

# Introducing the Lab Environment

Electrical Engineering 371 Lab 1

University of Washington



Name	Student ID	Signature
Ting-Yu Wang (Jacky)	1333301	
Jack Gentsch	1329060	

## **Table of Contents**

[ABSTRACT](#)

[INTRODUCTION](#)

[DISCUSSION](#)

[Design Specifications](#)

[Design Procedure](#)

[System Description](#)

[Software Implementation](#)

[Hardware Implementation](#)

[Test Plan](#)

[Test Specifications](#)

[Test Cases](#)

[Presentation, discussion, and analysis of the results](#)

[ANALYSIS OF ANY ERROR](#)

[ANALYSIS OF WHY THE PROJECT MAY NOT OF WORKED AND WHAT EFFORTS  
WERE MADE TO IDENTIFY THE ROOT CAUSE OF ANY PROBLEMS](#)

[SUMMARY](#)

[CONCLUSION](#)

[APPENDIX A](#)

[APPENDIX B](#)

[APPENDIX C](#)

[APPENDIX D](#)

[APPENDIX E](#)

## **ABSTRACT**

In lab one, our group builds, simulates, and tests three types of counters in Verilog HDL using three modeling levels as well as schematic design in Quartus. In addition, we compiled the example C program and implemented a car list price calculator. In order to validate our design, we used SignalTap, the iVerilog compiler, and the GTKwave simulator. We edited the C program in CodeBlocks, and ran the resulting file on our laptop computers.

## **INTRODUCTION**

The purpose of this lab is to help us refresh our knowledge from EE 271 such as utilizing the Quartus II software and DE1-SoC board. In addition, the group learned how to use new tools such as iVerilog, GTKwave, and SignalTap to aid the development of the four assigned counters. The target platform for our labs is the Cyclone V FPGA while the code development is within the Quartus II. Lastly, we've also programmed a car list price calculator in C for this lab assignment.

## **DISCUSSION**

### **Design Specifications**

The four 4-bit counters that we developed are the following:

- Ripple up counter, with active low reset, using a gate level model.
- Synchronous down counter, with active low reset, using a dataflow level model.
- Synchronous Johnson counter, with active low reset, using a behavioral level model.
- Synchronous down counter, with active low reset, using schematic level entry.

The designed counters will be tested using GTKwave as well as SignalTap simulation software.

A car list price calculator will be designed in C using CodeBlocks. A thorough specification of the program is given in Appendix A.

## Design Procedure

### Ripple Up Counter

The first counter that we developed was a ripple up counter. Being a ripple counter, the dual flip-flops were asynchronous - that is, only the first had a clock input fed from the source (e.g. the DE1\_SOC). The rest of the flip-flops had to depend on the first DFF to drive their clock inputs. The counter had to count up and have an active-low reset. The implementation using a gate-level design is below, utilizing the provided Verilog code to create the DFF's:

```
//Attaches outputs, clock, and reset to upper level module
output wire [3:0] out;
wire [3:0] flip;
input wire clk, rst;

//Construction of flip-flops. Connected to create an asynchronous up counter.
DFlipFlop ff0 (.q(out[0]), .qBar(flip[0]), .D(flip[0]), .clk(clk), .rst(rst));
DFlipFlop ff1 (.q(out[1]), .qBar(flip[1]), .D(flip[1]), .clk(flip[0]), .rst(rst));
DFlipFlop ff2 (.q(out[2]), .qBar(flip[2]), .D(flip[2]), .clk(flip[1]), .rst(rst));
DFlipFlop ff3 (.q(out[3]), .qBar(flip[3]), .D(flip[3]), .clk(flip[2]), .rst(rst));
```

**Figure 1: Ripple Up Counter Dual Flip-Flops**

Note that the output of each DFF fed back into its input. That is, D was set to the output Q of each DFF. When the inversion of each output,  $\sim Q$  feeds the adjacent DFF's clock, this results in every flip-flop toggling states when the previous output becomes low. This ends up creating a ripple-up counter, and because the clock is not shared across devices, it is asynchronous.

### Synchronous Down Counter

The second counter was a synchronous down counter with active low reset. In order to determine the logic for the counter, truth table and k-maps were used. This allowed us to find the boolean algebras for the inputs (d[3:0]) of the D flip flops. Finally, the algebras was assigned in terms of Verilog's logical operations. This resulted in our completed dataflow model for a synchronous down counter.

```
// Connecting DFFs to form a 4 bit counter
assign d[3] = (q[3] & q[2]) | (q[3] & q[0]) | (q[3] & q[1]) | (qb[3] & qb[2] & qb[1] & qb[0]);
DFlipFlop dff3 (q[3], qb[3], d[3], clk, rst);

assign d[2] = (q[2] & q[0]) | (q[1] & q[2]) | (qb[2] & qb[1] & qb[0]);
DFlipFlop dff2 (q[2], qb[2], d[2], clk, rst);
|
assign d[1] = q[1] ^ q[0];
DFlipFlop dff1 (q[1], qb[1], d[1], clk, rst);

assign d[0] = qb[0];
DFlipFlop dff0 (q[0], qb[0], d[0], clk, rst);
```

