ECE3270 Digital Computer Design: Lab 3

FUEL STATE MACHINE

March 8, 2019

Aayahna Herbert
Clemson University
Department of Electrical and Computer Engineering
aayahnh@g.clemson.edu

Abstract

The goal of this lab was to implement a state machine that acts as a fuel gauge. This design lab was split into two parts: (1) completing a VHDL design of the state machine, (2) implementing the machine on a DE1-SoC board, and (3) compiling and and executing the device in OpenCL. In this lab, all parts of the lab were successfully designed, tested, and executed on the DE1-SoC board, in OpenCL, and through simulation done in ModelSim.

1 Introduction

According to Chapter 8 of Brown [1], a finite state machine, or a sequential circuit, is a circuit that uses combinational logic and one or more flip-flops along with a clock signal to control the operation of the circuit. A FSM usually has a set of primary inputs, W, and produces a set of outputs, Z; the values of the outputs of the flip-flops are referred to as the state of the circuit. There are two types of FSMs: a Moore type and a Mealy type; a Moore type has its outputs depend only on the state of the circuit while a Mealy type has its outputs depend on both the state and the primary inputs, W. For this lab, a Moore type state machine was designed to keep track of fuel. The next sections will go into details about how each part of the lab was constructed and the reason it was done the way it was.

2 Fuel State Machine

2.1 Design Method

The design of the fuel state machine was made up of an entity and a behavioral architecture for the entity. The entity consisted of a clock, "clk," reset, "resetn", its primary state input, "w," and a pin to check the validity of the input data, "valid" as the inputs and a four-bit output, "z," that produces the state's output. The fuel state machine will keep track of the amount of gas that is in the tank by monitoring when the driver chooses to drive (when w=0) or refuel (when w=1) and also display the state the fuel is currently in.

Using the state encoding information shown in Figure 7.1 and a hand-drawn diagram of the algorithmic state machine shown in Figure 7.2, the behavioral architecture was made of a specification section for the state machine and two process statements to design how the state machine will function and when it will update. The specification of the state machine defined the signals, "y-present" and "y-next," and the 10 different states it can be in: RESET, FULL, DRIVING1, TWO-THIRDS, DRIVING2, ONE-THIRD, DRIVING3, EMPTY, REFUELING, and WALKING. The first process statement included input "w" and signal "y-present" in its sensitivity list and were used to help set up the condition for each state and the value for output "z" that would come as a result of the state. The second process statement included inputs "clk" and "resetn" in the sensitivity list and set up the conditions for when the state of the machine would update. The states are set up to update when input "valid" is HIGH and on the rising edge of the clock; if input "resetn" is LOW, then the state machine would go to RESET and nothing will update until "resetn" is HIGH. To make sure the process statements were constructed correctly, the hand-drawn state diagram was compared to the VHDL-generated state diagram shown in Figure 7.3, which validated the correctness of the process statements.

2.2 Testing and Simulation

When it came to testing the fuel state machine with the test bench, the test cases were made to go through every possible path the state machine could possibly go through;

it also tested the functioning of the input "resetn" to make sure it can perform a hard reset on the state machine if needed. When simulated in ModelSim, the wave forms, as displayed in Figure 7.4, reflected the expected results from the test bench, making it successful; it also included the wave forms of "y-present" and "y-next" to show the intended values for output "z."

3 Implementation on the DE1-SoC Board

In order to implement the fuel state machine on the DE1-SoC board, the keys, switches, and LED lights were activated for use on the board. The structure for the board used the instance of the fuel state machine to map the inputs "clk" and "resetn" to the keys, inputs "w" and "valid" to the switches, and the output "z" to four LED lights to display the current four-bit state of the machine. When compiled and uploaded to the board, the execution of the state machine was successful.

