Abdulkareem Alqahtani

Eyad Esmail

Khalid Almanae

Dr. Christopher Martinez

System on Chips

Lab Report for Labs 1 – 5

DEC. 2 2018

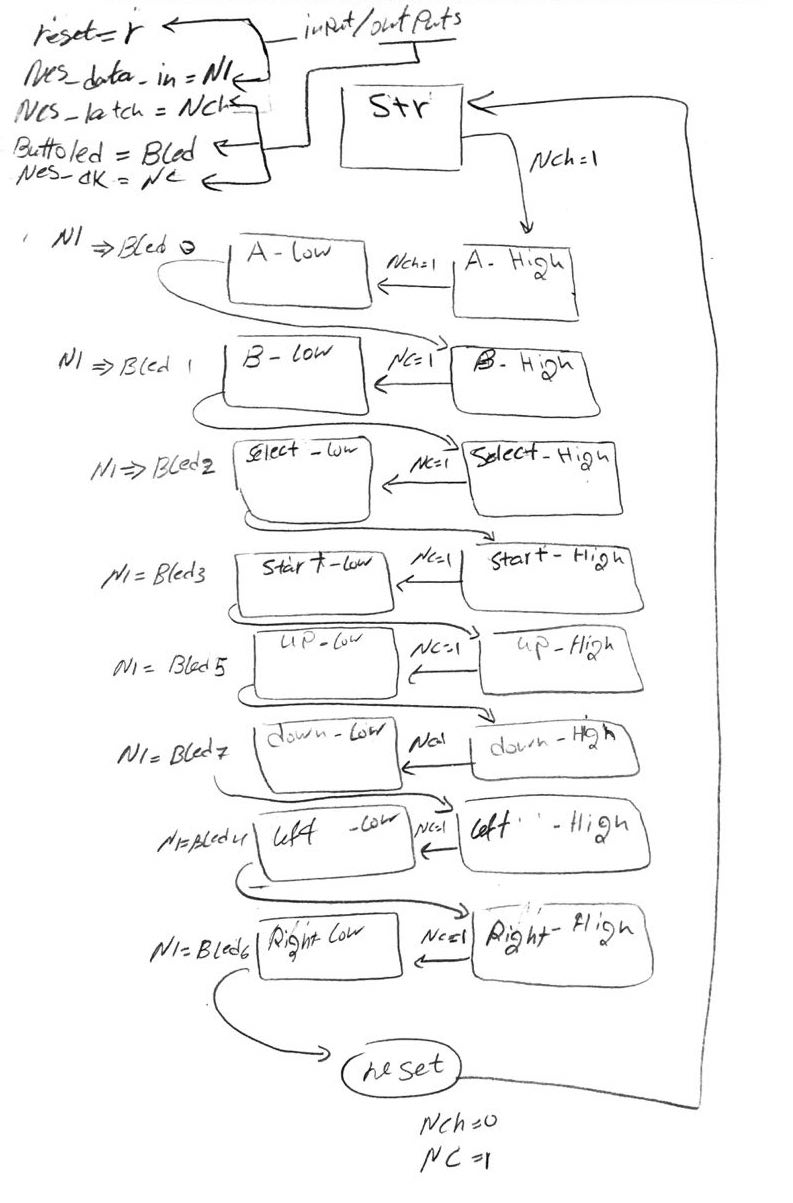
In the SOC class, we were required to develop an NES game using our knowledge from previous classes. The entire project is consisting of 8 different labs. In this project, the hardware we will be using are the Diligent Spartan-3E Starter and an NES controller. This paper will focus on the first part of the project (lab 1-5). Lab 1 is centered on the NES controller, functionality and implementation. We need to develop a VHDL code with a clock divider and a method to read how the controller responds to button press. In lab two, we started to work on the visual part of the project. By creating tile-maps using VHDL, we were able to display one color in each segment of the screen (each segment is 8X8 pixels). In lab3, we used the tile-maps we created in lab two and we added sprites that can be placed into the design. We created a sprite-map that supports 32 sprites. After that, we made the game characters that were split between more than one sprites as required. Next, we figured out the downsizing, reading sprites, and placing pixels equations. At this stage, we have two maps a foreground tile-map, and a background tile-map. In Lab 5, we created IP cores for the already done controller, the graphics. We also created a RAM to be able to interact between the cores and codes. The controller was dropped into our system with some minor change. The graphics processor required removing the written tile and sprite data, keeping the declaration only. Both the graphic and controller files required modifications to the user logic and the IP core file.

