Talal Jawaid

CPE 166

3/12/18

LAB 2: Binary Combinational Array Multiplier Design, Sequential Multiplier Design and LCD Interfacing

Professor Pang

**Part 1** – **Sequential Multiplier Design**

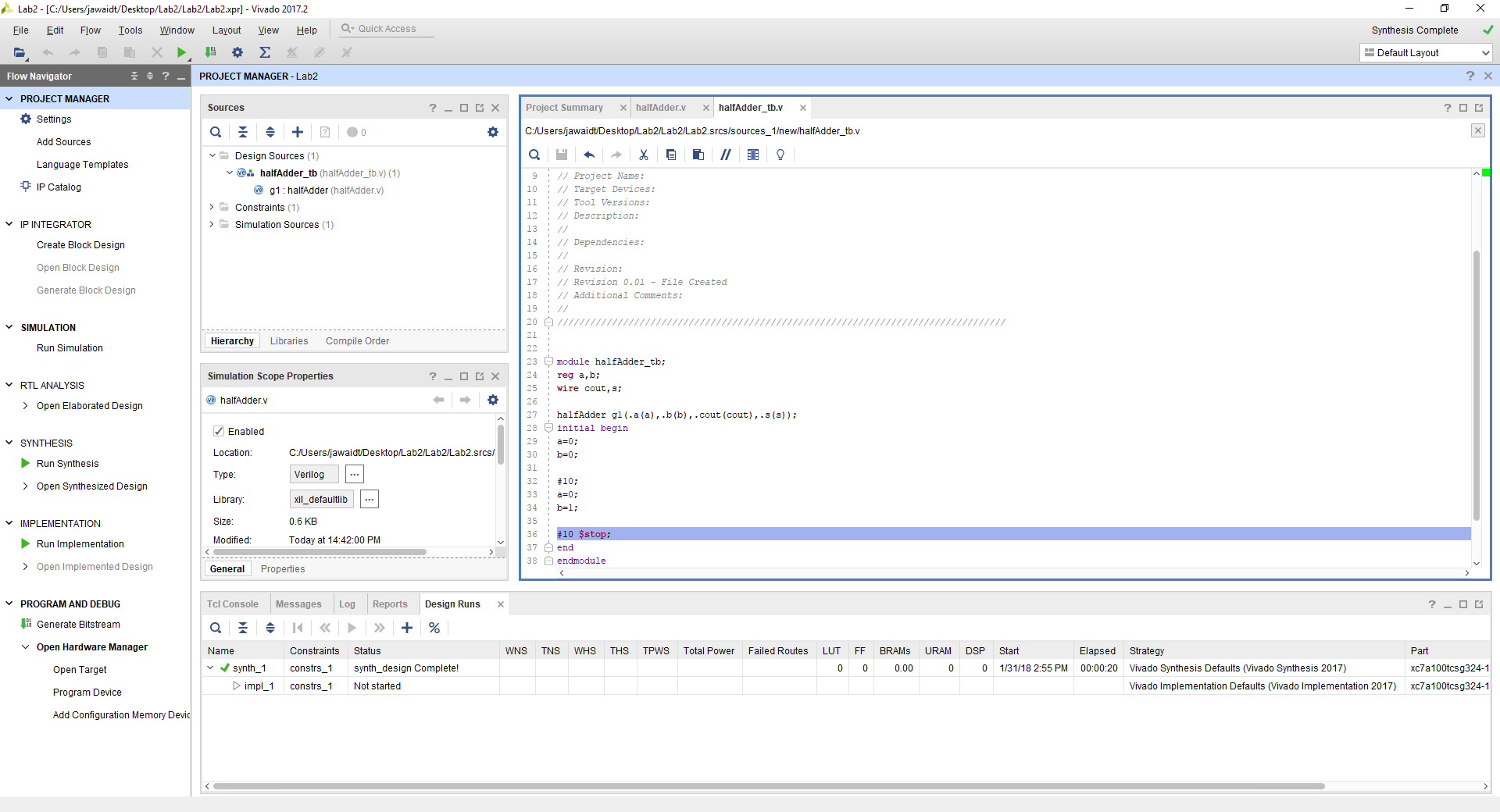
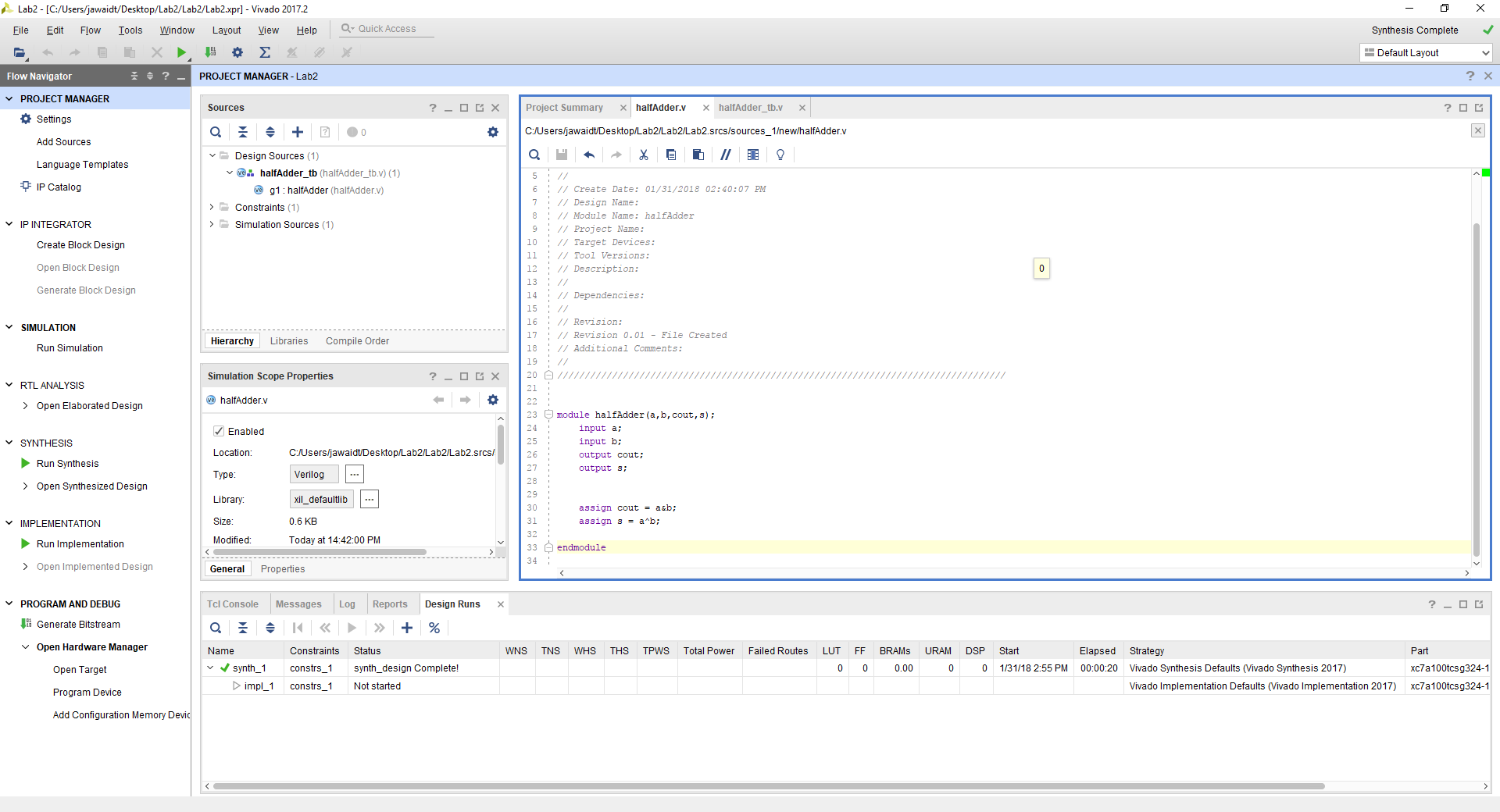
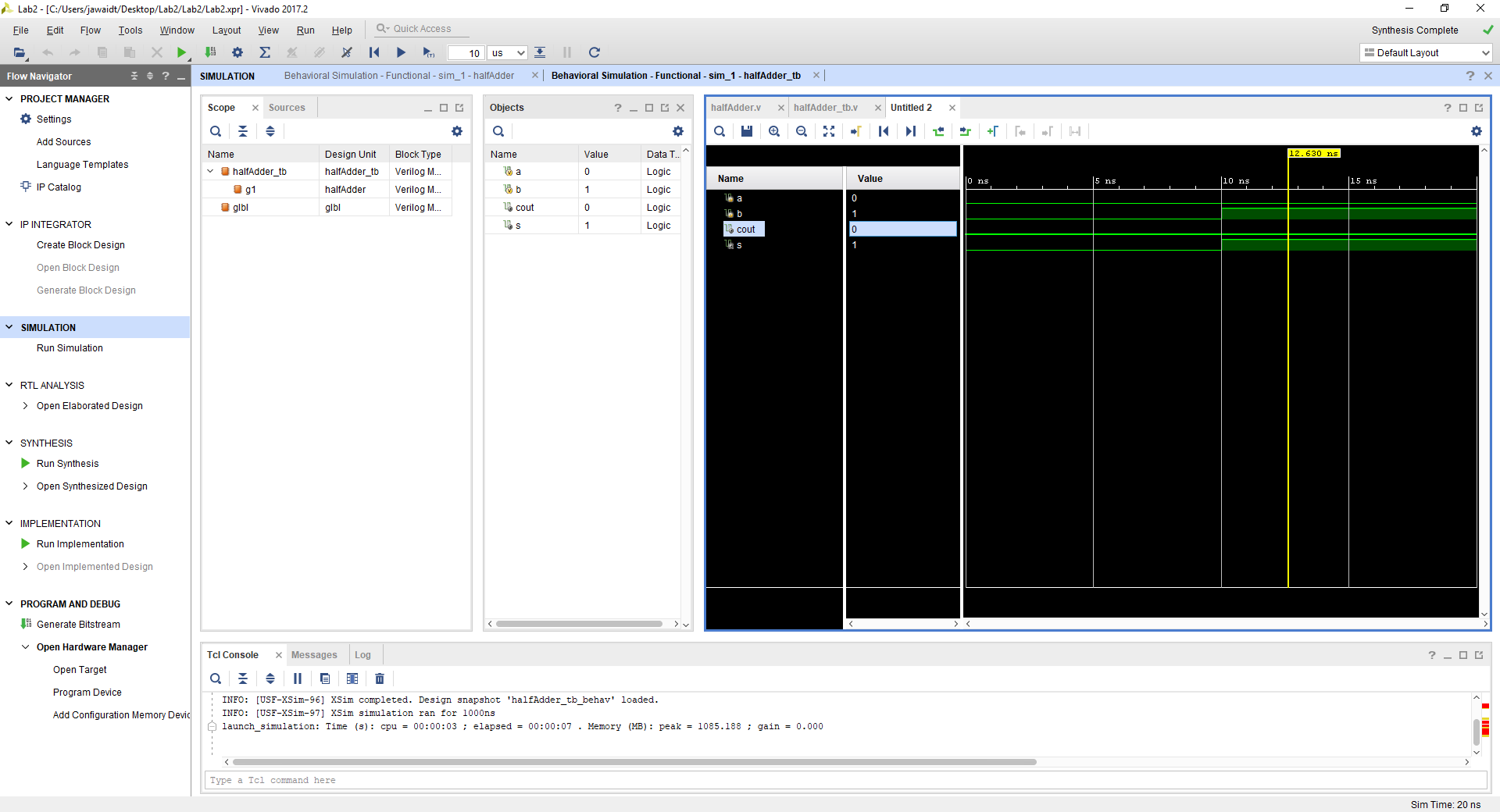
**Description**

