**MECHTRON 3TB4 Lab 1 Prelab**

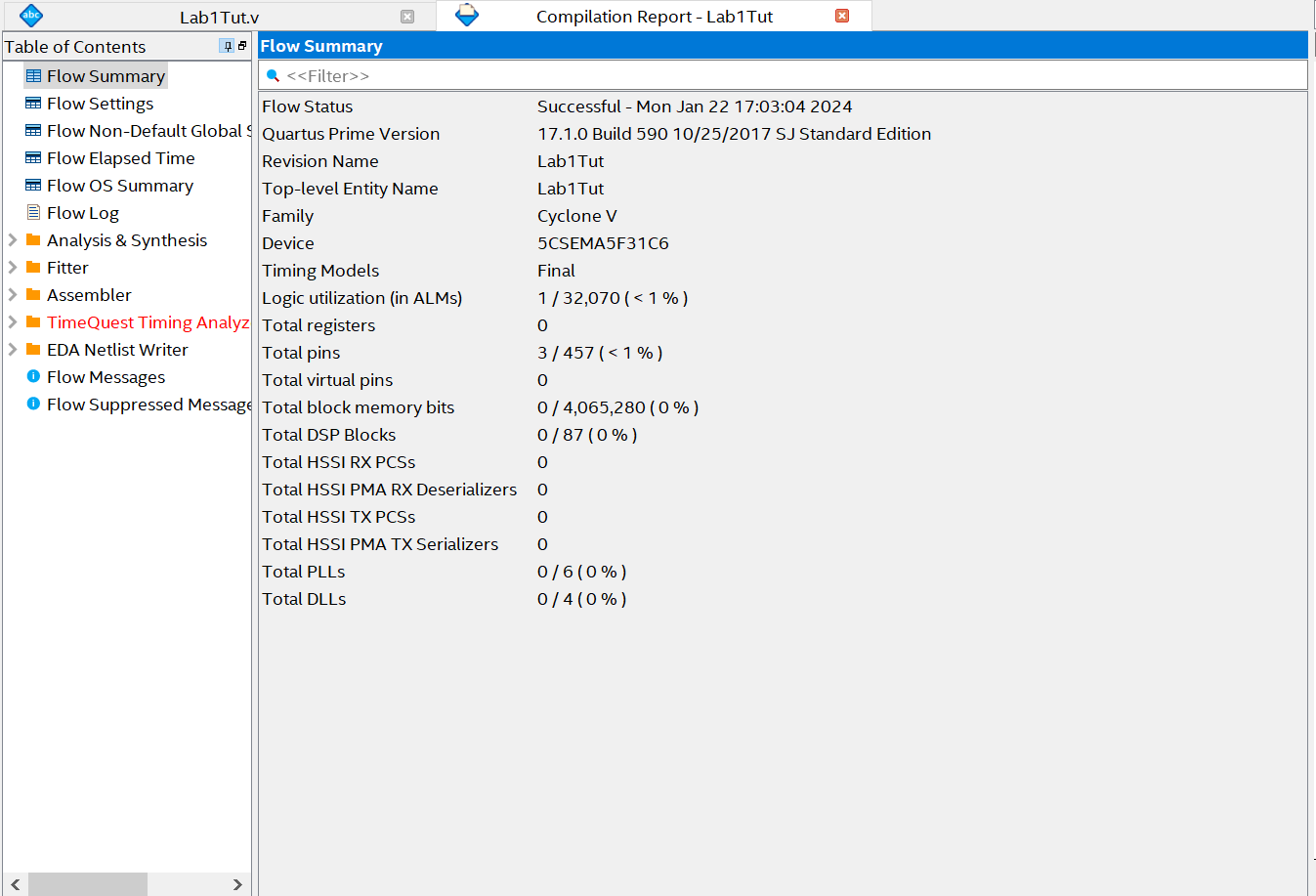
**Part 1:**

During the Lab 1 Tutorial, we installed Quartus Prime and created our first basic project to get familiar with the software. For our very first project, we implemented a simple XOR gate. We learned how to create the Verilog script to implement the logic for the XOR gate.