Lab #1 – NES Controller

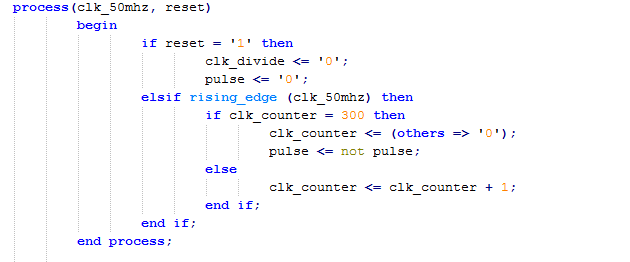
Description of the lab – inputs/outputs, explanation of timing for the controller:

We created the VHDL code to generate the signals and read the buttons press. The controller is a digital only controller with 8 buttons. The NES pin out are layout as the following: pin 1 is the ground, pin 2 is pulse (in), pin 3 is latch (in), pin 4 is data (out) then pin 7 is the power for the controller. The latch is responsible to let the controller know the status of the bottom (ether the bottom is pressed to not). The latch is a 12us high signal. The job of the pulse is to move the buttons through the data out. Pulse is 12us pre-cycle. The board has a 50 MHz clock. We needed to get 6us out of it. It was divided by a factor of 300 using a code that will be shown later. The buttons are read after each pulse cycle. The main code was written under the state machine idea. We created several states, two for each button, one reading the button and one writing to the LEDs on the FPGA.

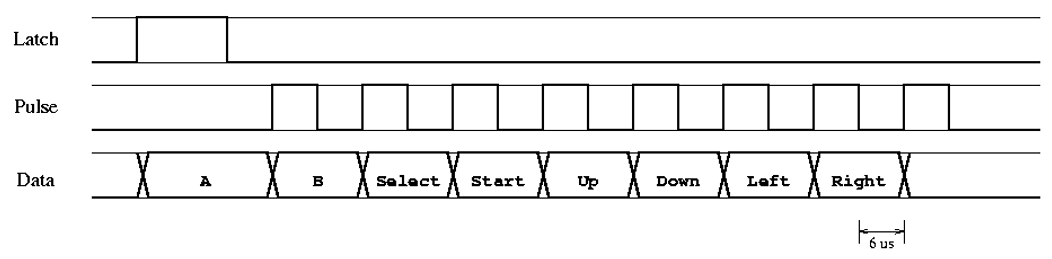
The state machine:

****

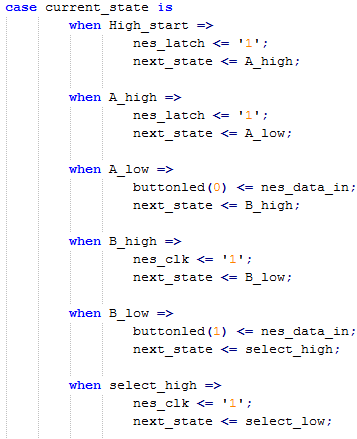
Code explanation:

* Clock Divider:

What the clock divider does is basically creating a delay on the main clock until a certain value is reached and then it increments. The FPGA clock is 50 MHz. we needed a 6 microsecond clock to work with the 12 microsecond cycle needed for the pulse and latch. The state machine and the clock divider were built based on the understanding of the following diagram.



* State machine:

The picture shown here is a snippet of the state machine code. The data requires a longer time than all other buttons. To do that, we created an extra state to wait. Then for all other states, there is a state to set the nes\_clk so we can read a signal and then the next state sets the button led to read the data in from the nes.

