CPEN 311 LAB 2 – Simple iPod

1. Directory / Path of .sof File:

harshil_rajesh_patel_35437326_Lab_2/rtl/template_de1soc/simple_ipod_solution.sof

2. Status of the Lab:

- Completed and tested 3 FSMs (keyboard interface, address calculator, flash control).
- The song can be played on the DE1-SOC from flash memory.
- It is possible to pause, play, play forward, play backward using the keyboard.
- It is also possible to increase and decrease the speed of the song using the DE1 keys.

3. Finite State Machine (FSM) – Diagrams:

a. *Keyboard Audio Control FSM:*

The state diagram shown below is for the **keyboard_audio_controller** module:

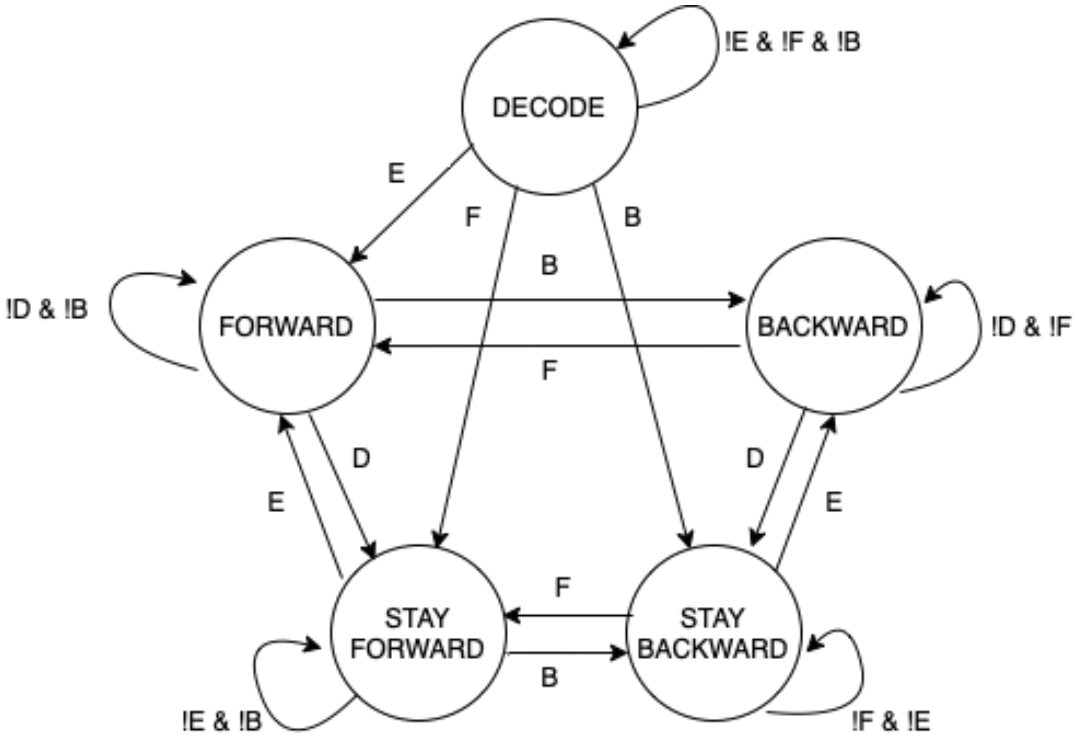

Figure 1: Keyboard Audio Controller State Diagram

In addition to the input signals shown, the outputs (that are sent to the **address_calc** module) for the states are as follows:

DECODE: **read_ready** = 0, **forward** = 0
FORWARD: **read_ready** = 1, **forward** = 1 [Sent to **address_calc**]
STAY FORWARD: **read_ready** = 0, **forward** = 0
BACKWARD: **read_ready** = 1, **forward** = 0 [Sent to **address_calc**]
STAY BACKWARD: **read_ready** = 0, **forward** = 0

b. Address Calculator FSM:

The state diagram shown below is for the **address_calc** module:

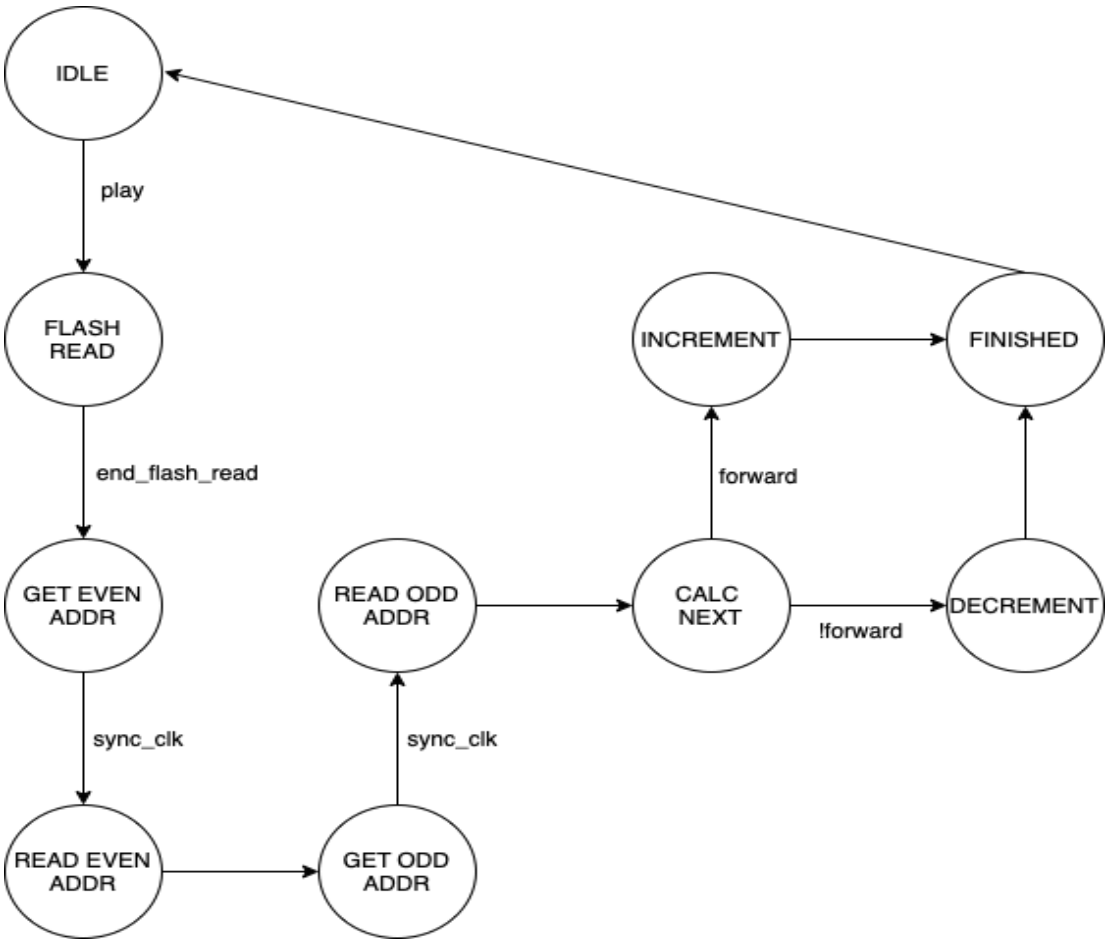

Figure 2: Address Calculator State Diagram

In addition to the input signals (including **play, forward** from **keyboard_audio_controller**) shown, the outputs for the states are as follows:

IDLE: **start_flash** = 0, **read** = 0, **finish** = 0
FLASH READ: **start_flash** = 1, **read** = 1, **finish** = 0 [Sent to **flash_control**]
GET EVEN ADDR: **start_flash** = 0, **read** = 0, **finish** = 0
READ EVEN ADDR: **start_flash** = 0, **read** = 0, **finish** = 0
GET ODD ADDR: **start_flash** = 0, **read** = 0, **finish** = 0
READ ODD ADDR: **start_flash** = 0, **read** = 0, **finish** = 0
CALC NEXT: **start_flash** = 0, **read** = 0, **finish** = 0
INCREMENT: **start_flash** = 0, **read** = 0, **finish** = 0
DECREMENT: **start_flash** = 0, **read** = 0, **finish** = 0
FINISHED: **start_flash** = 0, **read** = 0, **finish** = 1

c. Flash Control FSM:

The state diagram shown below is for the **flash_control** module:

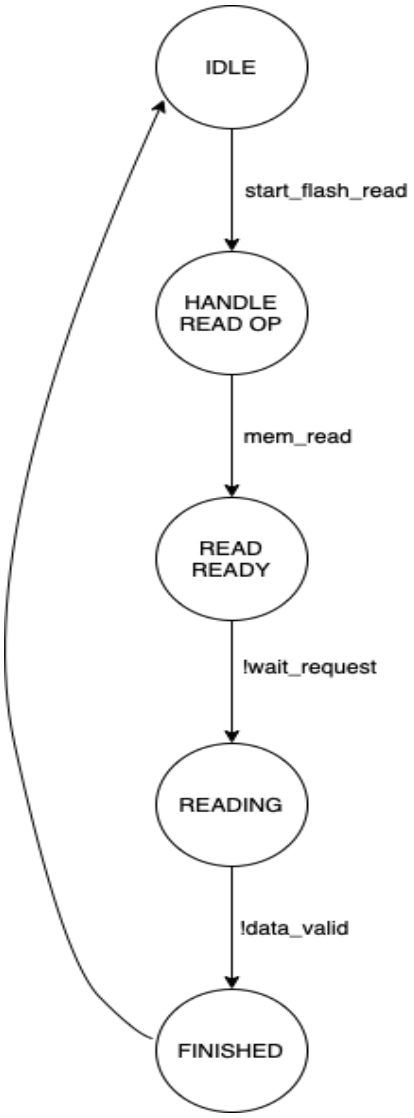

Figure 3: Flash Control State Diagram

In addition to the input signals (including **start_flash_read** from **address_calc**) shown, the outputs for the states are as follows:

IDLE: **end_flash_read** = 0
HANDLE READ OP: **end_flash_read** = 0
READ READY: **end_flash_read** = 0
READING: **end_flash_read** = 0
FINISHED: **end_flash_read** = 1 [Sent back to **address_ctrl**]

4. Simulation Screenshots and Explanation:

Keyboard Audio Control FSM:

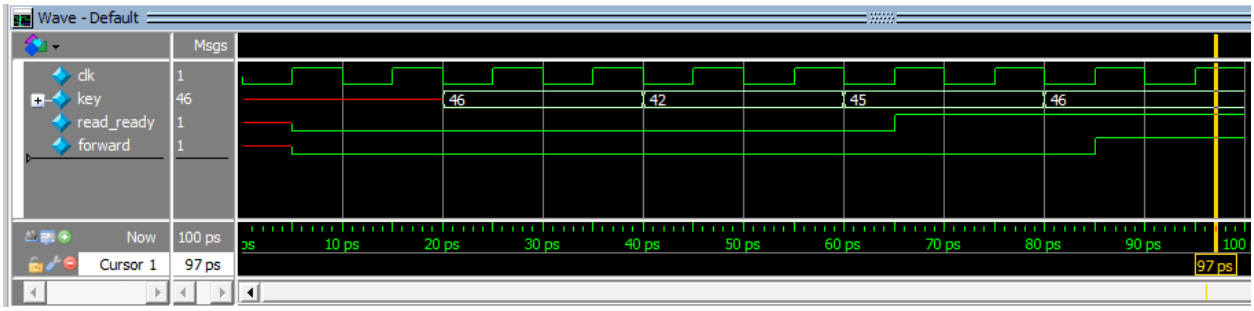

Figure 4: Keyboard Audio Control Simulation

- In the simulation above, I input different key values (corresponding to the ascii values).
- The **read_ready** output is 1'b1 when we are in the **forward_state** or **backward_state**.
- The **forward** output is 1'b1 when we are in the **forward_state**.
- We start with the key value 46h (key F) which causes us to go into the **stay_forward** state from the **decode** state. The outputs are both 1'b0 at this point.
- Next, we enter the key value 42h (key B) which causes us to go into the **stay_backward** state. The outputs are both still 1'b0 at this point.
- Then, we enter the key value 45'h (key E) at which point the song starts playing and so **read_ready** is 1'b1. The other output **forward** is still 1'b0 because we are playing in the backward direction.
- Now finally, when we enter key value 46h (key F) again, we are playing in the forward direction and thus, **forward** and **read_ready** are both 1'b1 (to be sent to **address_calc**).
- Hence, the **keyboard_audio_controller** module is working as intended.

Flash Control FSM:

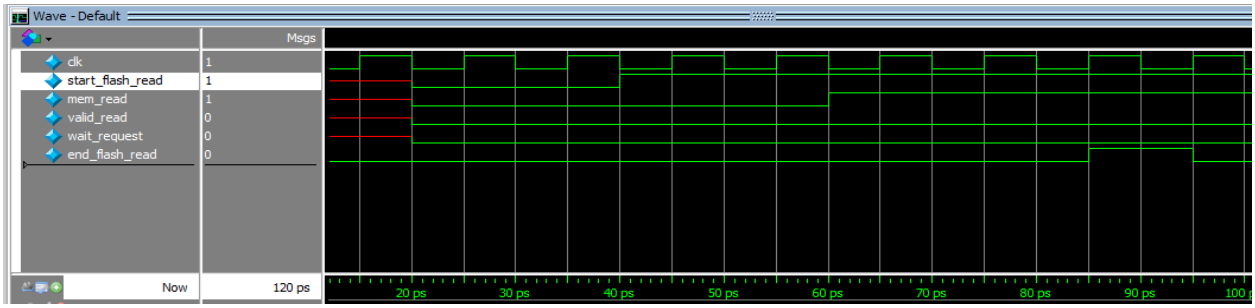

Figure 5: Flash Control Simulation

- In the simulation above, we can observe the following changes in state:
- Initially, all inputs are 1'b0 and so we are in the idle state. However, at 40ps **start_flash_read** is asserted and so we move into the **handle_read_op** state.
- Then, at 60ps, **mem_read** is asserted and we move into the **read_ready** state.
- After this, we move into the reading state as **!wait_request** is asserted after which we move into finished state at 85ps since **!valid_read** is true.
- At 85ps, we are done reading and are in the **finished** state due to which **end_flash_read** is 1'b1.
- This signal **end_flash_read** is sent to **address_calc** to indicate that a new address is needed.

Address Calculator FSM:

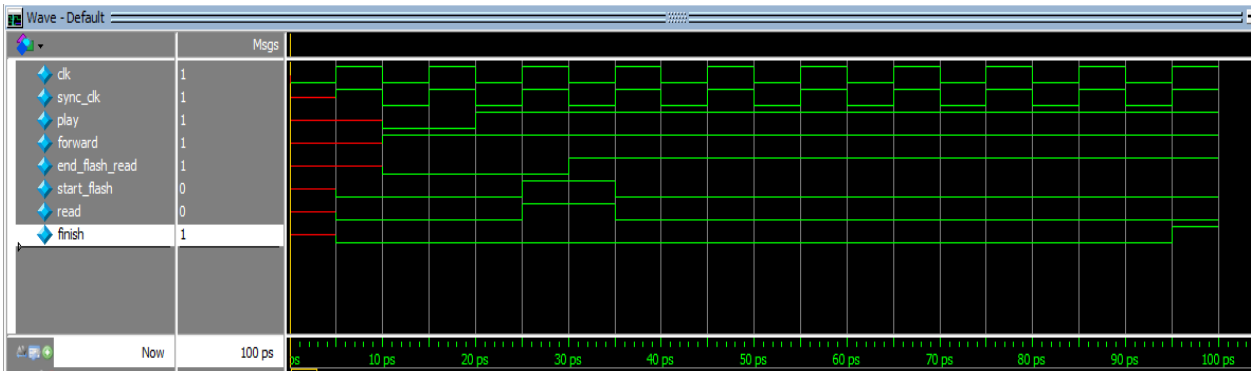

Figure 6: Address Calculator Simulation

- The simulation above is for the address_calc module.
- There are many signals involved in the working of this module so it will be easier to also refer to the comments in the .sv files for a theoretical explanation.
- Furthermore, the input signals shown are **clk, sync_clk, play, forward, end_flash_read** and the outputs shown are **start_flash, read**, and **finish**.
- There are additional signals like **audio_data**, **mem_addr**, and **audio_out** that I did not include in the waveform (because they contain many bits and are just long values and make the waveform harder to read). In addition, **byteenable** is also not added to the waveform because it is always set to 4'b1111. For further explanation on how these signals interact with other modules, refer to the comments at the top of **address_calc.sv**.
- Now coming to the simulation, we observe the following changes in state:
- Initially we are in the idle state, until play is set to 1'b1 at 20ps which causes us to transition into the **flash_read** state. At this point, we signal the **flash_control** module through **start_flash** (1'b1) and **read** (1'b1) is carrying out steps to read the flash at the current address. After it is done it sends back the signal **end_flash_read** which we simulate to happen at 30ps.
- This causes us to transition into the **get_even_addr** state where we start processing the next address to pass into the **flash_control** module.
- At the posedge of sync_clk, we read the even address in the **read_even_addr** state.
- We then repeat the previous two steps but in the **get_odd_addr** state and then the **read_odd_addr** state.
- Once we are done, we calculate the next address in the **calc_next** state and since we have asserted the **forward** signal to be true (1'b1), we move into the **increment** state to increment the current address to calculate the next address.
- Once done, we transition to the **finished** state and the **finish** signal becomes 1'b1 and is sent to the **flash_control** module to indicate that the data at the new address is ready to be read.
- Hence, through this simulation above, we can confirm that the module works correctly and as intended.

5. Signal Tap Screenshots:

Keyboard Audio Controller SignalTap:

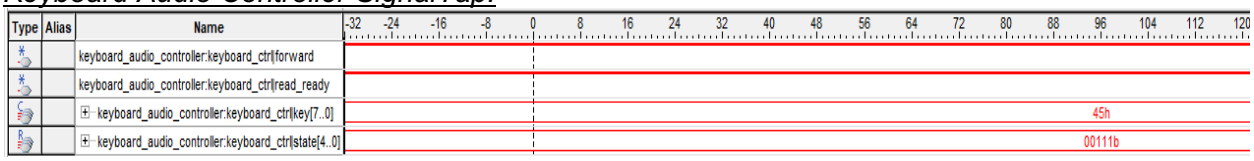

Figure 7: Keyboard Audio Controller (Play)

- In the SignalTap screenshot above, I entered the key E (45h) which corresponds to the play button.
- As a result, we are in the state 5'b00111 which is the **forward_state** (since we are playing in the forward direction). The signal **read_ready** is hence 1'b1.