A white background with black text

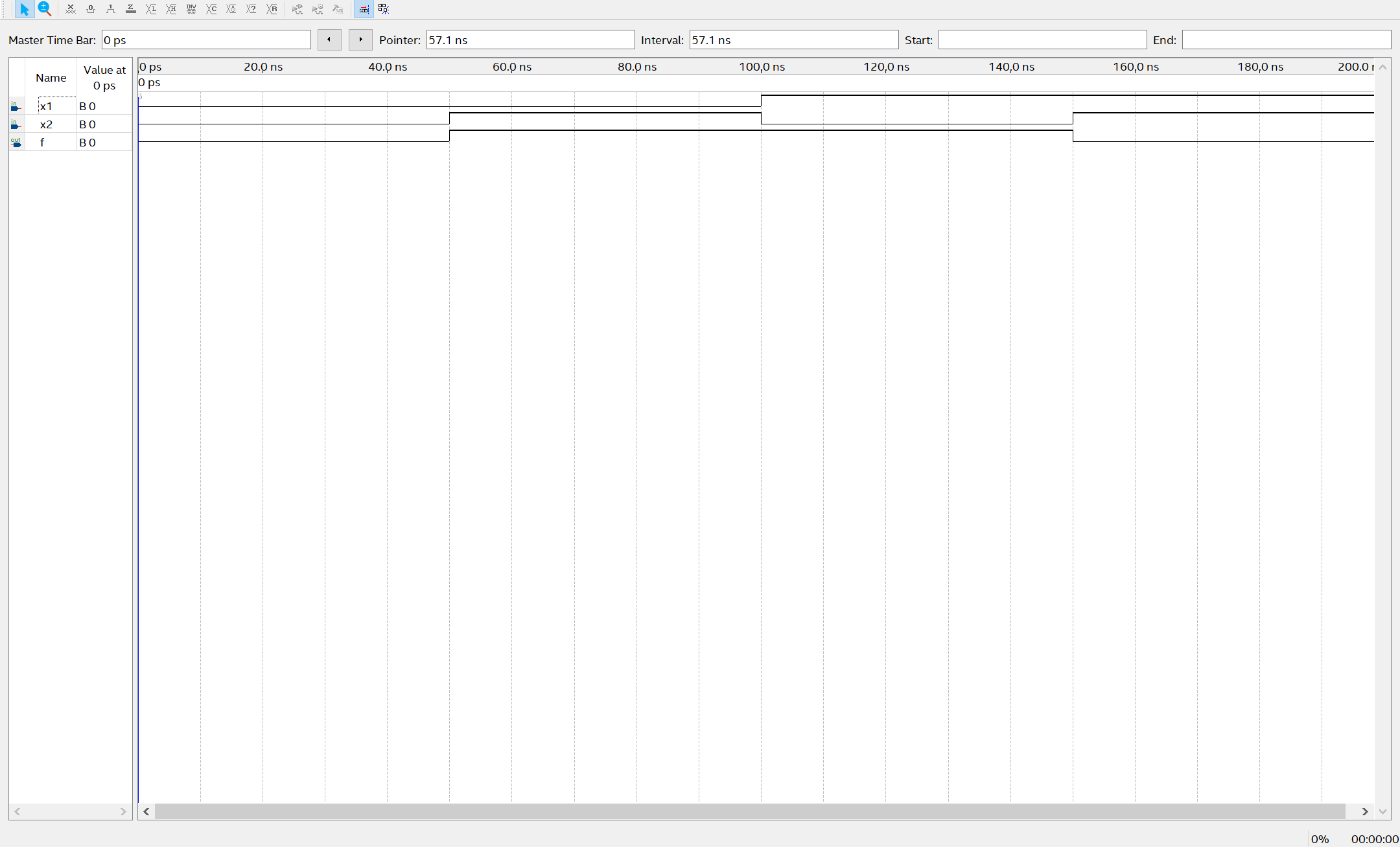
Description automatically generated

**Figure 1:** Screenshot of our XOR gate code in Verilog

Once the script was made, we made sure to compile our project to ensure there weren’t any errors.

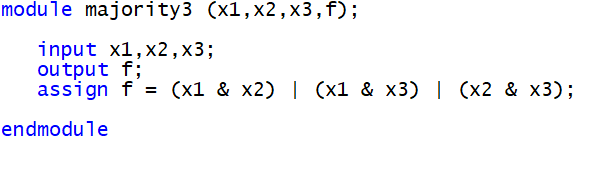
**Figure 2:** Screenshot of our XOR gate project compilation report

Once our project compiled, we then ran a simulation to verify the results of our project.



