The Simon State Machine

(Part 2)

Submitted to:

Dr. Gregg Chapman

Srinivastin Subramaniyan

Created by:

Group 18

Ansh Pachauri

Rushi Bhatt

Siwei Fan

ECE 2060

The Ohio State University

Columbus, OH

5 April 2024

Executive Summary

Lab 8 centered on building upon the functionalities implemented in Lab 8. The primary objective was to integrate a user interface (UI) into the existing Simon Game design. This UI would allow users to interact with the game through button presses.

To commence the lab, the group assembled the required materials: computer, USB cable, DE2 board, keypad, and Simon Game box. Following the launch of Quartus Prime, the team incorporated a keypad component into the schematic alongside the existing System Controller and Simon Game hardware components.

After integrating the keypad, the group designed the VHDL code to recognize user inputs from the keypad. This involved assigning functionalities to each button press within the various states of the Simon Game. Upon finalizing the code, the group compiled the project and downloaded it onto the DE2 board.

Successful compilation and download allowed for user interaction with the Simon Game box through the keypad. Users were able to initiate the game, input color sequences, and observe corresponding outputs on the game box. This accomplishment verified the successful integration of the user interface.

The following sections of this report will delve deeper into the background and objectives of the lab, the methodologies and processes employed, and the resulting schematics. The gorup will then explore the lab results and their significance, followed by a discussion section that explores key takeaways and learning points. The report concludes with a summary, acknowledgements, and a list of references.

Introduction

This lab's focus is on enhancing the user experience of the Simon Game by incorporating a user interface (UI). This UI would allow players to interact with the game through button presses. The lab began with assembling the necessary equipment: computer, USB cable, DE2 board, keypad, and Simon Game box. After launching Quartus Prime, the team integrated a keypad component into the existing schematic alongside the System Controller and Simon Game hardware.

Following meticulous integration, the group designed VHDL code to recognize user inputs from the keypad. This involved assigning functionalities to each button press within the various states of the Simon Game (wait state, autodisplay state, game state, win/lose state). Upon finalizing the code, the group compiled the project and downloaded it onto the DE2 board.

Successful compilation and download allowed for user interaction with the Simon Game box through the keypad. Players were able to initiate the game, input color sequences, and observe corresponding outputs on the game box. This confirmed the successful implementation of the user interface.

The following sections of this report will delve deeper into the background and objectives of the lab, the methodologies and processes employed, and the resulting schematics. The group will then explore the lab results and their significance, followed by a discussion section that explores key takeaways and learning points. The report concludes with a summary, acknowledgements, and a list of references.

Experimental Methodology

In this endeavor, Group 18 embarked on the task of overhauling the Test_Controller and modifying the code within the System Controller VHDL file to enable the functioning of the Simon game, amalgamating all preceding components from the semester, inclusive of an Audio block, RandomCounter, ButtonDebounce, and more. The System Controller, serving as the state machine orchestrating the Simon game, regulates all output signals to all other hardware based on the status of input signals. The requisites for this undertaking encompassed a DE2 Board, a USB Blaster Cable, Simon Game Box, and a speaker. The team referenced a video provided during the lab session, titled "Simon State Machine," and code outlines delineating five states: game, waits, autoplay, lose, and win states.

Significantly, attention was directed towards the CODEC, which served as the cornerstone of numerous labs conducted throughout the semester. A thorough comprehension of the CODEC, expounded upon in the report's introduction, facilitated the formulation of a functional schematic, underpinned by the conceptualization of digital logic. The CODEC encapsulates both the random number generator and the audio bus. The random number generator operates in conjunction with a 16-bit counter alongside the random sequence and multiplexor to yield the precise outcomes elucidated within this segment. It is noteworthy that the descriptions of debounce and audio synthesizer are integrated into the introduction as well.

Commencing the endeavor, Group 18 procured the System Controller VHDL file as provided. Subsequently, Group 18 devised a block and replaced the TEST_Controller. Thereafter, the group commenced the task of crafting the code for the system controller to synchronize all components seamlessly. The resultant schematic derived from substituting the TEST_Controller with the System_Controller was attained.

