Course Name: EMBEDDED SYSTEMS I / III

Course Number and Section: **14:332:493:03 / 16:332:579:05**

Year: Spring 2024

Final Project Report: Simon Says Game

Lab Instructor: Milton Diaz

Student Name and RUID: Karim Smires, ks1686, 206007000

Date Submitted: May 5, 2024

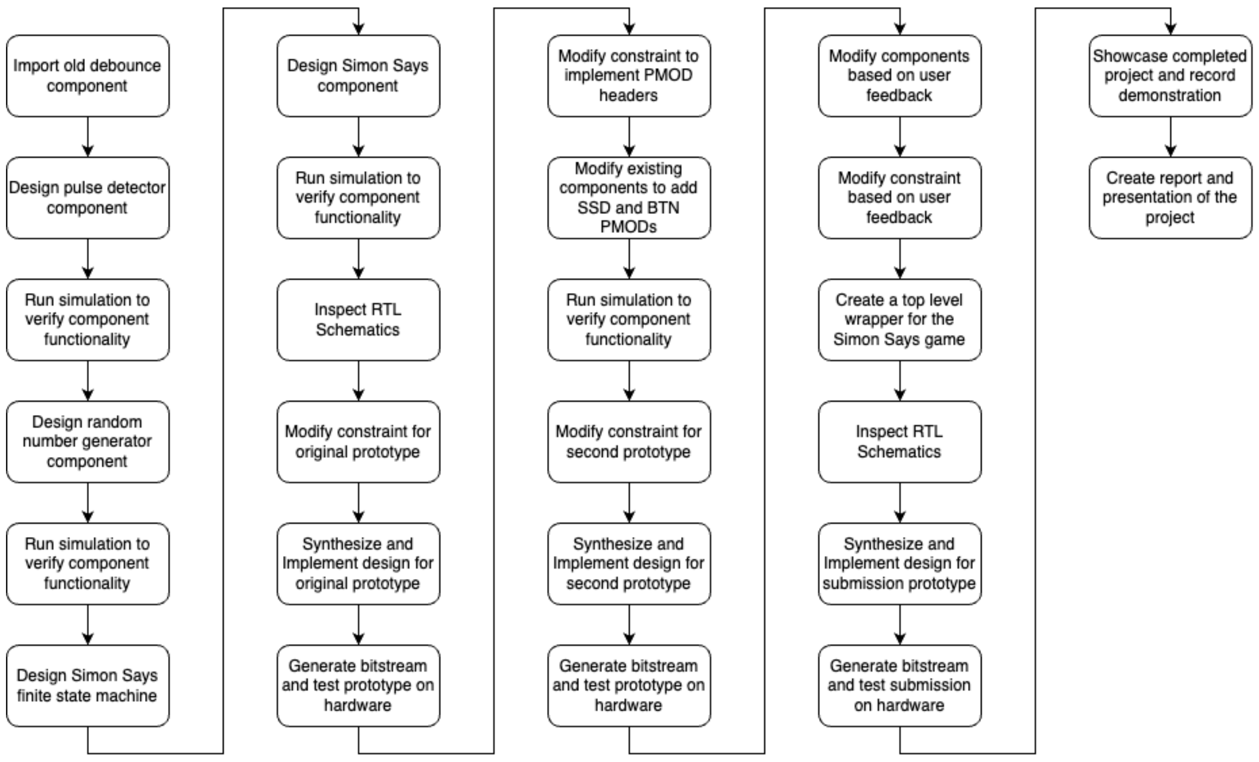
GitHub Link:
https://github.com/embedded-systems-1-spring-2024-labs/final-project-ks1686

<u>Purpose/Objective:</u>

The purpose of this final project is to combine all of the concepts and knowledge that we've acquired throughout the semester to create a custom-designed system from scratch. I decided that my final submission for this project would be a "Simon Says" game, using the seven-segment display and additional buttons, both paired with registers, to enable the functionality of the game. The project had me in the labs for a week straight, commuting to the labs and staying from 8:30 in the morning until the boards had to be collected. The first working prototype was achieved in a few days, with the remaining days being dedicated to polishing the game into a "consumer ready" version. Unfortunately, due to the deadlines of other final projects and exams, I was unable to complete the full vision of the project, but managed to create a working project that met the requirements given in the Canvas assignment page.

To create the Simon Says game, the debounce component was first reused. We then needed to ensure that we only pass a single output from the buttons, requiring the creation of a tweakable pulse detector, which in this embedded system relies on the falling edge of the debouncer. A custom random number generator that can accept seeds was also required to enable the randomization portion of the game. Since VHDL does not have a built in randomization function, this component had to be custom built and tested. Using these, a multitude of signals were created to enable the functionality of the finite state machine for the Simon Says game. The game logic for the state machine was straightforward, but required constant testing of timings to ensure that the game functioned as intended and was understandable to users. "Customer feedback" was implemented by having multiple "customers" play the game, with adjustments made based on the feedback. The registers are the core of this game, with random numbers being fed into registers with increasing amounts, maintaining storage to create the game's functionality of comparing random numbers with user inputs.

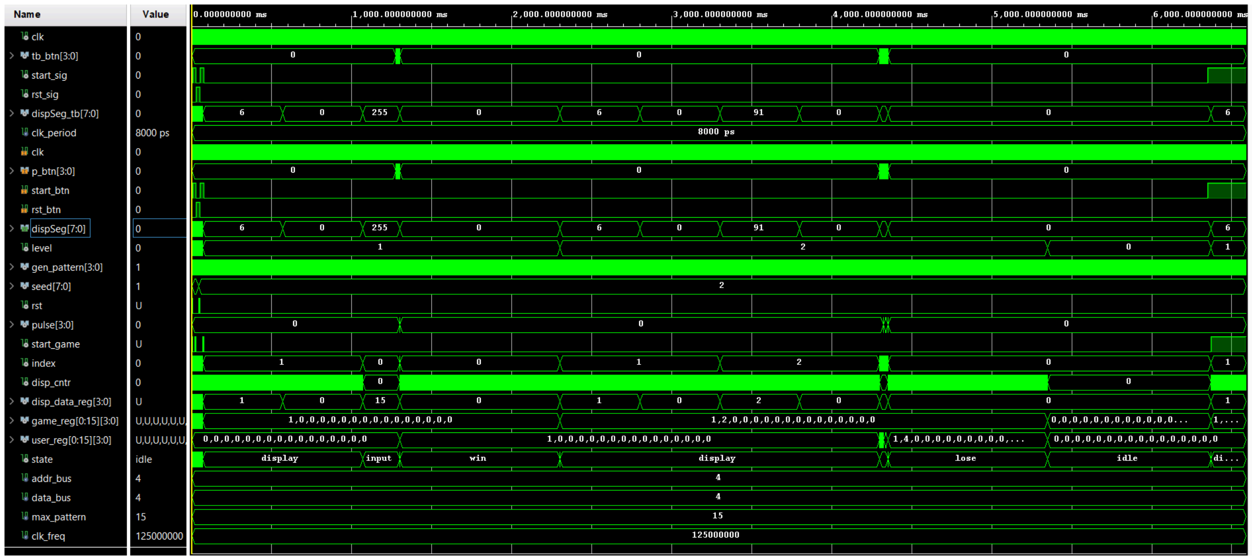
Theory of Operation:


Import old debounce component
Design pulse detector component
Run simulation to verify component functionality
Design random number generator component
Run simulation to verify component functionality
Design Simon Says finite state machine
Design Simon Says component
Run simulation to verify component functionality
Inspect RTL Schematics
Modify constraint for original prototype
Synthesize and Implement design for original prototype
Generate bitstream and test prototype on hardware
Modify constraint to implement PMOD headers
Modify existing components to add SSD and BTN PMODs
Run simulation to verify component functionality
Modify constraint for second prototype
Synthesize and Implement design for second prototype
Generate bitstream and test prototype on hardware
Modify components based on user feedback
Modify constraint based on user feedback
Create a top level wrapper for the Simon Says game
Inspect RTL Schematics
Synthesize and Implement design for submission prototype
Generate bitstream and test submission on hardware
Showcase completed project and record demonstration
Create report and presentation of the project