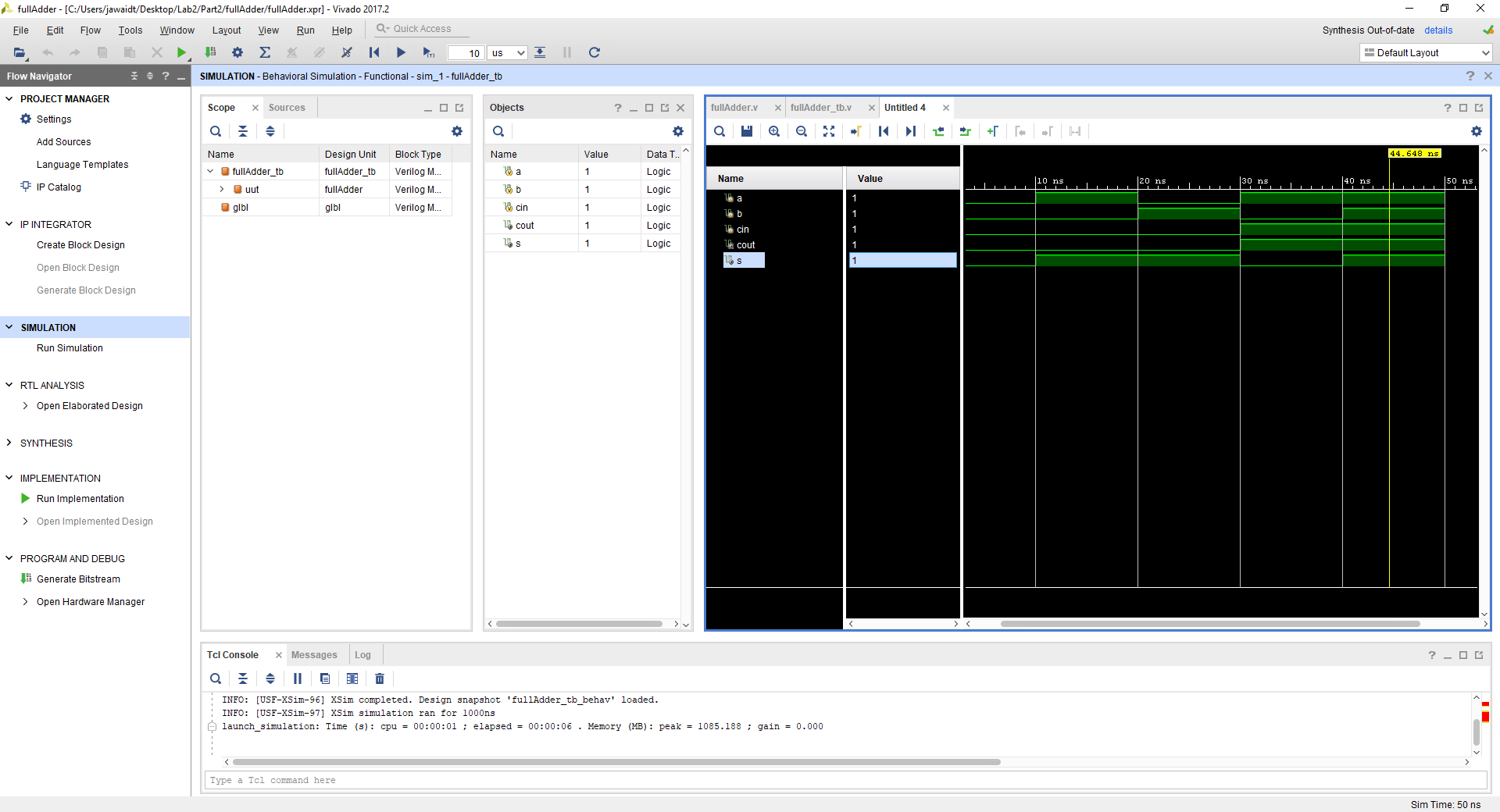
In this part of the lab, we had to implement a half adder, full adder, a four by four combinational multiplier Verilog code and test benches for each of those.

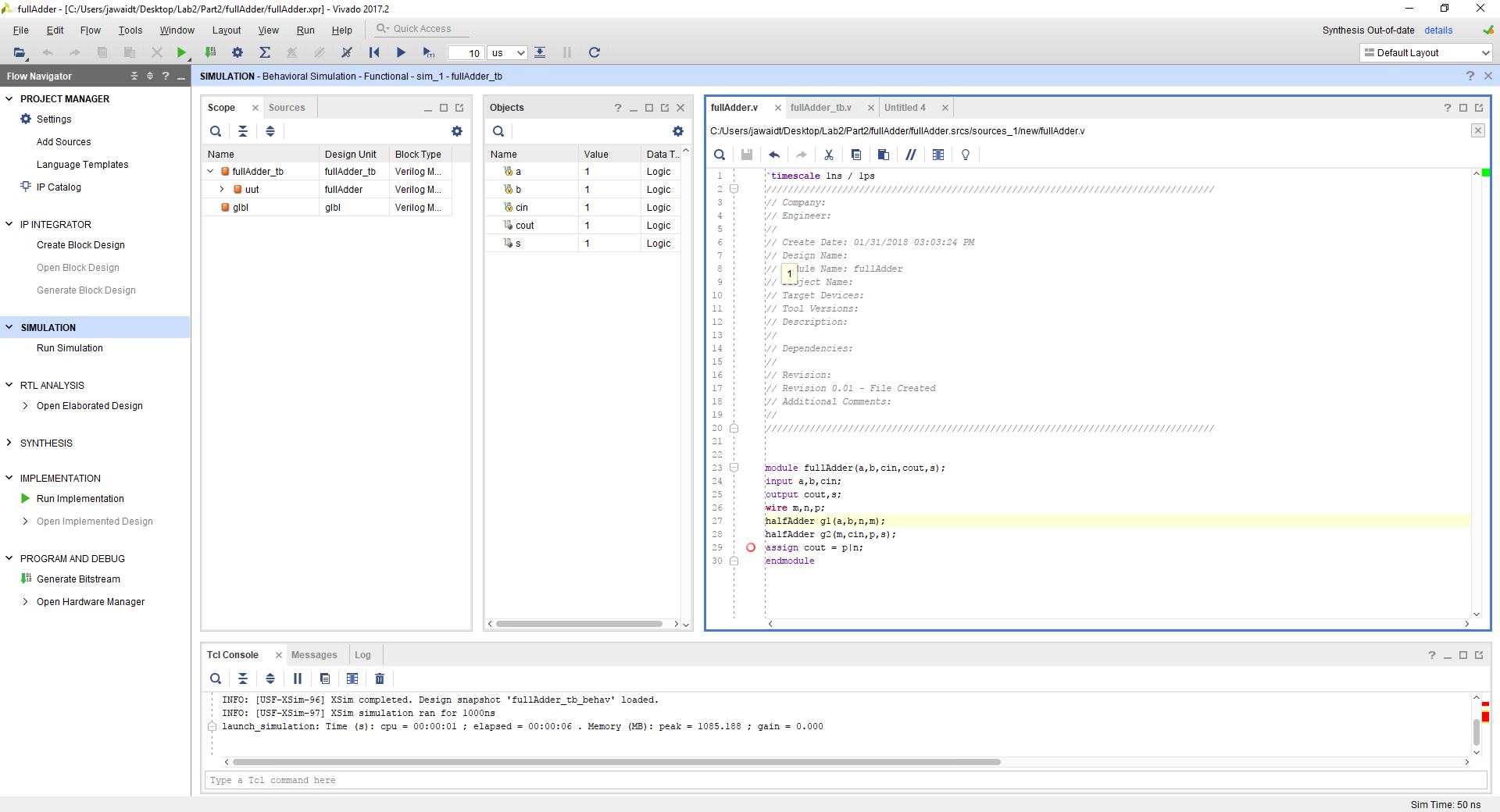
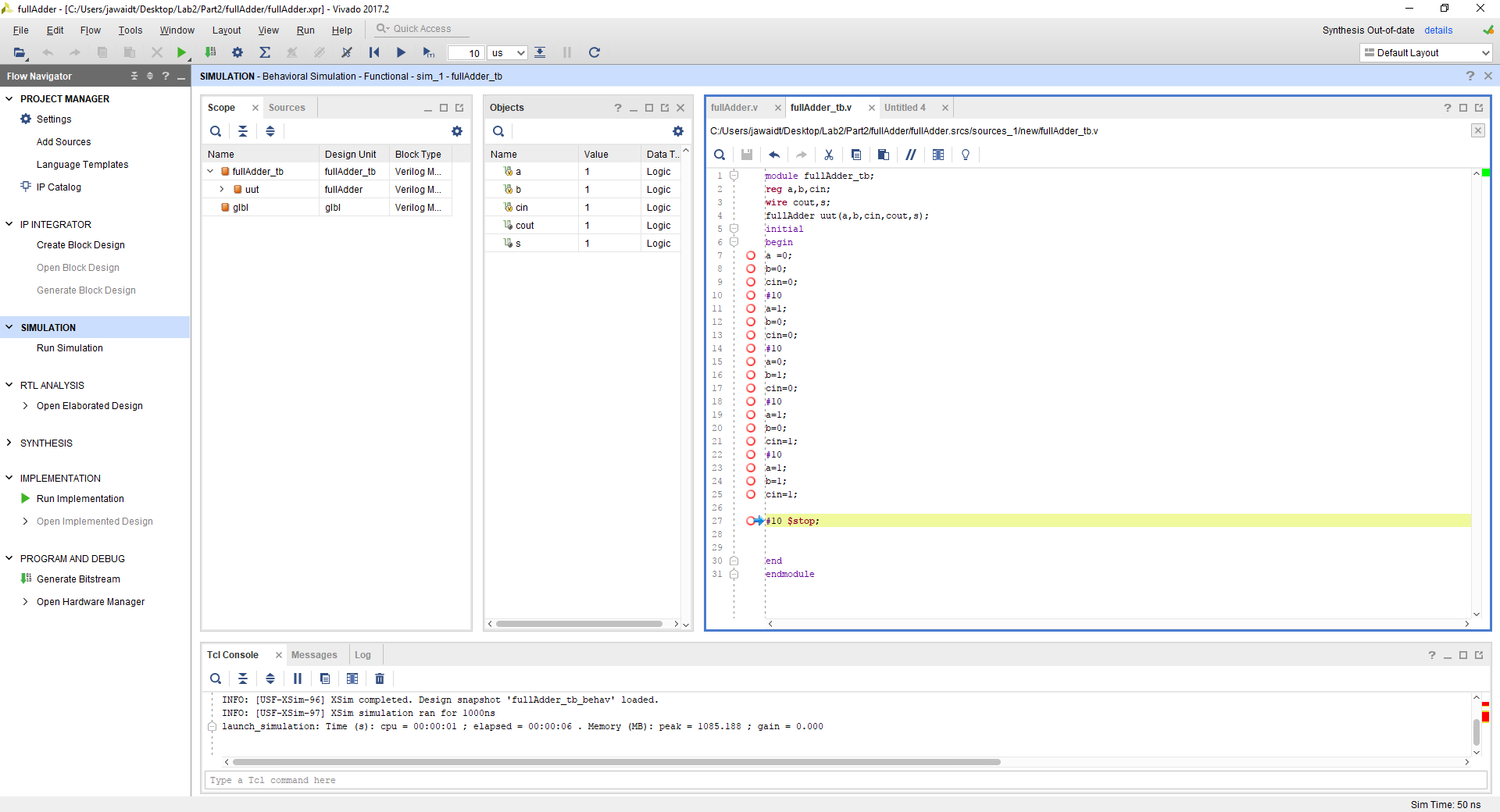
**Problem Definition (Procedure)**