Sizing of the chip. (Will be shown on a separate scanned document)

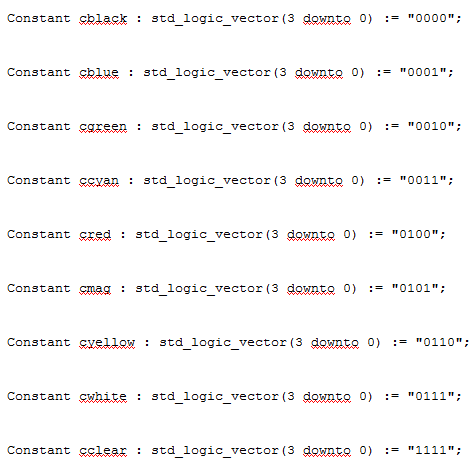
Lab #2&3 – Graphics

Description of the lab – inputs/outputs, explanation of timing for the VGA:

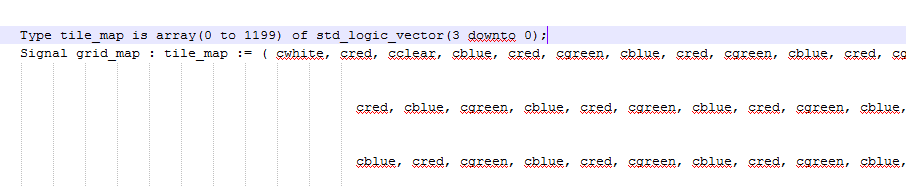
For labs two and three, we needed to create the graphics part for the project. We will use the VGA cable to connect the Spartan board to a monitor. We needed a method to downsize the graphics so that it can fit on the board’s memory. The original size is 640 x 480. We needed to downsize the resolution to 320 x 240. We will create a 40.30 tile map to access segments of the screen. Due to the downsizing, each segment is 8 x 8 pixels.

Code with explanation:

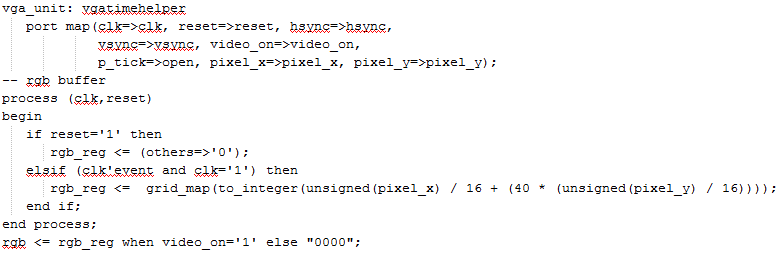
Lab 2:

We created our pallet using the code segment shown in the figure 1. Each color is assigned a 4 bit number.

Next we wrote out tile map, which is an array of 1200 elements.

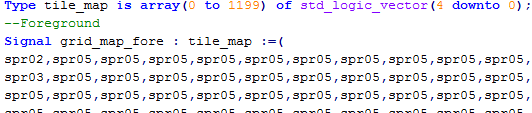


We then used the following code, and a code provided to us that deals with the timer and when the data is drawn to the screen. On the rising edge of the pixel clock, we send the grid map value of pixelx and pixely with its downsizing (division) properties.

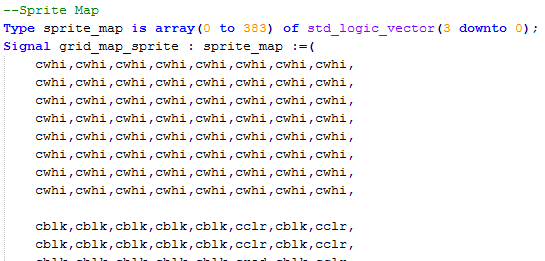


Lab 3:

For lab 3, we created our sprites and added them to the code. Now each pixel can hold a sprite which holds an 8 x 8 amount of different pixels containing our character parts. We created two different tile maps; background and foreground. The tile map now hold sprite numbers rather than colors.