**Figure 2: Synchronous Down Counter Code Fragment**

This logic was realized using the following K-Maps:

Q[1]Q[0] \ Q[3]Q[2]	00	01	11	10
00	1	0	1	0
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

**Table 1:** K-map denoting the output for the next sequence of Q[3] given Q[3:0]  
Results in:  $Q[3] = Q[3] * Q[2] + Q[3] * Q[1] + Q[3] * Q[0] + \sim Q[3] * \sim Q[2] * \sim Q[1] * \sim Q[0]$

Q[1]Q[0] \ Q[3]Q[2]	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

**Table 2:** K-map denoting the output for the next sequence of Q[2] given Q[3:0]  
Results in:  $Q[2] = Q[2] * Q[0] + Q[2] * Q[1] + \sim Q[2] * \sim Q[1] * \sim Q[0]$

Q[1]Q[0] \ Q[3]Q[2]	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	1	1	1	1
10	0	0	0	0

**Table 3:** K-map denoting the output for the next sequence of Q[1] given Q[3:0]  
Results in:  $Q[1] = Q[1] \oplus Q[0]$

Q[1]Q[0] \ Q[3]Q[2]	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

**Table 4:** K-map denoting the output for the next sequence of Q[0] given Q[3:0]  
Results in:  $Q[0] = \sim Q[0]$

### Synchronous Johnson Counter

The third counter was a synchronous Johnson counter with active low reset, using a behavioral level model. With the knowledge of the pattern for this counter, we found out that the most significant bit (MSB) of the next state will be the inverse of the LSB of the present state. In addition, the three lowest bits of the next state will be replaced by the three highest bits of the present state. Finally, the algorithms were implemented in terms of Verilog code within an always block.

```
ns[3] = ~out[0];
ns[2:0] = out[3:1];
```

**Figure 3: Synchronous Johnson Counter Code Fragment**

### Synchronous Down Counter - Schematic

The fourth and final counter was a four stage synchronous down counter, with a low reset, implemented in a schematic level entry. This utilized the same logic as our second counter, as they are both synchronous down counters with a low reset. The only difference is that this counter had to be implemented in Quartus' block symbol builder. The desired logic was broken down into individual gates, and then synthesized in the schematic editor.

The car list price calculator design procedure is under Appendix B.

## System Description

### Ripple Up Counter

The ripple up counter operated using the DE1-SoC on-board 50 MHz clock, which was stepped down to 0.75 Hz for observation on the board. The reset signal was mapped to SW[0] on the DE1-SoC, and the counter output was displayed on LEDR[3:0]. That is, LEDR[3] was the most significant bit, and LEDR[0] was the least significant bit. Each output bit is the result of a D flip-flop, and will flip states when the bit 1 lower than it becomes a zero. This results in the asynchronous behavior of the counter, and also creates the desired up counter behavior. Timing is limited to the 50 MHz clock, and no errors are possible besides hardware failure. The reset switch provides the only input, and controls whether the counter is allowed to run.

### Synchronous Down Counter

The inputs of this counter are CLOCK\_50 which serves as the clock and SW[9] which serves as the reset signal. On the other hand, the outputs for the counter is connected to LEDR[3:0] in order to visualize the outputs. Due to the synchronizing property, all flip flops are triggered by the same input clock which allows them to change state simultaneously. This clock produces a down counting sequence and loops forever until the reset signal is triggered.

### Synchronous Johnson Counter

Similarly to the previous two counters, the clock is derived from CLOCK\_50. Output bits are present on LEDR[3:0], and SW[9] will reset the counter (active-low). The bits count down synchronously due to the distribution of the global clock. The Johnson counter cycles through the intended eight states, as is required of a Johnson counter.

### Synchronous Down Counter (Schematic Level)

Please refer to Appendix D for an image of the schematic for this counter. The counter was originally designed using the logic calculated for the previous synchronous down counter. The gates were simply laid down in the block editor according to the desired logic, with as few gates

used as possible. Because the schematic level design had to be used with a 0.75 Hz clock (for in-class demonstration) and 50 MHz for computer simulation, the program had to be compiled in Verilog. One major problem with this process is that the Quartus compiler failed to assign the output pins to those of the DFF, resulting in a non-driven output. This bug was also seen in other group's projects, and was resolved by manually adding the assignment in the output Verilog file. The upper level module could then attach our counter to the DE1\_SoC, and utilized CLOCK\_50 as an input clock, SW[0] for an active-low reset switch, and LEDR[3:0] to display the output bits. There were no significant timing constraints with this counter due to the simplicity of the gate array, and the only errors were software-sided and resolved.

#### Car List Price Calculator

The inputs prompted from the user for this program are carCost, markup, salesTax as well as discount from the car dealer. On the other hand, the output is stored as listPrice which is calculated by taking the summation of the cost of the car, the markup price, the sales tax and less the discount. The final car list price will be displayed to the user. Any values entered by the user that are invalid will be prevented with error messages.

