**Embedded Systems**

**Project:** FPGsnAke

**Report on Milestone:** 2

**Student Name:** Michail Ouroutzoglou (2014030063)

**Date:** 10 / 12 / 2017

**Short revision on what is finished and future plans:**

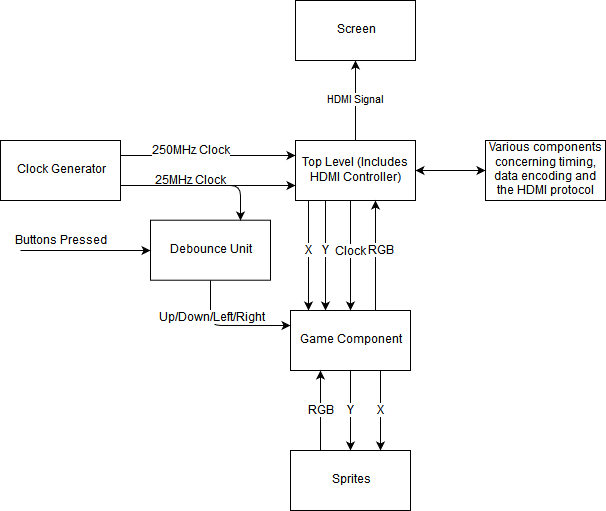
In the first milestone, we studied Verilog (since we only knew VHDL) and created some test code to practice. After learning the basics, we started testing the code that Xilinx provides us with to drive the HDMI output of the FPGA. We implemented our first sprites (using a python script) and added some linear movement on the sprites to test the control we had on the output. Finally, we experimented with counters to enable the triggering of events whenever a set amount of cycles pass (for example the movement of the sprites on screen).

In the second milestone, the plan was to properly handle the input from the push buttons on the FPGA in order to move the snake. We also need to count the score as well as being able to increase the length of the snake and draw it bigger every time it consumes an apple. Finally, add some logic to the circuit to detect if the snake has collided with the brick walls.

In the third milestone, we have to cover all the remaining aspects of the game which are detecting when the snake bit itself, changing the position of the apple every time it gets eaten, displaying the score on the screen and having a main menu and a game over screen.

**Elaboration on the second milestone’s goals and how they were achieved:**

First, we are going to take a look at a block diagram that describes the basic idea of the circuit thus far.



We have the Clock Generator module which provides the circuit with two clocks, the 250MHz one is needed for driving the HDMI port, while the 25MHz is the pixel clock.

Then, we have the Top-Level module which communicates with all the other components and most importantly the components that do all the work for the HDMI protocol. The top-level module talks with our game component by exchanging messages in the form of (X, Y) coordinates and (R, G, B) values.

The game component handles has all the logic required to handle it’s inputs, the movement of the objects on the screen, the collision detection e.t.c.

The game component also takes input by the debounce unit which is responsible for handling the input.

Finally, the game component talks with our sprite bank, where for every input (X, Y) an (R, G, B) value is returned back.

Now we are going to go through all the details of each functionality added in the second milestone.

* Input handling.

The input is taken from the four push buttons placed on the bottom left corner of the FPGA. The signal is then taken through a debounce unit, where the signal’s glitches are removed

(by glitches we mean the fluctuation of the signal until it stabilizes). Then, we forward the debounced signal to the game component where it is handled for the snaked movement.

The debounce unit works by remembering the previous state of the input signal, and starts counting cycles if it changes. So, for example when it changes from “0” to “1”, it counts 2^k cycles, and if it’s stable to “1” throughout all these 2^k cycles it then the output of the unit becomes “1”, otherwise it remains “0”. There is also, extra logic in the debounce unit, that outputs “1” for one cycle right after the 2^k cycles, and also outputs “1” for one cycle after 2^k from when the button is released. This is really useful for our case, since we want to get the input for one cycle, that is to prevent the snake from turning around uncontrollably.

* Snake movement.

After getting the debounced inputs (four of them: up, down, left, right), we use them in a simple FSM, where depending where the snake is looking, the input is either valid or invalid, and changes direction accordingly. What that means is that, if it is looking up or down, we can’t accept as input the buttons up or down, the same applies for left and right. That’s how we control the movement of the head. The rest of the body must follow the head’s movement. The way we solved this problem was by having a register for each part of the body and each time the head changes, the values are propagated from top to bottom.

* Snake growth\Score keeping.

We set up a counter that increases its value every time the snake is about to move in the position where an apple is located. This is the score, but it also is the length of the snake’s body. Since we already store the position for every part of the snake, we just need to use this counter as a way to know up to which depth should we search for when drawing the snake (or when it bit itself, but that’s in the next milestone).

* Drawing the snake.

In every frame, the game component is given (X, Y) coordinates and is asked by the HDMI component to return an RGB triplet. Thus, the game component must first determine what is to be drawn on that (X, Y) pixel. The way we solved it is by checking each and every part of the snake (up to the depth that the score counter limits us) and if any part of the snake is on that pixel then we return the appropriate RGB value.

* Collision detection with the brick walls.

This is simply solved by checking if the next position of the head is on the borders of the grid. If it is, then, for the time being, the snake just loops around the grid. Later, we can show a game over screen when the collision is detected.

The end.