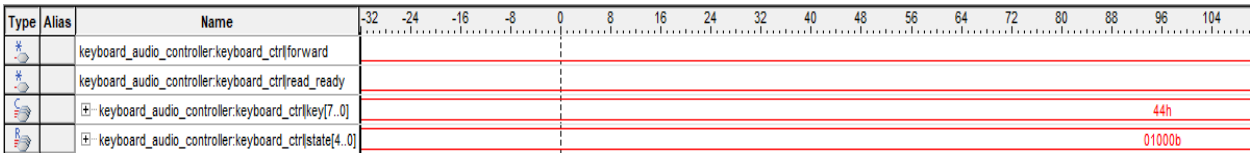

Figure 8: Keyboard Audio Controller (Pause)

- In the SignalTap screenshot above, I entered the key D (44h) next which corresponds to the pause button.
- As a result, we are in the state 5'b01000 which is the **stay_forward** state (since we are waiting in the forward direction). The signal **read_ready** is hence 1'b0.

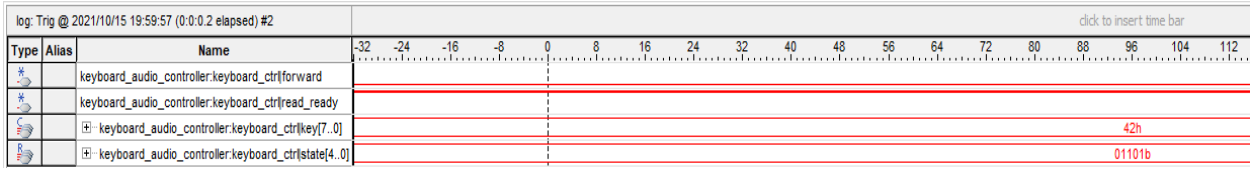

Figure 9: Keyboard Audio Controller (Backward)

- In the SignalTap screenshot above, I entered the key B (42h) next which corresponds to the play backwards button.
- As a result, we are in the state 5'b01101 which is the **backward_state** (since we are now playing in the backward direction). The signal **read_ready** is hence 1'b1.

Flash Control and Address Calculator SignalTap Screenshots on following pages ->

Flash Control SignalTap:

The SignalTap analysis for the **flash_control** module involves being in the **idle** state for most of the time and then in the **finished** state for a very short period where the signals change. This made it difficult to get proper screenshots from a real time analysis, but I managed to get screenshots of all major states.

Note: Ignore the interval -32 to -31 which shows the signal values of an older runtime (look to the right of that section). The section shows up because I had to set a time bar to capture the state change that occurs for a very short period.

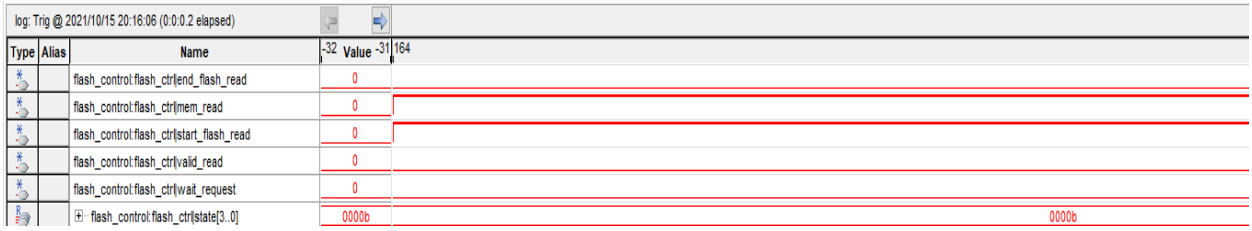

Figure 10: Flash Control (Idle)

- In the SignalTap screenshot above, we are in the **idle** state and all the signals are 1'b0.

