***Lab 2 –* Implementing A Full Adder**

## Review Item Comments Points (max)

1. Prelabs from each student complete (5)

and thoughtful

1. Introduction effectively presents the (5)

objectives and purpose of the lab. Methodology gives enough details to allow for replication of procedure.

1. Discussion opens with an effective (5)

statement on the goals of the lab, backs up the statement concerning appropriate findings, provides a sufficient and logical explanation for the statement, addresses other issues pertinent to the lab.

1. Results open with an effective statement of (5) overall findings, presents visuals clearly

and accurately, presents findings clearly and with sufficient support. You MUST

include screenshots of the test bench results for each part of the lab.

Conclusion convincingly describes what has been learned in the lab.

1. Other: (10)

References are included.

Tables and figures are formatted. Grammar and spelling are correct

Comment Blocks for ALL Verilog modules are filled

in with students names and module description/purpose Report is written clearly and to the point.

Overall, the team...

* + has successfully demonstrated what the lab was designed to teach
  + demonstrates clear and thoughtful scientific inquiry
  + has accurately measured and analyzed data for lab findings

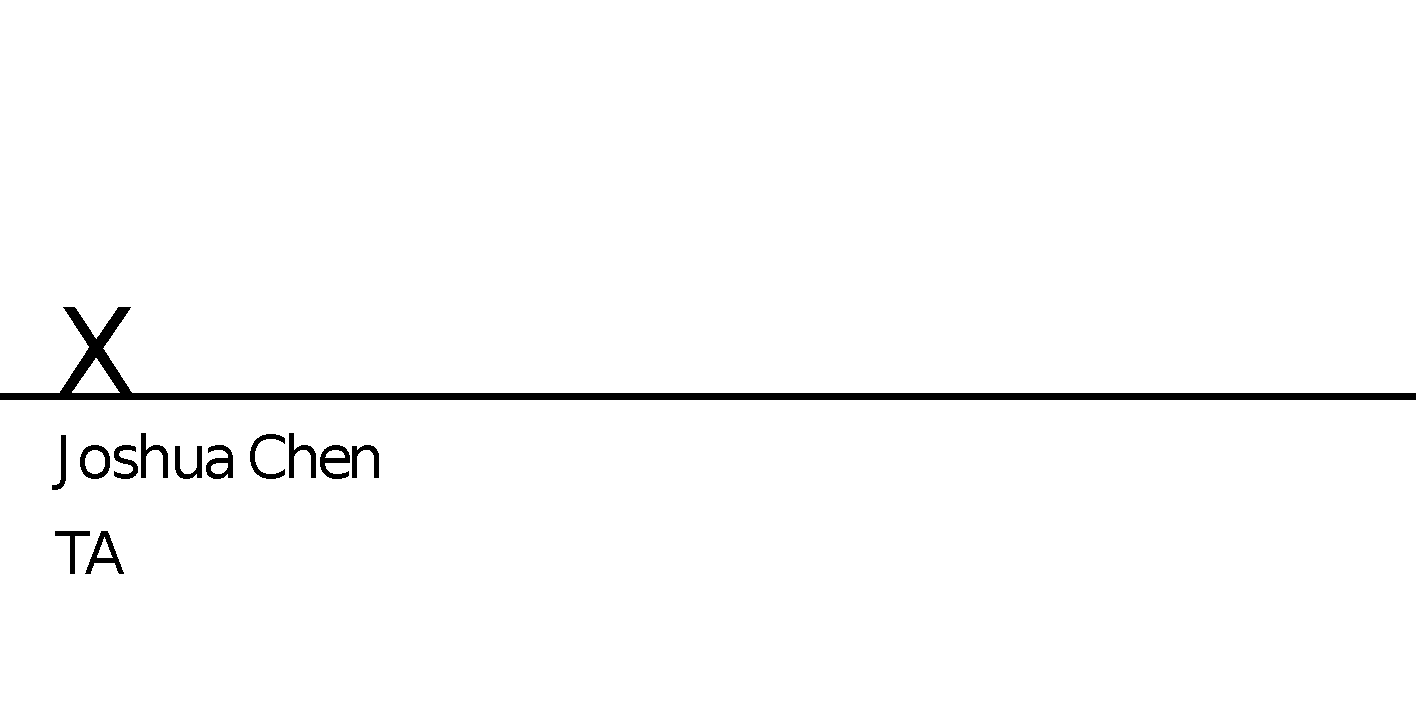
Total: (30)

|  |
| --- |
| EE260 Lab Lab Number  **Implementing a Full Adder/Two’s Complement Circuit** |

