Logic Circuit Simulator - Project Report

Contributions of Each Member

Mahmoud: He worked on the Event and the Gate classes. In the Event class, he made sure that it kept track of the different events' times, then the name of the signal being updated, and the new value it was going to take. He made the event class to represent different logic gates such as AND, OR, and NOT in addition to the functions that calculate their outputs.
Fatima: She implemented the functions to read both the circuit.v and .stim files and populate the necessary data structures. In the function used to parse the .v files, she made sure that we get only the useful data from the file such as the type of the gate, its inputs, and outputs, and use these extracted data to populate the gates vector. She implemented the function used to parse the .stim file so that it extracts useful information about the events and populates the priority queue.
Adham: He worked on the refresh_gates and the run functions. In the function used to refresh gates, he made a loop that checks each gate's current state and evaluates it. If there is a change in the gate's output, an event is added to the priority queue to accommodate the change. For the run function, he implemented it such that it goes through each event in the priority queue and makes the necessary changes in the .sim file accordingly.

Data Structures and Algorithms

Priority Queue: It is a min-heap priority queue, it stores events based on their timestamps. It was implemented such that earlier events are processed first, which allows the simulation to run in the correct order.
Vector of Gates: We implemented this vector such that it stores all the gate objects in the circuit. It allows us to go through and make any necessary changes in the gate's output while the circuit's state changes during the simulation.
Map for different signal values: This map matches each signal with each binary value. It keeps track of the current state of each signal, using the signals' names as keys. We made each signal start with -999 as a default value which means that the state of the signal is unknown for now. It gets changed afterward with an assigned real value. This map tells us the latest signal values throughout the simulation.
Event and Gate Objects: The event objects store the necessary information about each event including the event time, the signal name, and the new value assigned to it. Each event is a change in the circuit which will be then added to the priority queue based on its timestamp.
The gate objects represent every logical gate like AND, OR, and NOT, their inputs, and outputs in addition to their delays. Each gate object then gets added to the vector of the gates we mentioned earlier.

Algorithms:

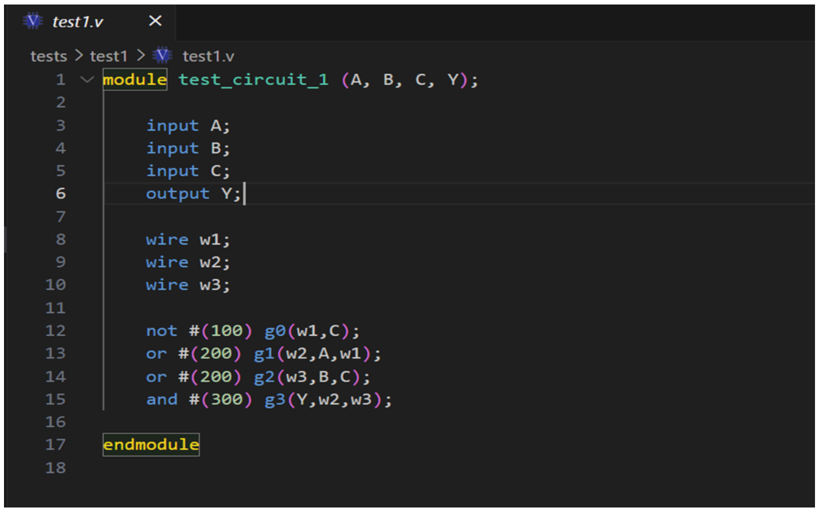
File Parsing: In the project, we need to do file parsing in two situations, the first one is to parse the .v file to extract the necessary information about the gates, and the second one is to parse the .stim file to get the events data. For the .v file reading, we went through each line in the file ignoring the empty lines, the lines that start with module, or endmodule, we then got the first words of the remaining lines to be the gates types, and after that, we extracted the inputs using regular expressions library, and then got the outputs and pushed them into the output vector, and we did the same with the delays as well and used this information to create Gate objects and populate the gates vector. Similarly, we used some helping functions to extract the timestamps, the input names, and the new values from the .stim file and populate the priority queue.

Gate Evaluation: the purpose of this is to compute the output of each gate based on its logical function and current input values. The steps we used in this are as follows, first, we checked the map of the signal values to get the current state of the gate inputs, and then based on the type of the gate the function performs the corresponding logical operation. And finally, it returns the output we got which allows the simulation to refresh the circuit with any new state updates.

Simulation: it simulates by processing the events in chronological order. First, it pops the earliest event from the priority queue and updates the signal values in the map. It keeps on doing this for all the events in the priority queue. If, at any time, the signal value differs from its previous state, the refreshGateOutpus function gets called to modify the gate's output accordingly. It keeps on doing this until the priority queue is empty.