### Software Implementation

#### Counters

We built a top level file for each of the counters to hook the SW[9] or SW[0] to the reset signal and the LEDR[3:0] to the output. We also included a clock divider in order to slow down the 50MHz, CLOCK\_50 to a 0.75Hz clock so that the output on the DE1-SoC board can be seen with human eyes. The DFFs are hooked up as (MSB) DFF3 > DFF2 > DFF1 > DFF0 (LSB). The schematic level synchronous down counter is in Appendix D. The logic connecting these DFFs varies with each counter, but the overall goal and design process was identical for each counter. The logic was either immediately realized (the asynchronous down counter was quite simple), or was simplified using a K-Map (see Design Procedure section for an example). The desired functionality was then built in Quartus using dataflow, behavioral, gate, or schematic level design.

#### Car List Price Calculator

The user will be prompted for the cost of the car (carCost), the dealer's markup rate (markup), the sales tax rate (salesTax) and the pre-tax discount rate (discount) by the dealer. After collecting all the necessary data, the program takes the summation of the cost of the car, the dealer's markup price as well as the sales tax and then subtracts them from the pre tax discount. Finally, the result will be stored in the listPrice variable, which is rounded to 2 decimal places,



as the output and displayed on the screen. Any float values entered by the user that are invalid will be prevented with error messages.

## **Hardware Implementation**

Please refer to software implementation for details on the interface between our Verilog code and the Cyclone V FPGA. For lab 1, all our designs are loaded onto the FPGA of the DE1-SoC board which is connected to our computer through USB cable. Quartus compiles our Verilog code into functional gates, and then burns these gates into the FPGA. Our design was focused on developing code that would be optimized before being compiled - using K-Maps (see Design Process) and other logic analyses, we could create Verilog code that was synthesized with very few gates in the Cyclone V. We also strove to use the features built into the DE1-SoC to help interaction with the user, as the LED makes a very readable output and the switch forms a useful reset button. Strangely, our counters in this lab do not have a notable critical timing path. This is because the output of the DFF is wired directly to the output, which is taken by Quartus and directed to drive the LED output. Thus, each output has an equal critical path, and is likely very small. Note that the DFF is a clock triggered device, and thus the gates input to D do not add to the critical path of the system. Therefore the only critical path that can be created is a result of the FPGA layout, and is likely a result of the relatively lengthy wires spanning the FPGA. See Appendix D for an example of our schematic - the output of the DFF is wired directly to the output.

## **Test Plan**

### **Counters**

Our goal in testing is to verify that the actual output matches the expected output. The reset button will reset the counter to an all-zero state when high. The reset in the low position should allow the counter to change state. In addition, the LEDs will light up in the proper pattern depending on the counter being tested.

### **Car List Price Calculator**

Our program yield the correct final price based on the human-provided parameters. Invalid ranges of values should be prevented, and request the user for a valid value. Non-integers and non-floats are unhandled exceptions, and should create undesired function of the program.

## **Test Specifications**

## Counters

Upon being released from the reset condition (active low), the down counter outputs are expected to undergo a down count from 15 to 0, returning to 15 after reaching 0. On the other hand, the up counter should do the opposite: count from 0 to 15 and back to 0 after reaching 15. The Johnson counter will create the traditional stream of 0's, then slowly add 1's until the 1111 state, and then add 0's until returning to 0000. Note that the incoming stream enters from the left, or the most significant bit. In addition, our test also needs to show that all the counters has active low reset. The reset should consistently reset the output to the all 0 state, regardless of the present state of the output bits during the reset activation. Holding reset should also maintain an output of 0. Unwanted switches and keys should be nonfunctional, and fail to impact the output.

## Car List Price Calculator

The calculator must be tested with valid values as well as invalid values. For any invalid input values such as negative car cost, negative or over a hundred percent markup rate, tax rate and discount rate, an error message should be displayed. When entering -1 for the cost of the car, an error message should be displayed and prompt the user for the same input again while entering a valid value such as 25000, the program will simply proceed and prompt the user for the next input. When entering a value such as 10 that is in range for the markup, sales tax or discount rate, the program should proceed as normal while input rates such as -1000 and -0.0001 will trigger error messages and prompt the user for the same inputs again. Re-prompting of inputs will continually occur until valid input is given by the user or when the user exit the program.x