**Submitted by:**

**Team: TheShockers**

Grade:



|  |  |
| --- | --- |
| Team Members: | Chad Dunap [cddunlap@hawaii.edu](mailto:cddunlap@hawaii.edu)  Ryan E  redquiba@hawaii.edu |
|  |  |
| Date Experiment Performed: | Jan 30, 2018 |
| Date of Submission: | Feb 4, 2018 |
|  |  |
| EE260 Lab Section | 2 |

|  |
| --- |
| Introduction |

Introduction effectively presents the objectives and purpose of the lab.

As with last week we will be creating, testing, simulating, and downloading different kinds of digital circuits to the Basys 3 board. We will be implementing an 1-bit adder, a 4-bit adder, and lastly a two’s complement full adder.

We will first design the 1-bit adder which is three one bit numbers that will be A, B, and Cin for the input for our project and then have two outputs of Sum and Cout. After this we will use this to implement the 4-bit adder which is the addition of the full adder where there are four bits where the Cout of the beginning is connected to the Cin of the next bit. For a 2’s comp adder we will design the program to convert a number B into -B in 2’s comp form, which is basically flipping all the bits and adding one to the least significant bit.

After the implementation of our codes and constraints we will create a test bench to simulate the functionality of our logic and check the timing diagrams. Once we can see the successful timing we desire we can check for glitches in the circuit and adjust accordingly, and then we will program the FPGA board where our logic will be wired to LED lights where we can see if we correctly designed the circuit.

|  |
| --- |
| Methodology |

Methodology gives enough details to allow for replication of procedure. (You may assume lab assignment is available. You do not have to regurgitate the lab instructions).

We will be following the procedures for designing the three different circuits and using our assignment for the pre-lab to create a 1-bit Full adder, 4-bit adder, and a 2’s complement full adder.

For the 1-bit full adder the design is simple three inputs and two outputs. The three inputs we will use for this lab are A, B, and Cin for the two outputs it will be Sum, and Cout. We then will design the gates for our full adder using the boolean expression from the pre lab. After this constraints will be made to select the appropriate switches and LED lights to be wired. For testing our logic we will create a test bench to test different simulations we would like to check, especially for when A=1 B=0 and then transitions to A=1 B=1. After this we generated a bit stream to test our design on the Basys 3 board.

Next we designed a 4-bit adder with only two inputs and three outputs. What’s different about the inputs is that it will have a 4-bit bus array ranging from 0-3 for our inputs of A and B. The output Sum is also a 4-bit bus array and the other two outputs will be OF\_S and C\_MSB. OF\_S will be implemented as a signed overflow flag and C\_MSB will be represented as the unsigned overflow flag that’s the carry-out of the MSB. We will have to wire the carry out from the first bit to the carry-in of the next bit to connect the adders together. Also we will give each one of the four full adders a U number for initiation. Once the assign statements are finalized we can add the constraints file. We will use similar a similar constraint file as the full adder except for each input they will be assigned four switches. The output Sum will be assigned to four LED lights and the overflow detections OF\_S and C\_MSB will be assigned to their own lights. Then we create a test bench for simulation testing where we will be testing for each case of our logic and to activate each of the overflow detections.