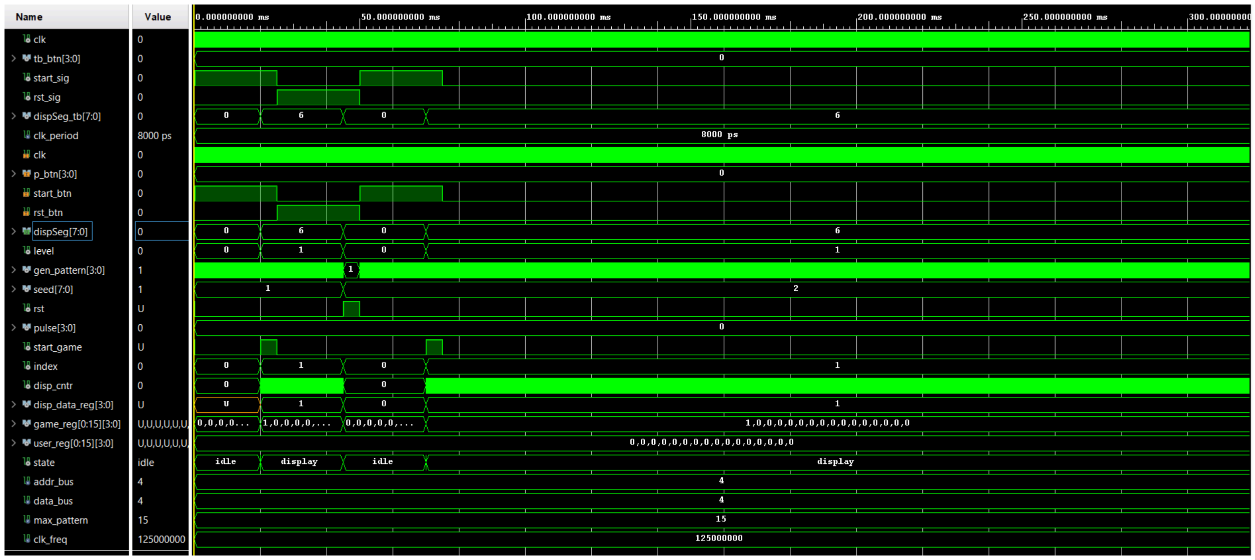
Simulation Waveforms:

1. Waveform for top_level and simon_game:

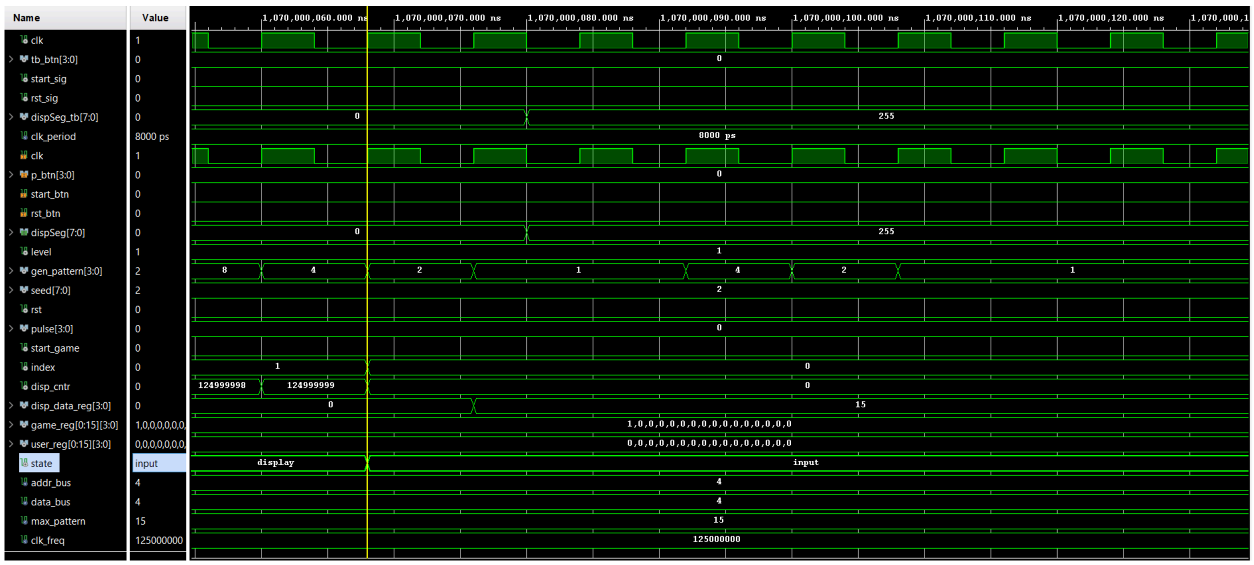
The top_level and simon_game components share the same waveform as the top_level is simply a wrapper to hide the signals of the simon_game component and make it easier for people to analyze the schematics and synthesis of the embedded system. Functionality wise, the inputs are the same thing. The first waveform below shows a game of two rounds being played:



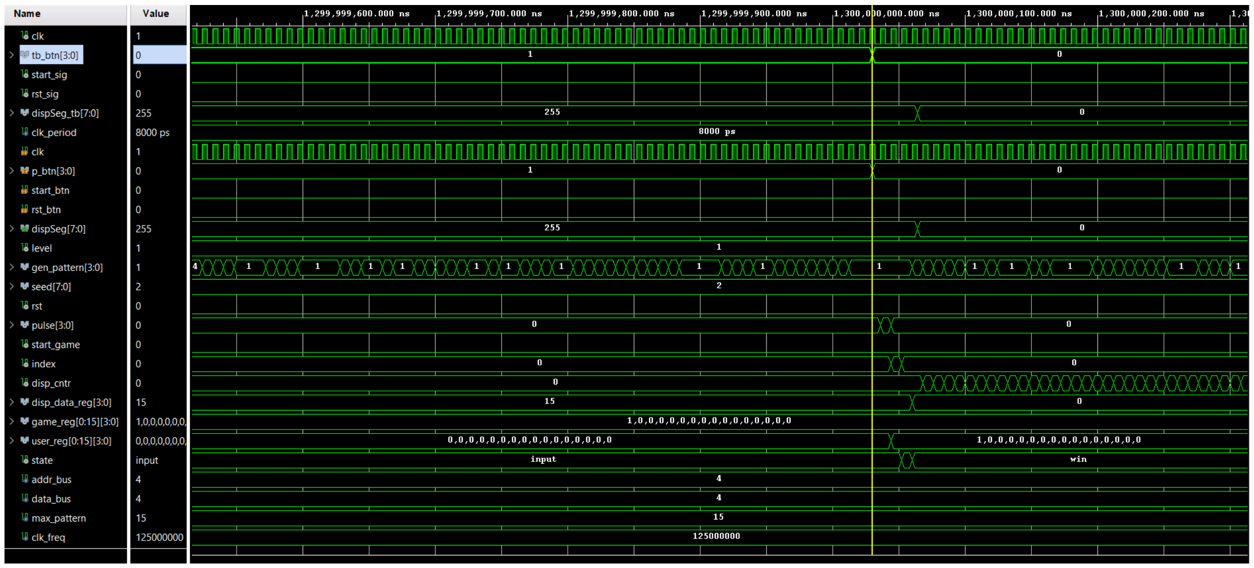
Here we can see here the overall operation of the game over the course of two rounds, with transitions from different states for displaying data, taking user input, and handling win or lose conditions. Zooming in below, we address some of the harder to see states and signals:



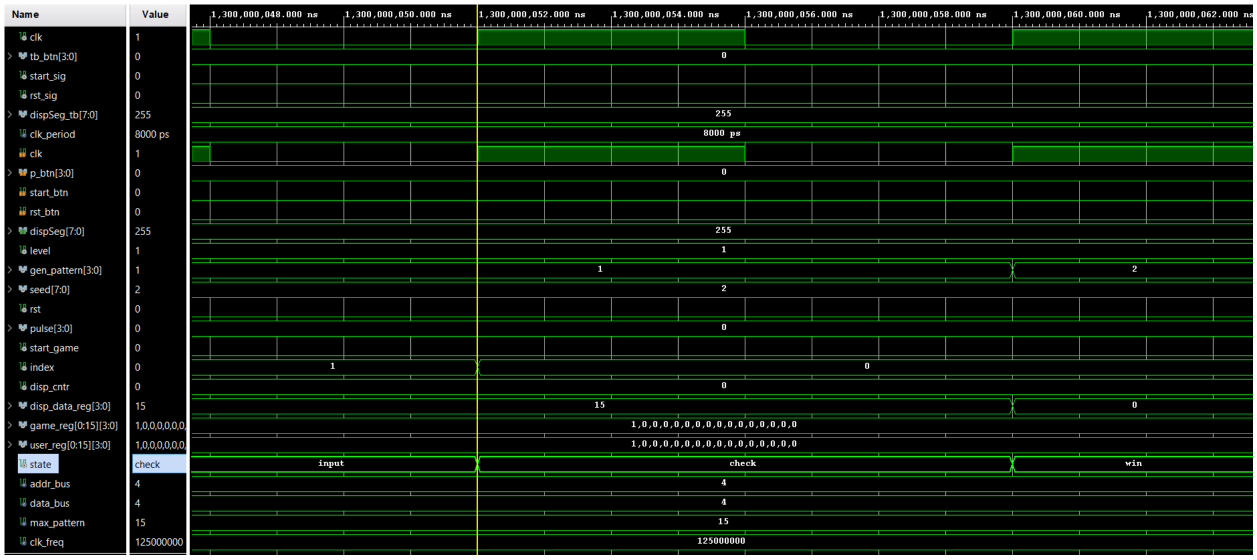
Here, we test the functionality of the start and reset buttons, and can see that the device transitions from the idle state to display, then back to the idle state, with the game register also being cleared. Note that the game continues with a new number being loaded into the game register, and the display responding accordingly. We then see the game continue as normal below:



We can see here the transition between display and input states as the disp_cntr signal reaches a specific threshold. Below, we can see the button press, generated pulse, and user input being fed into the register:

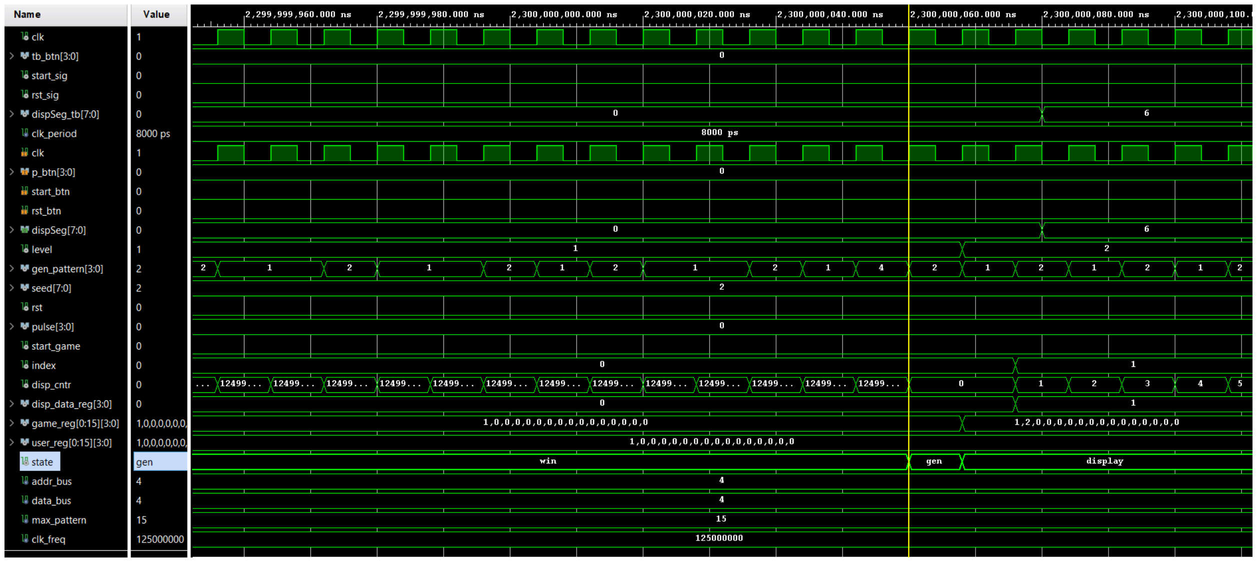


After this, we need to transition in to the check state, which can be seen below:



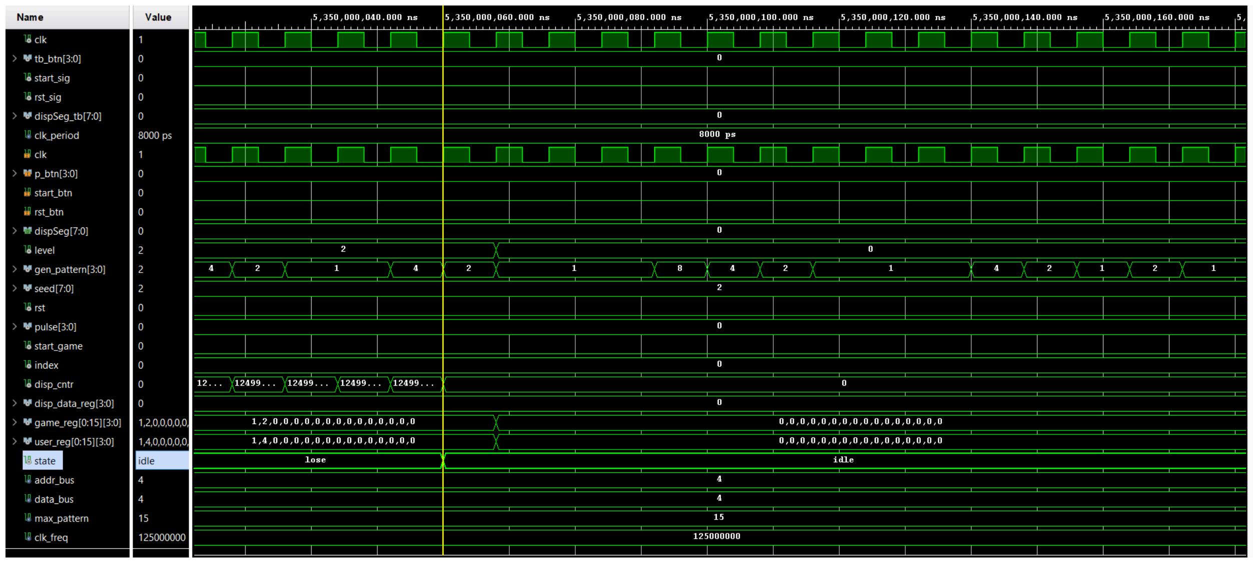
Name	Value
clk	1
tb_btn[3:0]	0
start_sig	0
rst_sig	0
dispSeg_tb[7:0]	255
clk_period	8000 ps
clk	1
p_btn[3:0]	0
start_btn	0
rst_btn	0
dispSeg[7:0]	255
level	1
gen_pattern[3:0]	1
seed[7:0]	2
rst	0
pulse[3:0]	0
start_game	0
index	0
disp_cntr	0
disp_data_reg[3:0]	15
game_reg[0:15][3:0]	1,0,0,0,0,0,0,
user_reg[0:15][3:0]	1,0,0,0,0,0,0,
state	check
addr_bus	4
data_bus	4
max_pattern	15
clk_freq	125000000

We can see the check state and the transition into the win state, as the data in the game_reg and user_reg match each other. Below, we can see the transition after our counter hits a threshold into the generate state, with a new number being added into the game_reg and the level incrementing:

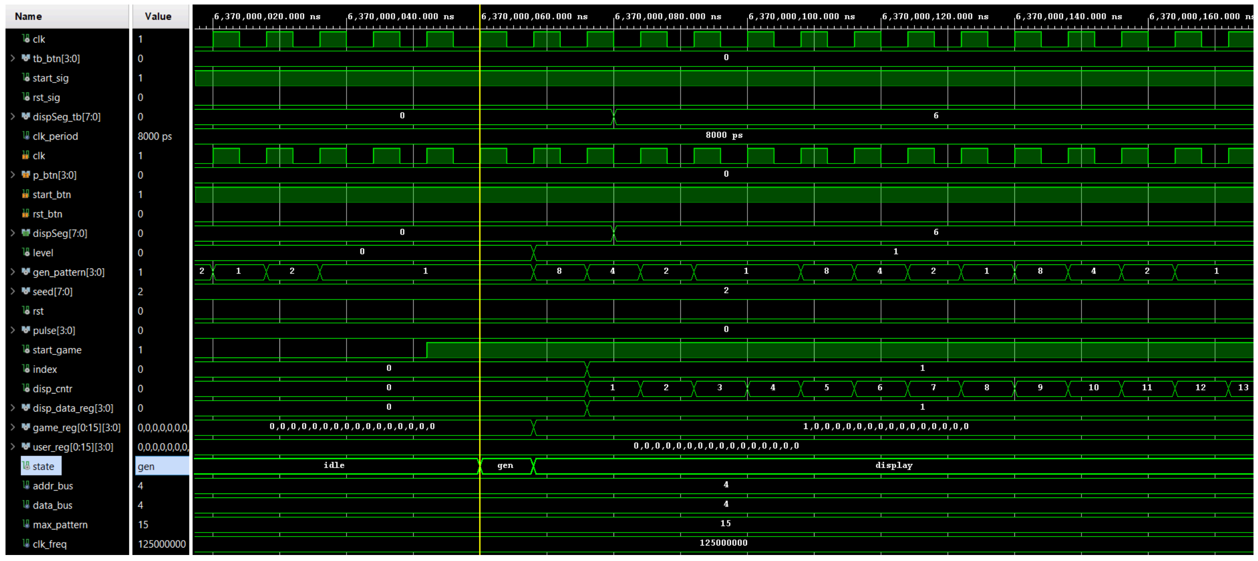


Name	Value
clk	1
tb_btn[3:0]	0
start_sig	0
rst_sig	0
dispSeg_tb[7:0]	0
clk_period	8000 ps
clk	1
p_btn[3:0]	0
start_btn	0
rst_btn	0
dispSeg[7:0]	0
level	1
gen_pattern[3:0]	2
seed[7:0]	2
rst	0
pulse[3:0]	0
start_game	0
index	0
disp_cntr	0
disp_data_reg[3:0]	0
game_reg[0:15][3:0]	1,0,0,0,0,0,0,
user_reg[0:15][3:0]	1,0,0,0,0,0,0,
state	gen
addr_bus	4
data_bus	4
max_pattern	15
clk_freq	125000000

Now, we test below a case where the user_reg and game_reg do not match, or a lose condition. The game resets the level and the machine goes back into the idle state:

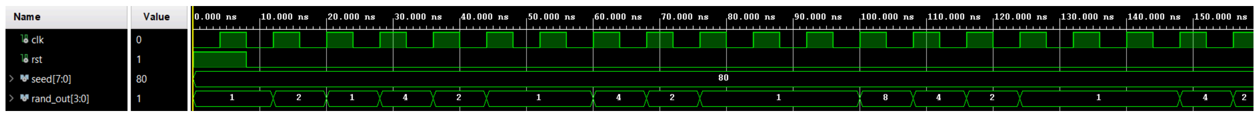


And finally, we showcase the transition from this idle state to a new game with the press of the start button below, finishing the functionality display of our embedded system:



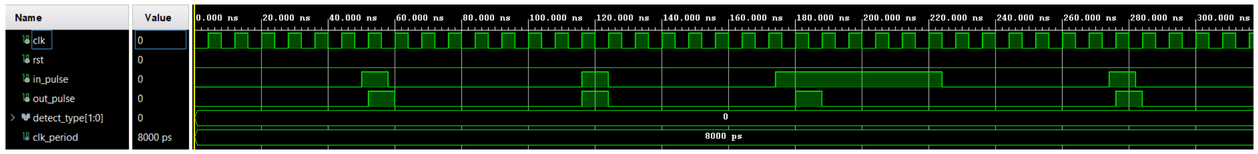
2. <u>random_generator</u>

Below is our random number generator showcasing the functionality of our component, with the random outputs visible over time below. The seed here is static, but changes in the simon_game component for a better randomization technique:



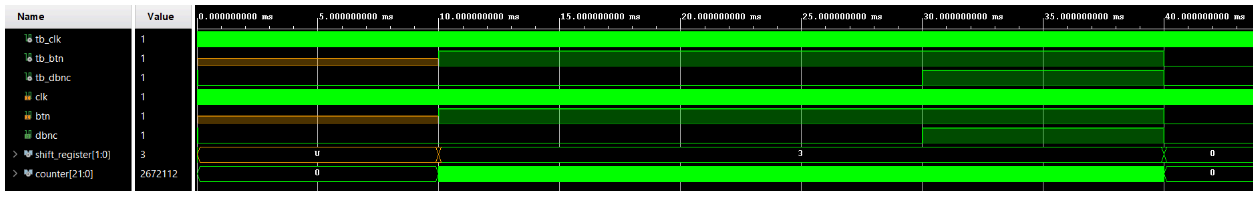
3. pulse_detector

Below is our pulse detector component, testing the pulse detector with a rising edge configuration. While it may at the start look unnecessary, pay attention to the third high in_pulse, where the corresponding out_pulse only occurs for a single clock cycle:



4. debounce

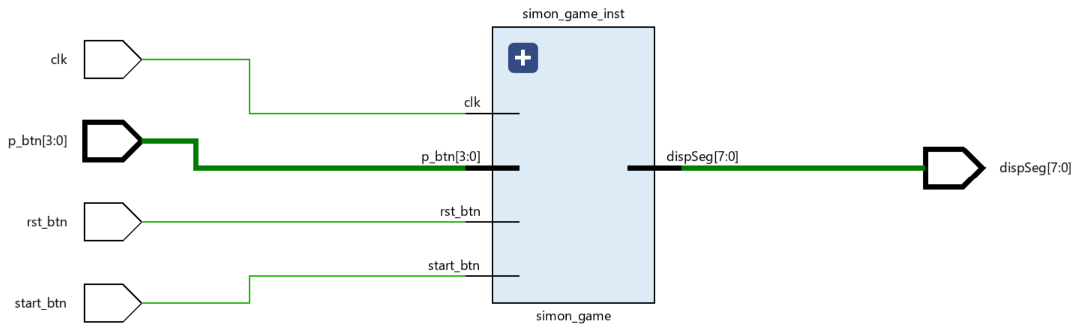
Below is our debounce component that was created in the original classroom labs. After 20ms, the debounce logic kicks in and outputs as intended:

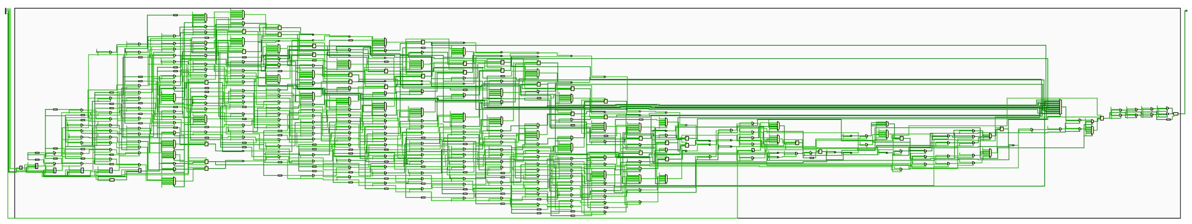


Vivado Schematics:

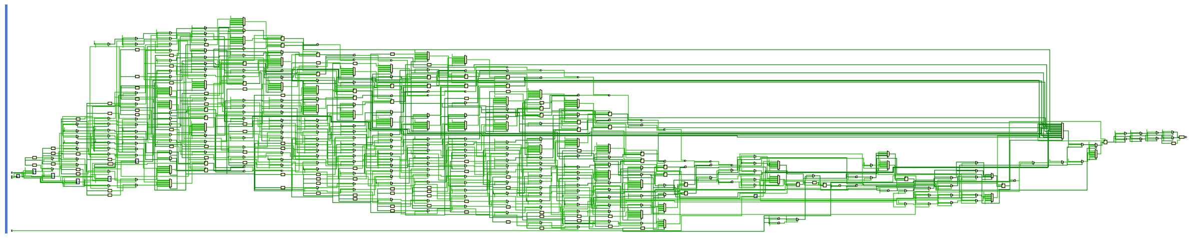
1. RTL Elaborated Designs

a. top_level and top_level expanded

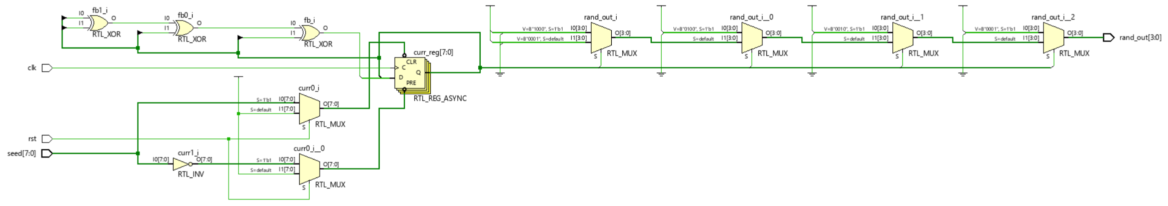




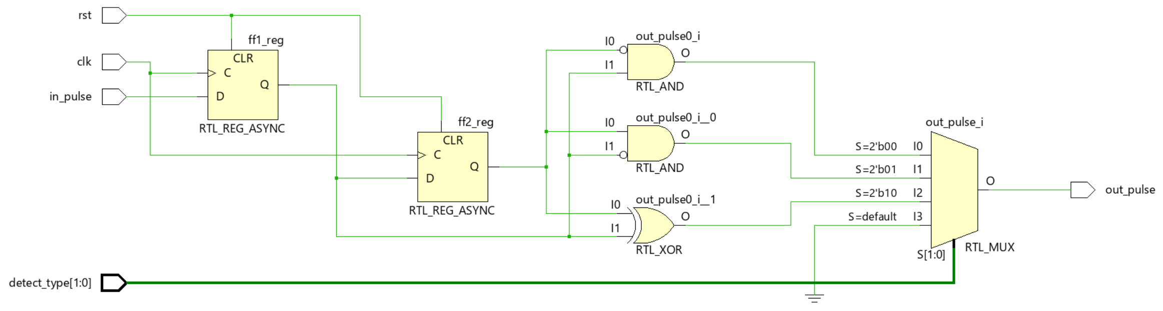
b. simon_game



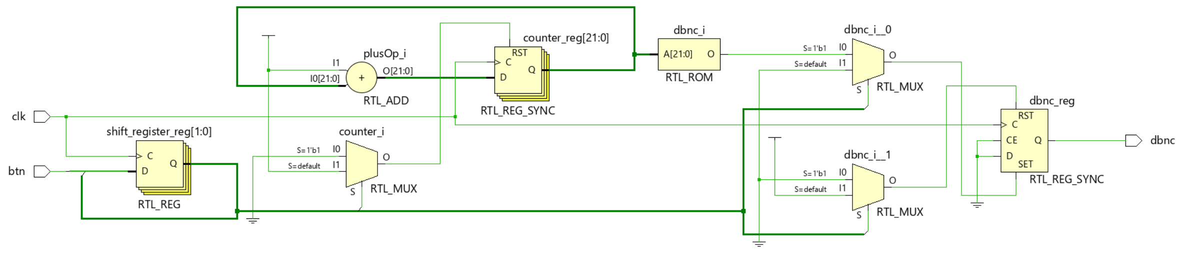
c. random_generator



d. pulse_detector

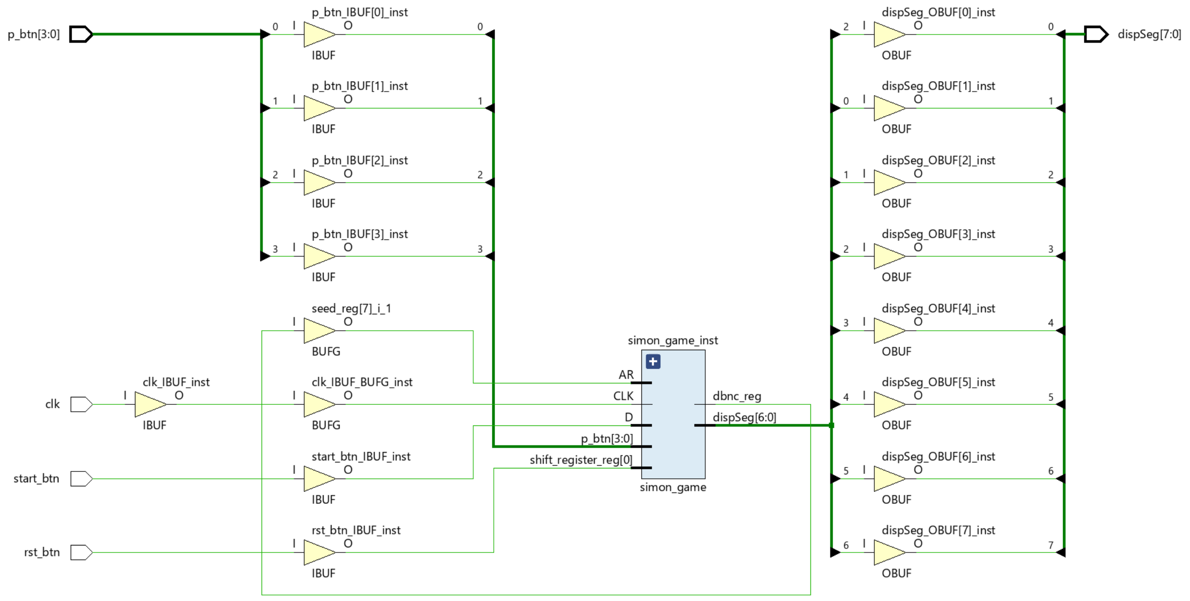


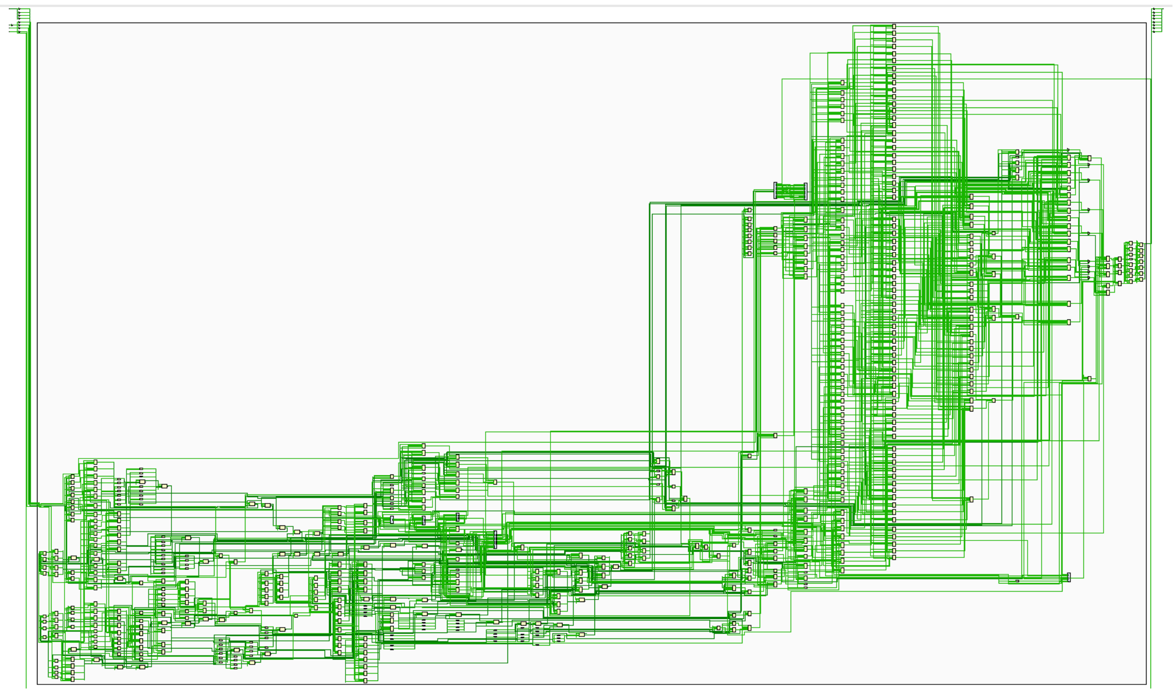
e. debounce



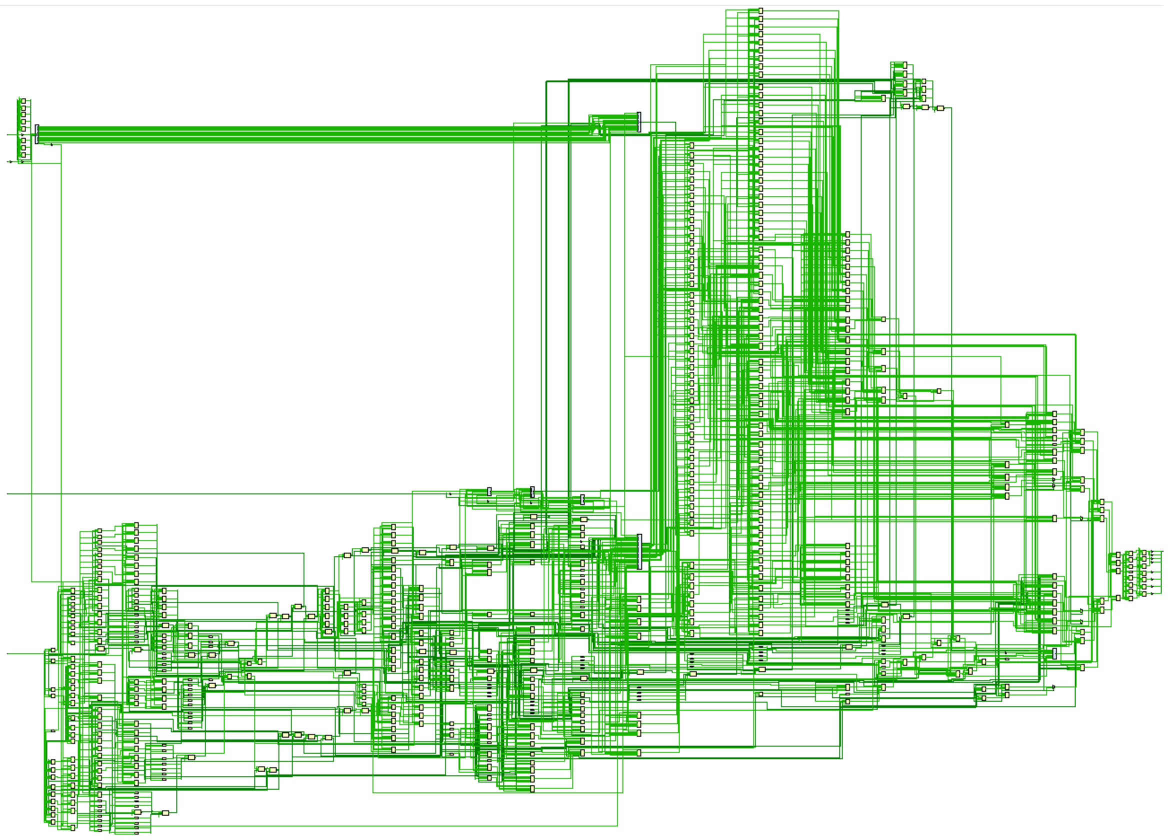
2. Synthesis Schematics

a. top_level and top_level expanded

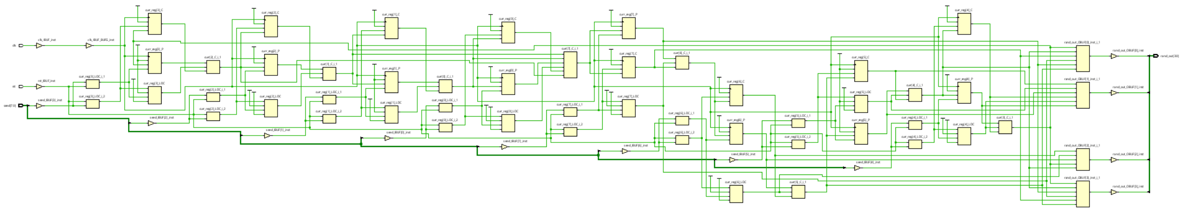




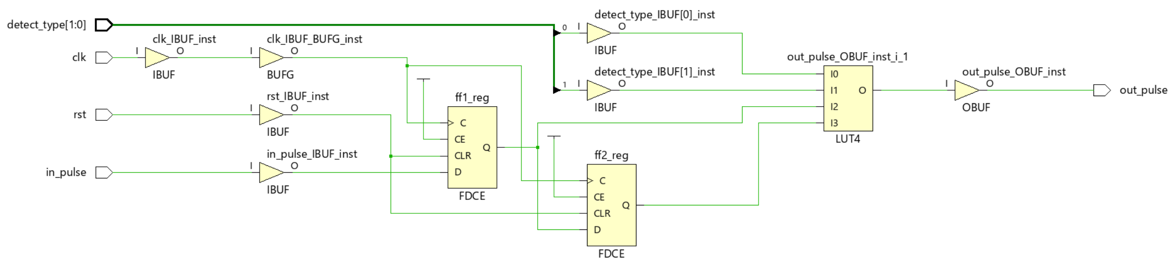
b. simon_game



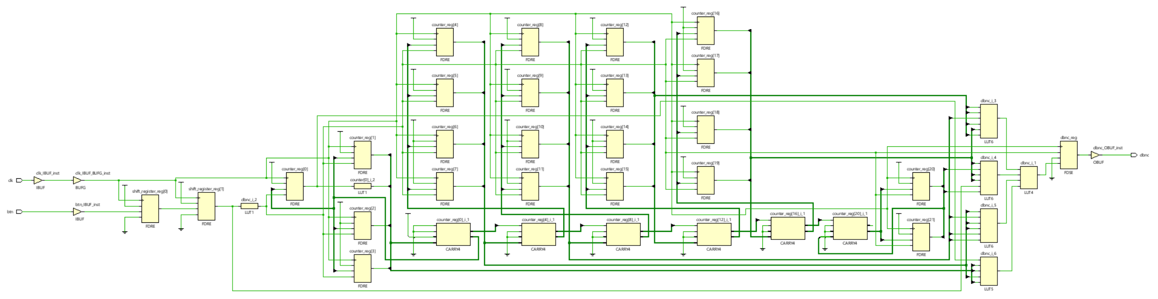
c. random_generator



d. pulse_detector

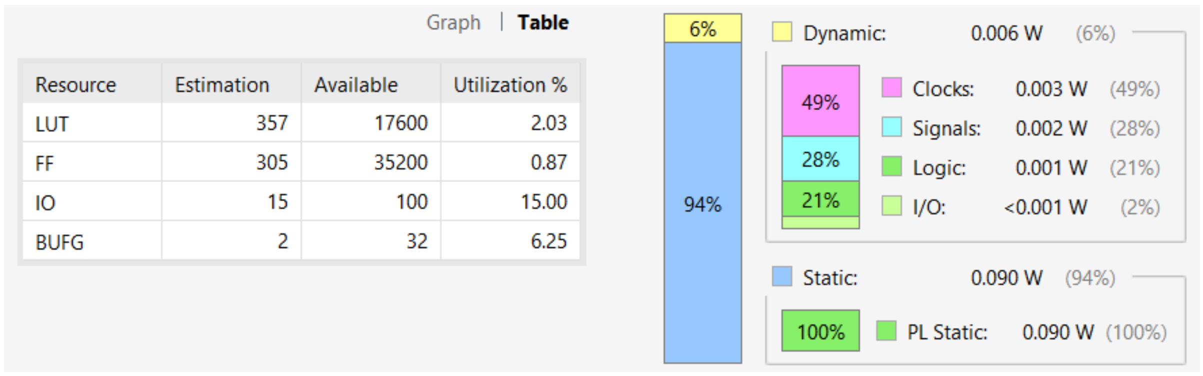


e. debounce



3. Power and Utilization

a. top_level



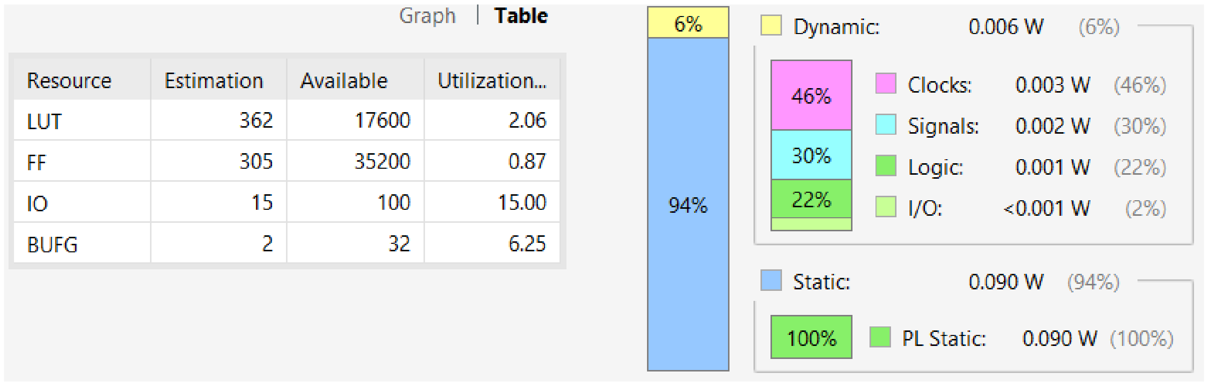
Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	357	17600	2.03
FF	305	35200	0.87
IO	15	100	15.00
BUFG	2	32	6.25

