Simon Game

ENGS 31 Final Project Report

By: Carolina Almonte and Kevin Ponce

Abstract

In our project we designed the Simon game. Our system has a selected number of randomized bits that create a game ID. The randomized game ID is selected by the enter button and displayed as a light sequence with sound. Then, the user is supposed to follow the game with the corresponding button presses. Our comparator then checks to see if the inputted button presses matches that of the game ID. There is a timer that counts down how much time the player has between button presses. The sequence increases as the level increases.

Table of Contents

1. Introduction

Our project is to build a game that displays a light sequence that increases as the game progresses. Using four LED lights that correspond to a different button. Each light has a different color and makes a different sound when the button is pressed. The objective of the game is to follow the game sequence in the order that it is displayed. We also need to have a randomized game each time so that the player plays a different game every time.

2. Design Solution

2.1. Specifications:

Our circuit takes is controlled by the *enter* button which chooses a randomized game id and displays it with sound. Once displayed, the user has about 5 seconds to press the corresponding button which is checked to see if it matches what was displayed. Our display consists of four LEDs that are arranged clockwise with green being at the twelve o'clock position, then red, blue and finally yellow. Each LED is accompanied with a button and a sound arranged in the same way. This process continues until the game is lost. Our inputs are the button presses and the game id displayed. Our outputs are the sound the LEDs turning on. See figure in Appendix A for an image of our circuit and to see clearly our inputs and outputs.

2.2. Operating Instructions

- Press the enter button which is the center button demonstrated on Appendix A.

- Once the game starts, it will display a light pattern. Then, you will be given time to press the button that correspond to that specific light. If you forget, the lights on the game are arranged in the same order as the buttons.
- As the game progresses, it will continue to display different light patterns as add-ons to the existing light pattern it first showed you. Your job is to follow that light pattern in the same order.
- You play until you press the wrong button or take too long between button presses.
- **Key things**:
    - Each light has a sound that corresponds to it. As you press the button it should light up and make a sound at the same time.
    - There is a little display on the left that prompts you to start before every game and after you've lost the game. It also shows you what level you are on and displays "-----" (dashes) when you should wait for the lights to display to press the button.
    - There's no winning in this game so play until you lose.

2.3. Theory of Operation

Our circuit is made up of a shift register, a multiplexer, decoders, and multiple counters. These components are sectioned into into three main phases. The displaying phase, the inputs/comparator phase, and the timer phase that works with the controller to create an interactive game. Each phase plays a key role in making the game work.

Below are the specific smaller components that are broken down to explain exactly what happens in each phase:

The GameID Counter:

This is a 50 bit register that we made using the linear feedback shift register from the Xilinx page (can be seen on Appendix B final image). This register XNORs the 50th bit, the 49th, the 24th bit, and the 23rd bit. Then, it loops the bits around to the beginning of the shift register and it continuously runs it until the game starts (the *enter* button is pressed) then it stops. The longer you wait to press the button, the more likely it is to generate a more randomized game. This data which is the *Game ID* is then sent to a multiplexer that generates the game data.

Step Counter:

This counter is combined with the *Game ID* from the GameID counter in the same multiplexer. The way it works is that this counter grabs the level that the player is on, then generates separate counts for each step taken. Once the number of steps equals the number of the level, the counter releases a terminal count. Then using this information it choose which data to pump out from the multiplexer. The data that will be pumped out is the Game_data which as the level progresses is if you take the level bit + 1.

Check box (comparator):

This grabs the button ID (2 bits) and the game data (2 bits) and tell you if the button presses have an equal value to the game data that was displayed.

LED Decoder:

This is used in the display phase and it is indicated by the led_en. It grabs the game_data (2 bits) then displays the corresponding LED. When it is in the timer phase, the signal *timeout* is low which displays the buttons that correspond to the LED. These buttons and LEDs are connected with sound. These are the notes for the sound using a piano scale: G3- 196 hz, C4 - 262 hz, E4 - 329 hz, G4- 392 hz.

Button Decoder:

This grabs the four buttons from the FPGA and converts it into a two bit number, called the *bttn_id*, that can be used for the comparator and and the LED-Decoder. This allows for easy access of the game seed as it can be read two bits at a time and easily matches with this id.