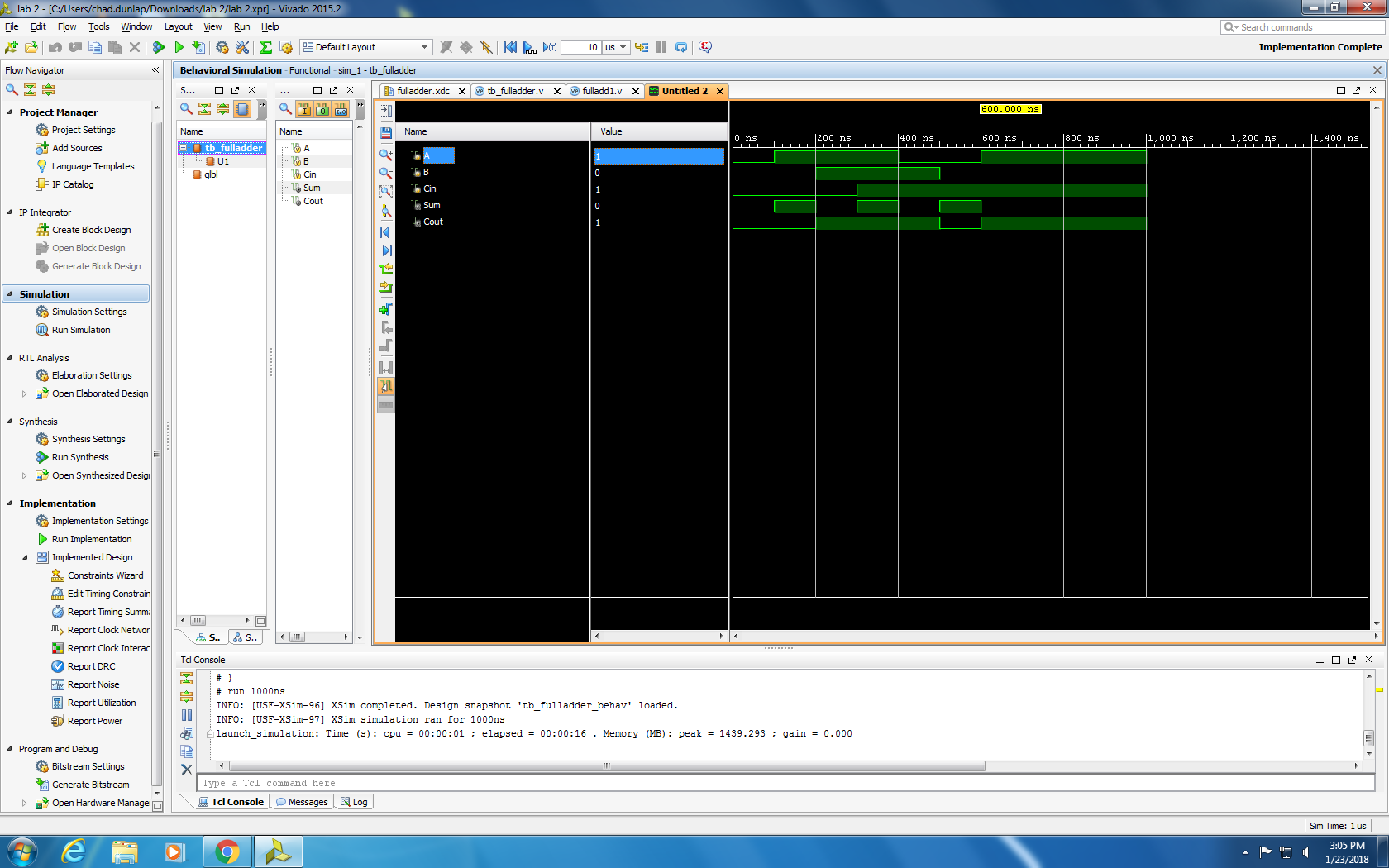
Lastly we will design a 2’s complement full adder. The theory behind two’s complement is that you take you 4-bit number (for this example) and you give the complement of each bit (flip the bits) then you add a one and the result is a two’s complement representation of the original number. The code behind this is simple we will take a value (B for this lab) and invert it then add one, the result will be negB both of which we be a four bus array number. We will test all 16 combinations in our constraints file to test for each possible case. The test bench will display the 16 combinations and their two’s complement equivalent. Then we will generate the bit stream to test our design on the Basys 3 FPGA board.