Testing

Five test circuits were created initially using the specified file format for .v and .stim files. The Verilog file included only Verilog primitives and used a non-ANSI style for the module header. Also, each port was declared separately as shown in Figure 1.



```verilog
module test_circuit_1 (A, B, C, Y);

    input A;
    input B;
    input C;
    output Y;

    wire w1;
    wire w2;
    wire w3;

    not #(100) g0(w1,C);
    or #(200) g1(w2,A,w1);
    or #(200) g2(w3,B,C);
    and #(300) g3(Y,w2,w3);

endmodule
```

Figure 1 test circuit 1

A GitHub workflow was added for continuous integration (CI) by adding a .yml file that performs basic checkout on the C++ code and then compiles the main.cpp and reports any errors if found.

To validate the output of the program, each test circuit was simulated using gtkwave.exe, and the results were compared manually with the output.sim file as well as the waveform generated by the Python script.

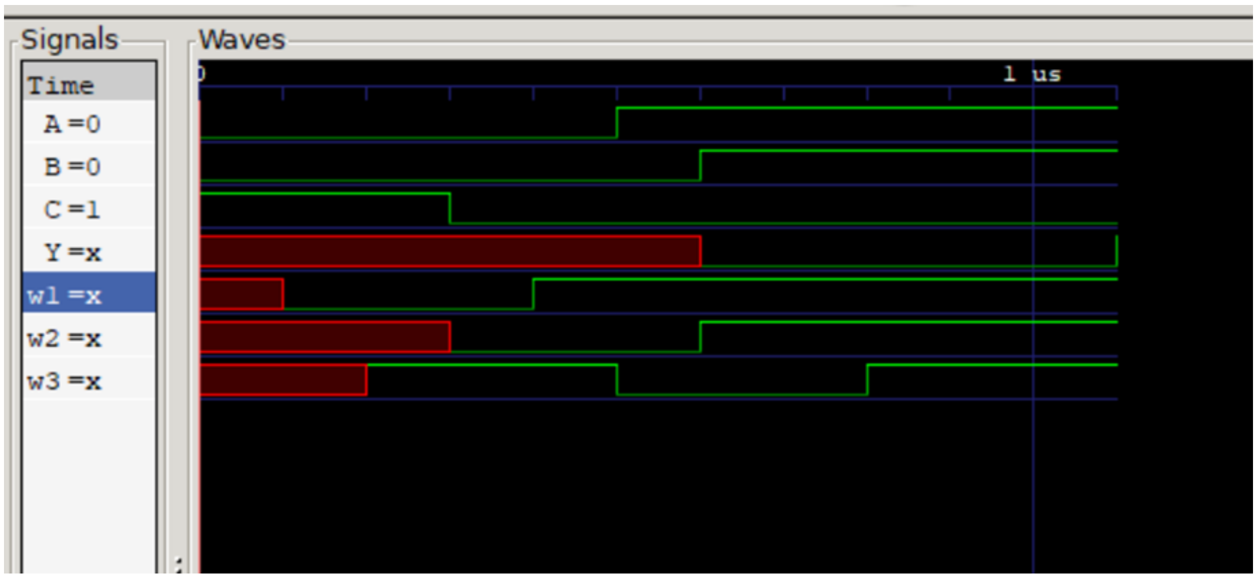
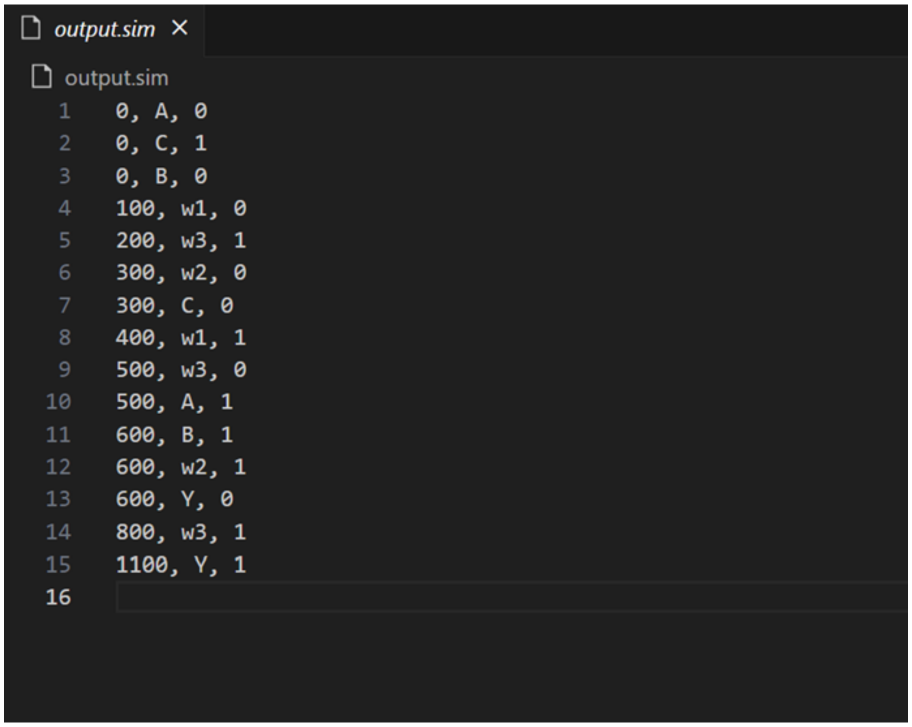