Level Counter:

In the level select of the controller this counter counts up one level. If the level reaches above 49, then the level stops counting. Thus, it will be stuck forever in the signal *level* which is the end of the data. (This is very unlikely!!! It is nearly impossible for anyone to make it to level 50 on this game).

Display Buffer and Timeout Counter:

This counter counts up to some arbitrary number. After many trials and test we discovered that the number for the display buffer was .7 seconds and for the timeout counter it was 5 seconds. The display buffer allows the LED to be on for a set amount of time. While the timeout prevents the game from going on for forever.

*For a more concise and clear visual of the theory of operation please refer to Appendix B which is our final RTL design for this project. To see the many versions of our RTL designs please reference Appendix B: original, version 2 , and final.

2.4. Construction and Debugging

We spent a really long time designing and debugging our game. We spent hours creating the RTL design so it was really easy to delegate the different parts of the project and focus on smaller parts to build up. During our planning process we divided our design into three main components: 1. How to display the full game. 2. How to have inputs and how that compares to the game. 3. The timers needed to interact with the controller in order to make sure that the timeout worked correctly. Having these three components as our main focus really allowed us to think about what we wanted our game to do and how we could possibly achieve that. While our game has changed since that original design - which can be accessed in Appendix B figure 1- you can tell that those three main components still remain.

We had many bugs during the construction of our game. Our main bug was that our controller wasn't working properly. It was displaying one bit ahead and as the game progressed the difference between the bits was getting bigger, which was a really big problem. It was mainly our fault because we didn't re-update our state machine to account for the fact that we changed our original design and we were now using a multiplexer for our data. So what we did to fix this was rewrite the entire controller again. Even then, we were still having issues because our controller wasn't counting correctly. It was trying to do everything in one clock cycle, and wasn't giving enough time for the check/ compare process to happen. To prevent this from happening we used a counter in the controller file to slow it down and prevent errors on the next clock cycle. Another bug that we encountered was that if a person pressed down twice by accident or intentionally there wasn't any feedback to let you know that it was happening. This usually happened in the compare phase, and it wouldn't allow display or sound to occur. We fixed this by changing the length of the counters to make sure it was more user friendly.

We coded our game very strategically. We did it in smaller parts then combined everything in the end. We started off by creating the code for the LED decoder and the buttons. Then, we built all of the counters and set them to an arbitrary number until we figured out which specific numbers worked well with our design. Then we decided to make a shell file which links the button and LED decoders and made sure that the two were directly wired to each other. Then we connected everything logically to the controller to test it out. Once that was done we decided to use the 7-seg display in order to help ourselves debug. We used this to make sure that the ID requested and game played matched up correctly. Once we solved the time discrepancies and bugs we added the extra features of our design which were the sound, lighting up the buttons when pressed before the game start and the display with specific instructions. Using the 7-seg display we modified it to display the word *start*, "-----" for waiting in between game display and button presses, and the level number. This was added to provide a more user friendly cue to press the buttons and a friendlier overall game. After that we added sound to the 4-LED decoder in doing so we had to debug the frequency to prevent the sound from cutting in and out. We also had to fix our debouncer to prevent premature exiting due to double clock.

3. Evaluation

Our design is very effective because it is version number three of our original design. We found a way to cut down two shift registers, and a block Rom, so we wouldn't change anything major with the design. We looked at how other groups did theirs and we still think ours is more concise and has the least room for unpredictable error. However, if we were to change something it would be making the game larger. While we have many randomized games, our game only goes up to level 50 and gets stuck there. We would also use different buttons. The buttons that we used were the ones on the Basys3 board and those were very finicky. We had to try many different measurements for the debouncer. Instead we would use the PMod four button array that were given to us. These buttons come debounced so our program wouldn't confuse one button press for two. And finally we would make our sound for each button sound more harmonic. It currently sounds too robotic and we would like to make them sound better.

4. Conclusions

I think we achieved all of our goals and more. We were able to finish our project early and present the earliest. This was mainly due to our great group dynamic and the fact that we invested many hours on this project. We had a working project that displayed a start, a waiting dash, the buttons all had their respective sound and LED color. They lit up and made sound even when the game hasn’t started.

