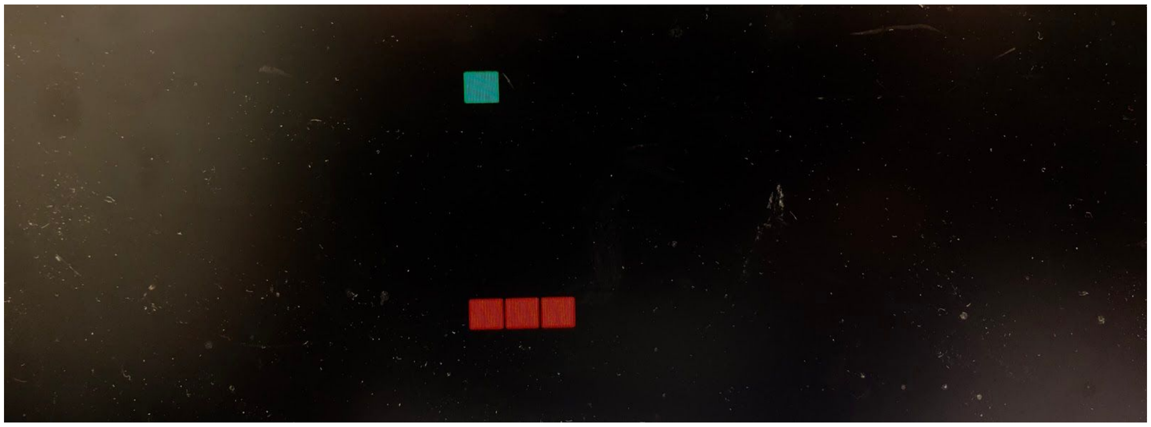
CSM152A - Final Project Report

Group Members:
Grant Roberts, 405-180-937
Ryan Brennan, 605-347-597
Yu Gao, 804-957-843

Section 5
TA: Kuo
Date Demoed: 3/12/2020

1) Introduction

For our final project we recreated the popular Nokia cell phone game "Snake". It is a simple game where a player controls a snake object on a screen with 4 movements(up,down,right,left). The object of the game is to get as many apples as possible without running into itself. The apple will spawn randomly and the snake can move through the borders of the walls. Each time the snake eats an apple, its length grows by one block. This piece of logic was actually one of the harder pieces of the project to implement as dynamic resizing is difficult in Verilog. We also allowed for the user to control the movements of the snake through the Digilent joystick add-on for the Nexys3 FPGA board. We implemented the graphics using the VGA connection on the FPGA.



2) Design Requirements

2.1) Random Number Generator(25%) -- Completed

We needed a random number generator for placing the apple at a random place on the screen. This we did by using a pseudo-random number generator. We found we had to take the modulo-10 of this random number so that the apple would always appear in the exact line of the snake. As the snake body was 10x10 pixels we had to be sure to place the apple on that size grid. This was to allow for getter collision detection with the head of the snake.

2.2) VGA Graphics(25%) -- Completed

The VGA graphics were the first thing we worked to implement as our entire game would rely on our ability to get the VGA graphics working. This took us about two full days to implement and we needed to learn the basics of VGA to get it working. We also found the monitor we were working on was slightly clipping on the left side of the screen, so we shifted things to the right.

2.3) Snake Growth(15%) -- Completed

Our original idea to complete the snake growth was to think about it like a linked list, however Verilog doesn't allow for dynamic allocation. So we decided to create an array of fixed length size with a flag bit to allow for that body part to either be on or off. So technically the snake is always at full length, however its nth body part will only show up if it has eaten n apples. This logic took a while to figure out and we also encountered a problem of the snake growing to max length upon eating the first apple. We had to bring the growth outside of the clock always statement so it would only grow by one piece each clock cycle. We did the growth by creating a register that contains all the body parts and would resize it based on this.