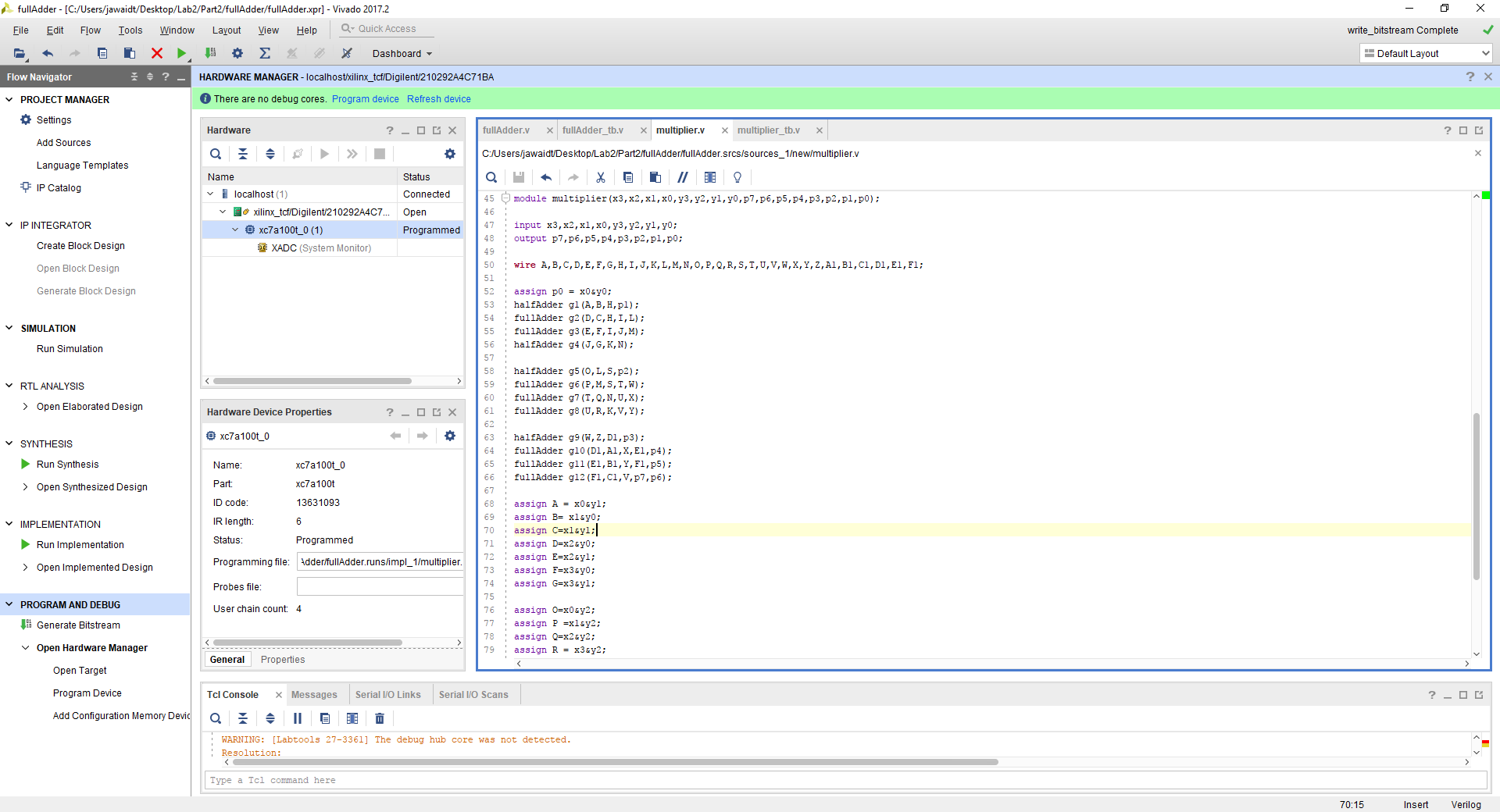
First we took the code that was provided to us for the half adder, full adder, and four by four circuit and implemented it in Verilog HDL. We used the Vivado IDE and used it to test the code and run it on our boards.

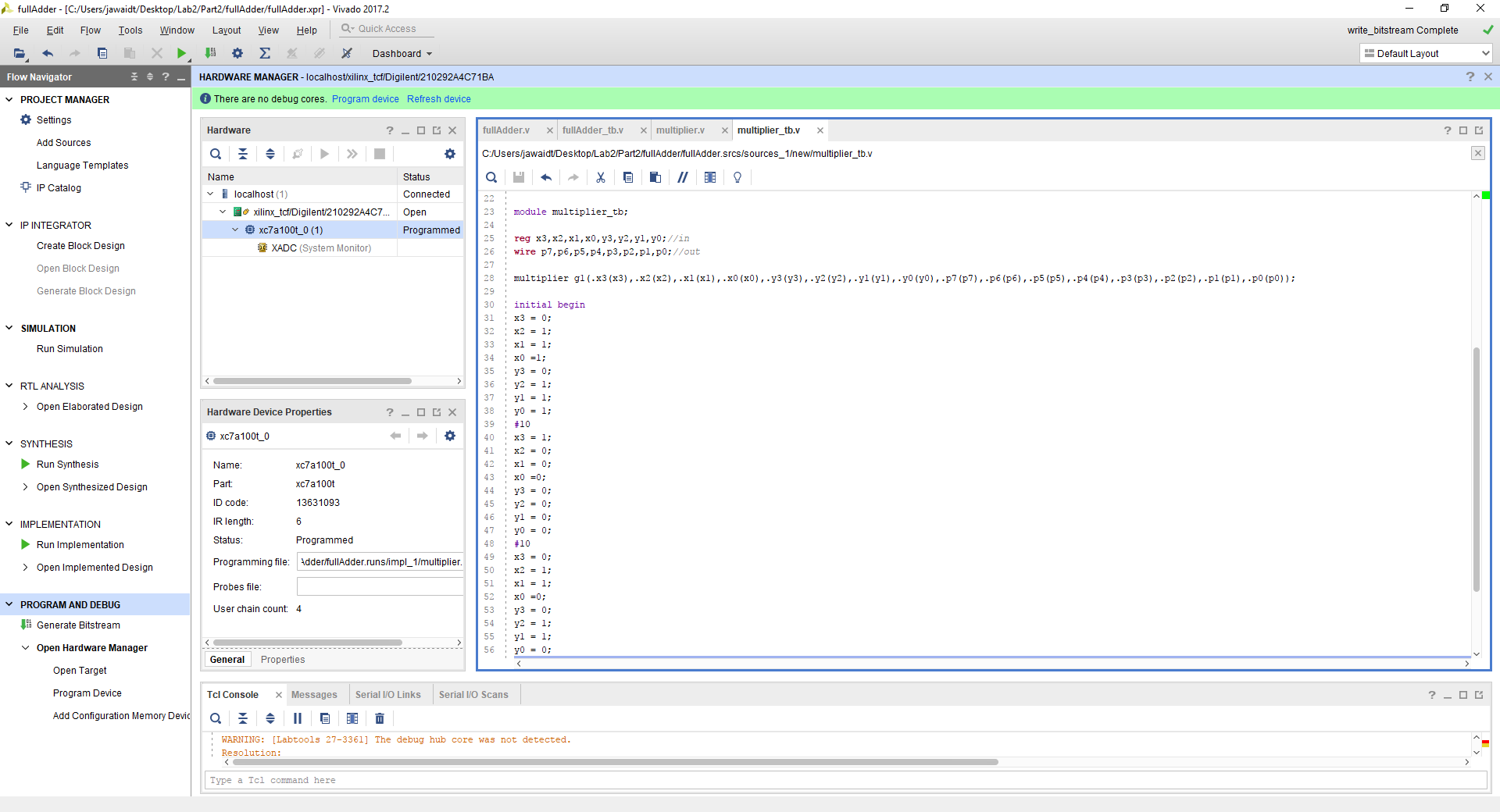
Using the given circuit design, we wrote out on the paper, what each component would be referred to in the code. We assigned these names so that it would be simpler to code in Verilog. Once we did that, we determined whether each variable would be an input, wire, output, or a type of logic gate. After that it was a matter of simply assigning values to each output variable and wire, and then using a test bench or board to test different values for the inputs. We were successfully able to get it working.