One tip for the future groups considering this project would be to start early and not procrastinate. You don't know what bugs you might encounter and a simple bug can take you hours or days to correct. Also, spend a lot of time on your RTL design before the review. It was super helpful that we did that super early and thoroughly because we always had a point of reference to fall back on if we messed up. Also it is important that on the early stages you work together because it avoids confusing and both partners are on the same page when moving forward.

Our project underwent three major changes. These changes can be seen in Appendix B version : Original, Version 2, and Final. These changes were:

Original

The game Id counter was used as our main data collector. It would count up (simple), once the counting stops the data went to the block Rom, it was stored there, then the select game number was generated. Using the select game number was generated it was supposed to be used to call a game number id predone on MATLAB. This number was 50 bits. Then the number got parallelly loaded into a 2 shift register. The shift registers would register as needed until the game was done. It would reset after finishing every level. The process would repeat until the player was done. The rest of the game design remains the same.

Version 2

Then we changed the counter to be the game itself. It counted up linearly and we used a multiplexer to select the bits and display the game, but this presented a problem. After a certain point in the game it was giving us/ showing a bunch of “000”. This was problematic when trying to display the game.

Final Version

To solve the problem from above we used the XNOR combinational logic to get a more random looking game that had less repeating numbers. It fixed the “000” problems and provided (2^50) - 1 games available to play. It also got rid of the shift registers, and the Block Rom.

5. Acknowledgements

We would like to thank Professor Hansen and Dave Picard for all of their help throughout the project. They provided us with all the tools and materials necessary in order to complete the project as well as provide insight on each part of the project. For example, using a multiplexor instead of the use of two shift registers in order to read the data. A huge thanks to all of the TAs, and even more so to Zeke Baker and Matt Gardner, who helped us complete our projects with their suggestions, and set of fresh--but experienced--eyes.

We worked together at first during the planning stages in order to create an overview of the project, and to avoid confusion on the process we would take. Once we were finished creating our top level RTL and State Machine designs, we split up in order to create the individual components of the project.

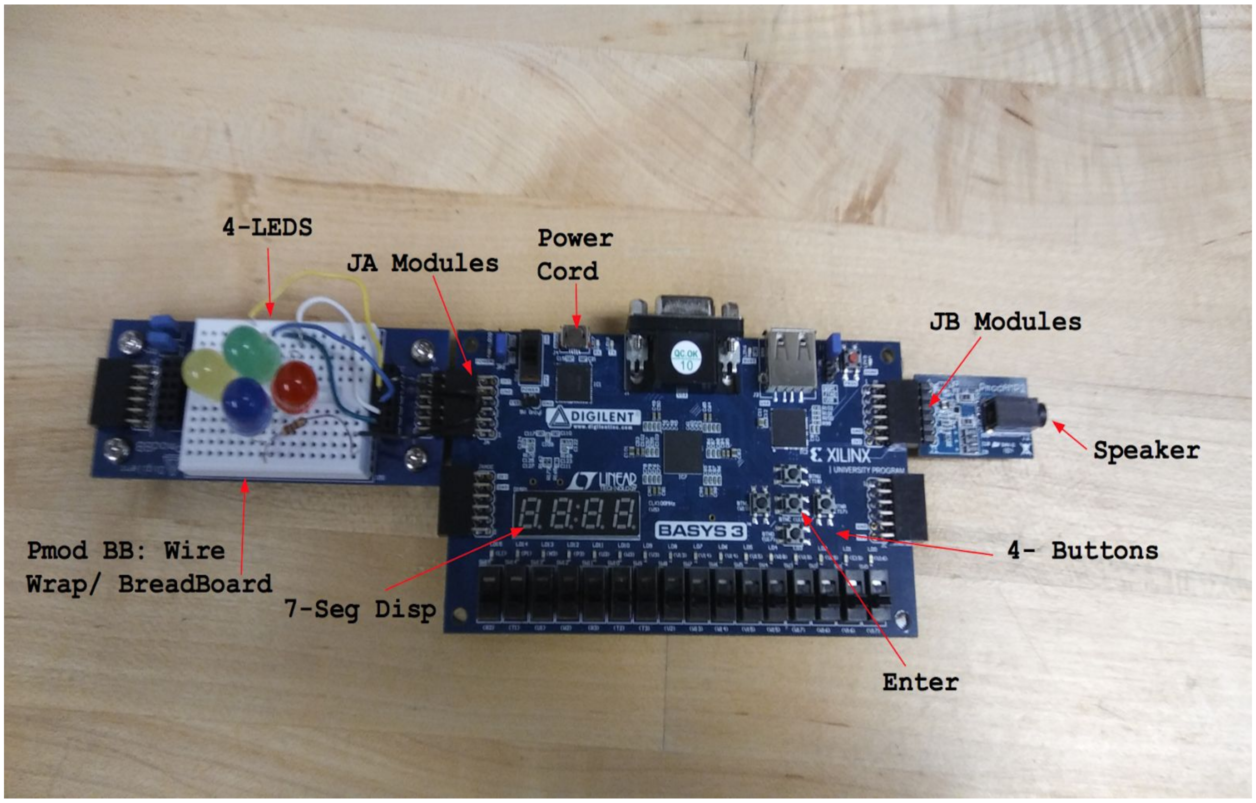
Carolina took care of the counters that were used for delaying, as well as creating and developing the game data we ended up using towards the end. Carolina also created the sound generators that were later implemented into the LED decoder. Kevin was in charge of coding the original LED decoder and button decoders, as well as writing down the logic for the state machine. We then met up to combine all of the files together as well as code the testbenches to debug together. Carolina took a large role in doing the physical testing of the game and assuring for a friendly user interface. Kevin commented the majority of the code. There was a relatively even split of work for the report. All throughout we met often, and did a constant back and forth of ideas, leading to a very smooth group dynamic that let us finish ahead of the curve and reach most of our stretch goals.