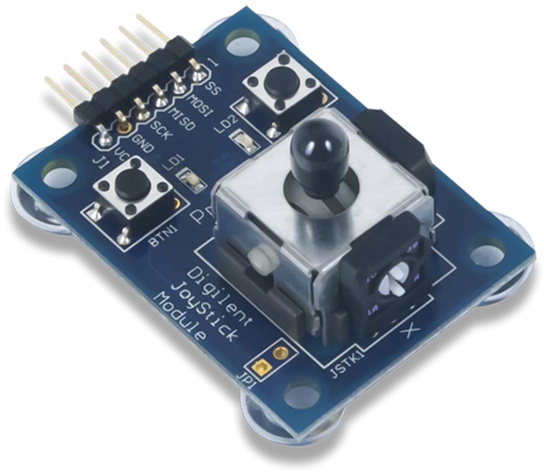
2.4) Snake Movement(10%) -- Completed

Snake movement was accomplished by assigning the snake head an (x,y) position on the screen. Each clock cycle the snake would move in a direction by one block of size 10. The direction was decided by the user. If no new direction was chosen, the snake would simply stay in the current direction. So we had the direction only update/change upon movement from the user. We decided to go with the "Snake 2" movement implementation where the snake can pass through the walls of the screen. So if the snake is going to run into the top of the screen, the snake will appear as if it is coming out of the bottom of the screen. We also had to implement collision detection for both the snake and the apple. If the snake runs into itself, the game is over and we did this by checking if any part of the body was both on and at the same (x,y) position as the head of the snake. The game also needed a way to tell if the snake head was about to run into an apple. We used the same logic for the collision of the body for the apple.

The other tricky part of the movement was having the rest of the snake body follow the snake head. We originally thought of it doing it as a for loop and having the first element update its new position, then having the movements cascade down to the others. However we weren't able to implement a for loop in Verilog in an always loop. So we would just remove the last piece of the body and "add" a new piece to the head to make it appear as if all the pieces were following.

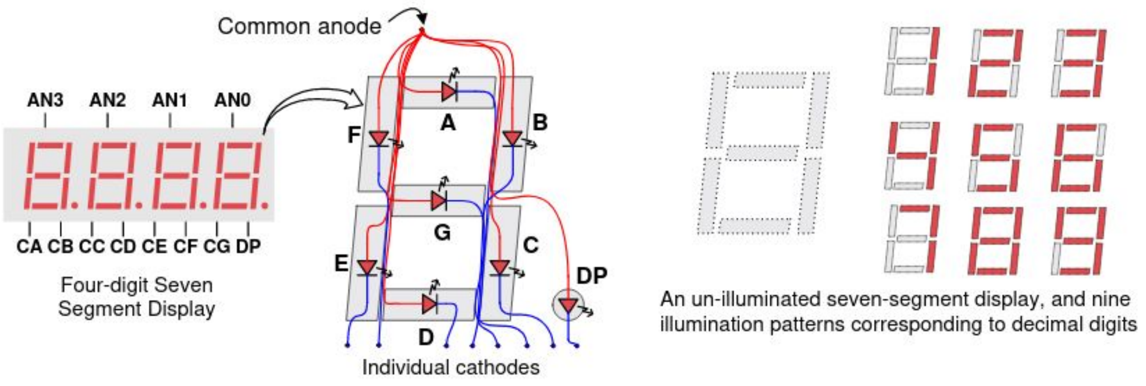
2.5) Joystick implementation(10%) -- Completed

We implemented the joystick by going to the Digilent JSTK website and following the manual/code tutorials they provided[1]. We took their implementation of having the potentiometer numbers appearing on the 7-seg display and ran it on our FPGA board. We did this to get a baseline of the numbers the joystick reads so we can have thresholds for up, down, right, left on our snake movement. We found this to be a difficult exercise in learning how to take somebody else's code and rework it to fit our needs. Once we were able to extract the readings from the joystick, we replaced our if-else statement cases that were currently relying on the boards switches and dropped in the readings from the joystick.



2.6) Score Counter(10%) -- Completed

One of the smaller components of the game was to keep a counter on the 7-segment display for each time an apple is eaten. We borrowed the logic for the 7-segment display from lab 3 and implemented it in our game. We would simply increment the counter each time an apple was eaten.