|  |
| --- |
| Results |

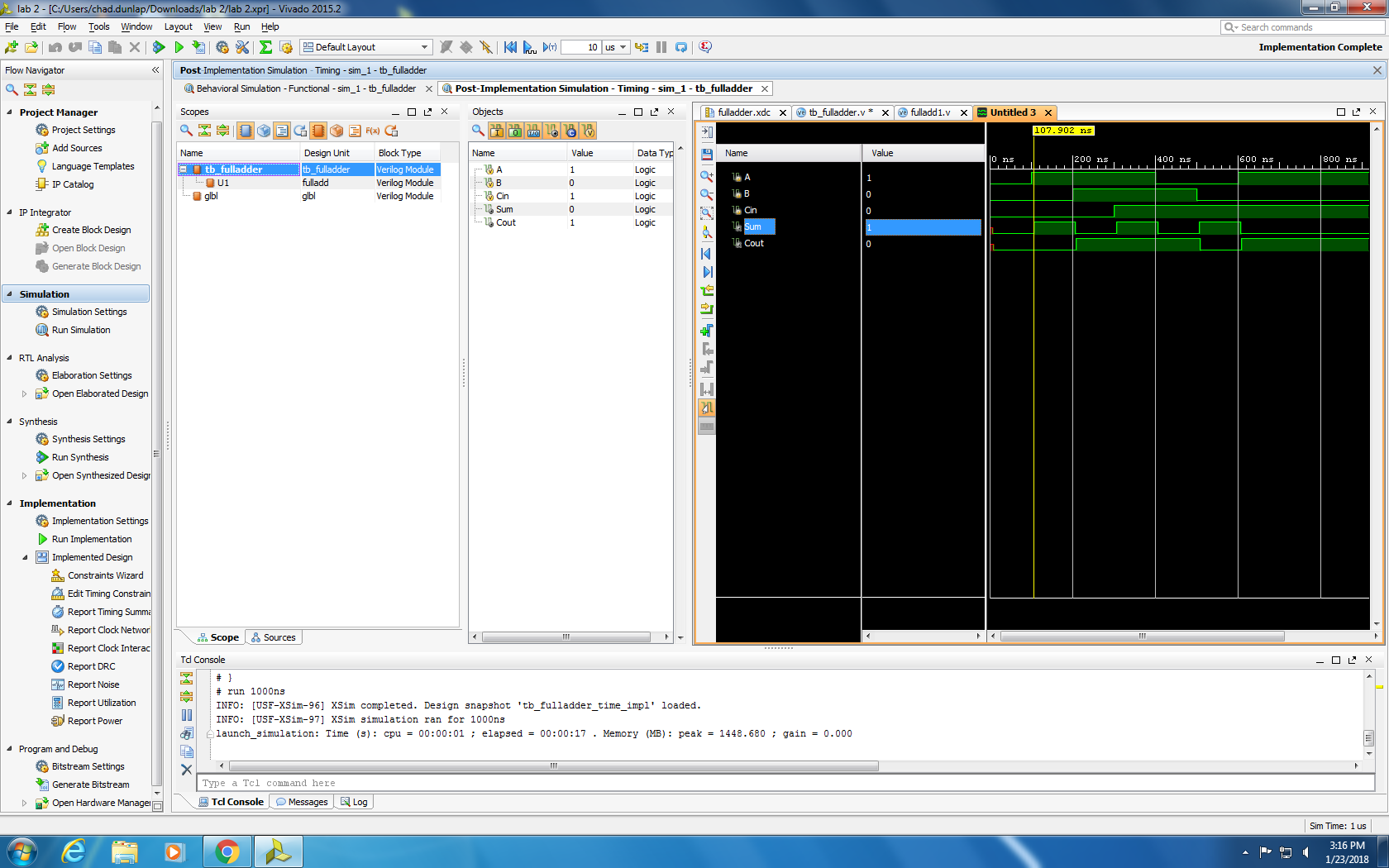
Results open with an effective statement of overall findings presents visuals clearly and accurately, presents findings clearly and with sufficient support. You MUST

Our design went just as planned for all three designs. We were able to successfully test all the cases we wanted to see and check the simulation to make sure there were no timing issues. We did find some glitches that will be discussed shortly, but overall the lab went as planned. We will show the test bench pictures to display our findings and show the results of our design.