## Test Cases

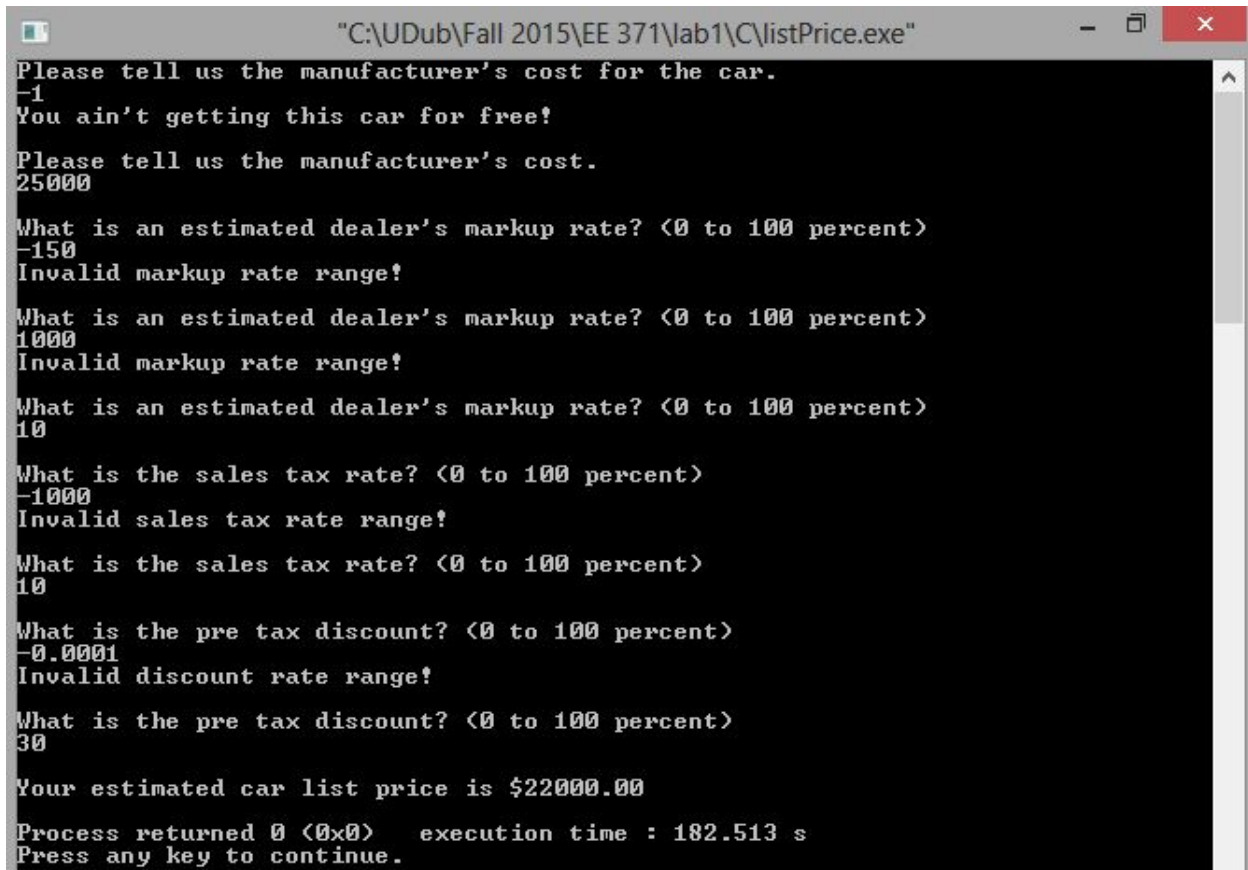
### Counters

To test the proper function of each counter, testing can be broken down into two phases. The first is computer simulation: utilizing programs like GTKwave and Quartus' SignalTap, we can run our program with a 50 MHz clock and verify their functionality. GTKwave is useful for testing the reset switch at 50 MHz, because there is more control over when the clock and reset switch is activated. For example, the reset switch can be toggled during operation, held low, or left untouched for many clock cycles. The resulting waveform can be manually analyzed by the engineer to verify proper functionality. The expected output of each counter is elaborated in depth in the Test Specifications section, but some important qualities is that each counter is low-reset, and the second and fourth counter should have identical functionality (despite being created in different mediums). After verifying functionality in GTKwave, SignalTap can be used to analyze delays and timing more accurately. SignalTap is a much more precise program when it comes to timing, so it can be used to analyze proper timing of each signal. Finally, the clock can be slowed down and the program compiled onto the DE1-SoC to allow for human testing.

In-person testing can be useful to throw switches near a clock edge or time reset signals at specific intervals, and analyzing the output at a much slower clock speed. Generally by this point all major bugs should have been detected, and the engineer can decide if the counters are functioning as intended.

### Car List Price Calculator

We will manually test our program using input values described in the test specification. The program is expected to handle erroneous float inputs, and display the proper messages. To ensure that the output for our program is correct, we must manually calculate the expected list price and compare it with the final program output.



```
"C:\UDub\Fall 2015\EE 371\lab1\C\listPrice.exe"
Please tell us the manufacturer's cost for the car.
-1
You ain't getting this car for free!

Please tell us the manufacturer's cost.
25000

What is an estimated dealer's markup rate? (0 to 100 percent)
-150
Invalid markup rate range!

What is an estimated dealer's markup rate? (0 to 100 percent)
1000
Invalid markup rate range!

What is an estimated dealer's markup rate? (0 to 100 percent)
10

What is the sales tax rate? (0 to 100 percent)
-1000
Invalid sales tax rate range!

What is the sales tax rate? (0 to 100 percent)
10

What is the pre tax discount? (0 to 100 percent)
-0.0001
Invalid discount rate range!

What is the pre tax discount? (0 to 100 percent)
30

Your estimated car list price is $22000.00

Process returned 0 (0x0)    execution time : 182.513 s
Press any key to continue.
```

Figure 4: Car List Price Program In Action

### Presentation, discussion, and analysis of the results

#### Ripple Down Counter

The ripple down counter fortunately operated as expected. The LED output remains in the all zero state when the reset switch is low, and begins counting upwards in binary upon being switched high. The LEDs cycle from the 1111 state to the 0000 state to restart the counting loop.