Next, we created our sprite maps. Each sprite was created using excel. We created the shape by highlighting cells and then converting everything to names. We have 32 sprites all held in an array.



We used the following formulas to link the sprites to the tile maps.

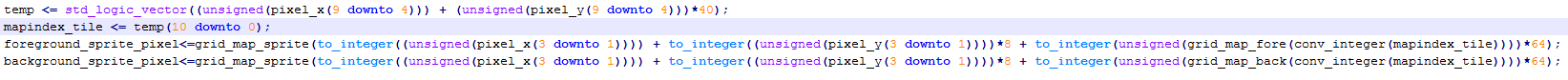
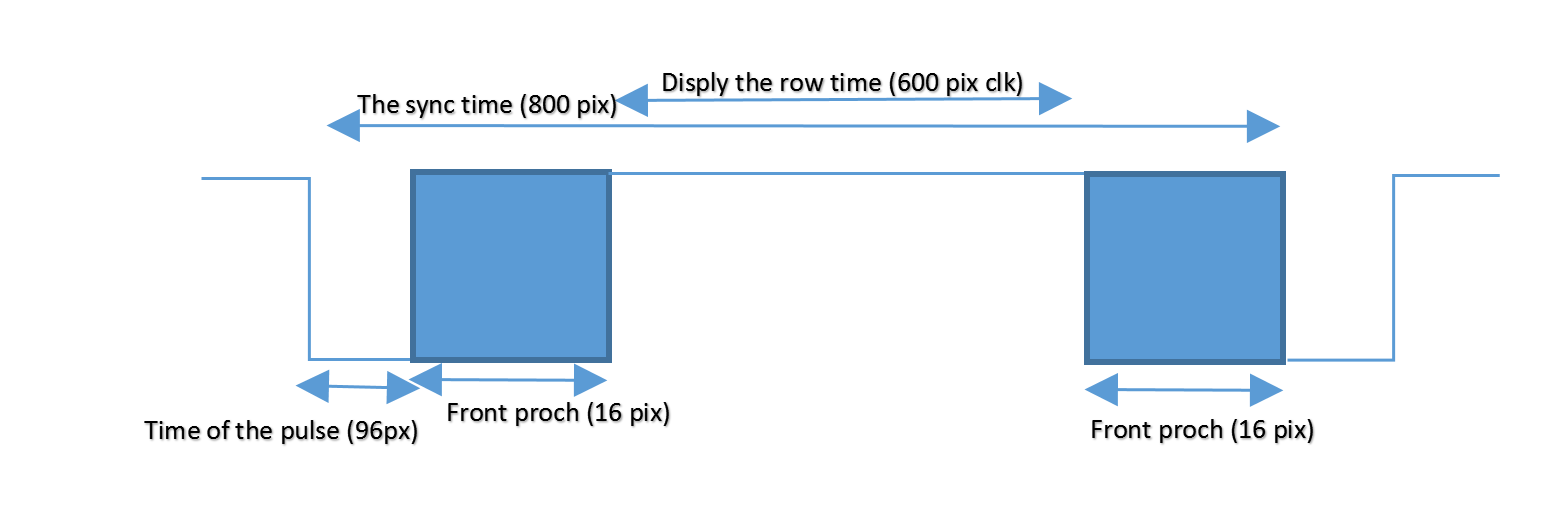
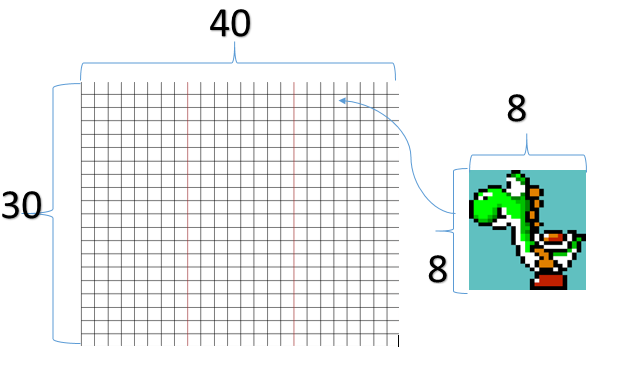


