ECE 3270: Laboratory 3

LAB 3: STATE MACHINES

October 25, 2017

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

Abstract

Laboratory three focuses on creating modular VHDL entities to create a single design with multiple purposes. The laboratory was divided in two parts: Traffic State Machine and implementation of Traffic State Machine with OpenCL software. The laboratory consisted on designing a Traffic State Machine to be used in part two and implement it with OpenCL software. The outcome of the design was to be able to implement a VHDL entity with OpenCL software to obtain output in human readable form. The purpose of the laboratory was to reinforce the design on software Quartus II, simulation software ModelSim, programming language VHDL, and the use of OpenCL software.

1 Introduction

The third laboratory for ECE 3270, Digital Computing Design, had the purpose of introducing the student to the design of a State Diagrams using the Quartus II, ModelSim, and OpenCL software as well as the VHDL programming language. Laboratory Three was divided into two parts. Part one consisted of designing a Moore state machine that operated like a traffic signal. Part two consisted of implementing the Moore state machine created from part one with the OpenCL with the purpose of displaying the machine's output in human readable form.

2 Design of Moore State Machine

2.1 Traffic Signal

In part one of the laboratory, the design of a Traffic Signal machine was to be created. The Traffic Signal state machine consisted of reading a 3-bit binary number and keeping track of machine's current state with the purpose of using this information to determine the machine's next state which corresponded to an 8-bit value. The machine contained six possible states: "Flash Red", "Red", "Green", "Yellow", Left Green", and "Right Red". Each of these states were represented by 8-bit binary values: 01111111', "11111011", "11111110", "11111101", "11110111", "11011111" correspondingly. The design of the Traffic Signal state machine was approaches in two steps. First, the machine's state transition were determined. The state transitions were calculated by using the state diagram provided on the Laboratory Three Manual (Figure 1)[2] and determining the next possible state based on the current state of the machine. The calculation can be seen on Figure 2. Second, software Quartus II was used to design a top level entity. The Traffic Machine was made a top-level VHDL entity, which was composed of four inputs and one output. The inputs were composed of three one-bit inputs named "clock", "reset", and "ivalid" and a 3-bit binary number named "input". The output consisted of a 8-bit binary value named "output". The architecture of the Traffic Machine consisted of an internal signal named "trafficColors". The purpose of the internal signal was to store an 8-bit binary value which corresponded to one of the six possible states of the Traffic Machine which was assigned based on the four inputs and current state of the machine. The state machine designed in Quartus II software can be seen in Figure 3. It should be noted that State Diagram on Figure 3 does not contain input "ivalid" which was not required to be implemented on part one.

2.1.1 Tables and Figures

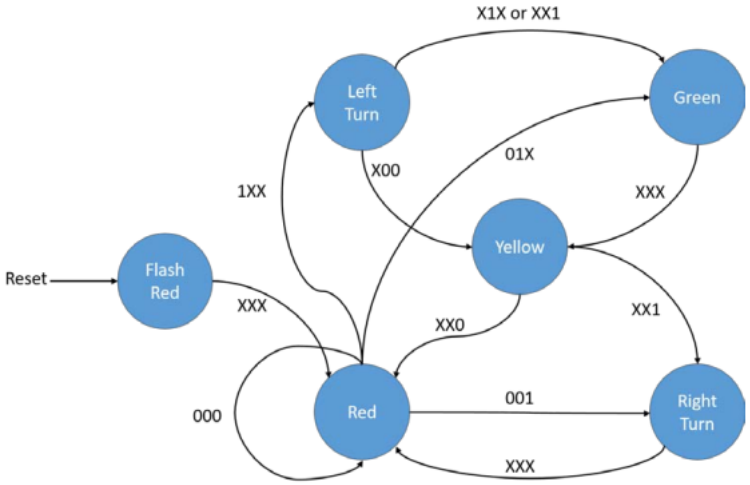

Figure 1: State Diagram from Laboratory 3 Manual

Current State	3-Bit Input	Clock input	Ivalid Input	Reset Input	Next State
Flash Red	XXX	1	1	1	Red
Red	000	1	1	1	Red
Red	001	1	1	1	Right Red
Red	01X	1	1	1	Green
Red	1XX	1	1	1	Left Green
Yellow	XX0	1	1	1	Red
Yellow	XX1	1	1	1	Right Red
Right Red	XXX	1	1	1	Red
Green	XXX	1	1	1	Yellow
Left Green	X1X OR XX1	1	1	1	Green
Left Green	X00	1	1	1	Yellow
ANY	XXX	0 OR 1	0 OR 1	0	Flash Red