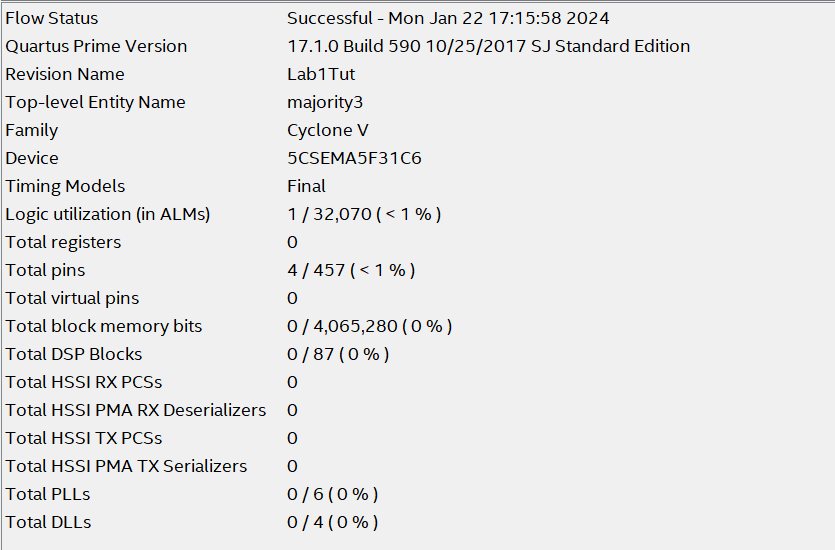
**Figure 3:** Screenshot of our XOR gate project simulation results

After implementing the XOR gate, we were given a second project to implement. There were three inputs and the output had to be 1 if at least 2 inputs are set to 1. The Verilog script to implement this project is shown below:



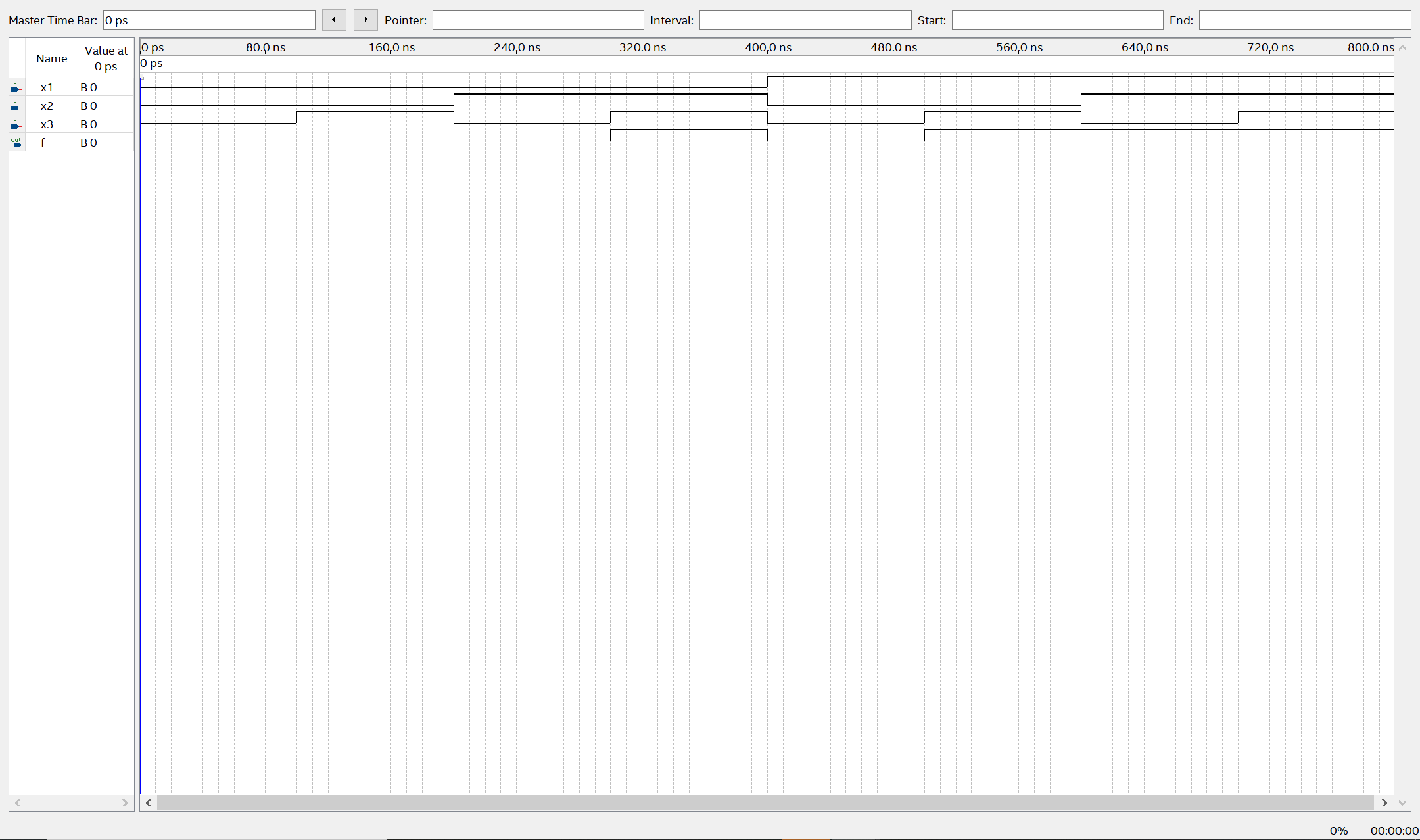
**Figure 4:** Screenshot of the Verilog code for the second project