Activating the reset switch during operation will cause the counter to restart counting from 0000. The counter functioned as intended under strenuous testing, and deemed entirely functional.

### Synchronous Down Counter

When we untoggle (0) then toggle (1) the active low reset, all the LEDR[0] to LEDR[3] light up (0000). The LEDR[3:0] then begins to down count from 0000, 1111, 1110, 1101, 1100, 1011, 1010, 1001, 1000, 0111, 0110, 0101, 0100, 0011, 0010, 0001 and to 0000 then back to 1111. This behavior matches the expected output pattern. The down counter also responded to triggering reset in the middle of operation, and properly resumed counting from the 0000 state.

### Synchronous Johnson Counter

When we untoggle (0) then toggle (1) the active low reset, all the LEDR[0] to LEDR[3] being in the all zero state. Then the LEDR[3:0] begins to create the desired pattern from 0000, 1000, 1100, 1110, 1111, 0111, 0011 to 0001 and back to 0000. As we designed a Johnson counter, this behavior matches the expected output pattern. Triggering a reset during operation will properly return the system to the all zero state.

### Synchronous Down Counter (Schematic)

The schematic level down counter created an output identical to that of the second counter. The active-low reset behaved as expected, resetting the system's state to 0000. Holding reset maintained this state as desired, and the counter would resume by counting down to 1111, 1110... before returning to 0000. This output is as expected, our schematic level model behaved as we intended.

### Car List Price Calculator

The program was tested with several extreme cases and the output matched the externally calculated value. In addition, all the invalid input values were handled properly. An example test run can be seen in the previous test case section. Note that inputs such as a string were unhandled errors, and would ruin operation of the program as expected. Our thorough testing validated that the program is functioning properly.

### Questions

One major question raised involving computer simulation of our clocked circuits is how the simulated signals (i.e. those in SignalTap) compare to those in an ideal device. As can be seen in Appendix E, the SignalTap waveforms appear to trigger simultaneously with the edge of the clock. This happens to be ideal behavior, as no transport or inertial delay is visible in the system.

## **ANALYSIS OF ANY ERROR**

One of the errors we encountered was during the erroneous construction of the K-Map for the synchronous down counter which resulted in an improper counting sequence. However, this error was quickly discovered and resolved by redrawing the K-Map. Our car list price calculator was unable to handle strings, such as the input “\$900” or “20%”. The error could be resolved by implementing try and catch statements into our code, but due to this being a lab introducing the C programming language, we felt it was unnecessary to handles such cases. The largest error in our program occurred during the development of the schematic level down counter. In order to develop a clock divider for testing on the DE1-SoC, we needed to use the Quartus Verilog synthesizer. However, the synthesizer strangely failed to assign the output pins Q[3:0] to the wires they were attached to in the schematic view. After discussing the subject with another lab group, a teacher assistant, and the professor, it was determined that it was acceptable to manually code the required assign statement. This error is likely due to a bug within the Quartus Verilog synthesizer, because other groups experienced an identical error in their synthesized code.

## **ANALYSIS OF WHY THE PROJECT MAY NOT OF WORKED AND WHAT EFFORTS WERE MADE TO IDENTIFY THE ROOT CAUSE OF ANY PROBLEMS**

Other than the errors outlined in the above section, our design worked as expected. No error was left unfixed by the conclusion of our design process.

## **SUMMARY**

In this lab, we developed several counters using Quartus and the Verilog programming language. These counters were designed using either the dataflow, behavioral, gate, or schematic level models. Each was simulated in programs such as GTKwave and SignalTap, as well as observed and tested on the DE1-SoC’s Cyclone V FPGA. Our counters were developed utilizing analysis techniques developed in EE 271, and programmed/designed using Verilog concepts developed in EE 371. The results of our operational counters verified that our design process soundly created functional counters. For the latter part of our project, we developed a car list price calculator using the C programming language, which is a higher-level programming language when compared to Verilog HDL.

## **CONCLUSION**

This lab utilized our knowledge of hardware design and applied it to the languages of Verilog and C. The development of counters using four different methods helped us recall the various

methods to write Verilog, and helped us use the DE1-SoC, Quartus II, and various simulation software. Our design process was largely developed in this lab, which may be able to utilize in future labs. This project was very effective at introducing the necessary software required for the rest of the course, and should provide a good foundation for the rest of the quarter with regards to the entire lab process.