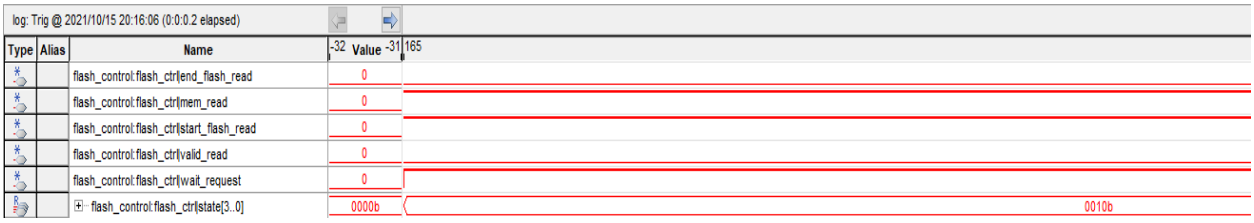

Figure 11: Flash Control (Handle Read Op)

- In the SignalTap screenshot above, we are in the **handle_read_op** state and all the signals are also 1'b0.

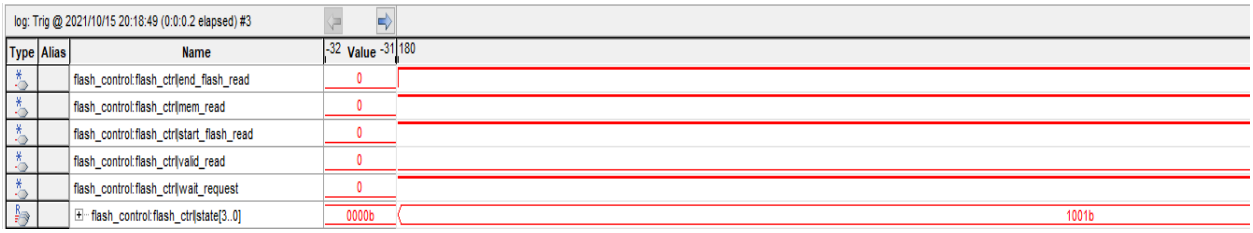

Figure 12: Flash Control (Finished)

- In the SignalTap screenshot above, we are in the **finished** state. The signal **end_flash_read** which needs to be sent to the **address_calc** module is 1'b1 as seen by the bolded red line.

Address Calculator SignalTap:

Just like for the **flash_control** module, the SignalTap analysis for the **address_calc** module involves being in the **idle** state for most of the time and then in the **finished** state for a very short period where the signals change. This made it difficult to get proper screenshots from a real time analysis, but I managed to get screenshots of all major states.

Note: Ignore the interval -32 to -31 which shows the signal values of an older runtime (look to the right of that section). The section shows up because I had to set a time bar to capture the state change that occurs for a very short period.

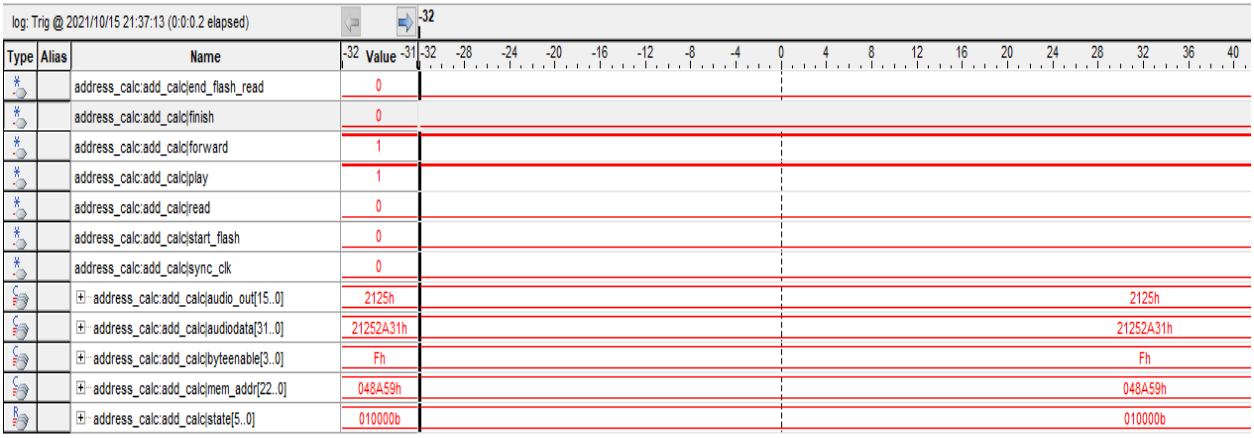

Figure 13: Address Calculator (Get Odd Addr)