Figure 2 gtkwave.exe simulation of test 1 circuit



```
0, A, 0
0, C, 1
0, B, 0
100, w1, 0
200, w3, 1
300, w2, 0
300, C, 0
400, w1, 1
500, w3, 0
500, A, 1
600, B, 1
600, w2, 1
600, Y, 0
800, w3, 1
1100, Y, 1
```

Figure 3 ouput.sim for test 1 circuit

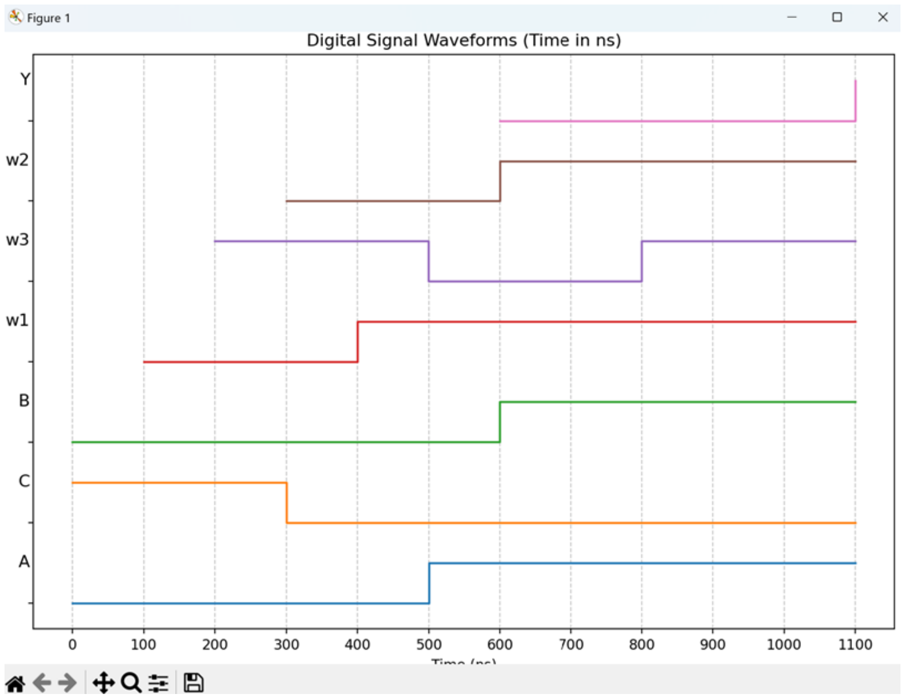
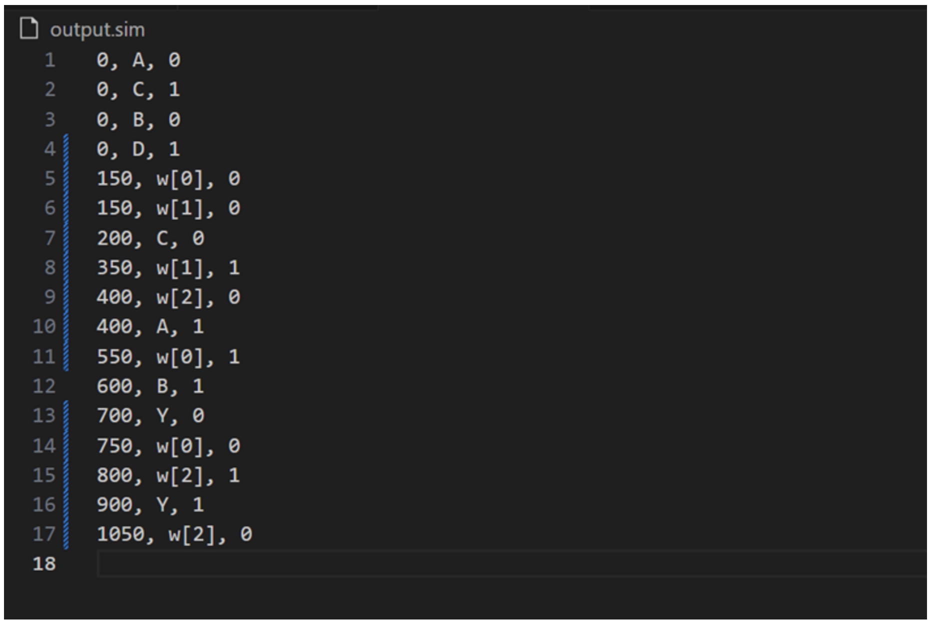