6. References

The references that we used were files from previous labs and notes. Especifically information regarding the mux7seg. The monopulser was taken directly from Lab 4. We also used the digilent website to gather specific information to guide us in programing the sound chip. We used Wikipedia and Xilinx as references on random number generation and for which notes to use for sound. You can access this information in Appendix I.

7. Appendices

7.1. Appendix A : Front Panel



JA Modules: Used to connect the LEDs

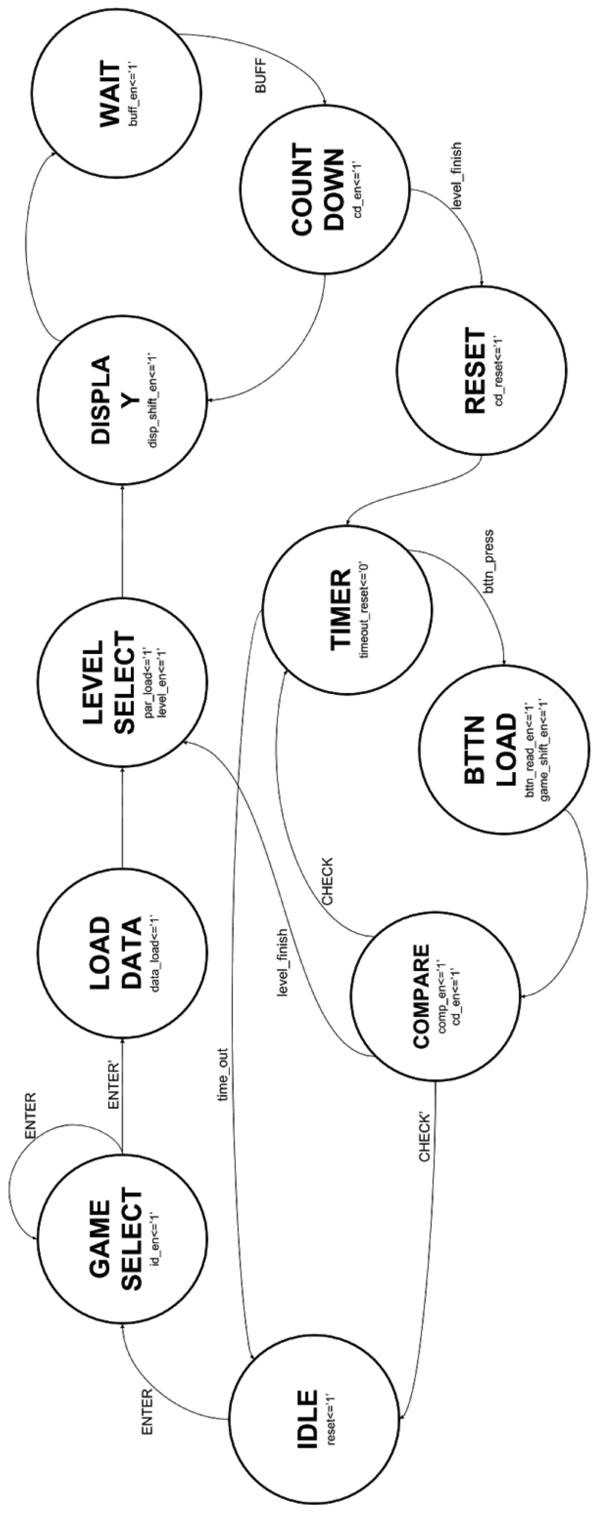
JB Modules: Used to connect the sound

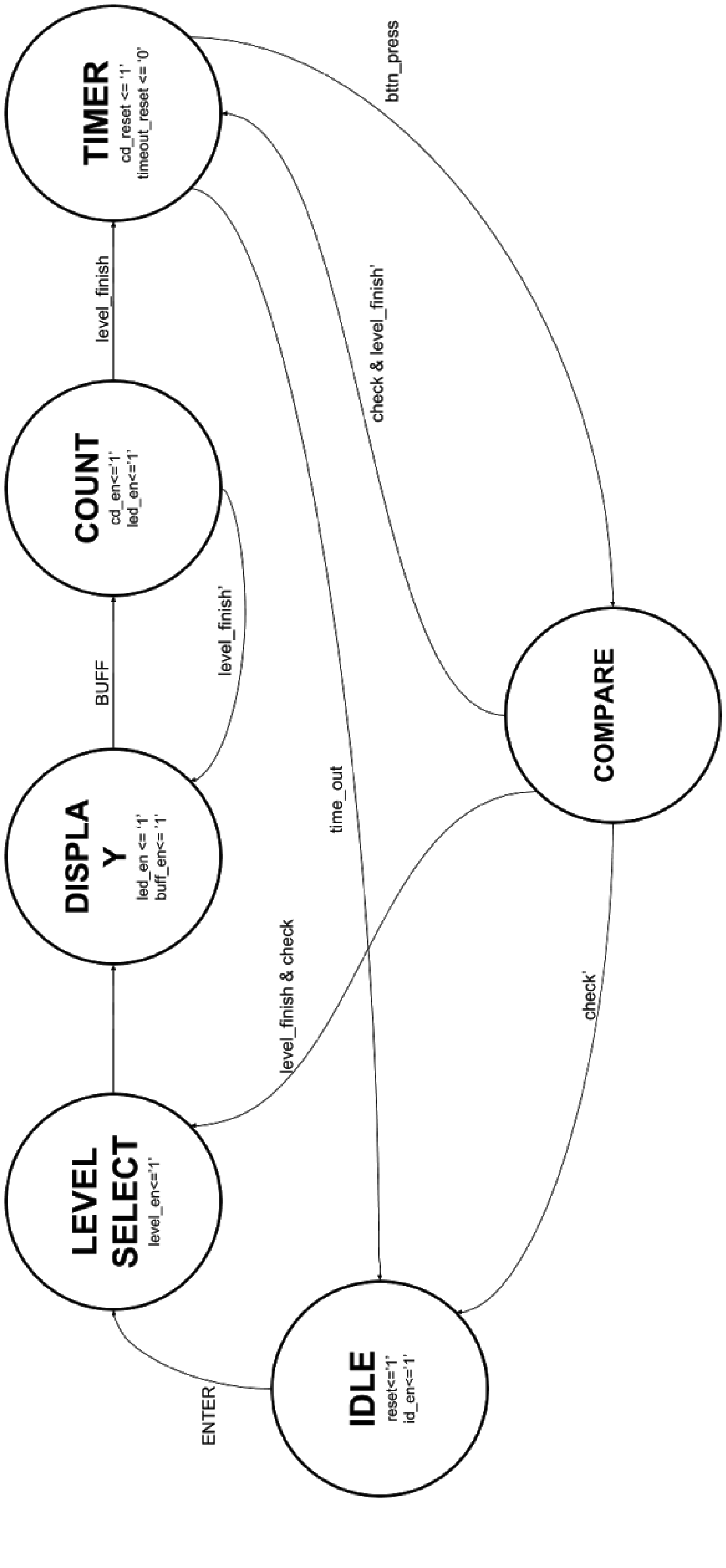
Enter: Starts the game

7-Seg Disp: Displays “Strt”, “----”, “L-0#”