Following this, the team initiated the modification of the given code format in the System_Controller VHDL file. The objective of this endeavor was to impart specific behaviors to the System_Controller to facilitate the functionality of the Simon Game. The initial focus was

on the Wait State. To accomplish this, the team delineated behaviors under each if and elseif statement. To activate an LED, the team set the LED variables to ‘0’ (e.g., blue_LED <= ‘0’;). Conversely, to deactivate the LED, the team set the LED variables to ‘1’ (e.g., red_LED <= ‘1’;), owing to the LEDs being low true. Additionally, the team reset the ticks after the execution of the if-statement structure by setting the ticks variable to 0 (e.g., ticks <= 0;).

Following the Wait State, the team proceeded to implement the Autodisplay State using the following guideline. To execute the autodisplay state, Group 18 assigned the value of audio_value to audio_blue since this represents the blue sequence (audio_value <= audio_blue;). Subsequently, the team activated the blue LED (blue_LED <= ‘0’;), followed by enabling the clock sequence (clk_en_seq <= ‘1’;), incrementing the auto_counter (auto_counter = auto_counter + 1;), and resetting the ticks (ticks <= 0;). This procedure was replicated for the other three colors to execute the VHDL code for this state effectively.

The third state implemented was the game state. For the game state, the group formulated if statements for two scenarios: one scenario for when a button is pressed and another for comparing the pressed button with the generated sequence. Upon pressing each button, the group incorporated the following actions: activation of the corresponding LED, loading of the corresponding audio value, preservation of the corresponding color value in ‘button,’ and resetting of the timer (ticks).

The aforementioned four requirements were fulfilled through the following lines of code: blue_LED <= ‘0’; // audio_value <= audio_BLUE; // button <= blue_button; // ticks <= 0;. This implementation was replicated for each button and its corresponding color. The subsequent aspect of the game state unfolded as follows. The three requisites for each if and elseif statement involved incrementing play_counter, enabling the clock, and resetting button to no_button. This was realized through the following code: play_counter <= play_counter + 1; // clk_en_seq <= ‘1’; // button <= no_button;.

Subsequent to this, the team programmed the win state of the game. To enact the win state, the team activated the LEDs and sound in ascending order of their audio frequency, alongside the corresponding LED color. The reference for the win state is elucidated below:
To illuminate the blue LED, the team set the blue LED to ‘0’ (blue_LED <= ‘0’;). Furthermore, each color was aligned with its corresponding audio frequency (audio_value <= audio_BLUE;). This sequence was replicated for the three other colors as well. At the culmination of the if-statement structure, the team reset the ticks to 0 (ticks <= 0;) and transitioned back to the Wait State (STATE <= WAITS;).
Group 18 also programmed the lose state of the game, which entailed playing the Wrong audio for one second. Following one second, the team reset ticks, reverted the value of button to ‘no_button,’ and transitioned back to the Wait state. The reference for this is articulated below:

To reset the ticks, Group 18 assigned the value to 0 (ticks <= 0;). Subsequently, the team set the value of button to no_button (button <= no_button;), and transitioned back to the Waits state (STATE <= WAITS;).

Upon completion of programming each state, Group 18 compiled the project and rectified all syntax errors. The functionality of the Simon Game was demonstrated through various outcomes. In the WAITS state, after programming the board, the LEDs sequenced in a circular pattern. The game commenced by displaying the first LED and audio tone of the sequence upon pressing and releasing KEY[0]. Matching the LED/audio pattern displayed with a corresponding button press was executed correctly. The AUTODISPLAY sequence was presented with one additional state. Upon matching the 5th state in the sequence, the WIN state audio tones were generated in order of increasing frequency to simultaneously display the associated LED. Intentionally losing a game demonstrated the WRONG tone when this event occurred.