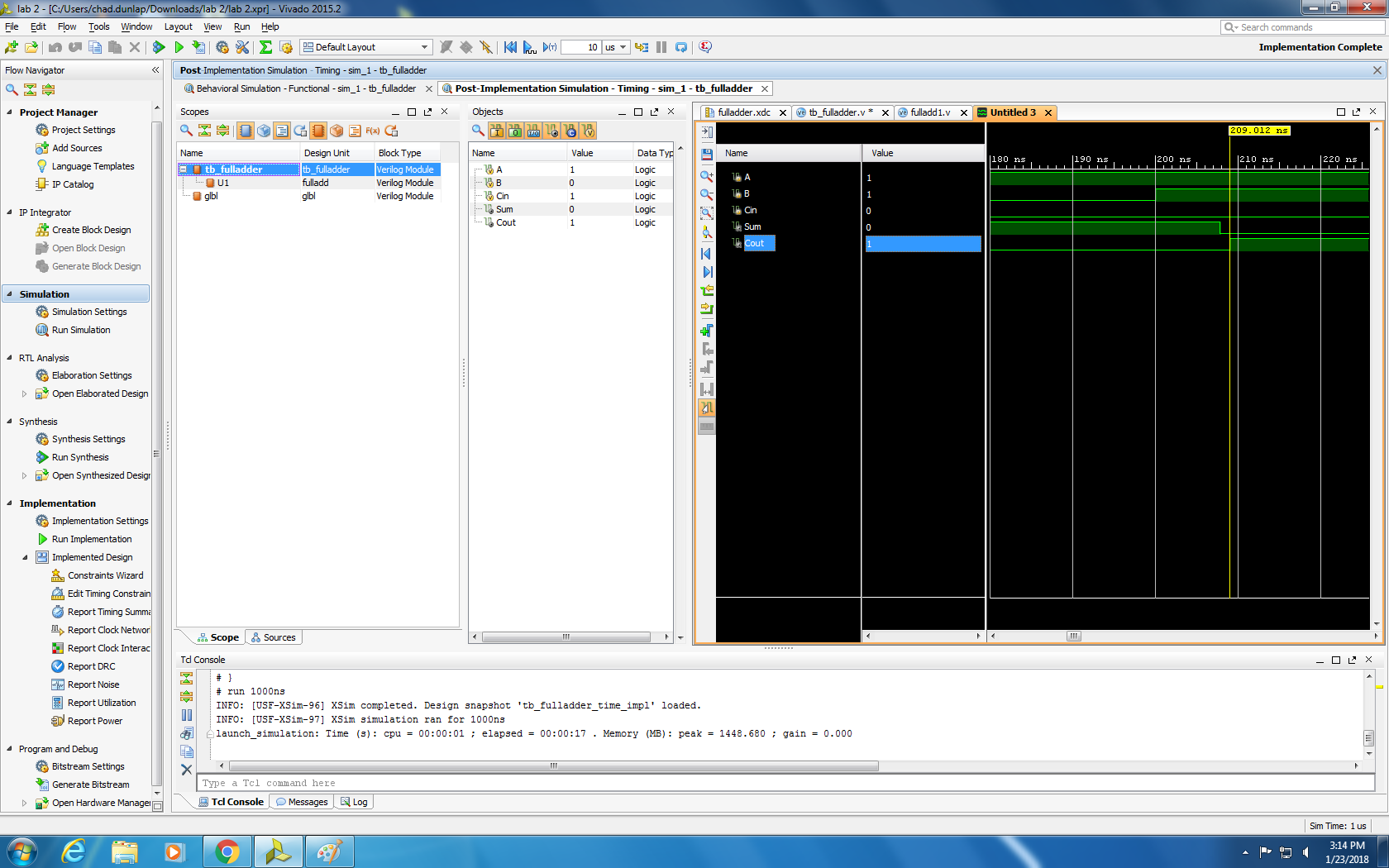
For our first picture this is the test bench behavioral simulation of the Full adder we have captured the picture at the point when A=1, B=0, and Cin=1 and the outputs were Sum=0 and Cout=1, which is what we wanted.



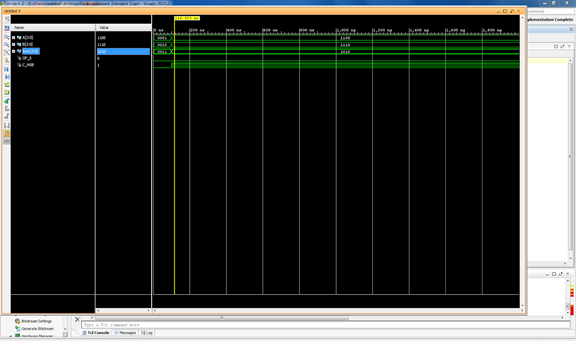
This is a screen-shot of our post implementation timing simulation of the Full adder, and we can see that when A=1, B=0, Cin=0 Sum=1 and Cout will equal 0 which is just the way we need our program to run.



