ECE3270: Laboratory 2

LABORATORY 2: OPENCL LIBRARIES

October 6, 2017

Rodrigo Ignacio Rojas Garcia
Clemson University
Department of Electrical and Computer Engineering
rrojas@clemson.edu

Abstract

The OpenCL laboratory focuses of reinforcing the creation of modular VHDL entities to create a single design with multiple purposes. The homework was divided into two parts: 3-bit to 8-bit Decoder and OpenCL implementation. The laboratory consisted on designing a 3-bit to 8-bit to be used in part two and implement it with OpenCL software. The outcome of the design was to enter a 3-bit binary number and enable bit as input and display the output in human readable form by using OpenCL software. The output was displayed as different traffic light colors. The purpose of the homework was to introduce the student to the OpenCL software and to reinforce the utilization of Quartus II and ModelSim software for VHDL design and simulation.

1 Introduction

The laboratory was composed of two parts. Part one consisted on designing a 3-bit to 8-bit decoder with an enable pin. The design of the circuit was to be done in software Quartus II, in the VHDL programming language, and tested on software ModelSim. Part two of the laboratory, consisted of implementing the 3-bit to 8-bit decoder designed on part one as a component with the purpose to use OpenCL Libraries to send data to the decoder to obtain human readable output.

2 Part One

2.1 3-bit to 8-bit Decoder

Part one of the laboratory consisted on designing a 3-bit to 8-bit decoder with an enable pin. A decoder is a combinational logic circuit which converts a binary value input and outputs a set of binary bits that are associated to the binary value input.The design of the decoder was based on the figure provided in the homework manual[2] (Figure 1). The design was completed through the Quartus II software and VHDL programming language. The decoder was made a top level entity which contained two inputs and one output. One of the inputs consisted of a 3-bit binary number with possible values between 0 and 7 and was named "input", and the other input consisted on the enable bit which could have a value of either 1 or 0 and was named "enable". The output consisted of an 8-bit binary value in which each bit was given a value of either 1 or 0 and was named "output". The logic behind the decoder was to assign value 0 to the corresponding bit based on the 3-bit binary number input and if and only if the enable bit had the value of 1. The bits that were not chosen were given value of 1. For example, if the three-bit binary number was "010" the 8-bit output would be "11111101". If the enable bit was of value of 0, all 8 bits were given a value of 1 even if a certain bit was chosen.

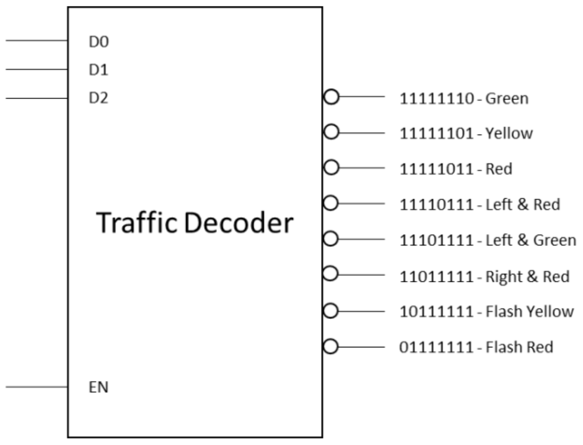
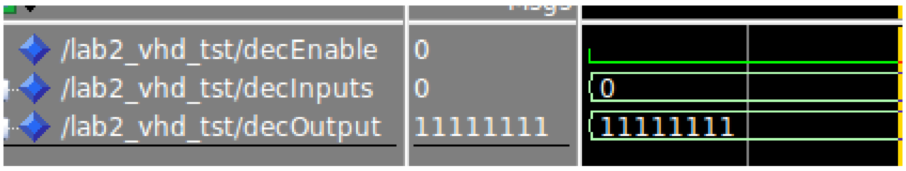


Figure 1: Decoder Diagram

2.2 Simulation of 3-bit to 8-bit Decoder

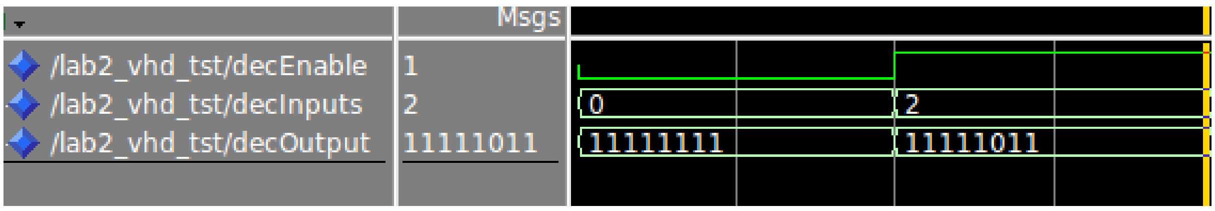
After the design of the 3-bit to 8-bit decoder was complete, simulation was done through software ModelSim to test and verify that the output obtained from the decoder was correct. The simulation of the decoder consisted on six different cases which tested the output of the decoder based on values of the three-bit binary number and enable inputs.



```
/lab2_vhd_tst/decEnable   0
/lab2_vhd_tst/decInputs   0          0
/lab2_vhd_tst/decOutput   11111111   11111111
```