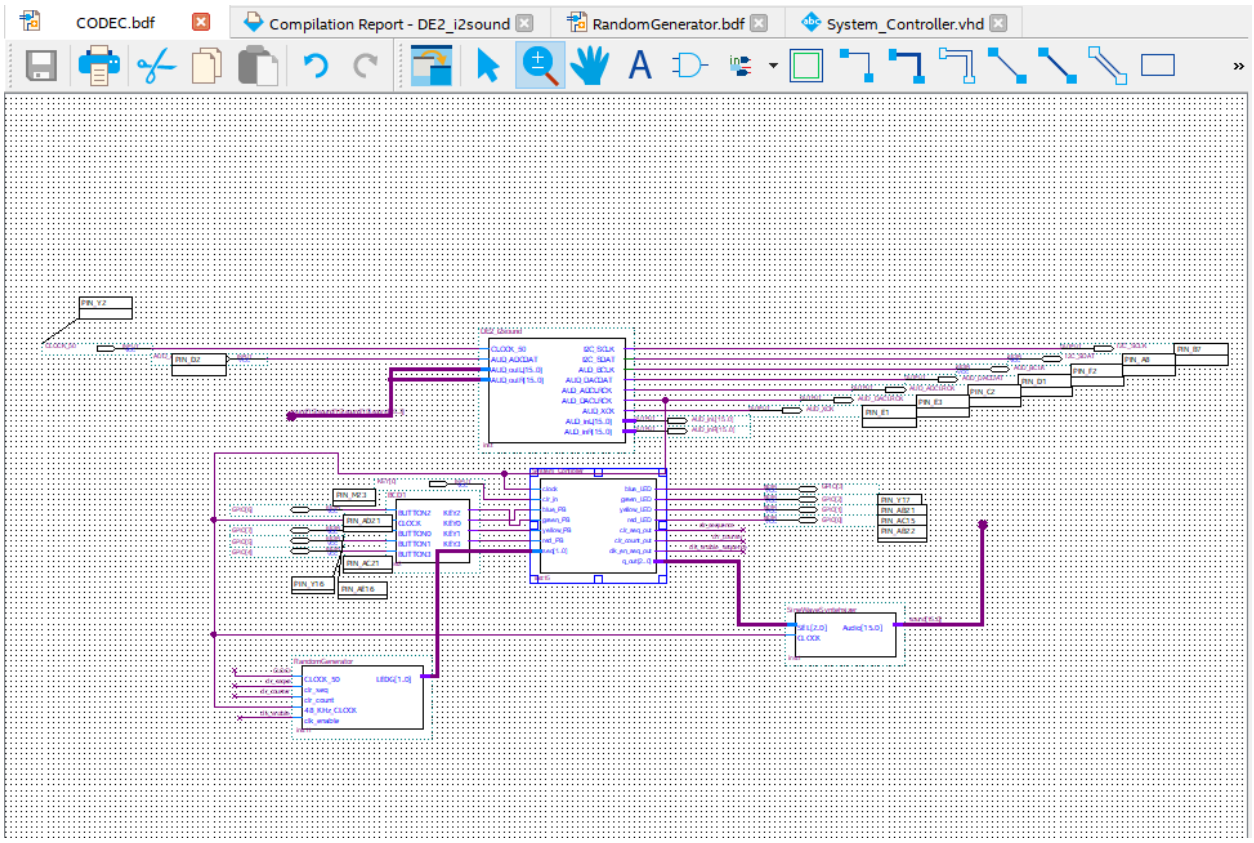


Figure 1 - Top-Level Schematic with System_Controller

Results

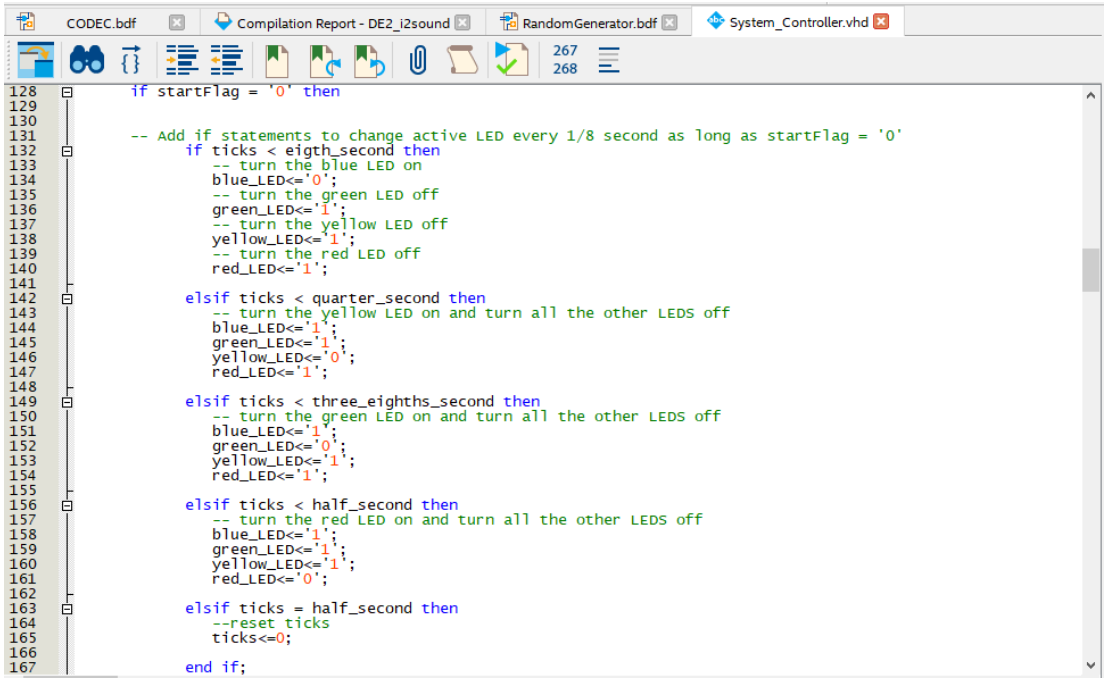
Lab 8 provided the team with a comprehensive understanding of each facet of the lab and how diverse topics covered could be seamlessly amalgamated to craft a successful experiment. The primary focus of Lab 8 was the development of a robust top-level schematic of the CODEC coupled with the system controller. The system controller underwent programming via VHDL code, implementing five fundamental states: the Wait State, Autodisplay State, Game State, Win State, and Lose State. The team crafted each of these states, followed by successful compilation.

Subsequently, the team proceeded to program the code, designating the CODEC as the top-level schematic. Upon completion, the Simon game underwent rigorous testing to validate its functionality. The initial assessment involved the WAITS state, where programming the board led to the anticipated circular LED sequence patterns.

Moving forward, the Game State was tested by pressing KEY[0] on the DE2 board, resulting in the successful display of the first LED and its corresponding audio tone for the sequence. Subsequent to this, the team scrutinized the operation of the Autodisplay State. By pressing various buttons on the Simon Game box, the team confirmed that each LED accurately displayed the designated pattern and audio, in accordance with the Autodisplay code.