6%
94%

Dynamic:		0.006 W	(6%)
49%	Clocks:	0.003 W	(49%)
28%	Signals:	0.002 W	(28%)
21%	Logic:	0.001 W	(21%)
	I/O:	<0.001 W	(2%)
Static:		0.090 W	(94%)
100%	PL Static:	0.090 W	(100%)

b. simon_game



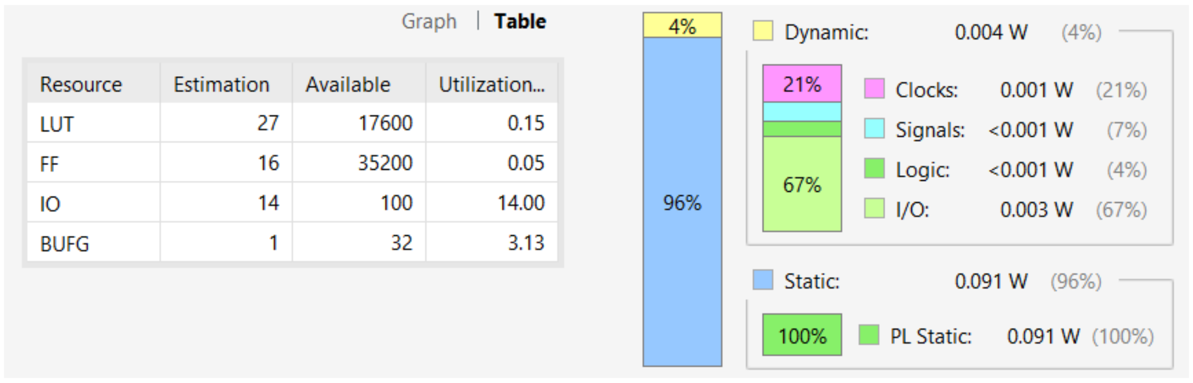
Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	362	17600	2.06
FF	305	35200	0.87
IO	15	100	15.00
BUFG	2	32	6.25

6%
94%

Dynamic:		0.006 W	(6%)
46%	Clocks:	0.003 W	(46%)
30%	Signals:	0.002 W	(30%)
22%	Logic:	0.001 W	(22%)
	I/O:	<0.001 W	(2%)
Static:		0.090 W	(94%)
100%	PL Static:	0.090 W	(100%)

c. random_generator

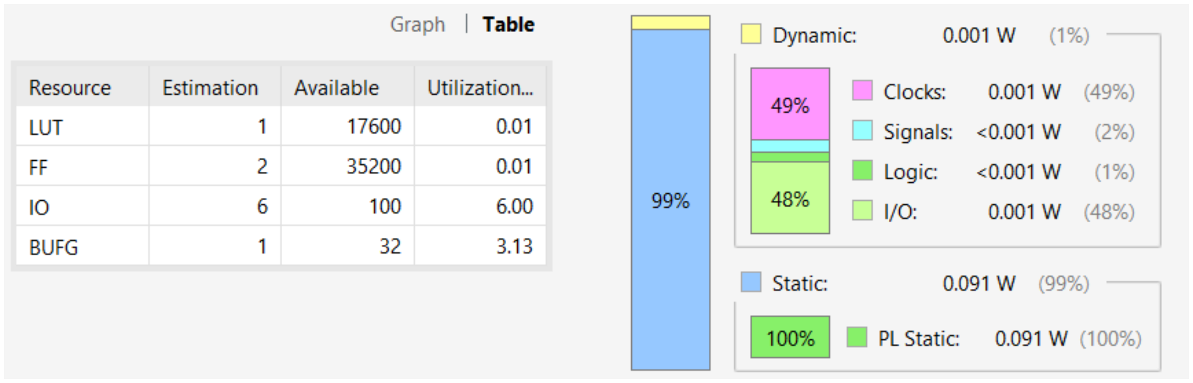


Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	27	17600	0.15
FF	16	35200	0.05
IO	14	100	14.00
BUFG	1	32	3.13

Dynamic:	0.004 W	(4%)	
Clocks:	0.001 W	(21%)	21%
Signals:	<0.001 W	(7%)	
Logic:	<0.001 W	(4%)	
I/O:	0.003 W	(67%)	67%
Static:	0.091 W	(96%)	96%
PL Static:	0.091 W	(100%)	100%

d. pulse_detector