Figure 2: First Case

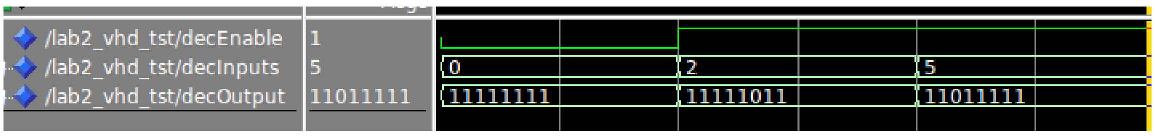
First case consisted of three-bit binary number being of value 0 and the enable bit being a value of 0 as well. The expected output was for all 8 bits to be of value 1 because the enable pin is of value 0, therefore value ”11111111” is obtained.



```
                          Msgs
/lab2_vhd_tst/decEnable   1
/lab2_vhd_tst/decInputs   2          0          2
/lab2_vhd_tst/decOutput   11111011   11111111   11111011
```

Figure 3: Second Case

Second case consisted of three-bit binary number being of value of 2 and the enable bit being of value of 1. The expected output was for bit 3 to be of value of 0. The expected output was for the 8-bit output to be ”11111011” because the enable pin is of value of 1, and the third bit corresponds to the the number 2.



```
/lab2_vhd_tst/decEnable   1
/lab2_vhd_tst/decInputs   5          0          2          5
/lab2_vhd_tst/decOutput   11011111   11111111   11111011   11011111
```

Figure 4: Third Case

Third case consisted of three-bit binary number being of value 5 and the enable bit being of value 1. The expected output was for bit 6 to be of value 0. The expected output was for the 8-bit output to be ”11011111” because the enable pin is of value 1, and the sixth bit corresponds to number 5.

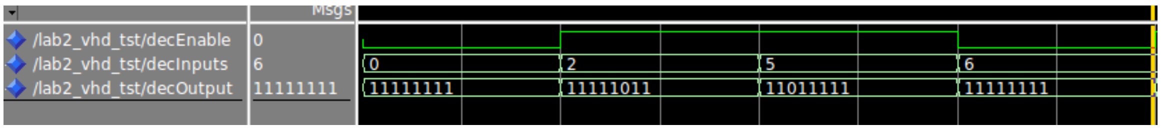


Figure 5: Fourth Case

Fourth case consisted of three-bit binary number being of value 6 and the enable bit being of value 0. The expected output for all 8 bits to be of value 1 because the enable pin was of value of 0, therefore value ”1111111” was to be obtained.

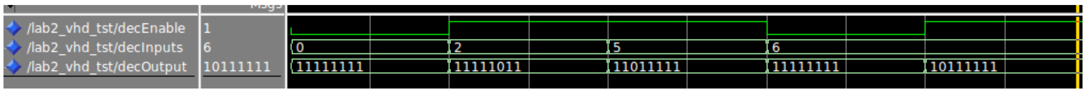


Figure 6: Fifth Case

Fifth case consisted on three-bit binary number being of value 6 and the enable bit being of value 1. The expected output was for bit 7 to be of value 0. The expected output was for the 8-bit output to to be ”10111111” because the enable pin was of value 1 and seventh bit corresponds to number 6.

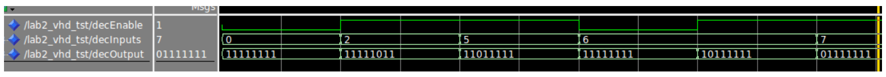


Figure 7: Sixth Case

Sixth case consisted on three-bit binary number being of value 7 and the enable bit being of value 1. The expected output was for bit 8 to be value of 0. The expected output was for the 8-bit output to be ”0111111” because the enable pin was of value of 1 and eight bit corresponds to number 7.