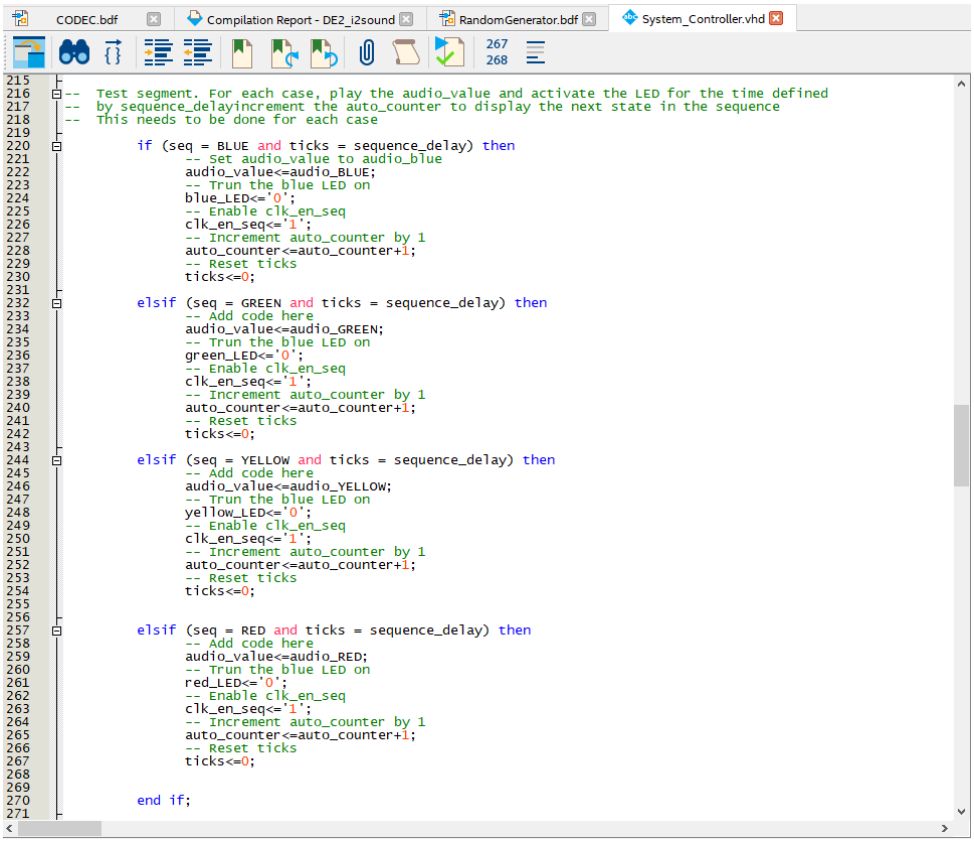
Following these tests, attention was directed towards validating the Win State. The team endeavored to match up to the fifth state within the sequence, thereby triggering the generation and playback of WIN tones through both audio and the Simon Box display for each LED. Lastly, the team conducted tests on the Lose State by intentionally pressing an incorrect sequence to evoke the playing of the WRONG tone.

These experiments substantiated the successful outcomes of each VHDL code, as depicted below. Additionally, the top-level schematic featuring the system controller implementation is presented at the experimental methodology of this report, and the corresponding VHDL code to each state of the game is displayed below. .




```
128     if startFlag = '0' then
129
130
131     -- Add if statements to change active LED every 1/8 second as long as startFlag = '0'
132         if ticks < eigth_second then
133            -- turn the blue LED on
134            blue_LED<='0';
135            -- turn the green LED off
136            green_LED<='1';
137            -- turn the yellow LED off
138            yellow_LED<='1';
139            -- turn the red LED off
140            red_LED<='1';
141
142         elsif ticks < quarter_second then
143            -- turn the yellow LED on and turn all the other LEDS off
144            blue_LED<='1';
145            green_LED<='1';
146            yellow_LED<='0';
147            red_LED<='1';
148
149         elsif ticks < three_eighths_second then
150            -- turn the green LED on and turn all the other LEDS off
151            blue_LED<='1';
152            green_LED<='0';
153            yellow_LED<='1';
154            red_LED<='1';
155
156         elsif ticks < half_second then
157            -- turn the red LED on and turn all the other LEDS off
158            blue_LED<='1';
159            green_LED<='1';
160            yellow_LED<='1';
161            red_LED<='0';
162
163         elsif ticks = half_second then
164            --reset ticks
165            ticks<=0;
166
167         end if;
```