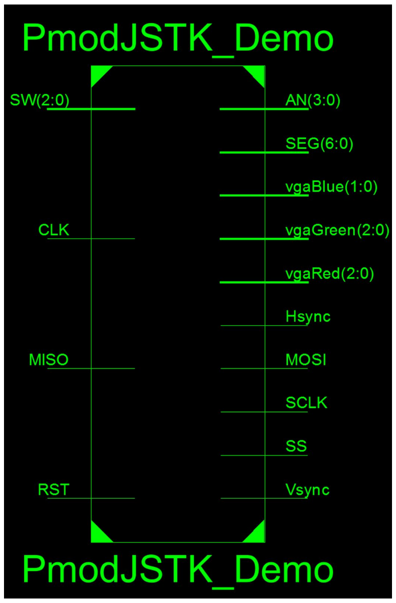
2.7) Food placement(5%)

This part was relatively simple once we had our random number generator. We passed the random number we generated for both an x and y component and gave it to the apple. This would then place the apple at a location on the VGA. Once the apple had been eaten we would run the random number generator again and place the apple again.

3) RTL Schematic

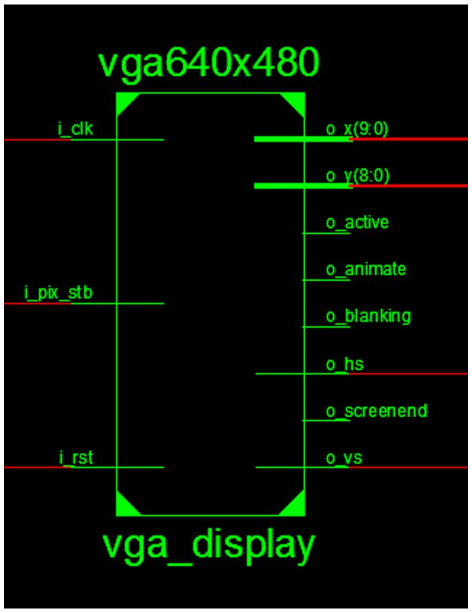
3.1)Top Module

The top module schematic is shown below. We used 3 switches to control what content was shown in the seven segment display, and one button for reset. SPI pins MOSI, MISO, SS, SCLK for communication between board and joystick.



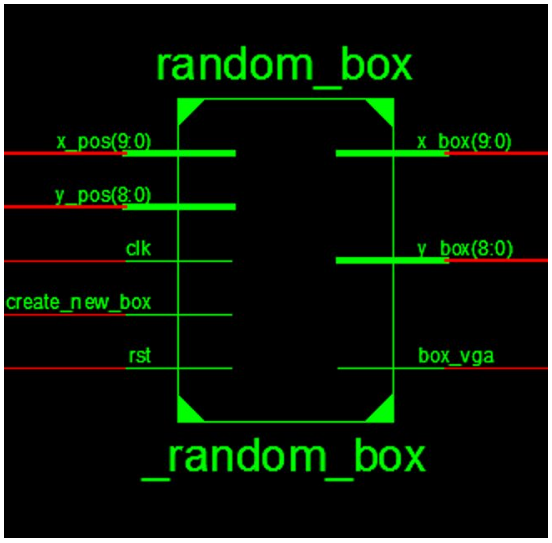
3.2) VGA display module

The VGA display module schematic is shown below. It takes a board clock and reset button as input, and outputs the x and y counter, which indicates where the VGA counter is.



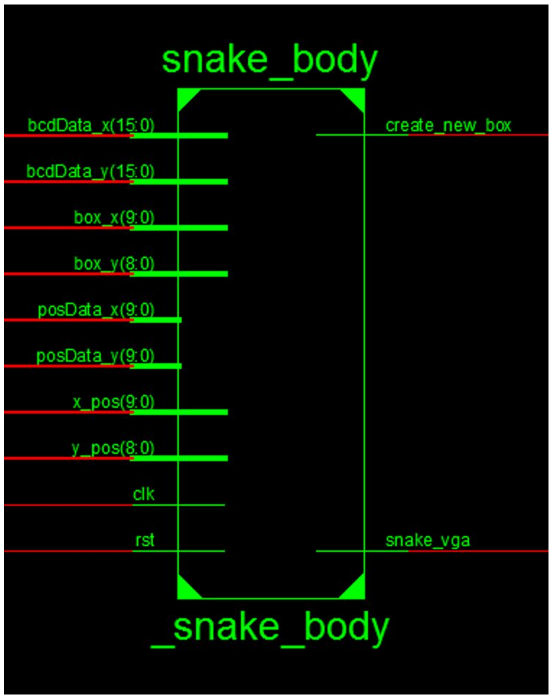
3.3) Random box generator module

The random box module schematic is shown below. It takes the x and y counter from the VGA module, and output box_vga to indicate when to display the box during the VGA screening process. It also takes the input "create_new_box" signal, which indicate the time to generate two random numbers.