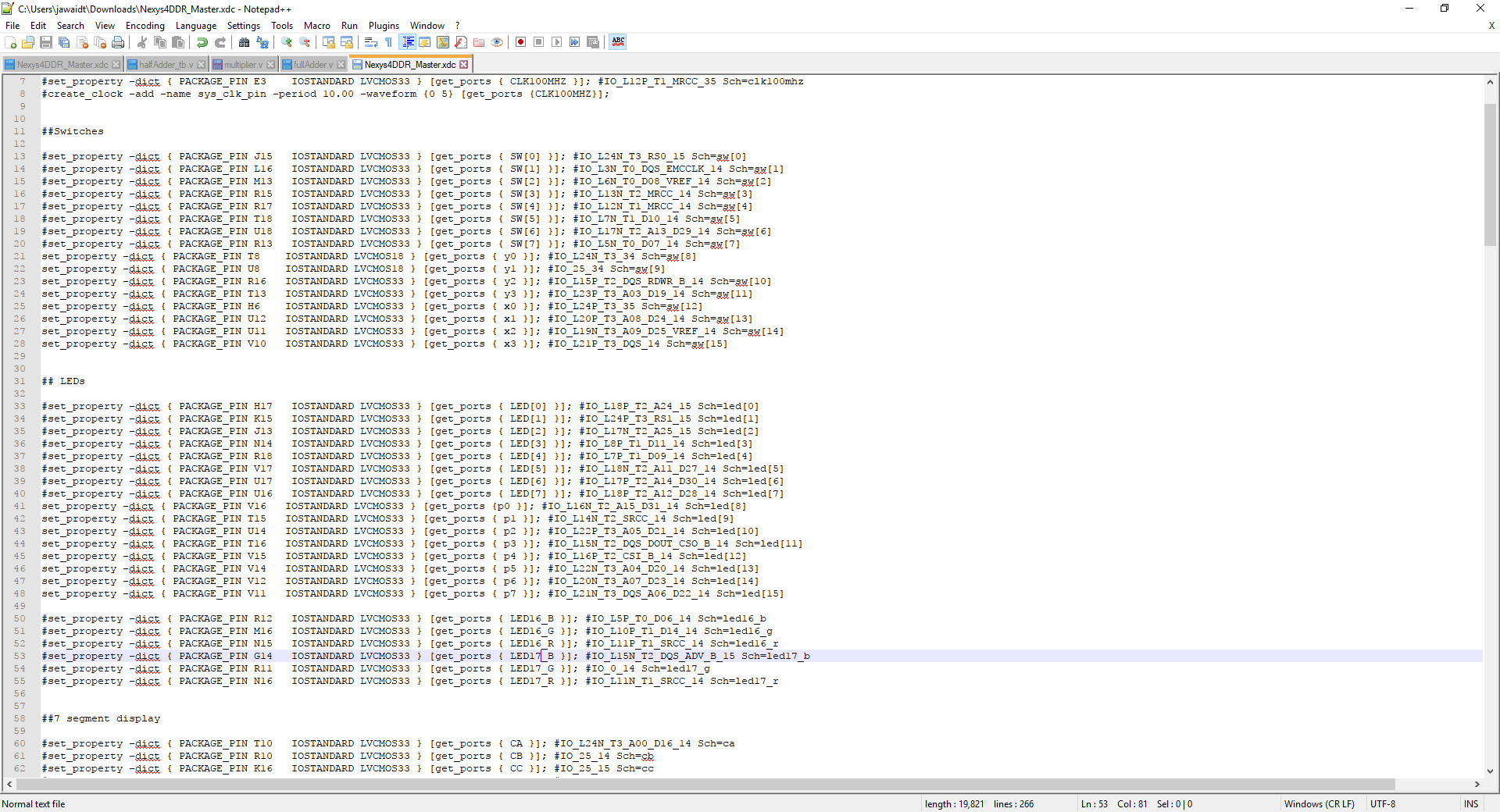
**Engineering Data**











**Conclusions**

In conclusion, we were able to succesfully implement the 4 by 4 binary combinational multiplier. This was an easy task as all the code was provided to us and all we had to do was import the code into Vivado and run it.

**Part 2 – Implementing a 4 X 4 Combinational Multiplier**

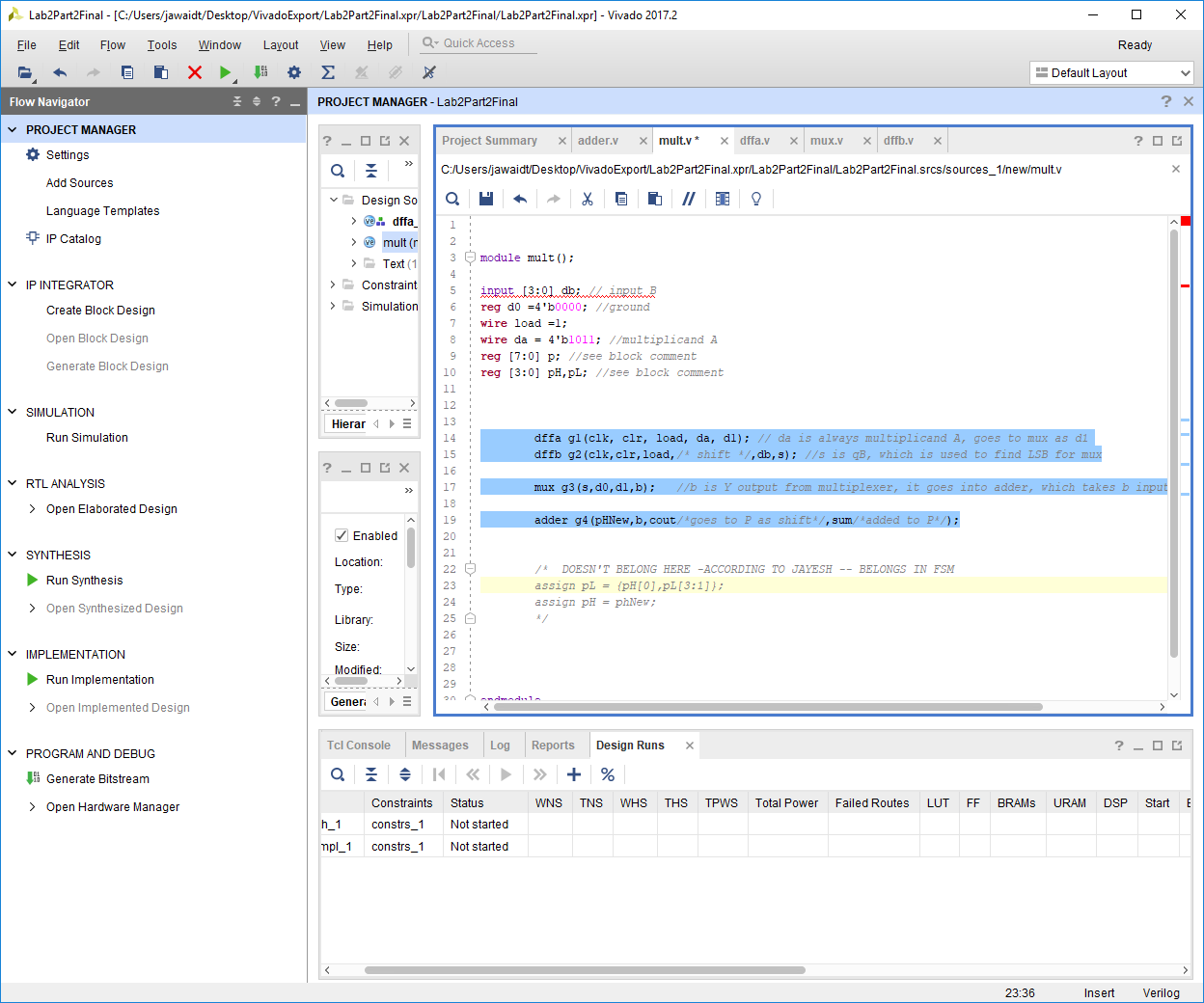
**Description**

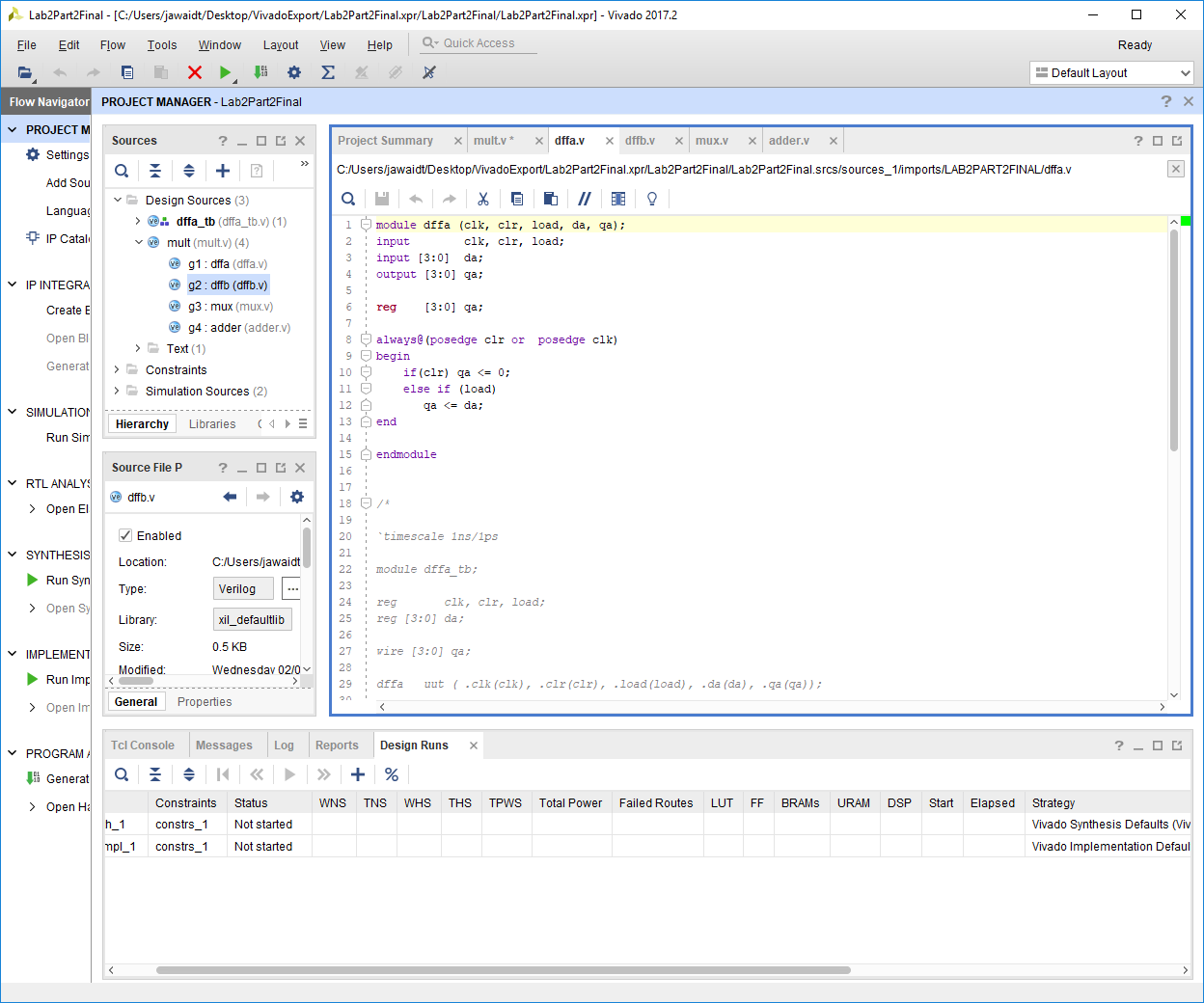
In this part of the lab we had to implement a four by four combinational multiplier. This was done by using the diagram provided to us along with the design specifications. This was a difficult task as we did not know how to bring together all of the different components within one module.