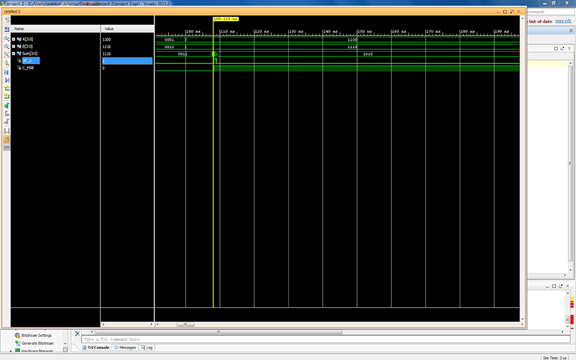
In this last picture of the Full adder it is of the Post implementation timing simulation, but our focus is on the switch from A=1 B=0 to A=1 B=1. Here we notice a glitch of about 9 ns for Cout and 8 ns for Sum these glitches are due to complications in the logic of our design. What we imagine will happen is that when A=1 B=0 and Cin=0 we should see Sum equal 1 and there will be no carry which is Cout and this is indeed what we noticed, but when we switched from B=0 to B=1 the Sum should equal 0 and Cout should equal 1 we can see a delay in both Sum and Cout.



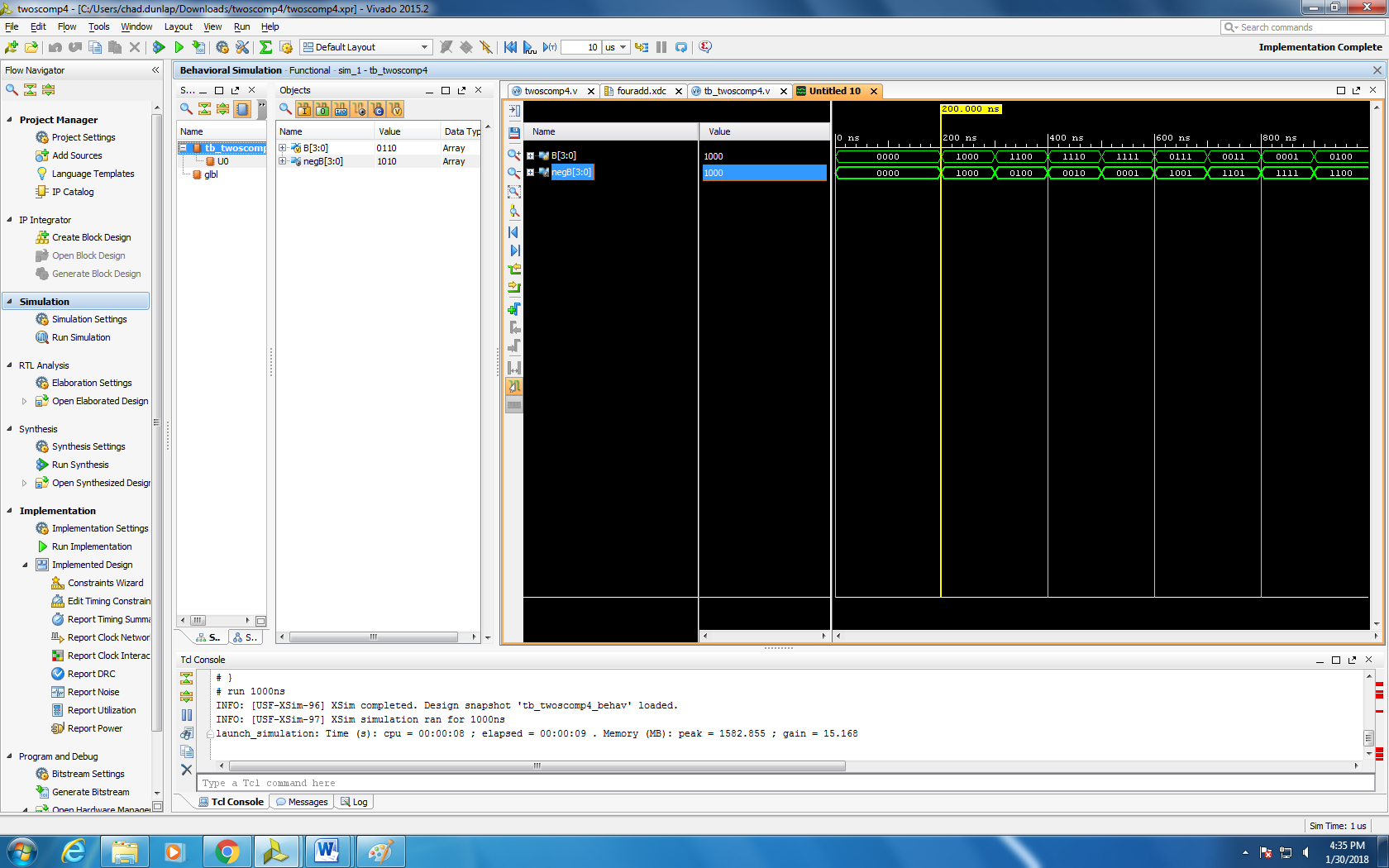
This next picture is the behavioral simulation for the 4-bit adder