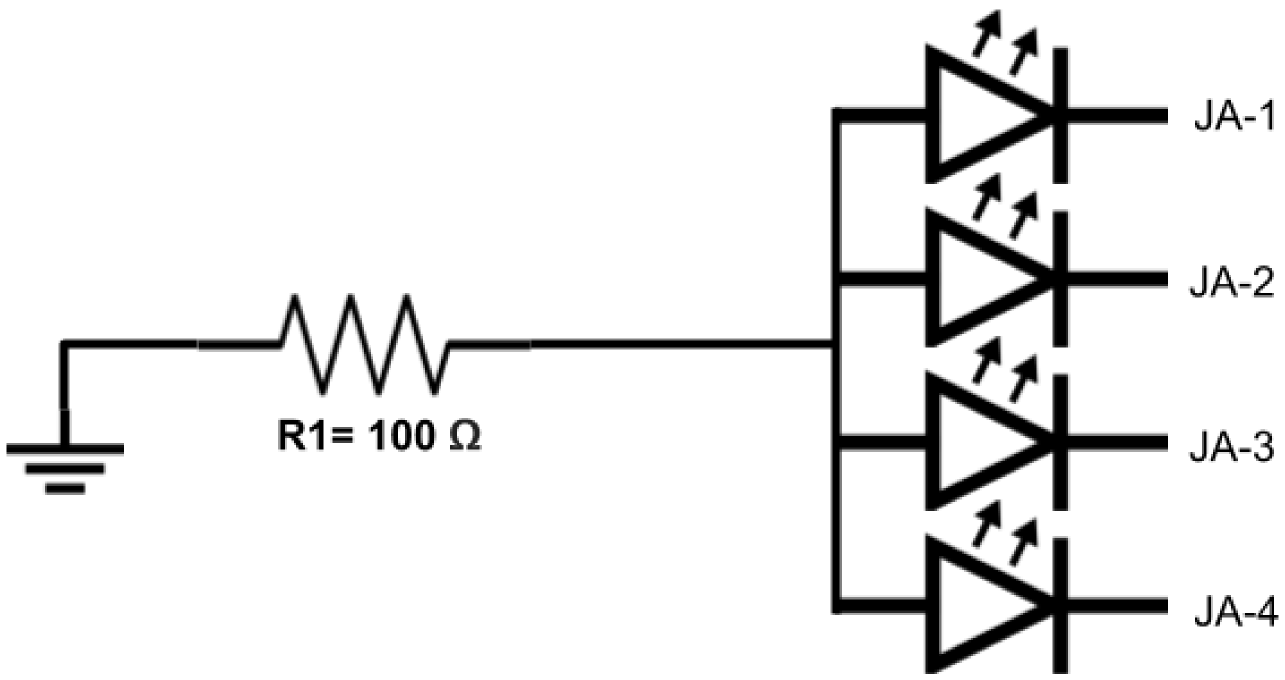
7.2. Appendix B : Block Diagrams

7.3. Appendix C : State Diagrams

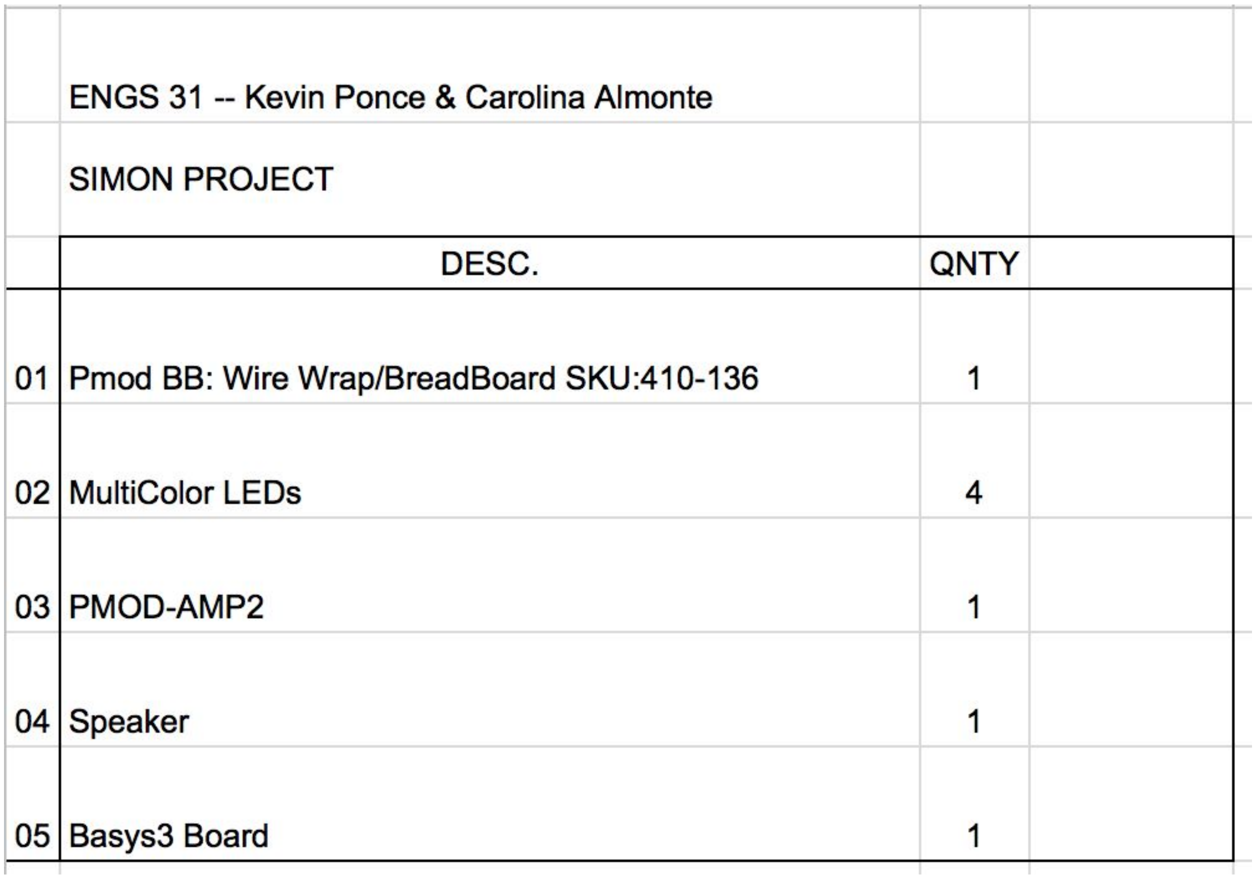



TIMER
cd_reset <= '1'
timeout_reset <= '0'
bttn_press
level_finish
check & level_finish'
COUNT
cd_en<='1'
led_en<='1'
level_finish'
BUFF
DISPLA
Y
led_en <= '1'
buff_en<= '1'
time_out
COMPARE
level_finish & check