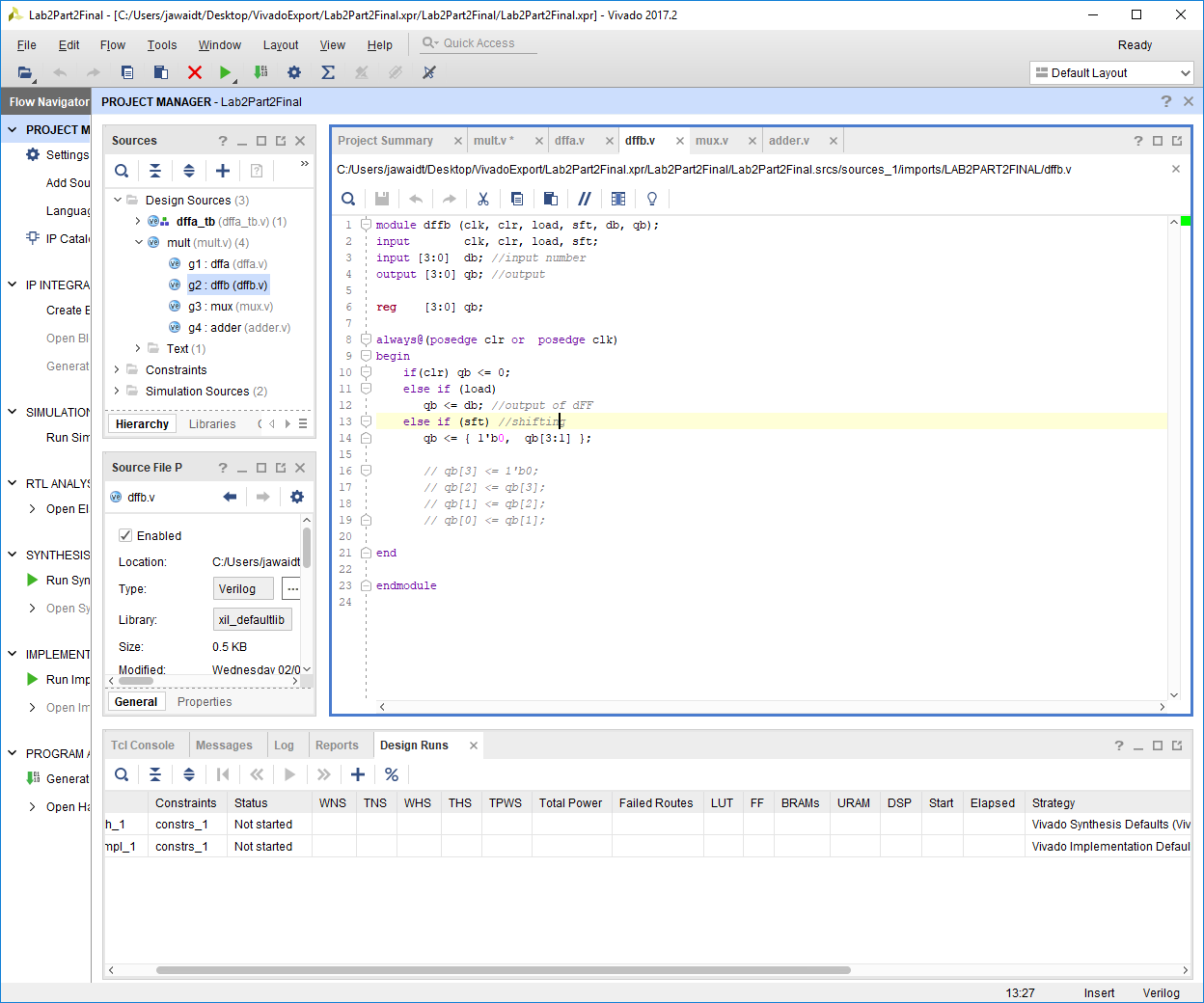
**Procedure**

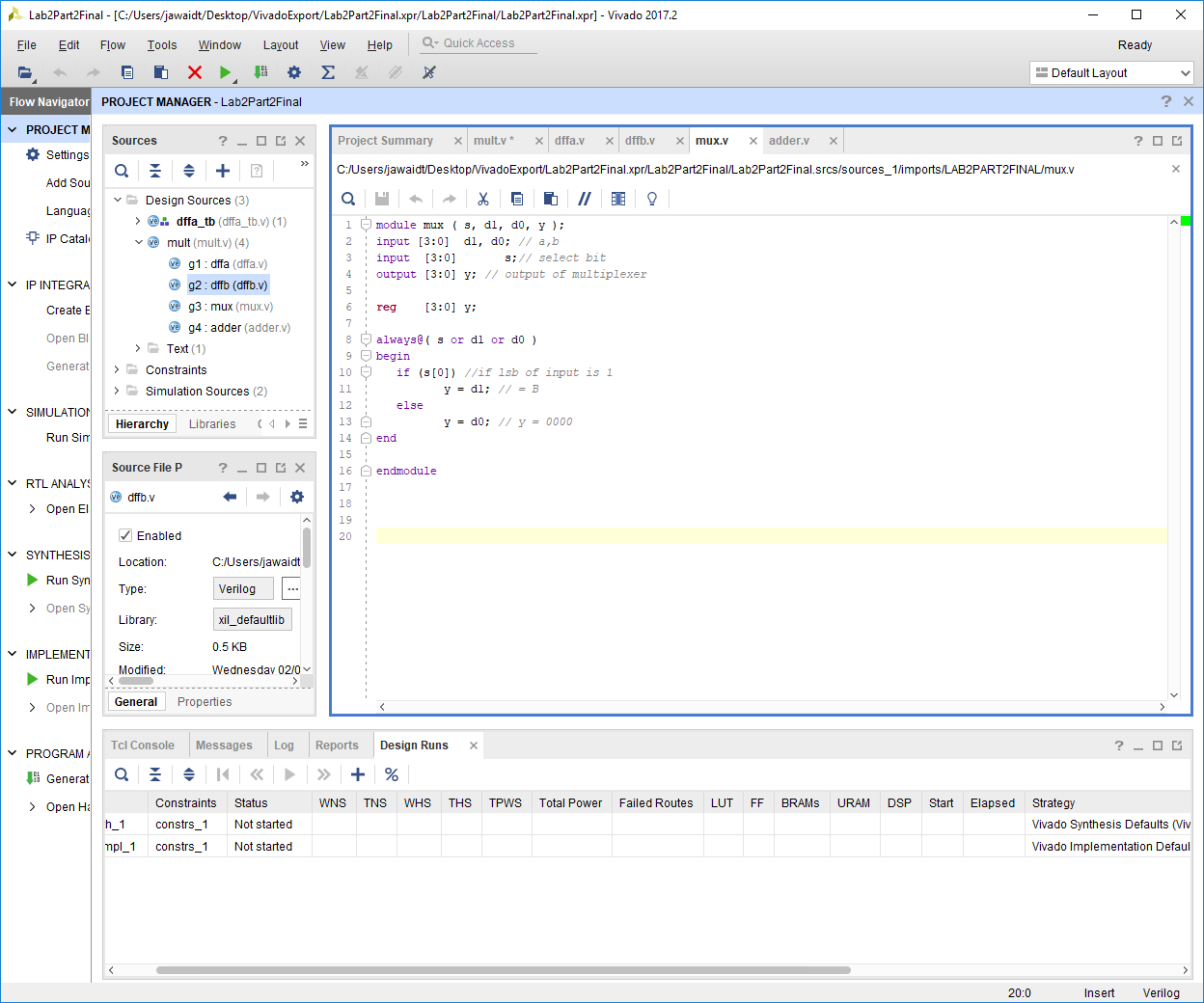
We used the design specifications as a starting point for knowing what our inputs and outputs are. We had to implement a 4 bit multiplicand operand, a 4 bit multiplier operand, a RESET, a START, and a CLK signal. The outputs were supposed to be simply P, but eventually they included P high and P low as well since it was easier to catch errors if we knew what Ph and Pl were. We implemented modules for mult, dffa, dffb, mux, fsm, and top. We then only had to implement a test bench for top to bring it all together.

