start_protocol
Flash_FSM_Controller:Flash_FSM|restart_start_protocol~reg0
Flash_FSM_Controller:Flash_FSM|audio_out[7..0]
Flash_FSM_Controller:Flash_FSM|current_address[22..0]
Flash_FSM_Controller:Flash_FSM|data_out[31..0]
Flash_FSM_Controller:Flash_FSM|flash_mem_address[22..0]
Flash_FSM_Controller:Flash_FSM|flash_mem_readdata[31..0]
Flash_FSM_Controller:Flash_FSM|state[2..0]
Flash_FSM_Controller:Flash_FSM|state[2]
Flash_FSM_Controller:Flash_FSM|state[1]
Flash_FSM_Controller:Flash_FSM|state[0]
-1024 -512 0 512 1024 1536 2048 2560 3072 3584 4096 4608 5120 5632 6144 6656 7168
E7h E8h E9h EDh F2h F8h FDh
0116A0h 0116A1h 0116A2h
E8E7E7E8h F8F2EDE9h
0116A0h 0116A1h
E8E7E7E8h F8F2EDE9h
4h 5h 0h 3h 4h 5h 0h 3h

- I have included some pictures of the SignalTap waveforms for my flash FSM that displays the behavior of the signals while the song is playing from the start
- The first picture corresponds to the second picture as they were taken at the same time, while the third picture was taken at a later time in the song
- If more picture are needed, please let me know and I can demonstrate them in-person during the lab
- As one can see, the state transitions behave properly and they proceed sequentially as defined in the code in my modules (I zoomed out so some state transitions that were really quick were hard to see on the pictures)
- Some highlights include:
    - The flash_mem_address signal as you can see updates correctly during the state transition from state 3’b000 to 3’b001

- The audio_out signal changes with each state transition for the states `OUTPUT1 to `OUTPUT4 (state 3-6) although state 6 is not visible from the picture because the transition is very quick
- Play_enable signal is high indicating the audio is playing as expected
- Current_direction is low indicating the audio is playing forwards