## **APPENDIX A**

### **Car List Price Calculator Requirement**

#### **Abstract**

The purpose of this program is to compute the list price for a selected car. In this piece of document, we will briefly talk about the inputs, output and the major functions of the program.

#### **Introduction**

The list price for a car is calculated from the summation of the cost of the car, dealer's markup price, sales tax and less the pre tax discount. The program will calculate the list price from the inputs given by the user and display the list price to the user.

#### **Inputs**

The user will need to enter four inputs which include the cost of the car, the markup rate, the sales tax rate and the discount rate. These inputs will be used to calculate the output.

#### **Outputs**

The final output, list price of the car, calculated from the inputs above will be displayed to the user.

#### **Major Functions**

This program will calculate the list price of the selected car based on the information given from the user through the inputs. By summing the car cost, markup price, sales tax and deduct the discount, the final list price of the car will be returned and displayed to the user of the program.

## APPENDIX B

### Car List Price Calculator Design Specification

#### Abstract

The car list price calculator will be used to determine the list price for a selected car. In this piece of document, we will discuss more in depth about the inputs and output of the program such as the range of legal values as well as the error handlings and their consequences.

#### Introduction

This program calculates the list price for the selected car. It takes in inputs from the user and reject invalid inputs by displaying error messages. The calculated list price will be displayed to the user.

#### Inputs

The program takes in the cost of the selected car, the markup rate, the sales tax rate and the pre tax discount rate as inputs and calculate the markup price, sales tax value and the discount value. **The inputs from the user MUST be an integer or float type value.** These values will be used to calculate the output for the program. The program will reject any inputs that are out of the specified range and display an error. The program will re-prompt the user for the same input until the range is satisfied. Any non-integer or float type value will result in the program entering an infinite loop or crashing, as it is an intentionally unhandled error.

#### Output

The final output, list price of the car, will be calculated based on the inputs and it will be displayed to the user. The output is rounded to two decimal places. The list price will always be positive since the cost of the car is restricted to positive only and that all the inputs are restricted between 0% to 100%.

#### Major Functions

The program will take in inputs such as the cost of the car, the dealer's markup rate the sales tax rate and the pre tax discount rate from the user based on the selected car. The program will detect

any invalid inputs and display error messages while re-prompting the user for the valid inputs. When all inputs are acquired, the final list price of the car, rounded to two decimal places, will be calculated and displayed to the user.

## **APPENDIX C**

### **Workload Distribution**

The workload was distributed equally amongst the team members. However, Beck left two days before the report due day which (tragically) forced the remaining two members to take on a three man job. After Beck left, the work distribution is as follows:

Jacky - The designing, coding, validating and simulating of the second down counter, Johnson counter and the C program.

Jack - The designing, coding, validating and simulating of the ripple up down counter and the schematic level down counter.

Both members contributed equally on the write ups for the lab report.