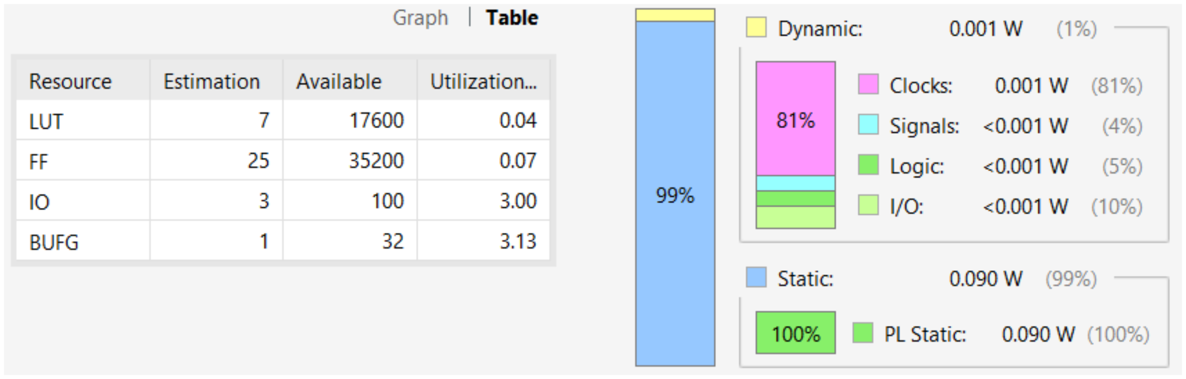


Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	1	17600	0.01
FF	2	35200	0.01
IO	6	100	6.00
BUFG	1	32	3.13

Dynamic:	0.001 W	(1%)	
Clocks:	0.001 W	(49%)	49%
Signals:	<0.001 W	(2%)	
Logic:	<0.001 W	(1%)	
I/O:	0.001 W	(48%)	48%
Static:	0.091 W	(99%)	99%
PL Static:	0.091 W	(100%)	100%

e. debounce



Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	7	17600	0.04
FF	25	35200	0.07
IO	3	100	3.00
BUFG	1	32	3.13

Dynamic:	0.001 W	(1%)	
Clocks:	0.001 W	(81%)	81%
Signals:	<0.001 W	(4%)	
Logic:	<0.001 W	(5%)	
I/O:	<0.001 W	(10%)	
Static:	0.090 W	(99%)	99%
PL Static:	0.090 W	(100%)	100%

Changes to constraint file (Zybo_Master.xdc):

The constraint file that was used in this project, Zybo_Master.xdc, was designed to provide functionality for this embedded system. The lines for the clock, 3 PMOD headers and two buttons were uncommented to allow for the unique functionality of the Simon Says gameplay.