Figure 2 - Wait State VHDL code




```
215
216 --  Test segment. For each case, play the audio_value and activate the LED for the time defined
217 --  by sequence_delayincrement the auto_counter to display the next state in the sequence
218 --  This needs to be done for each case
219
220         if (seq = BLUE and ticks = sequence_delay) then
221               -- Set audio_value to audio_blue
222               audio_value<=audio_BLUE;
223               -- Trun the blue LED on
224               blue_LED<='0';
225               -- Enable clk_en_seq
226               clk_en_seq<='1';
227               -- Increment auto_counter by 1
228               auto_counter<=auto_counter+1;
229               -- Reset ticks
230               ticks<=0;
231
232         elsif (seq = GREEN and ticks = sequence_delay) then
233               -- Add code here
234               audio_value<=audio_GREEN;
235               -- Trun the blue LED on
236               green_LED<='0';
237               -- Enable clk_en_seq
238               clk_en_seq<='1';
239               -- Increment auto_counter by 1
240               auto_counter<=auto_counter+1;
241               -- Reset ticks
242               ticks<=0;
243
244         elsif (seq = YELLOW and ticks = sequence_delay) then
245               -- Add code here
246               audio_value<=audio_YELLOW;
247               -- Trun the blue LED on
248               yellow_LED<='0';
249               -- Enable clk_en_seq
250               clk_en_seq<='1';
251               -- Increment auto_counter by 1
252               auto_counter<=auto_counter+1;
253               -- Reset ticks
254               ticks<=0;
255
256
257         elsif (seq = RED and ticks = sequence_delay) then
258               -- Add code here
259               audio_value<=audio_RED;
260               -- Trun the blue LED on
261               red_LED<='0';
262               -- Enable clk_en_seq
263               clk_en_seq<='1';
264               -- Increment auto_counter by 1
265               auto_counter<=auto_counter+1;
266               -- Reset ticks
267               ticks<=0;
268
269
270         end if;
271
```

Figure 3 - Autodisplay state VHDL code

```
-- For each pushbutton, test for press
-- If pressed:

      if blue_PB = '0'  then
            -- turn on blue LED
            blue_LED<='0';
            -- load blue audio
            audio_value<=audio_BLUE;
            -- save color in variable 'button'
            button<=blue_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif green_PB = '0'  then
            -- Add code here
            -- turn on blue LED
            green_LED<='0';
            -- load blue audio
            audio_value<=audio_GREEN;
            -- save color in variable 'button'
            button<=green_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif yellow_PB = '0'  then
            -- Add code here
            -- turn on blue LED
            yellow_LED<='0';
            -- load blue audio
            audio_value<=audio_YELLOW;
            -- save color in variable 'button'
            button<=yellow_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif red_PB = '0'  then
            -- Add code here
            -- turn on blue LED
            red_LED<='0';
            -- load blue audio
            audio_value<=audio_RED;
            -- save color in variable 'button'
            button<=red_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif (button /=no_button) then  -- This means a button has been pressed
            ButtonFlag <= 1;    -- but no button is pressed currently (all released)

      end if;
```

Figure 4 - Game state VHDL code (part 1)

```
-- For each pushbutton, test for press
-- If pressed:

      if blue_PB = '0'  then
            -- turn on blue LED
            blue_LED<='0';
            -- load blue audio
            audio_value<=audio_BLUE;
            -- save color in variable 'button'
            button<=blue_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif green_PB = '0'  then
            -- Add code here
            -- turn on blue LED
            green_LED<='0';
            -- load blue audio
            audio_value<=audio_GREEN;
            -- save color in variable 'button'
            button<=green_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif yellow_PB = '0'  then
            -- Add code here
            -- turn on blue LED
            yellow_LED<='0';
            -- load blue audio
            audio_value<=audio_YELLOW;
            -- save color in variable 'button'
            button<=yellow_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif red_PB = '0'  then
            -- Add code here
            -- turn on blue LED
            red_LED<='0';
            -- load blue audio
            audio_value<=audio_RED;
            -- save color in variable 'button'
            button<=red_button;
            -- reset timer (ticks)
            ticks<=0;

      elsif (button /=no_button) then  -- This means a button has been pressed
            ButtonFlag <= 1;    -- but no button is pressed currently (all released)

      end if;
```