4 Implementation in OpenCL

For OpenCL, the VHDL file for the OpenCL version of the state machine was provided in the .zip file for this lab with the entity for OpenCL completed, only needing the structure of the device needing to be built. The structure for OpenCL used the instance of the fuel state machine to map the inputs "clk," "resetn," "w," and "valid" to OpenCL's inputs "clock," "resetn," lowest bit of "datain," and "ivalid," respectfully. The output "z" was mapped to the lowest four bits of OpenCL's output "dataout" while the rest of its unused four bits were set to 0. The VHDL file for the fuel state machine was added to the "device" folder and the XML file; the files the "device" folder and in the "aocl" folder were compiled to get the executable and the .aocx file. When the executable and the .aocx file was added to the SD card and implemented on OpenCL, to program for the state machine functioned correctly.

5 Results

When both parts of the lab were modified to be tested on the Altera DE1-SoC Board and OpenCL using guidance from Smith [2], they compiled and functioned correctly. The common issue I ran into - outside of simple syntax issues - when working on the lab was the OpenCL not being able to compile as a result of me incorrectly setting unused bits in "dataout" to 0. When the final part was tested on the board and OpenCL with all of the other parts combined with it, the board and the terminal displayed the correct states based on the value of "w", meaning they were a success.

6 Conclusion

This lab has given me the chance to test and apply my knowledge about finite state machines while improving my skills and knowledge on OpenCL and VHDL coding, behavioral and entity elements and techniques, learned from class lectures. By doing this lab, I have a clearer understanding on how each of the components made in this lab work and what they all need in order to do their jobs correctly. If I could go back and change how things were done throughout the lab, I would change a couple of lines of code to be written in a way that makes the VHDL file shorter and easier to read, like combining the two process statements into one process statement.

References

[1] S. Brown and Z. Vranesic. Fundamental of digital logic with vhdl design. Textbook, New York, USA, 2009.

[2] M. Smith. Digital computer design lab manual. Manual, South Carolina, USA, 2019.

7 Appendix

You will need a total of **10** states: **Reset**, **Full**, **2/3**, **1/3**, **Empty**, **Refueling**, **Walking**, and the rest are **Driving** states. Encode them as follows:

RESET:	**"0000"**
FULL:	**"0001"**
2/3:	**"0010"**
1/3:	**"0111"**
EMPTY:	**"0100"**
REFUELING:	**"1100"**
WALKING:	**"1101"**
DRIVING:	**Use the remaining values for your other 3 states**