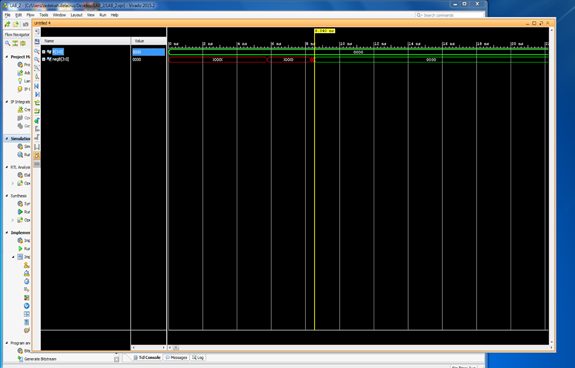
This is a picture of the 4-bit adder post implementation timing simulation. We can see that a glitch occurs at 108 ns when the input is 1100 and 1110 what the sum should read is 1010 but it has a signed bit overflow for a second then it is ok after this. Could be due to complications in the logic again and the timing it takes to switch is not exactly instant.



The next picture is our two’s complement full adder behavioral simulation the picture was cut off but here is where we inserted 16 different combinations to test each one and make sure that the correct out is there. The first 9 combinations shown are correct, as the two’s complement will flip the bits and then add one.



This last picture is the post implementation timing simulation where it displays a glitch in the beginning because the correct values have not been taken in. After some time we see that it goes back to normal.



|  |
| --- |
| Discussion |

Discussion opens with an effective statement on the goals of the lab, backs up the statement concerning appropriate findings, provides a sufficient and logical explanation for the statement, addresses other issues pertinent to the lab.

The goals of this lab was to understand and test a 1-bit full adder, 4-bit full adder, and a two’s complement full adder. We wanted to generate different scenarios and test each one to make sure our Verilog logic was correct. We were able to complete this for all three of the projects. For the 1-bit Full adder we wanted to test the carry output and sum which set us up to increase the full adder to a 4-bit full adder. For the 4-bit adder we wired the Cout to the next Cin adder and completed the circuit an additional 3 times. The last was to implement a 2’s complement full adder and test 16 different cases. Each of the three projects had some type of glitch associated with the complexity of the logic and showed us that in the real world the system will not be perfect so we have to account for those errors accordingly. Coming into this lab we were unsure what a two’s complement was, but after completion we understand very clearly.

|  |
| --- |
| Conclusion |

Conclusion convincingly describes what has been learned in the lab.

In this lab we learned how to design and test a full adder, a 4-bit adder, and finally the two’s complement. We started out just adding A and B, and looking at the three different outputs. From there we constructed a 4-bit adder, using the Cout output from the previous adder, which would then become the Cin input into the next adder. Then we moved on to the more complicated two’s complement, which utilizes both the full and the 4-bit adder. The two’s complement basically flipped the bits then added a one. The new bit would then be the negative value of the flipped bit. This is very useful when minusing bit because it gets rid of a negative zero which in nature doesn’t exist.

|  |
| --- |
| References |

Include your references here

ECE260 Lab 2 manual

Wakerly Digital Design Principles and Practices