Figure 5 - Game state VHDL (part 2)

```
---------------------------------- WIN STATE ---------------------------------------

when WIN =>

            if ticks < quarter_second then
               -- Activate BLUE LED
               blue_LED<='0';
               -- Play Blue LED audio
               audio_value<=audio_BLUE;

            elsif ticks < half_second then
               -- Activate YELLOW LED
               yellow_LED<='0';
               -- Play YELLOW LED audio
               audio_value<=audio_YELLOW;

            elsif ticks < three_quarters_second then
               -- Activate RED LED
               red_LED<='0';
               -- Play RED LED audio
               audio_value<=audio_RED;

            elsif ticks < two_and_one_quarter_second then
               -- Activate GREEN LED
               green_LED<='0';
               -- Play GREEN LED audio
               audio_value<=audio_GREEN;

            elsif ticks = two_and_one_quarter_second then
               -- reset ticks
               ticks<=0;
               -- Go back to WAITS state
               STATE<=WAITS;


            end if;
```

Figure 6 - Win state VHDL code

```
---------------------------------- LOSE STATE ---------------------------------------

when LOSE =>

-- Play WRONG audio for one second, then return to WAITS state
audio_value<=wrong_audio;

         -- set audio_value to the 'wrong' audio value here

         if ticks = one_second then
               -- reset ticks
               ticks<=0;
               -- set button value to no_button
               button<=no_button;
               -- Go back to Waits state
               STATE<=WAITS;
         end if;


      end case;

   end if;

end process;

end Behavioral;
```

Figure 7 - Lose state VHDL code

Discussion

The experimentation with the pseudo-random number generator revealed its straightforward operation: employing eight inputs to yield a single output selected at random. Each of the eight inputs holds an equal probability of being chosen, providing the multiplexer with a pool of choices. The "RandomCounter" component embedded within the generator plays a pivotal role in feeding inputs to the multiplexer, ensuring a random selection by altering the multiplexer's functionality whenever an input is selected.

The introduction of the TEST_CONTROLLER in this laboratory marks a significant departure from the previous sound decoder. This new component facilitates the creation of the Simon Game Box by executing a novel functionality. It accepts separate inputs for the clock and button colors, subsequently outputting them to their respective colors and sequences, thereby enhancing the game's dynamics.

The elaboration of the five primary states—Game State, Win State, Lose State, WAITS State, and Autodisplay State—provides insight into the intricate functionality of the system. The Game State, for instance, is responsible for comparing the pressed button value with the generated sequence, triggering various events such as illuminating lights and setting timers. Each state is composed of specific hardware blocks within the CODEC.bdf, delineating the hardware components involved in executing the respective functionalities.

Expanding the number of game states from 5 to 8 necessitates modifications in the code structure, introducing additional states by coding new conditions and updating relevant variables. Conversely, scaling up to 16 game states requires hardware augmentation, such as incorporating 8 buttons instead of 4 in the Simon Game Box, thereby expanding the range of possible combinations and game states.

Throughout the progression of the laboratories, the team acquired a diverse array of skills, spanning from circuit development using Quartus Prime software to understanding the nuances of ModelSim software and the intricacies of the Simon Box game. An alternative pedagogical approach involves familiarizing students with the internal operations of the Simon Box game and the underlying class objectives prior to engaging in the lab exercises, thereby fostering a deeper understanding of the lab procedures and objectives.

Conclusion

In summary, this lab served as a pivotal learning experience for Group 18 in understanding the diverse applications of the System Controller VHDL file in implementing various functionalities within the Simon Game. Through code alterations, the team gained insights into enhancing the intricacies of the Simon Game's operation, resulting in rhythmic outcomes when compiled onto the DE2 sound board box.

The implementation of the System Controller code by Group 18 encompassed key states such as the wait state, autodisplay state, game state, and win/lose state. By refining these aspects of the code, the lab culminated in a successful and precise demonstration of the Simon State Machine on the Simon Game Box.

Overall, this concluding lab presented a remarkable challenge, providing an opportunity to apply accumulated knowledge from previous labs. It enabled the group to craft a comprehensive state machine that encapsulated all the elements learned throughout the course of the lab sessions.

Acknowledgments

All members of Group 18 contributed equally and participated in all lab activities. In terms of the lab report, Siwei wrote the discussion and conclusion sections. Ansh described the executive summary, and introduction, and edited the document. Rushi wrote the experimental methodology and results sections.

References

- Lab report template - Lab-Report-Submission-Instructions
- Lab 8 document - The-Simon-State-Game2