Figure 2: Traffic Machine State Table

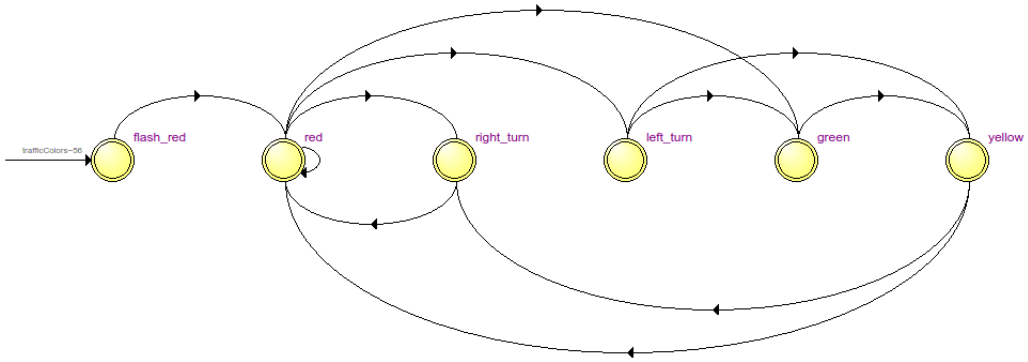

Figure 3: Quartus II State Machine Design without ivalid

2.2 Simulation of Traffic Signal

After the design of the Traffic State Signal Machine entity, simulation was performed on the software ModelSim to test seven different cases to confirm that the machine was in the correct state based on the inputs and last machine's state. Each test consisted of entering different 3-bit binary values into variable "input", toggling value of input "clock" between 0 and 1, keeping constant value of inputs "ivalid" and "reset" to 1 and checking that output "output" displays the correct state of the machine by comparing it to values of Figure 2. It should be noted that state "Left Green" corresponds to "Left Turn" and state "Right Red" to state "Right Turn".

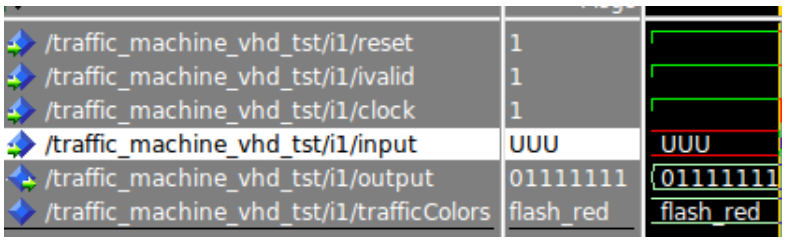

Figure 4: First Case

In the first test case, input "input" is not given a value by which the value of the output "output" is expected to be "011111111". The output is expected to be this value because the state machine is to be started on state "Flash Red". See Figure 2 to compare.

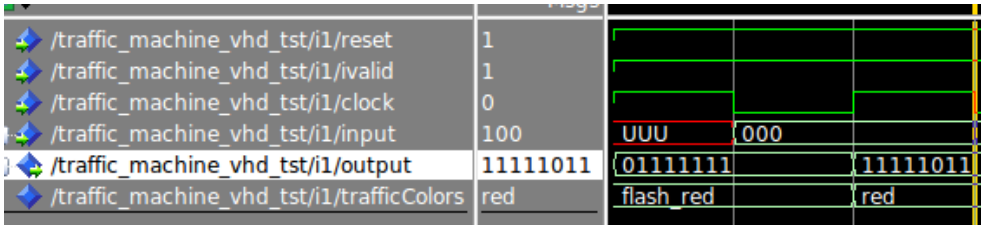

Figure 5: Second Case

In the second test case, input "input" is given value of "000" which is expected to give output value "11111011". The output is expected to be this value because last state of machine was of "Flash Red" and because input is of value "000" it goes to state "Red". See Figure 2 to compare.