## APPENDIX D

### Schematics

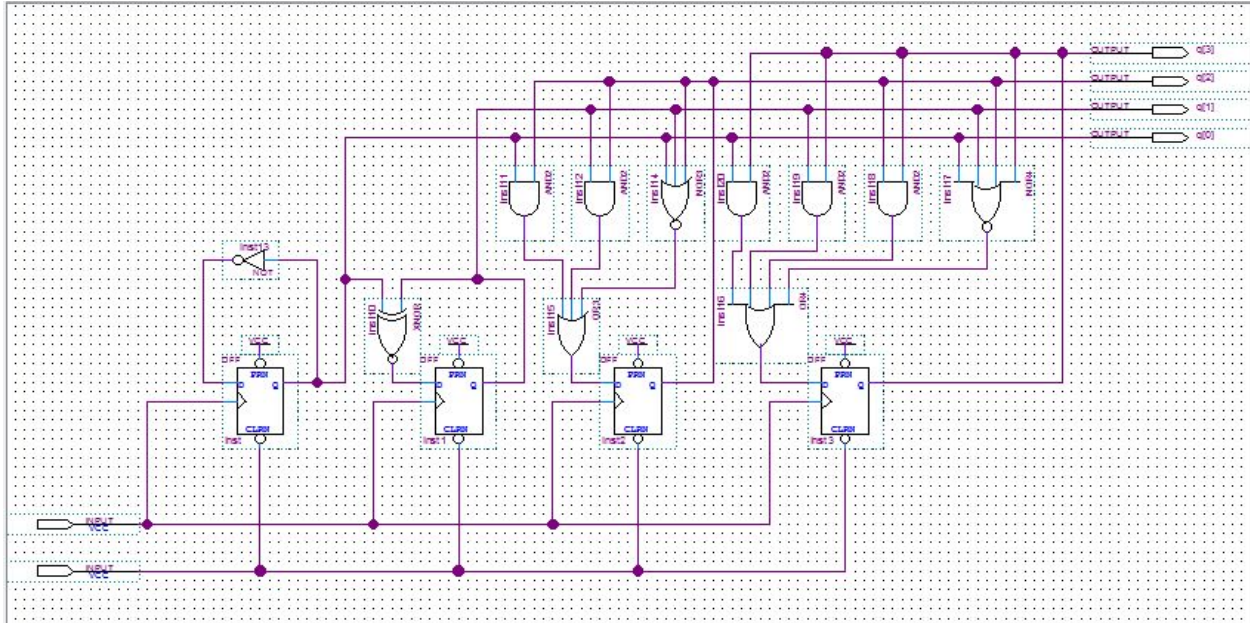


Figure 5: Schematic Level Model For Synchronous Down Counter

## APPENDIX E

### Simulation Screenshots

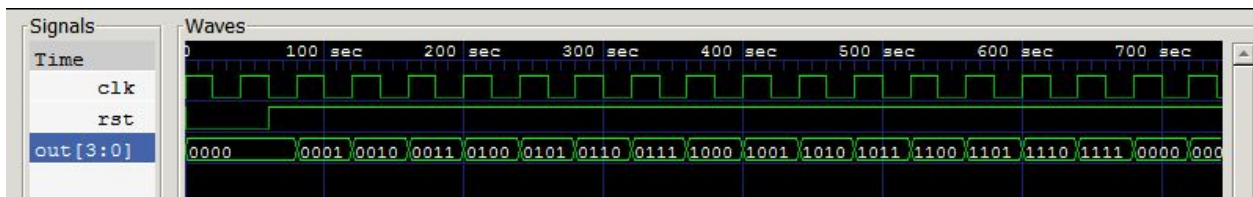


Figure 6: GTKWave For Ripple Up Counter

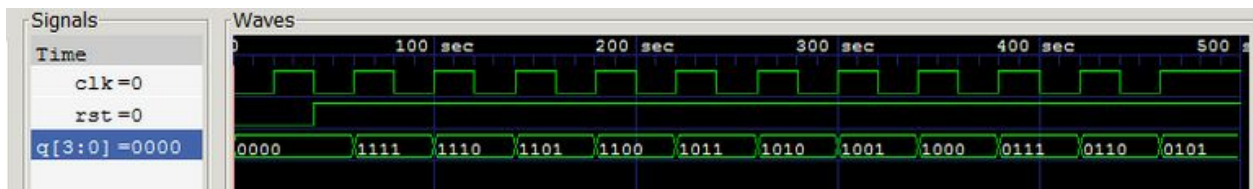


Figure 7: GTKWave For Synchronous Down Counter

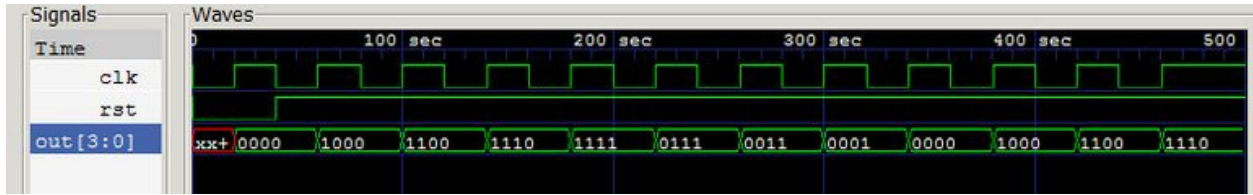


Figure 8: GTKWave For Synchronous Johnson Counter

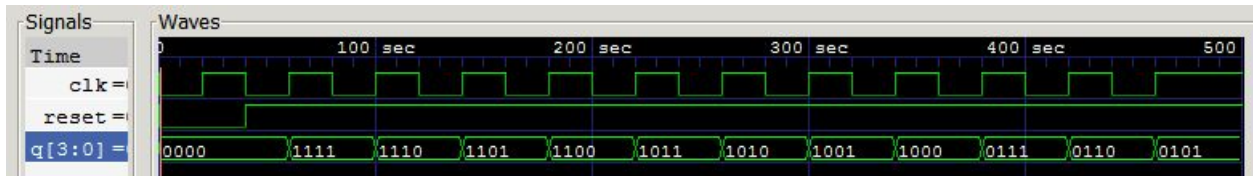


Figure 9: GTKWave For Synchronous Down Counter (Schematics)