3 Part Two

3.1 3-bit to 8-bit Decoder and OpenCL

Part two of the laboratory consisted of using the 3-bit to 8-bit decoder designed on part one as a component with the purpose of combining it with OpenCL software. The OpenCL software was used to send data to the decoder with the purpose of entering different test cases, save the results, and give the corresponding output of each test case in human readable form. The human readable output was the different colors on a traffic light based on the inputs. The implementation of the OpenCL software was based through the manual provided on Canvas[1]. The combination of the OpenCL software and decoder was approached in different steps. First, the DE1_SoC board had to be switched from the default JTAG configuration to the OpenCL configuration. To accomplish this, the SW10 switches located in the back of the board had to be set value "110101". Configuration and location of pins can be seen on Figure 8. Second, the main entity called "traffic.vhd" file located in the "TrafficLight.zip" was obtained. The main entity was called "traffic" which was composed of five inputs and three outputs. The inputs used in this homework were "datain", and "ivalid". The output used in this homework was "dataout". Third, the main entity was modified so the decoder component could be used. The decoder was composed of two inputs and one output. One of the inputs corresponded to the 3-bit binary number called "input" and the other input corresponded to the enable bit which was named "enable". The output corresponded to the 8-bit binary value which was named "output". The decoder's inputs and outputs were mapped to the main entity's inputs and outputs. Inputs "ivalid" and "datain" from the main entity were mapped to inputs "enable" and "input" from the decoder correspondingly. The output "dataout" from the main entity was mapped to output "output" from the decoder. Fourth, the OpenCL software was tested. The OpenCL software was tested by running script "run.sh" located in "TrafficLight/aocl_traffic/emulation" directory with the purpose of testing that the correct output was being obtained from the test inputs entered by the OpenCL software into the decoder without compiling the program. The result can be seen in Figure 9. Fifth, compile the program. After the emulation was completed and the correct output was checked, the compilation of the program was performed so it could be loaded to the DE1_SoC board. The compilation of the program was done by first running the "make" command in the directory "TrafficLight/aocl_traffic", and after completion the same command was ran in directory "TrafficLight/aocl_traffic/device". Sixth, transfer compiled files to SD card. After compilation was done, executable file "traffic" located in "TrafficLight/aocl_traffic" directory and file "traffic.acox" located in "TrafficLight/aocl_traffic/device" directory, were transferred to an SD card. Inside the SD card, a folder named "traffic" was created in directory "home/root/" and files "traffic" and "traffic.acox" were transfered there. Seventh, execute compiled files in DE1_SoC board. After files were transferred to the SD card, the card was plugged into the DE1_SoC board. The board was accessed through a terminal by entering the command "minicom" to the terminal. Root user was chosen when prompted. After accessing the board, the directory "home/root/traffic" was accessed and command "chmod +x traffic" was entered so executable file "traffic" could be executed. Then, command "./traffic" was entered to execute traffic executable and obtain the decoder output as the color of the traffic light(Figure 10).

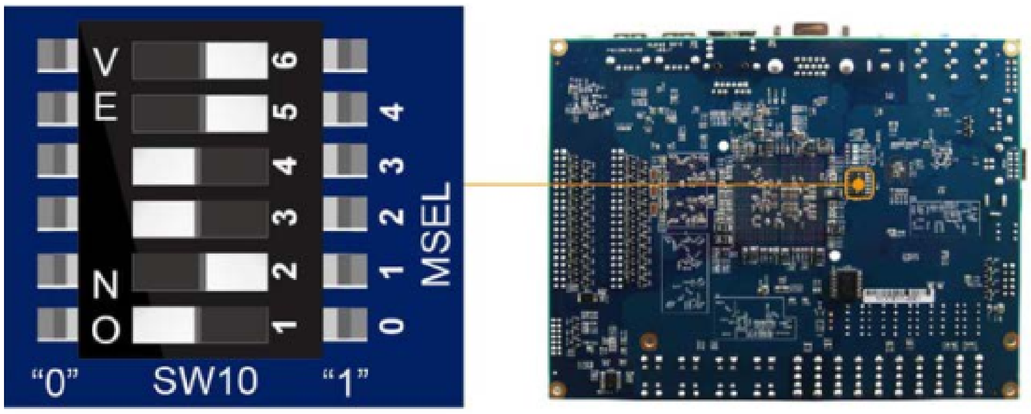
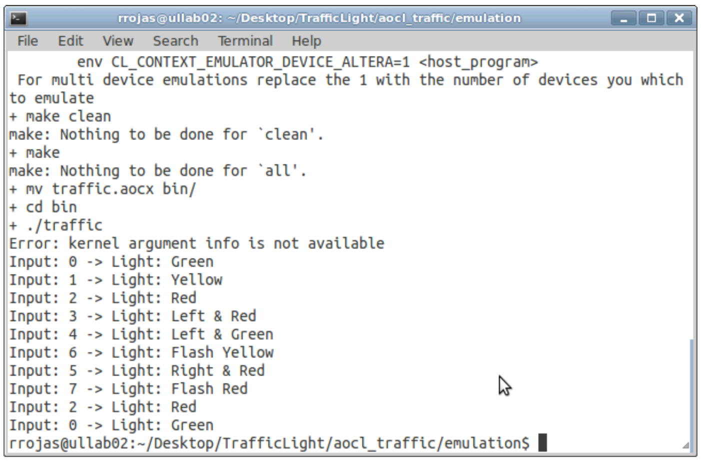