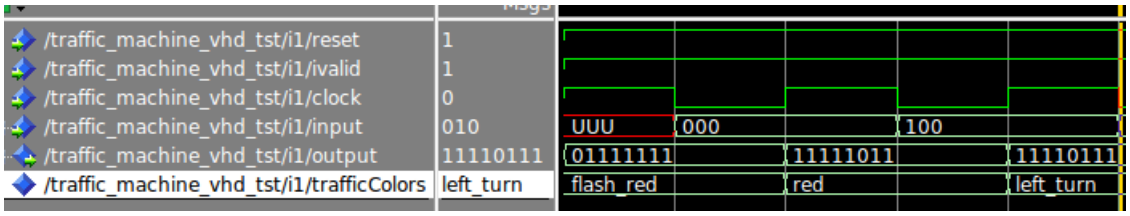

Figure 6: Third Case

In the third test case, input "input" is given value of "100" which is expected to give output value of "11111011". The output is expected to be this value because last state of the machine was "Red" and because input is of value "100" the next state should be "Left Green". See Figure 2 to compare.

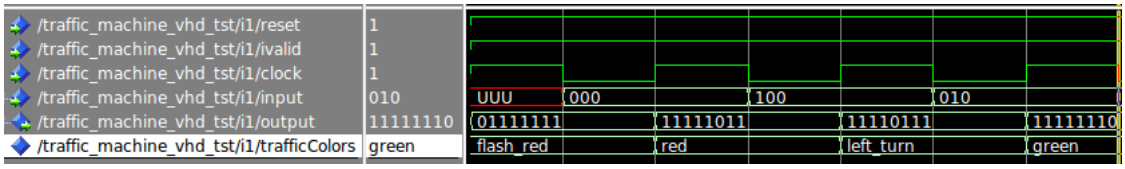

Figure 7: Fourth Case

In the fourth test case, input "input" is given value of "010" which is expected to give output value of "11111110". The output is expected to be this value because last state of the machine was "Left Green" and because input is of value "010" the next state should be "Green". See Figure 2 to compare.

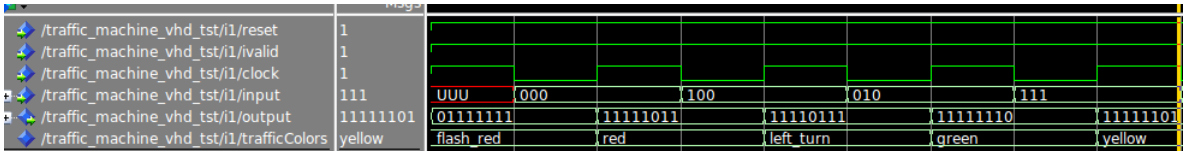

Figure 8: Fifth Case

In the fifth test case, input "input" is given value "111" which is expected to given output value of "11111101". The output is expected to be this value because last state of the machine was "Green" and because input is of value "111" the next state should be "Yellow". See Figure 2 to compare.

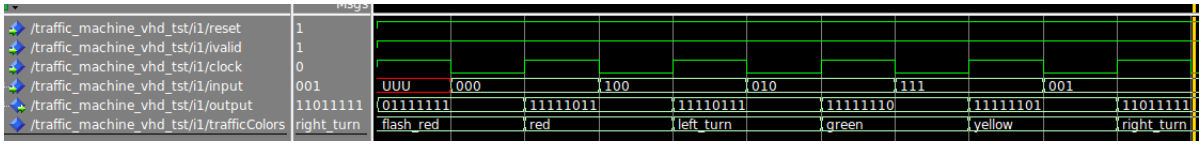

Figure 9: First Case

In the sixth case, input "input" is given value "001" which is expected to give output value of "11011111". The output is expected to be this value because last state of the machine was "Yellow" and because input is of value "001" the next state should be "Right Red". See Figure 2 to compare.

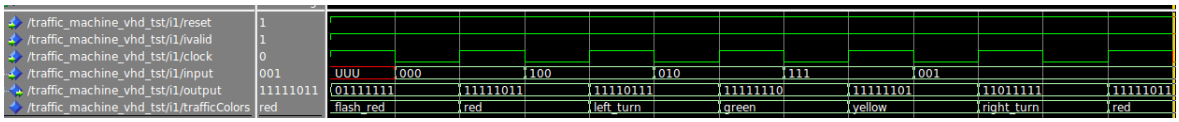

Figure 10: Seventh Case

In the seventh case, input "input" keep value "001" which is expected to give output value of "11111011". The output is expected to be this value because last state of the machine was "Right Red" and because input does not matter, the next state should be "Red". See Figure 2 to compare.