Figure 7.1: Fuel State Machine Encodes

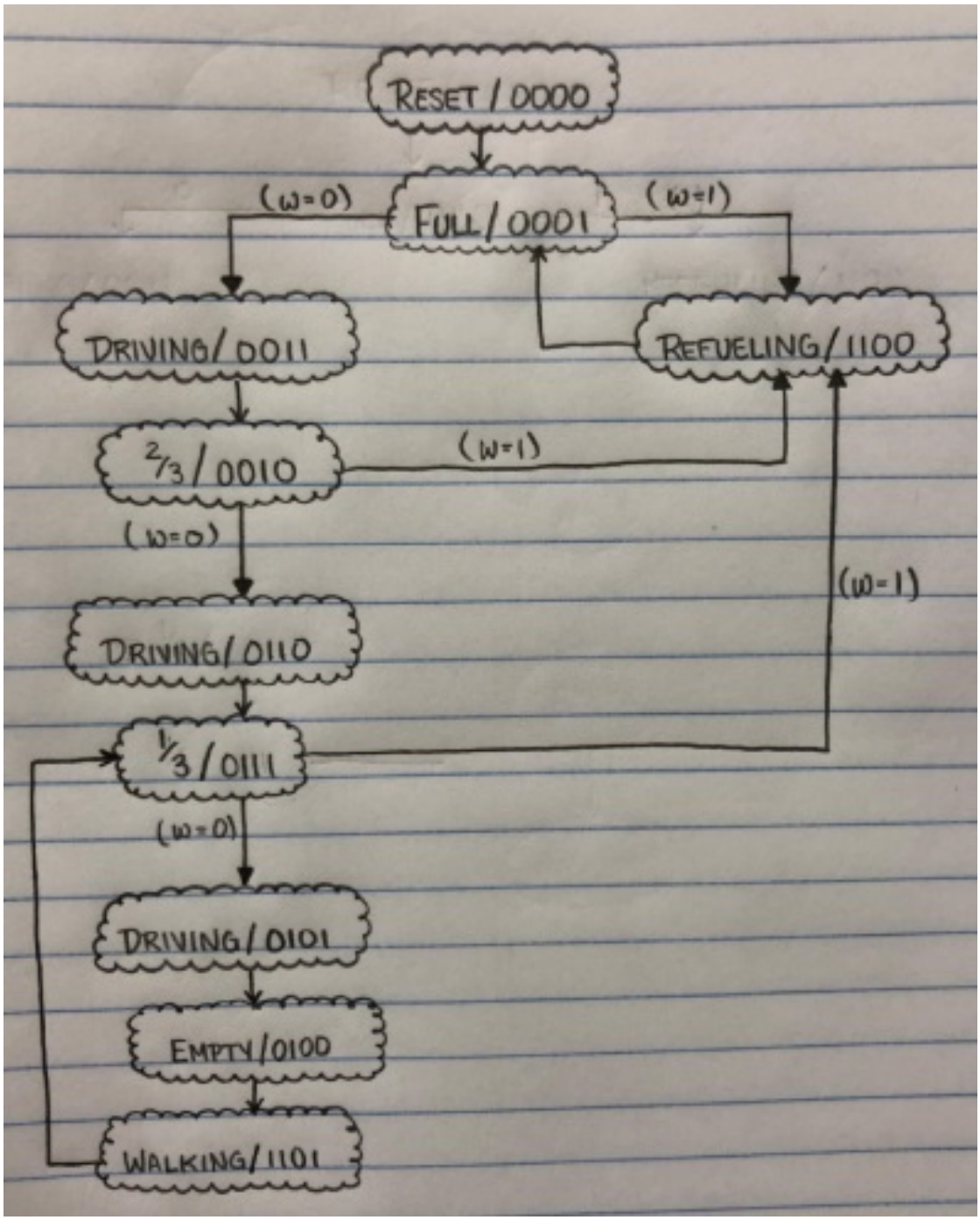

Figure 7.2: Hand-drawn Fuel State Machine Diagram

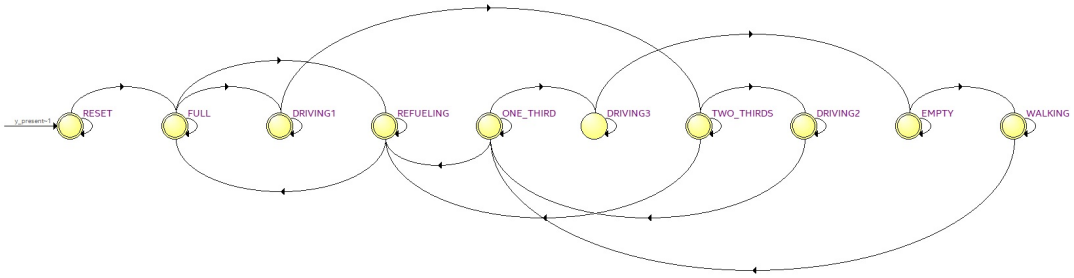

Figure 7.3: VHDL-Generated Fuel State Machine Diagram

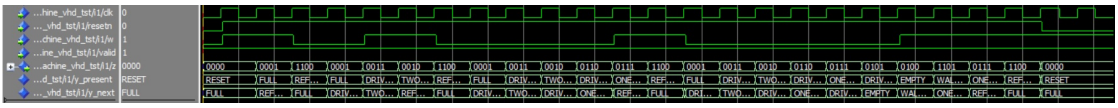

Figure 7.4: Fuel State Machine Simulated Test Waves