<u>**Answers to Additional Questions:**</u>

1. <u>Embedded Design Process</u>
    a. Performance:

    The performance requirements of the embedded system are fairly straightforward. The game relies on storing random values and increasing the number of said values stored, requiring that we enable the functionality of registers and conditional statements to make the comparisons in a manner that is both fast but also understandable by players.

    The game needs to be responsive to user inputs and provide users with an understandable interface to interact with the game. The seven segment display allows users to see exactly what numbers they need to input once the random sequence is generated. The user inputs are then requested after all numbers have been displayed. To accomplish this, we use counters and finite state machines to ensure that the game logic and user interface operate at the same frequency, but restrict themselves based on conditionals.

    The memory portion of the game is accomplished via registers, which support 16 unique 4-bit numbers. The numbers in the game are currently limited to numbers 1-4, as there are only 4 buttons on the button PMOD. All of these ideas and components combined ensure that performance requirements are met.

b. Size:

The size requirements of the system are related to the actual specifications of the Zybo board, with the restrictions that the board has in mind used when designing this embedded system. Referencing the Zybo specifications and the post-implementation utilization tables, IO is currently our only major issue. Therefore, we could more likely further complicate the design by adding a keyboard instead of the button PMOD, and implement a larger register size, creating a far more complex game.

Another possibility that could have been added was VGA or audio output, but the IO would start to slowly become more and more of a concern as we continue to add more of these functionalities.

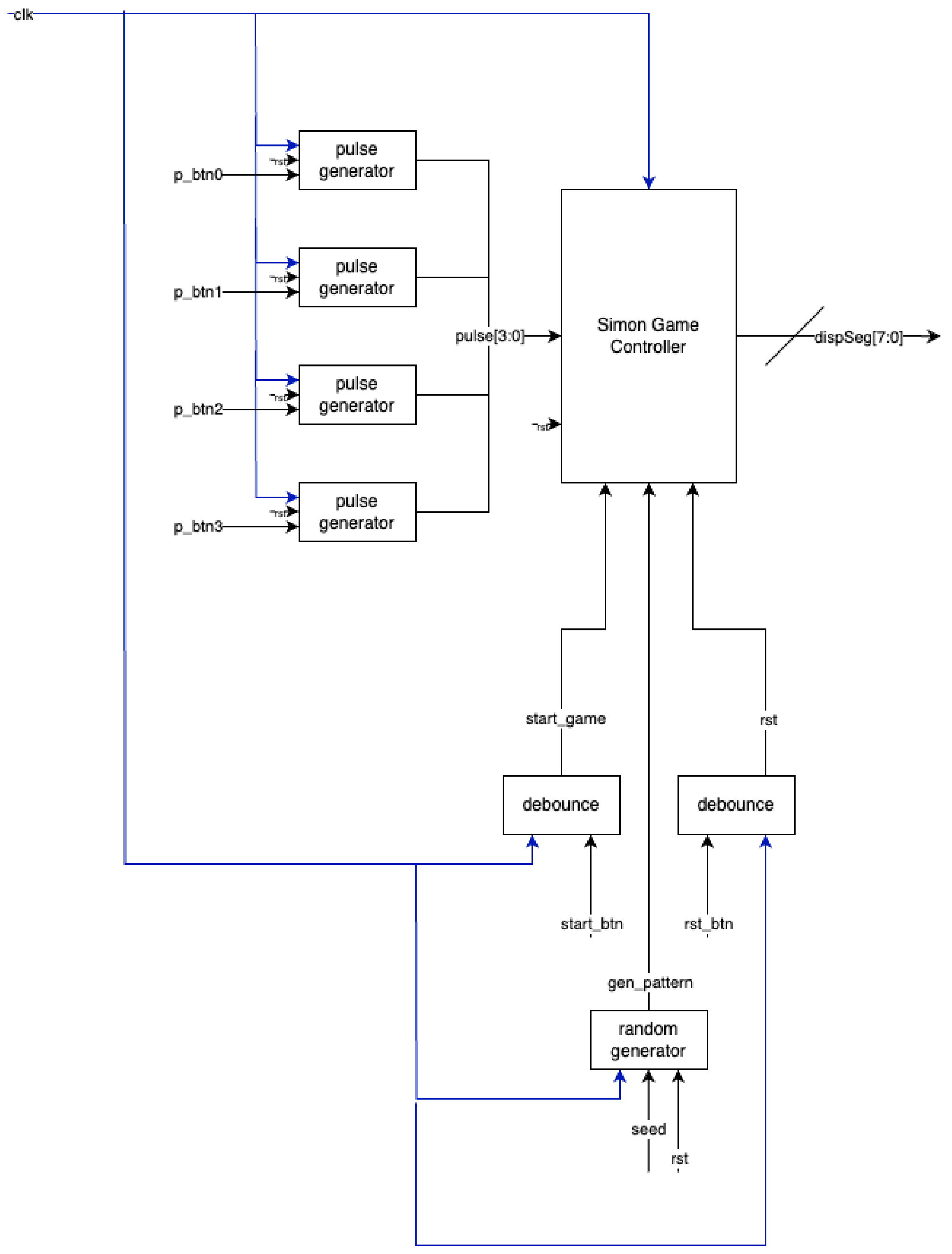
c. Power and Energy:

Similar to the requirements listed in the size portion, the power and energy requirements are tied to the specifications of the Zybo board. The Zybo accepts a maximum of 5V and a maximum of 4A over the barrel jack. The maximum of 20W that the board provides is FAR greater than 0.1W our embedded system consumes, meaning that we have no issues with power or energy consumption.

d. Limitations

With our current system, we have absolutely zero concern with limitations on the Zybo 7000 board. When attempting to recreate the Simon Says game from my childhood, the hardest portion was designing the conditions and states for the program itself. The board was more than capable of running the logic that was designed, and with the addition of PMODs, we were still only using 15% of the IO on the board. This can be a bit of a concern though, as when we may attempt to add new features like display or audio output, we may begin to run into issues. Logically, the game functions incredibly well on the board.

2. System Block Diagram:



<u>Conclusion:</u>

This project was a culmination of all knowledge and techniques that I learned in this course. Rather than starting with a very complicated design that would potentially cause issues with utilization of the board, I decided to opt into a simpler design that I could perfect and build on instead. The game logic was fairly straightforward, but there were some challenges that needed to be solved, introducing components like a pulse generator and a random number generator.

When working on this project, there were several observations and obstacles that I found. First, timing was a major concern, since timings would make or break the actual logic of the game. I also had to consider the playability of the game, requiring me to bring several other people to play. Some of these players were quite stupid, while others were a little too good, meaning I had to tweak timing logic multiple times to finalize the Simon Says gameplay.

While there were many other features that I wished to implement, due to the timing of other projects and exams, I opted to instead focus on providing a fully working perfect project that met all the requirements rather than being overly ambitious. While this may not earn me as many points as I hope, I prefer to solidify a passing grade than risk not having a submission at all.

<u>Follow Up:</u>

I'm very happy that I chose to take this course, as it was a very large shift from the mathematics and software programming courses that I normally take. It was a great experience to learn about embedded systems.