3.4) Snake body module

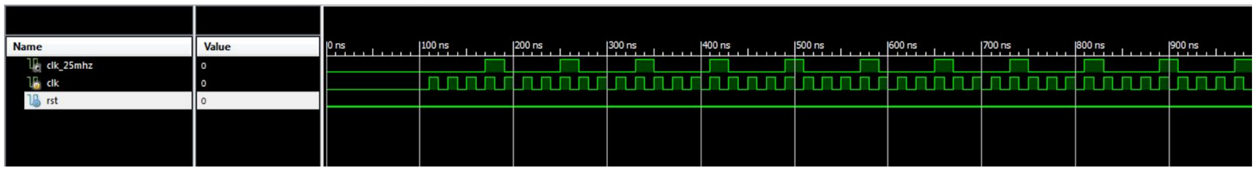
The snake body module schematic is shown below. The snake body module is similar to a random box module, except the bcdData input, which are the data read from a joystick. bcdData_x and bcdData_y is the control signal for turn snake to left, right, up and down respectively.



4) Testing

4.1) VGA clock requirement

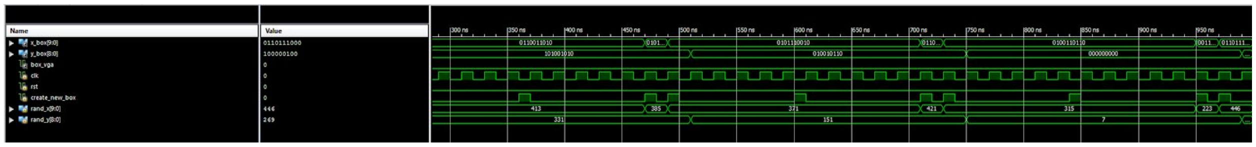
Since we wanted to create 480x640 60Hz VGA display, the needed clock frequency is 25MHz. Here is the screenshot from the clock divider module. Since the clock on the board is default 100MHz, there is one tick every four clock cycle.



Name	Value
clk_25mhz	0
clk	0
rst	0

4.2) Random box generator

As shown in the screenshot, as the signal "create_new_box" becomes high, two random numbers are generated. These two numbers are corresponding to the x and y location of the new box.



Name	Value
x_box[9:0]	0110111000
y_box[8:0]	100000100
box_vga	0
clk	0
rst	0
create_new_box	0
rand_x[9:0]	446
rand_y[8:0]	269

5) Conclusion

The snake project was a great way not only apply a little from something we learned in each lab so far, but also to add new knowledge such as the VGA and the joystick. The joystick was a great way of learning how to use add-on modules for the FPGA board. It was also a good exercise in reading other people's Verilog code and trying to get it to adapt to our needs. The VGA was similar as we followed a tutorial on getting it setup and then adjusted it for our game. The score counter used knowledge from lab 3 and we essentially just dropped in the 7-segment code from that lab into our project. We had to use states for the snake game and we were introduced to states in lab2. We didn't incorporate much from lab1 outside of general Verilog knowledge.

We faced several challenges such as completing everything on time. We ended up having to go to office hours to finish the joystick implementation. The VGA graphics took longer than we anticipated to get working so we found ourselves against the clock early on. We got caught up by doing some of the work from home and using the class hours for testing and debugging the code we wrote at home as we didn't have access to the FPGA board outside of class. In the future we think a game with more pieces interacting would be fun such as adding obstacles for the snake in the game or small power ups that make the snake move faster. In general we are happy with our implementation and feel that it encompasses a lot of what we have learned in this class and a few things we learned on our own.

References:

1. https://reference.digilentinc.com/reference/pmod/pmodjstk/reference-manual
2. https://timetoexplore.net/blog/arty-fpga-vga-verilog-01#footnote-25mhz
3. https://reference.digilentinc.com/pmod/pmod/jstk/example_code