**Report On Milestone:** 1

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

**Date:** 19 / 10 / 2017

**Project:** FPGsnAke

For the first milestone, I had to connect the FPGA board to the monitor using the HDMI cable. My first source of knowledge came from Xilinx’s [documentation](https://www.xilinx.com/support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf), where I studied about the HDMI protocol.

Then, I downloaded the demos present in this [link](https://secure.xilinx.com/webreg/clickthrough.do?cid=154258) (from inside the above pdf file). One demo (dvi\_demo.v) accepts an input and forwards it to an output. The other demo (vtc\_demo.v) tests the FPGA’s connectivity to the screen using various modes (VGA, SVGA etc.). For my purposes the second demo is ideal to use and build upon as a base for my snake game project.

Due to the fact that all the above code was in Verilog and the fact that I wanted to be able to tinker with the said code to build my game upon, I had two choices. I either had to translate the whole from Verilog to VHDL or I had to learn Verilog. Considering my limited deadline, I thought I couldn’t afford learning a new language, so I had to start translating the existing code as soon as possible. This process took me 2 days and after various failed efforts I gave up on the idea of translating the code from Verilog to VHDL. That left me with my last option, to learn Verilog.

I started out by studying tutorials on the internet and building small projects to try out things I’ve learned. Namely, I built, a two-bit comparator, a d flip flop, an n-bit d flip flop using the d flip flop as a component, a one-bit adder, then an n-bit adder, an n-bit shift register, a Moore FSM, a Mealy FSM, a debounce unit and a four-bit multiplier using the booth algorithm.

After I was finished with the above, I was feeling quite competent with taking up the process of editing the demo code.

For VGA dimensions (640x480) a 25MHz clock is needed to drive the pixels. The HDMI protocol though says that the information is sent using 4 differential outputs (8 outputs: red+, red-, green+, green-, blue+, blue-, clk+, clk-). In addition to that the RGB data is encoded so as instead of sending 24 bits (8 bits each) I need to send 30-bits (10 bits each), thus I need a clock at 250MHz to transmit all that data during one cycle of the 25MHz clock.  
(Notice that the clk+ and clk- are referring to the 250MHz clock)

The way the “painting” works, is by having two counters, one for the X position and one for the Y position on the screen. These two are passed to a module, alongside to the pixel clock (25MHz), that is responsible for returning an 8-bit triplet (RGB) and then this data is transmitted as noted above.

I removed the demo module that was responsible for the above and replaced it with my code. The first thing I tried to do was to “paint” a white rectangle. That was fairly easy as all I had to do was check if the (X, Y) position was in specific positions and if it was I returned   
RGB = xFFFFFF otherwise I returned x000000. Then, I tried to change the colors of the rectangle using the location of the position vector (X, Y). For example, R = X [7:0], G = Y [7:0] and B = {X [7:4], Y [3:0]}. After a couple more successful efforts, I wanted to do something more useful for my project. I needed to be able to draw images.  
  
In order to tackle this problem I needed to be able to store the image data in the FPGA with an easy and fast way. What I came up with was to build a rom file for each image that can take as input a X and Y position and can return me the RGB data. Turns out this is easily accomplished with a case statement where for each {Y, X} a value is assigned to the output of the rom. The only remaining problem was to find a way to create these rom files out of images. This is where, my knowledge on, Python came useful as I used it to read the data of images, rescale them down to 32x32 (or whichever dimensions I wanted) and generate a Verilog file, completely dynamically and reliably.

So, I am now able to load up images on the FPGA and draw them on the screen on a specific point by defining a x\_pos and a y\_pos signal for each of the images, and then checking with a big if statement which image is responsible for each pixel on the screen, if any, to get the color from that image.

Finally, I wanted to add some basic movement on the images, just to get the gist of it. In order to do that implemented a basic counter that would count up frames and move an apple on the screen by one pixel horizontally 32 time per second. That means, that I needed to count up to 25.000.00 / 32 = 781.250 on the counter before changing the x\_pos of the image. This worked really good and I actually timed the time it took for the apple to get from the left side of the screen to the right side and it was 20 seconds, that is, it moved 640 pixels on the right which is equal to the width of the screen, which is brilliant. Finally, I added some movement on the vertical axis just to test it out, which good worked as well.  
  
  
No external sources were used throughout the above (except from the HDMI interface code and the tutorials on basic Verilog spread all over the internet).

All the code I created can be downloaded from [here](https://mega.nz/#!mxdwGIpY!YX3moMPF9iLOOM-t9c7Ijpy_o7CF_pCRXE_x-se569Q).

The end.