check'
LEVEL
SELECT
level_en<='1'
IDLE
reset<='1'
id_en<='1'
ENTER

7.4. Appendix D: Circuit Diagram



7.5. Appendix E: Parts List

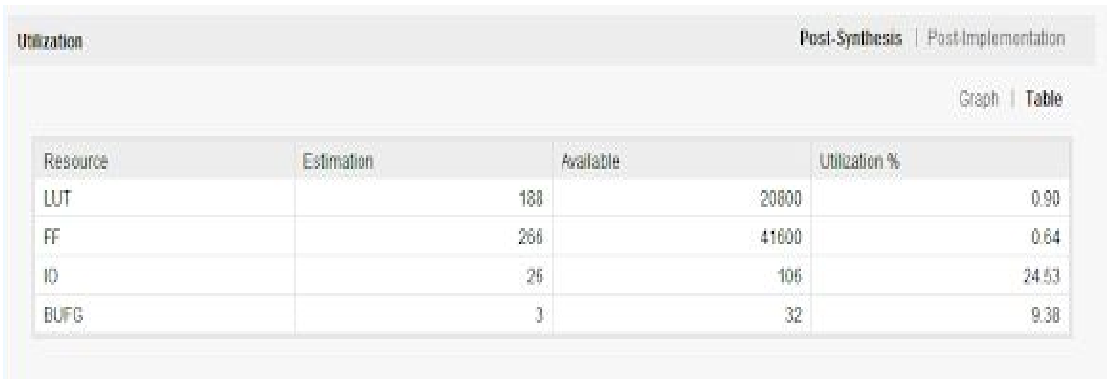


ENGS 31 -- Kevin Ponce & Carolina Almonte

SIMON PROJECT

	DESC.	QNTY
01	Pmod BB: Wire Wrap/BreadBoard SKU:410-136	1
02	MultiColor LEDs	4
03	PMOD-AMP2	1
04	Speaker	1
05	Basys3 Board	1