Before simulating this script, we made sure to compile our project to ensure there were no errors.



**Figure 5:** Compilation Report with the Majority3 script

We then simulated our project to verify that it was implemented correctly.



**Figure 6:** Simulation results of the project with the majority3 script

**Part 3:**

1. The reg data type is a data type used to represent variables that store some kind of constant value (until it gets manually changed to something else). The wire data type is a data type that is used to represent a connection from the output of one module to the input of another, where the stored value isn’t constant and changes depending on what the wire datatype is connected to.
2. Wire data types can be used on the left-hand side of the assign statement, as wire data types need to be assigned to the output of some other module. One way of assigning a wire data type to the output of another module is through the assign statement.
3. Modules can have inputs, outputs, or inouts (simultaneously input and output). All inputs, outputs, and inouts are usually declared at the top of the module.
4. Continuous assignments are assignments where the left-hand side variable changes in value depending on values of other variables on the right-hand side of the assignment statement. Blocking assignments are assignments where the evaluation of the right-hand side and the assignment to the left-hand side both occur at the same time for all assignment statements. Non-blocking assignments are assignments where the right-hand side gets evaluated first for all assign statements, then the values of the right-hand sides for all assign statements get assigned to their respective left-hand side.
5. To implement combinational logic, we don’t need to rely on values from previous states. Therefore, the program can be created inside of an initial block or always block to evaluate something once or repeatedly using present values. To implement sequential logic, we need to rely on values from previous states. Therefore, the program can be created inside of an always block, where the program repeats and re-evaluates values depending on values from past iterations of the always block.
6. To avoid inferred latches, you need to make sure that every variable that is getting used in the module is assigned to something, or is defined by some value.
7. The key difference between the << and <<< operator is that the << operator does not consider the first bit being a signed bit and by default shifts in zeros. The <<< operator however considers the first bit as a sign bit, and will shift in zeros if the sign bit is 0 or shift in ones if the sign bit is 1.
8. To define an array with 6 elements being 7 bit wires, the following code would be used:

**wire [0:6] array[0:5];**

**Part 4:**

Here are the screenshots of the Verilog code for each of the basic logic blocks:

A screen shot of a computer code

Description automatically generatedA screenshot of a computer program

Description automatically generated

**Figure 7:** D Flipflop **Figure 8:** D Flipflop with Active Low Sync Reset

**A computer code with black text

Description automatically generated**A screen shot of a computer code

Description automatically generated

**Figure 9:** D Flipflop with Active Low Sync and Enable **Figure 10:** D Latch with Sync Enable

A screenshot of a computer program

Description automatically generatedA screenshot of a computer code

Description automatically generated

**Figure 11:** 4to1 Multiplexer **Figure 12:** 4-bit Counter

**Part 5:**

The truth table for my name, Lazar, is given below:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Val** | **I3** | **I2** | **I1** | **I0** | **S0** | **S1** | **S2** | **S3** | **S4** | **S5** | **S6** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| L | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| A | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Z | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| A | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| N/A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Part 6:**

Using algebraic manipulation, an expression can be found for S0 and S1 by ANDing all the inputs together (when S0 and S1 = 1) and ORing them together. Therefore, the expressions for S0 and S1 are:

**Part 7:**

A screenshot of the code for implementing the logic of the seven-segment display is shown below:

A screenshot of a computer code

Description automatically generated

**Figure 13:** Screenshot of Part 7 code