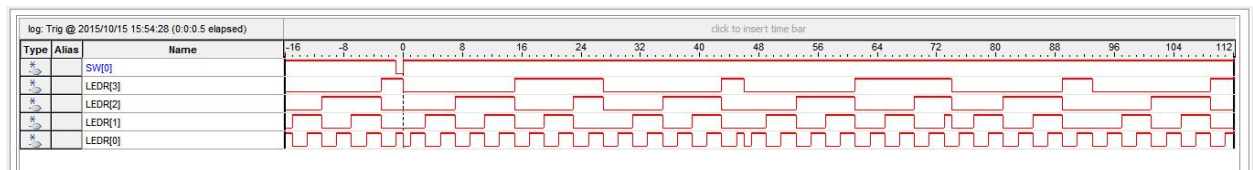


Figure 10: Ripple Up Counter Trigger On Rising Edge Of Reset

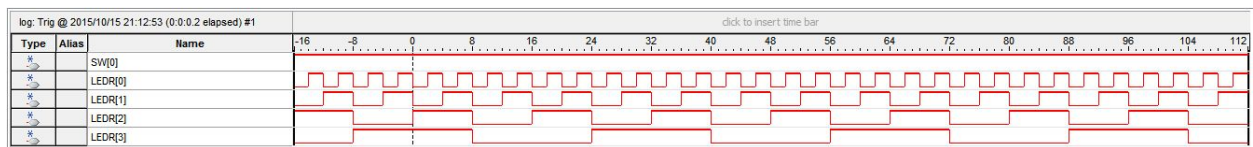


Figure 11: Ripple Up Counter Trigger Following Third State

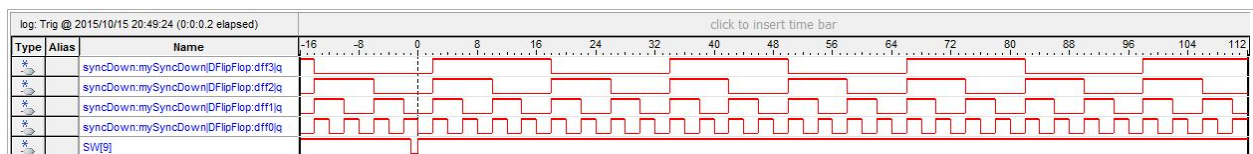


Figure 12: Synchronous Down Counter Trigger On Rising Edge Of Reset

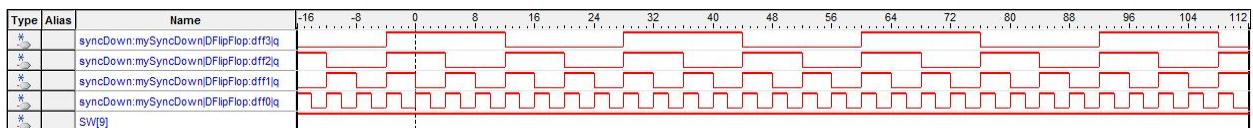
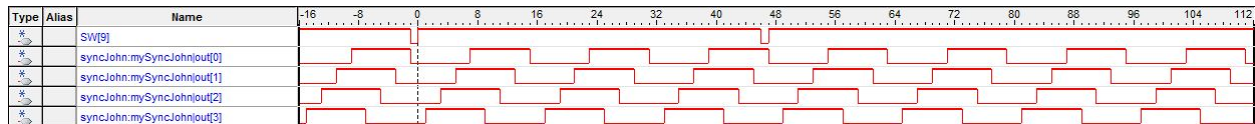
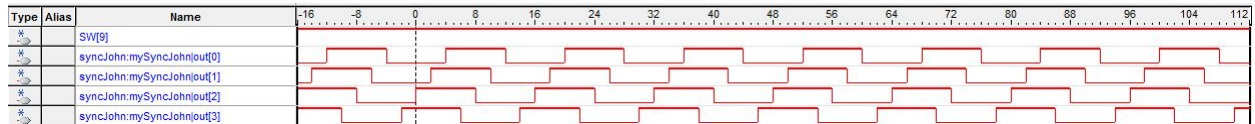


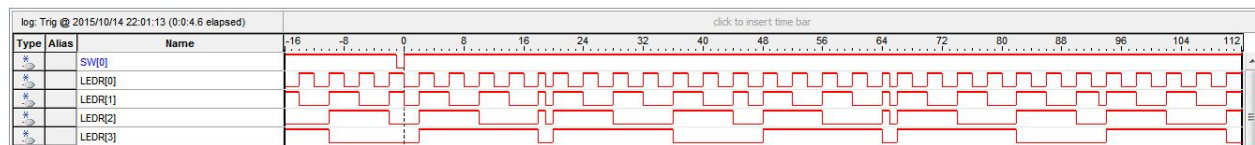
Figure 13: Synchronous Down Counter Trigger Following Third State



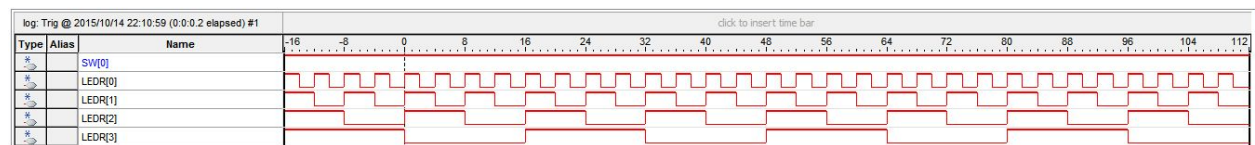
**Figure 14: Synchronous Johnson Counter Trigger On Rising Edge Of Reset**



**Figure 15: Synchronous Johnson Counter Trigger Following Third State**



**Figure 16: Synchronous Down Counter (Schematics) Trigger On Rising Edge Of Reset**



**Figure 17: Synchronous Down Counter (Schematics) Trigger Following Third State**