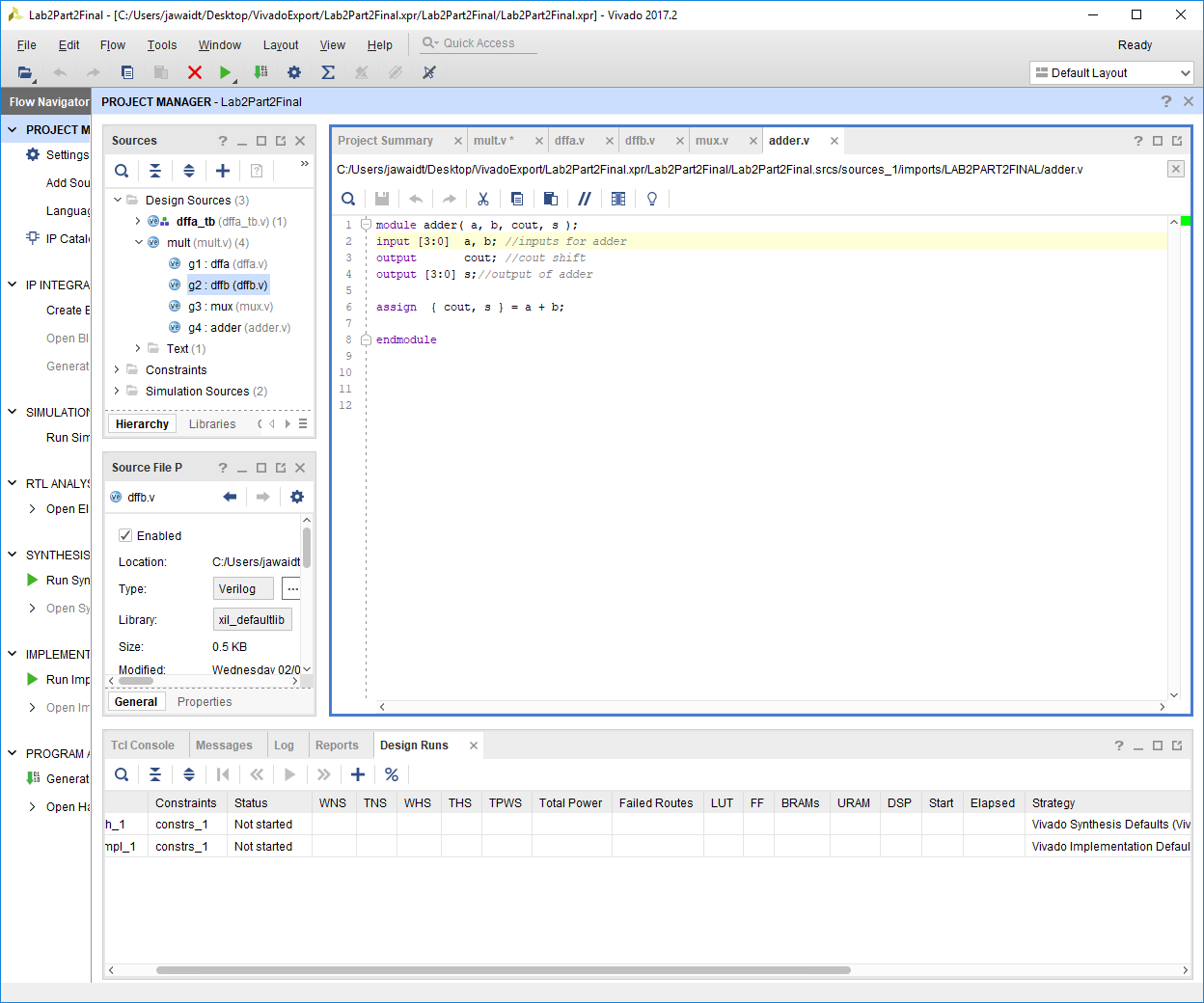
**Engineering Data**

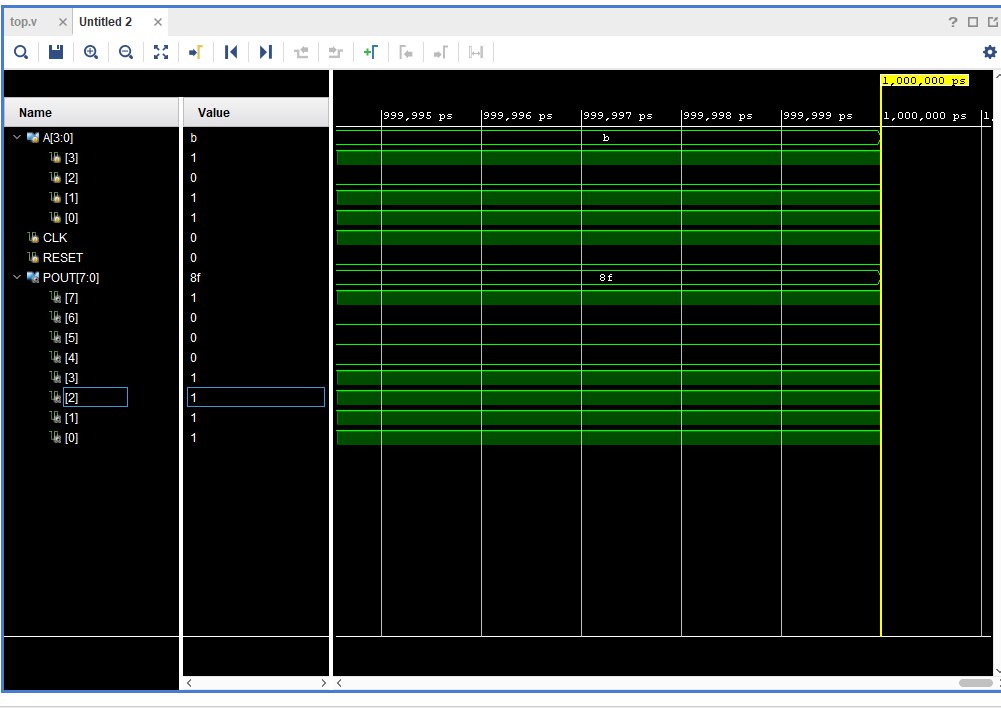


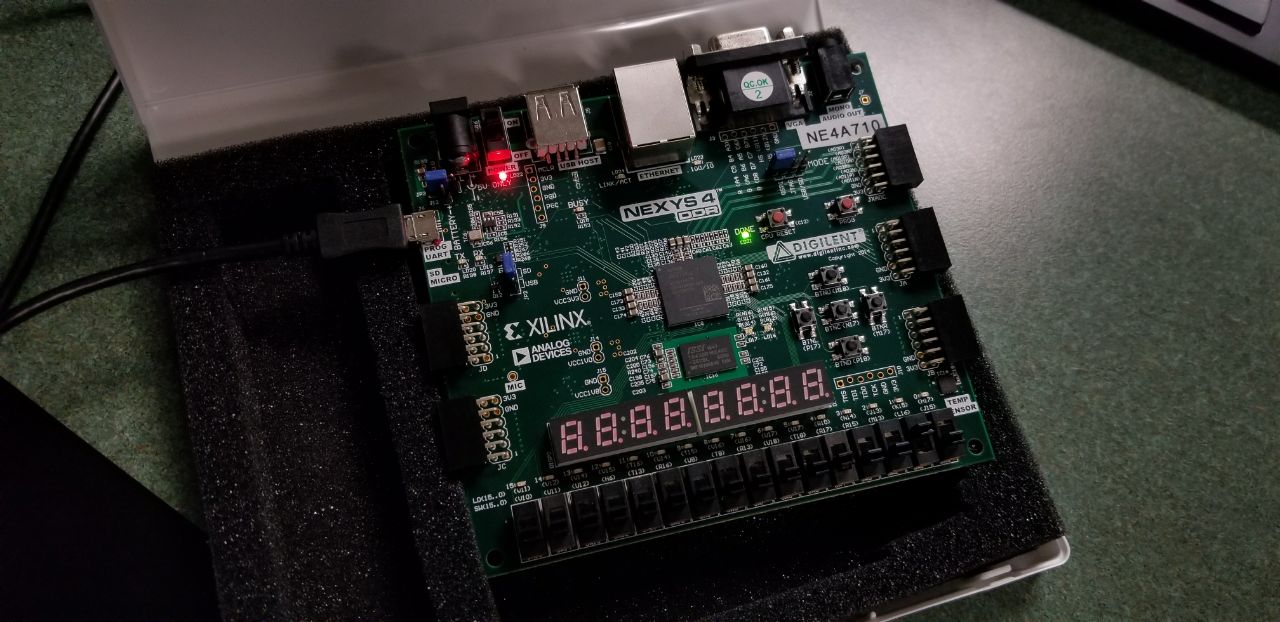


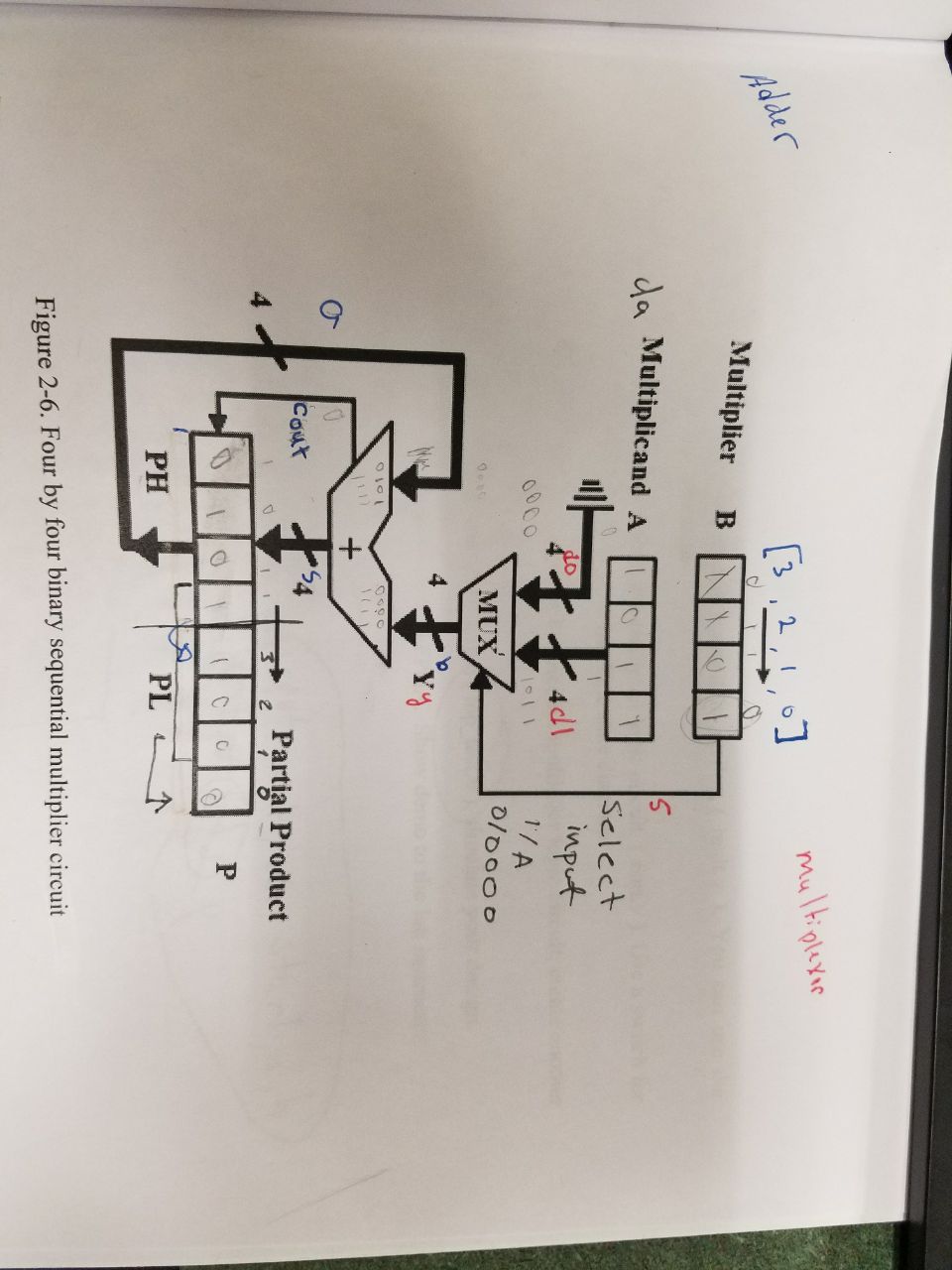










****

**Conclusions**

In conclusion, we were able to successfully get the combinational multiplier working. This was a tough task as it required us to assign variables to several different points of the circuit diagram but it was relatively simple in terms of complexity. It was tough to determine the different modules we would need and the best way to bring them all together, but use of a fsm and a module to control it were enough to get our code to work. Through the use of the board and test bench for the top module, we are able to confirm that our code works successfully as a combinational multiplier.