7.6. Appendix F: Resource Utilization


Utilization

Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	188	20800	0.90
FF	266	41600	0.64
IO	26	106	24.53
BUFG	3	32	9.38

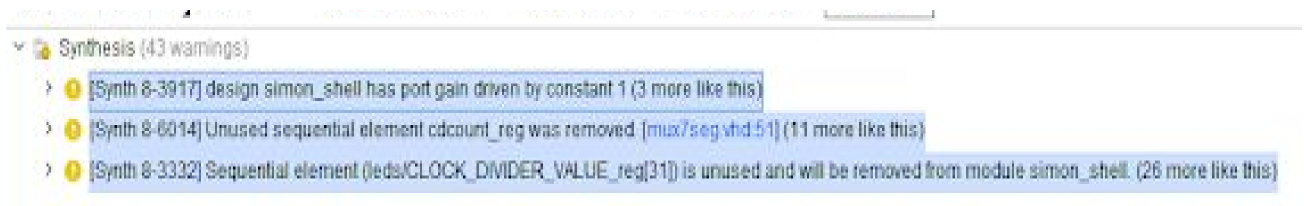
```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     3|
|2     |CARRY4 |    52|
|3     |LUT1   |    27|
|4     |LUT2   |    17|
|5     |LUT3   |     9|
|6     |LUT4   |    62|
|7     |LUT5   |    56|
|8     |LUT6   |    49|
|9     |MUXF7  |     7|
|10    |MUXF8  |     1|
|11    |FDRE   |   265|
|12    |FDSE   |     1|
|13    |IBUF   |     6|
|14    |OBUF   |    20|
+------+-------+------+
Report Instance Areas:
+------+-----------------+-----------------+------+
|      |Instance         |Module           |Cells |
+------+-----------------+-----------------+------+
|1     |top              |                 |   575|
|2     |  bttns          |bttn_decoder     |    20|
|3     |  controller     |simon_controller |   123|
|4     |  display_buffer |disbuff_count    |    50|
|5     |  game_id        |GameID_count     |    81|
|6     |  leds           |led_decoder      |    89|
|7     |  level_counter  |level_count      |    27|
|8     |  mux            |mux7seg          |    39|
|9     |  step           |step_count       |    31|
|10    |  timer          |timeout_count    |    48|
+------+-----------------+-----------------+------+

Synthesis finished with 0 errors, 0 critical warnings and 41 warnings.
97 Infos, 43 Warnings, 0 Critical Warnings and 0 Errors encountered.
```

7.7. Appendix G : Critical Timing

Our minimum clock period of our design is 7.279ns, with the max clock speed being 137.4 MHz, which is faster than the clock speed of the FPGA (at 100 MHz). Majority of the delay is due to the pulse width modulation for the sound generator, which accounted for 4.5ns, or 62%. The rest was accounted for hold and set-up times for the LUTs.

7.8. Appendix H: Residual Warnings

We had many residual warning but they have no effect the implementation of our design in hardware. They are just there to let us know that we didn't use a few things and that there are recommended changes that we can do.



```
Synthesis (43 warnings)
  [Synth 8-3917] design simon_shell has port gain driven by constant 1 (3 more like this)
  [Synth 8-6014] Unused sequential element cdcount_reg was removed. [mux7seg.vhd:51] (11 more like this)
  [Synth 8-3332] Sequential element (leds/CLOCK_DIVIDER_VALUE_reg[31]) is unused and will be removed from module simon_shell. (26 more like this)
```

In the image:

*First one is okay since we want a constant signal for the gain and shut off.

*Last ones are vivado optimizations.

7.9. Appendix I : Xilinx Page

7.10. Appendix J: Waveforms

7.11. Appendix K: VHDL Code

7.12. Appendix L: VHDL Testbench Code