Diagram of the manipulation of pixel x and pixel y values from the timing:



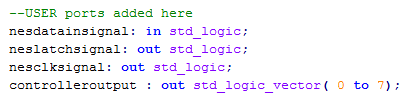
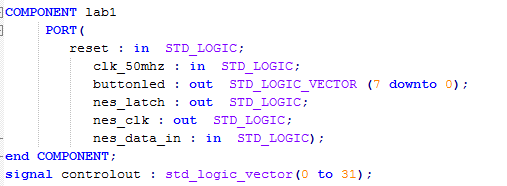
Lab #5 – IP Cores:

Description of the lab – inputs/outputs, explanation of how data is transferred from the CPU to the controller and graphics

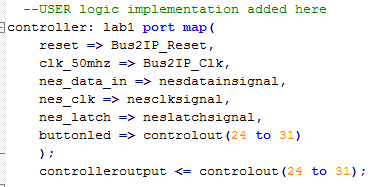
The main point of lab 5 is to create the ip cores for our system. We created two ip cores; one for the controller and one for graphics. Each ip core was assigned registers as needed. Register were needed to save the data given out of the vhdl code to the system and passing it to the upper modules to be used.

To create the controller core, we added our controller code to the ip core file. Then we modified the user logic and the ip core. The user logic has our code added to it as a component. It also takes the outputs/inputs from the code, assign it to signals to be passed to the upper module. Snippets of the code will be shown below.

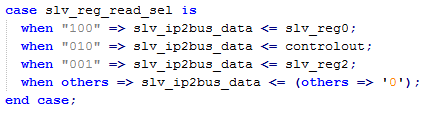
User Logic:

Signals to be assigned the inputs and outputs from the file.

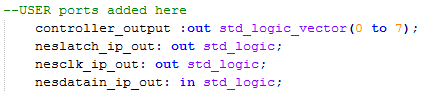


Ins/outs from the file assigned to the board buses and signals created

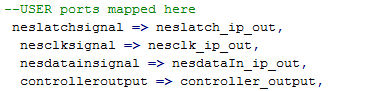


Signals from the controller assigned to the slave register

IP core:

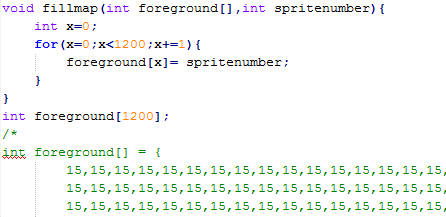


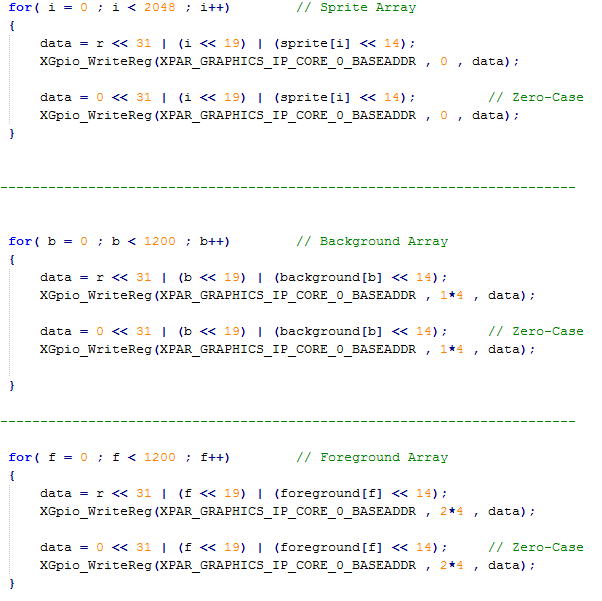
Signals to be assigned the inputs and outputs from the user logic.



Ins/outs from the user logic assigned to the signals created

Finally we wrote the C code that interacts with the processor to print to the screen:





Sizing of the chip will be shown on an attached pdf file.