- In the screenshot above, the FSM is in the **read_odd_addr** state (6'b010000). It is signalling the flash to read the data held at the lower 16 bits of the address of the flash memory.
- In addition, **forward** and **play** (from the **keyboard_audio_controller**) are also 1'b1 as seen.

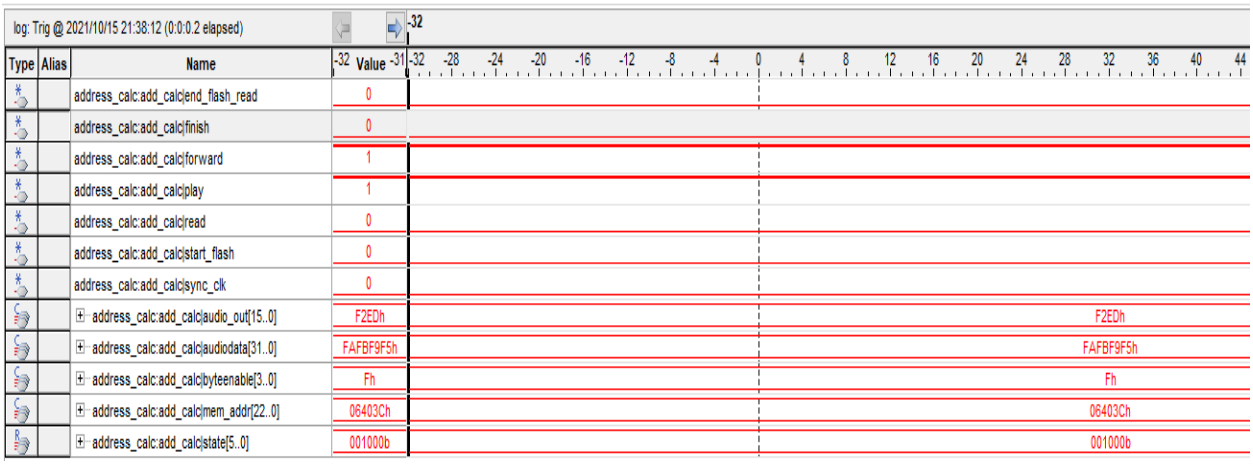

Figure 14: Address Calculator (Get Even Addr)

- In the screenshot above, the FSM is in the **read_even_addr** state (6'b001000). It is signalling the flash to read the data held at the upper 16 bits of the address of the flash memory.
- In addition, **forward** and **play** (from the **keyboard_audio_controller**) are also 1'b1 as seen.

6. How I ran the simulations:

- The simulations were run making a ModelSim project lab_2_sim.mpf and uploading the testbench (_tb.sv) files.
- Upon adding the files to the project, they were compiled and then uploaded using 'vsim module', and then the commands 'restart -f', and 'run -all'.
- Each of the three FSMs were tested and their waveforms analyzed.
- The annotations/explanations of the waveforms can be found in the earlier sections of this document, while the .do files are also available to view in the ./sim folder.
- The waveforms were used to debug the modules.

7. Additional Info:

- References from the lecture slides are mentioned in comments where possible.
- Any calculations are also mentioned in comments.
- There are comments explaining how the different FSMs interact with each other through certain input/output signals.
- All .sv and _tb.sv files are found in /rtl/template_de1soc
- All screenshots are found in /doc
- All .do files are in /sim
- The modelsim project .mpf file is found in /sim