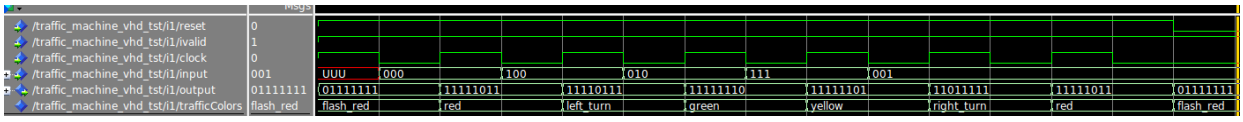

Figure 11: Eighth Case

In the eighth case, input "input" keeps value "001", but input "reset" is given value of "0" which is expected to give output of value "01111111". The output is expected to be

this value because if reset is of value "0" it will reset to the first state which corresponds to "Flash Red". See Figure 2 to compare.

3 OpenCL Implementation

In part two of laboratory three, the implementation of the Traffic State Machine with OpenCL software was to be created. The specification of laboratory three regarding the OpenCL implementation consisted of four parts: downloading tar file "Traffic.tgz" from Canvas and extract it to ULLAB machines, edit VHD files "traffic.vhd" and "traffic_.machine.vhd", and compile host and device binaries with the purpose of copying them to a SD card, set up and program DE1_.SoC board to execute program stored on SD card.

3.1 Obtaining Tar File

The first part consisted of logging into Canvas and downloading tar file named "Traffic.tgz" which can be found under "Lab 3 State Machine" assignment files. After the file was downloaded, the file was extracted in one of the ULLAB computers located in the laboratory room.

3.2 Modifying VHD Files

The second part consisted of modifying VHD files "traffic.vhd" and "traffic_.machine.vhd". These two files can be found in directory "Traffic/aocl_.traffic/device" which corresponds to extracted tar file "Traggic.tgz". First, the VHD file "traffic_.machine.vhd" was modified. This file corresponded to the traffic machine designed on part one. The modification of the file consisted of copying the same exact code from part one of the laboratory into this file. After this was done, the modification of VHD file "traffic.vhd" could proceed. The file consisted of a main entity called "traffic" with 5 inputs and 3 outputs. The inputs consisted of four single bit inputs called "clock", "resetn", "ivalid", and "iready", and 7-bit input named called "datain". The outputs consisted of a single bit outputs called "oready" and "ovalid", and 7-bit output called "dataout". The architecture of the entity was composed of a single component named "traffic_.machine". This component corresponds to the Traffic State machine designed on part one of the laboratory. The component contained the same four inputs and single output as the designed from part one as well as the same naming for inputs and output. The top-level entity "traffic" used the component "traffic_.machine" to obtain it's output and save it on "dataout". To obtain this result, inputs "clock", "reset", "ivalid", "input" from component "traffic_.machine" were mapped to inputs "clock", "resetn", "ivalid", "datain" of the top-level entity "traffic". The input "clock" was used as a clock in the entity, "resetn" was used as a reset to go back to the first state, "ivalid" was used to tell the machine that the input data was valid and allow the machine to go the next state, and finally the 3 lower bits of input "datain" were used as input to determine the state of the machine. The output of the component "traffic_.machine" was mapped to output "dataout" of top-level entity "traffic" which

corresponds to the current state of the machine. The design of the Moore Traffic State Machine made on Quartus II can be seen on Figure 13. All steps regarding design were based on laboratory manual provided by instructor[2]

3.3 Compilation of Program and Storage of SD Card

After the modification of the VHD files, compilation of the program could proceed. First, a terminal was opened in the directory "Traffic/aolc_.traffic" from the extracted "Traffic.tgz" file. This directory contained file named "Makefile" which was executed by using the command "make", the outcome of executing this command was an executable file named "traffic". Second, directory of terminal was changed to "Traffic/aolc_.traffic/device". This directory also contained a file named "Makefile" which was executed by using command "make", the outcome of executing the command was an executable file named "traffic.aocx". The purpose of compilation was to obtain the files "traffic" and "traffic.aocx" with the purpose of executing them on board DE1_.SoC board. After the compilation of the VHD files was complete, the executable files "traffic" and "traffic.aocx" were transferred to a SD card. The process of transferring the executable files to the SD card could only be completed in a computer running a Linux Operating System. Once the SD card was inserted into the computer, a folder named "lab3" was created in directory "home/root" in which executable files "traffic" and "traffic.aocx" were stored at.