Figure 8: OpenCL pin Configuration

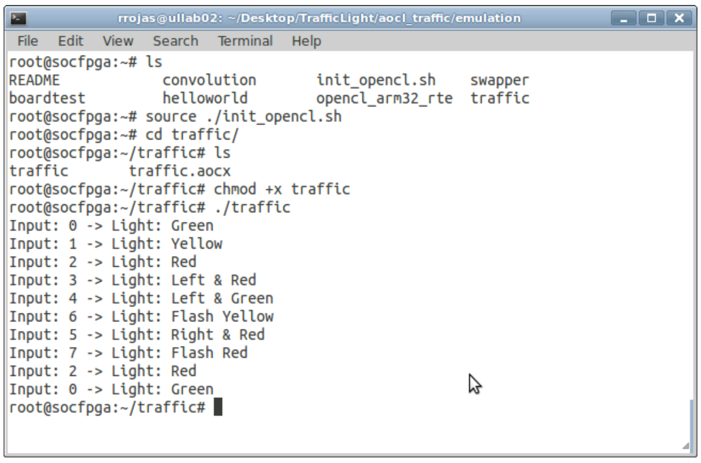
3.2 OpenCL and Decoder Results



rrojas@ullab02: ~/Desktop/TrafficLight/aocl_traffic/emulation
File Edit View Search Terminal Help

```
        env CL_CONTEXT_EMULATOR_DEVICE_ALTERA=1 <host_program>
 For multi device emulations replace the 1 with the number of devices you which
to emulate
+ make clean
make: Nothing to be done for `clean'.
+ make
make: Nothing to be done for `all'.
+ mv traffic.aocx bin/
+ cd bin
+ ./traffic
Error: kernel argument info is not available
Input: 0 -> Light: Green
Input: 1 -> Light: Yellow
Input: 2 -> Light: Red
Input: 3 -> Light: Left & Red
Input: 4 -> Light: Left & Green
Input: 6 -> Light: Flash Yellow
Input: 5 -> Light: Right & Red
Input: 7 -> Light: Flash Red
Input: 2 -> Light: Red
Input: 0 -> Light: Green
rrojas@ullab02:~/Desktop/TrafficLight/aocl_traffic/emulation$
```

Figure 9: Emulation Output


rrojas@ullab02: ~/Desktop/TrafficLight/aocl_traffic/emulation
File Edit View Search Terminal Help

```
root@socfpga:~# ls
README            convolution       init_opencl.sh    swapper
boardtest         helloworld        opencl_arm32_rte  traffic
root@socfpga:~# source ./init_opencl.sh
root@socfpga:~# cd traffic/
root@socfpga:~/traffic# ls
traffic       traffic.aocx
root@socfpga:~/traffic# chmod +x traffic
root@socfpga:~/traffic# ./traffic
Input: 0 -> Light: Green
Input: 1 -> Light: Yellow
Input: 2 -> Light: Red
Input: 3 -> Light: Left & Red
Input: 4 -> Light: Left & Green
Input: 6 -> Light: Flash Yellow
Input: 5 -> Light: Right & Red
Input: 7 -> Light: Flash Red
Input: 2 -> Light: Red
Input: 0 -> Light: Green
root@socfpga:~/traffic# 
```

Figure 10: DE1_SoC Decoder Output

4 Conclusion

The purpose of the laboratory was to introduce the student to the OpenCL software as well reinforcing the design in VHDL programming language using software Quartus II and ModelSim. The outcome of the laboratory taught the student how to create and design modular entities which can be implemented with OpenCL software. The homework accomplished this by making the student design a 3-bit to 8-bit decoder and impalement OpenCL software to obtain human readable output.

References

[1] Canvas. *Compilation and Usage.pdf.* Canvas, 2017.

[2] Canvas. *Traffic Decoder - OpenCL Libraries.pdf.* Canvas, 2017.