**Part 3 Decoder**

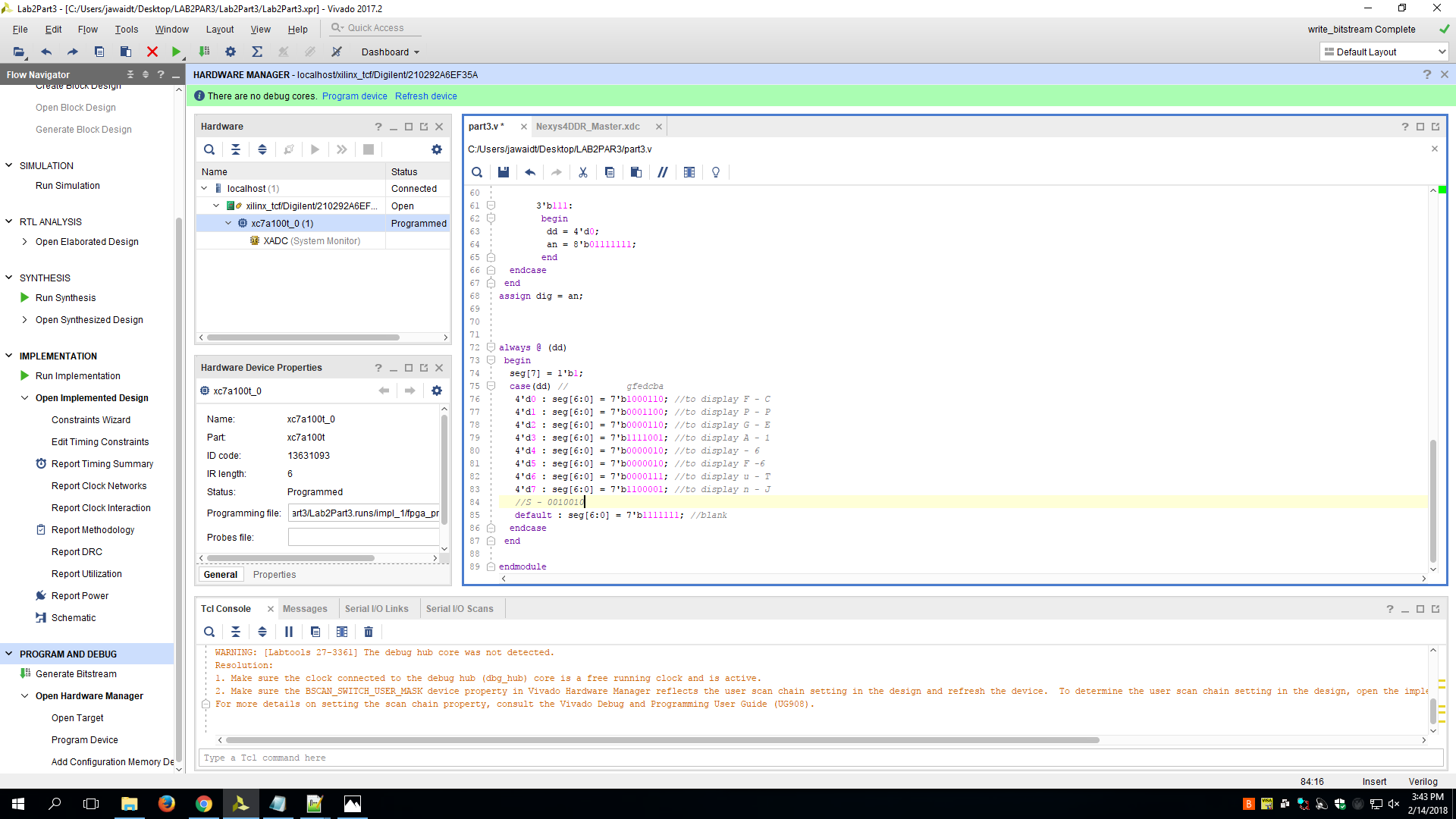
**Description**

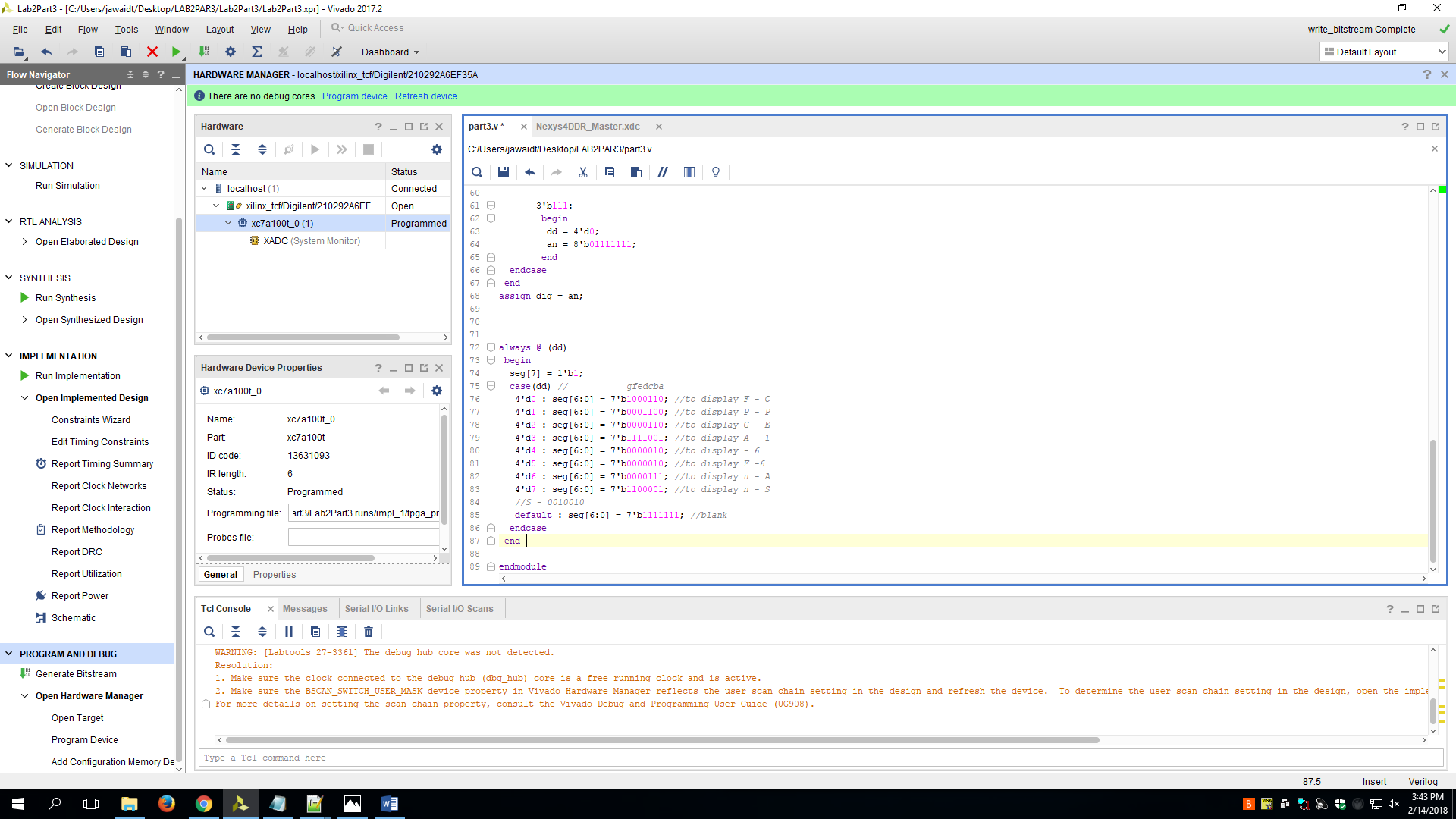
In this part of the lab, we had to implement the given code and use it to write our name. This part was simple as we only had to create a table for what our letters would be on a seven segment display and then use that to rewrite the values already provided. The values already provided spelled out FPGA FUN.