3.4 DE1_SoC Configuration and Program Execution

After the executable files were compiled and stored in an SD card the process of configuring the DE1_SoC board's hardware and software configurations and program execution could proceed. First, the DE1_SoC board was configured to run on an OpenCL configuration. The DE1_SoC board as default runs in the JTAG configuration, to configure it to the OpenCL configuration, switch SW10 on the back of the board had to be set to value "110101". See Figure 12 for reference. After this was completed, the microSD card could be inserted into the board and upon powering on the board, red LEDs started showing a binary counting sequence. After the hardware OpenCL configuration was completed, the software configuration had to be completed. The software setup consisted on accessing the DE1_SoC board through a ULLAB computer. To do this, a terminal was opened and command "minicom" was entered. This command made the board start initialization. Then, the type of user was prompted and user "root" was entered. After board was accessed, command command "./init_opencl.sh" was entered which configured the board to run OpenCL code. All steps regarding DE1_SoC configuration were based on manual provided by laboratory instructor[1]. After the software configuration was completed and execution of executable files transferred to the SD card could be executed. To accomplish this, directory "root/home/lab3" was accessed. In this directory, executable file "traffic" and "traffic.aocx" were located here. To run these executable files, command "chmod +x traffic" was entered which made file "traffic" an executable file. After this, command "./traffic -t" was entered which executed the file and an output was obtained. The output consisted of the different states that the Traffic light could output which corresponded to

the output that the Traffic State Machine designed on part one could output (Figure 14).

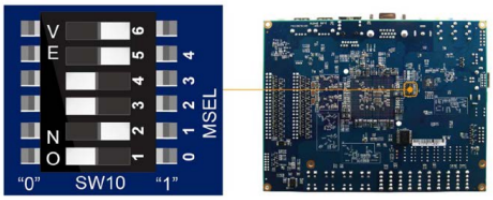

Figure 12: DE1_SoC board Configuration

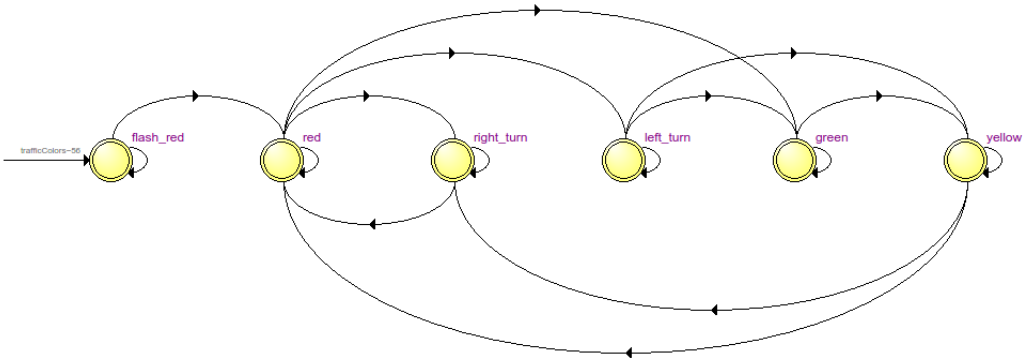

Figure 13: Quartus II State Diagram with ivalid

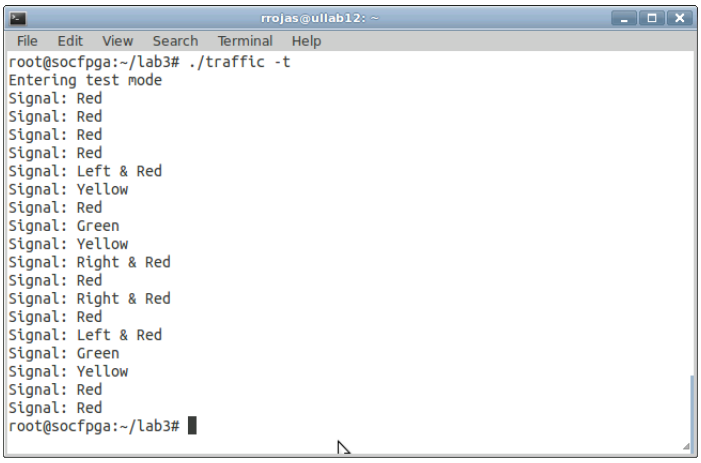

Figure 14: Traffic State Machine OpenCL Simulation Output

4 Conclusion

The purpose of laboratory three was to reinforce the use of the VHDL programming language, design software Quartus II, simulation software ModelSim, and implementation software OpenCL to the student. The outcome of laboratory three taught the student how to create and design modular entities and implement it with OpenCL software. The laboratory accomplished this by making the student design a Traffic State Machine and implementing it with OpenCL software to obtain human readable output as traffic light states.

References

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

[2] Canvas. *Traffic State Machine.pdf*. Canvas, 2017.