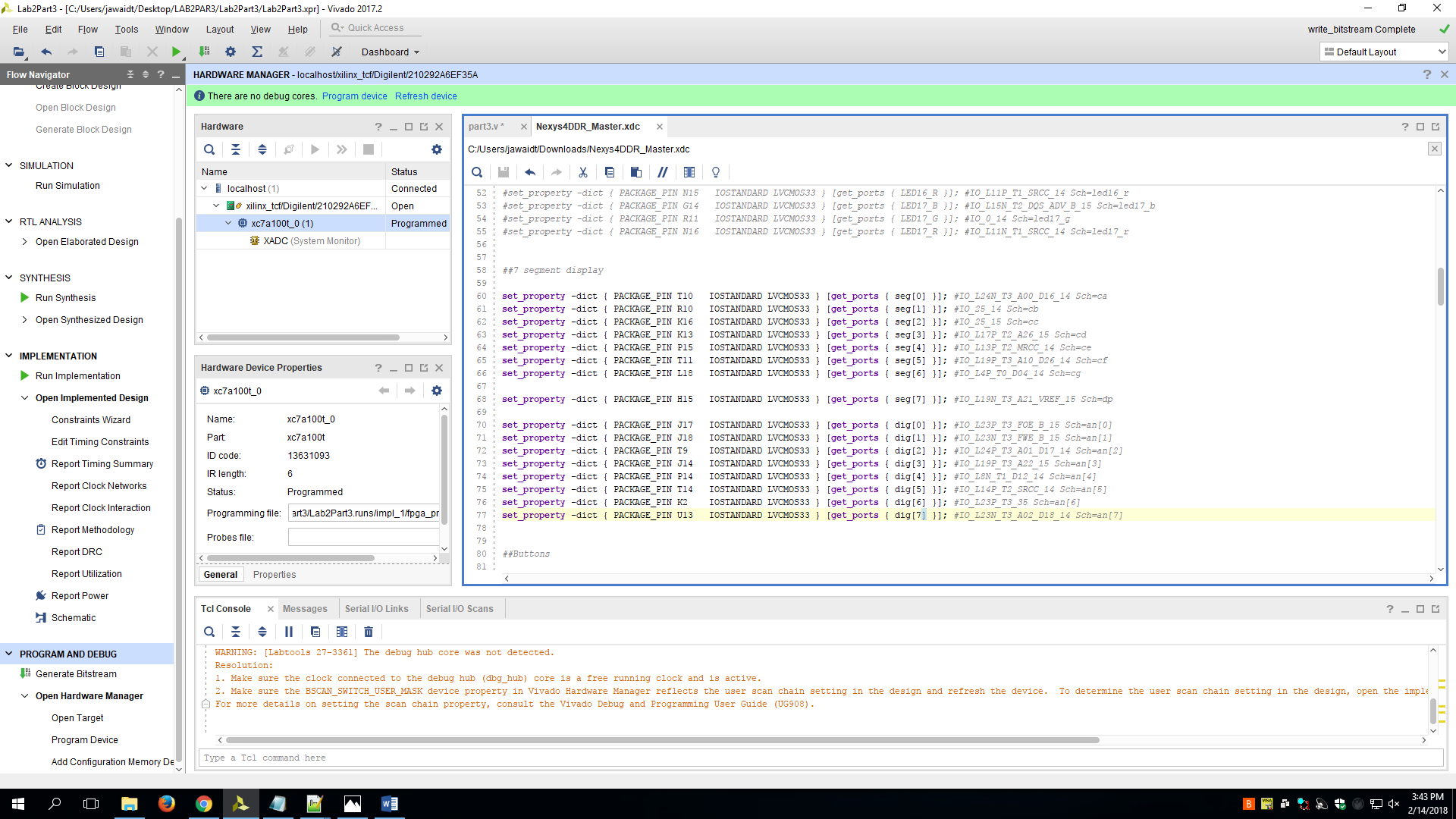
**Procedure**

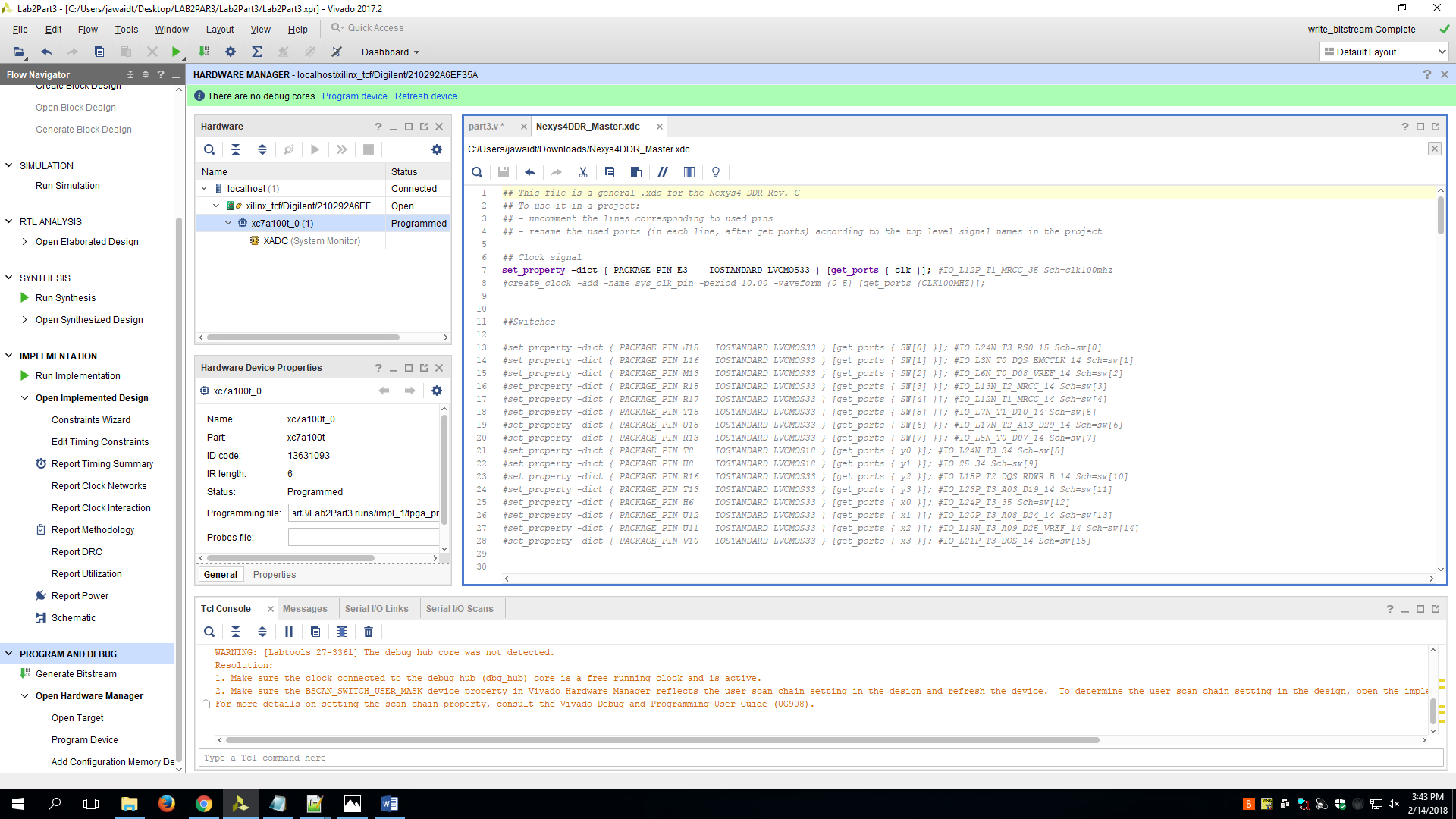
Using a table, we determined what our CPE 166 + Initials would be on a seven segment display. We split the display from a through g and then we assigned zeroes to the ones that needed to be lit up. This was a very simple task.

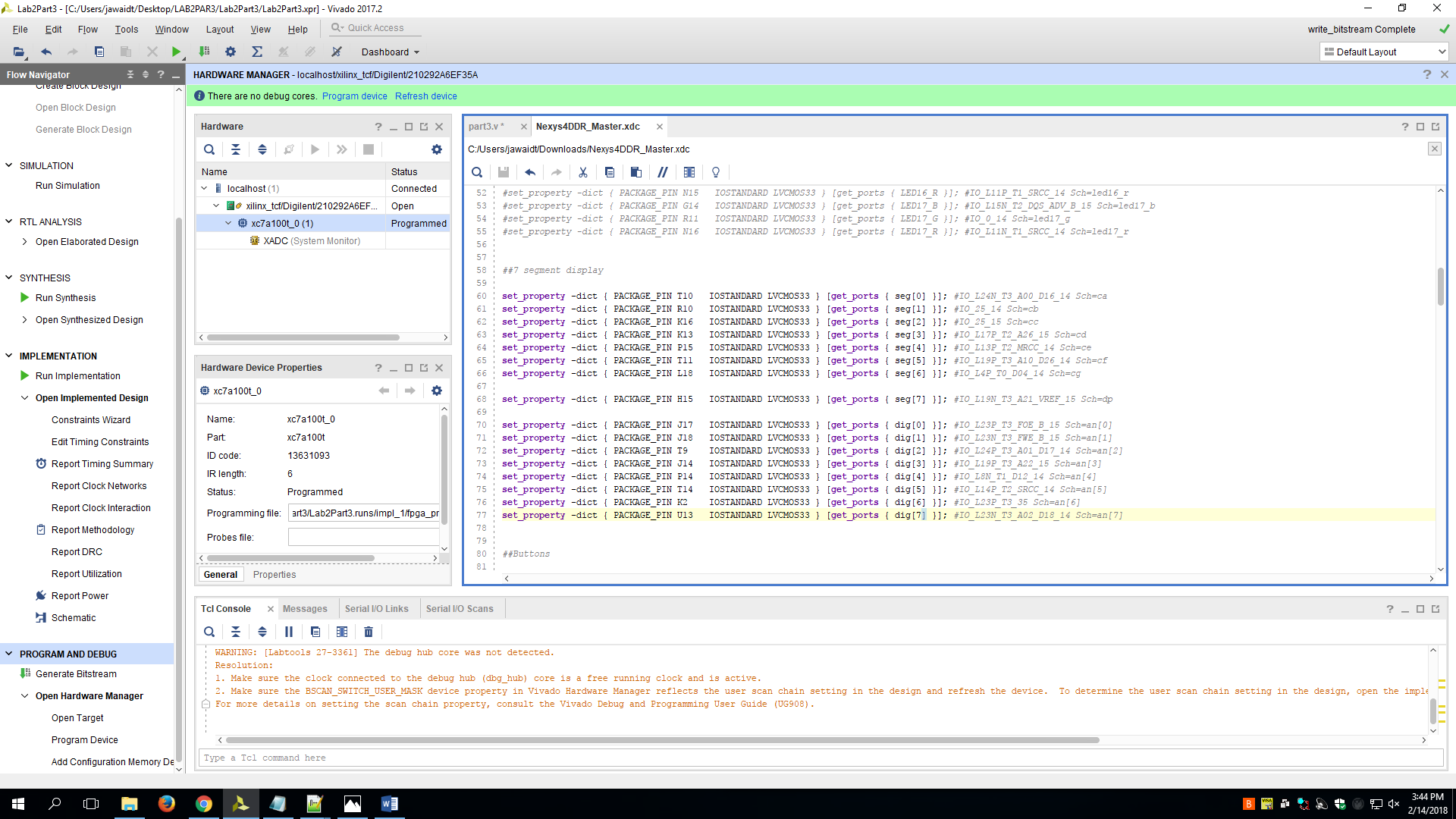
**Engineering Data**

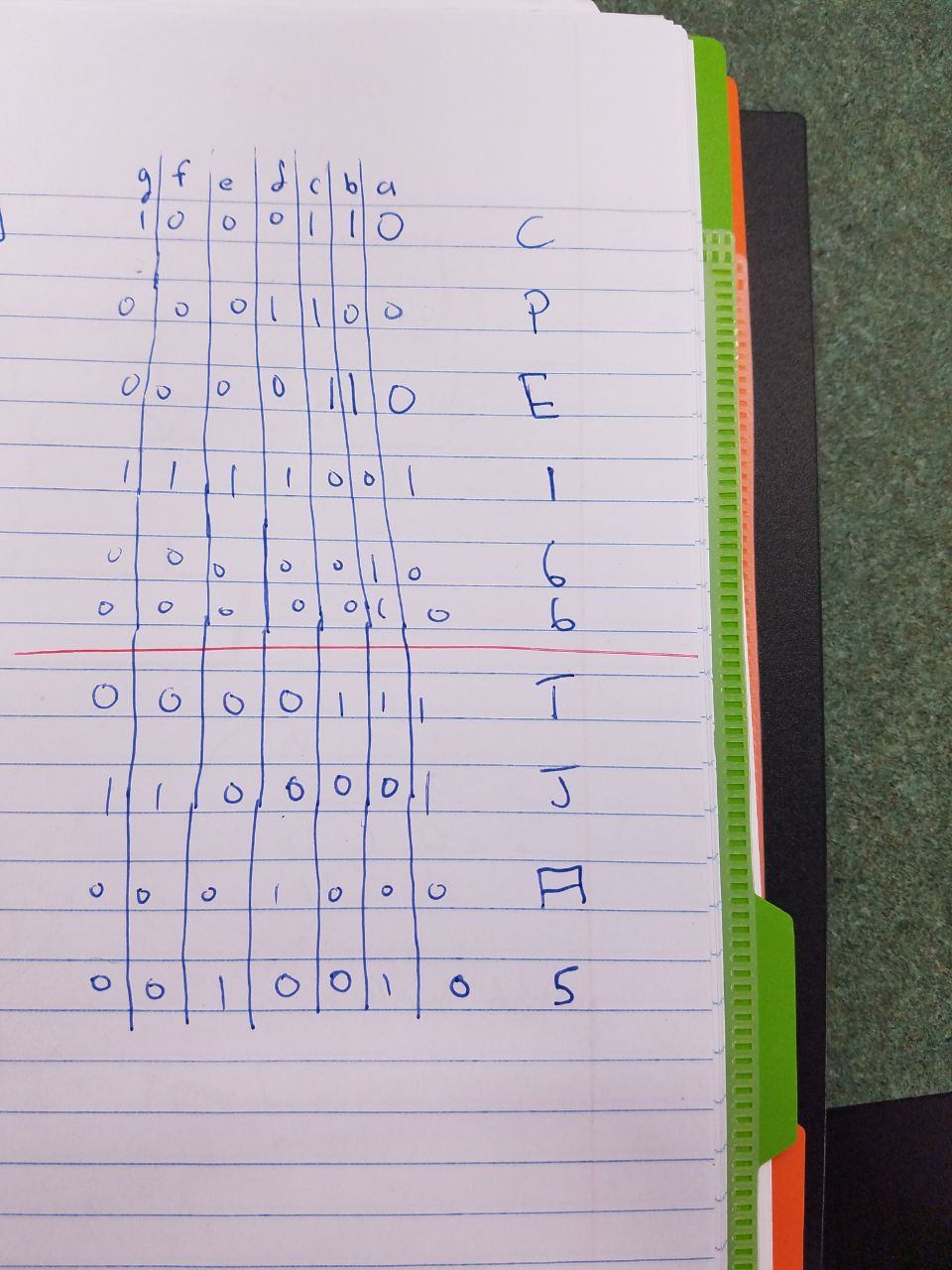












**Conclusions**

In conclusion, this part was very simple to do as we only had to change 8 seven bit values in the provided code. All we really had to was import the code and change those values. Since some of the letters in FPGA FUN are the same as in CPE166TJ, we reused the values already provided to us for those letters.