Figure 4 The output waveform for test circuit 1

In the first set of tests, some bugs were observed. One bug was having an empty line at the end of the test2.stim file that caused a runtime error. The bug was resolved accordingly in the code. A similar bug was the presence of whitespace before or after the name of the output or delay time in the declaration of the Verilog primitive, which was resolved by updating the regex expression used in the readVFile() function in the Simulation class to deal with whitespaces.

The tests were updated afterward to include a test 6 circuit that caused an infinite loop – the output of an XOR gate was connected to the input of the same gate while the other input was logic "1," which operated as a NOT gate. Furthermore, the test 2 circuit was edited to include a vector of wires – which required editing the regex expression again to allow reading square brackets \[\] in the readVFile() function.

Eventually, all tests were passed successfully. The following are the images of the output for the final version of tests 2 and 6.



```
0, A, 0
0, C, 1
0, B, 0
0, D, 1
150, w[0], 0
150, w[1], 0
200, C, 0
350, w[1], 1
400, w[2], 0
400, A, 1
550, w[0], 1
600, B, 1
700, Y, 0
750, w[0], 0
800, w[2], 1
900, Y, 1
1050, w[2], 0
```

Figure 5 output.sim for test circuit 2

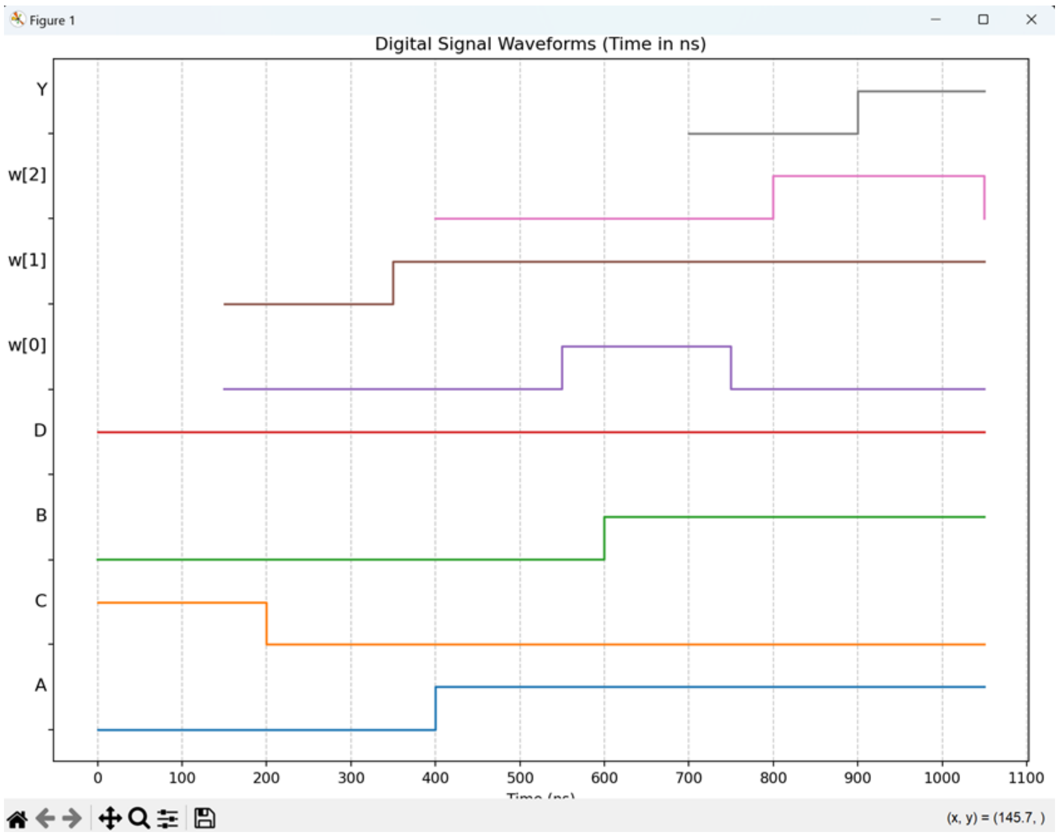
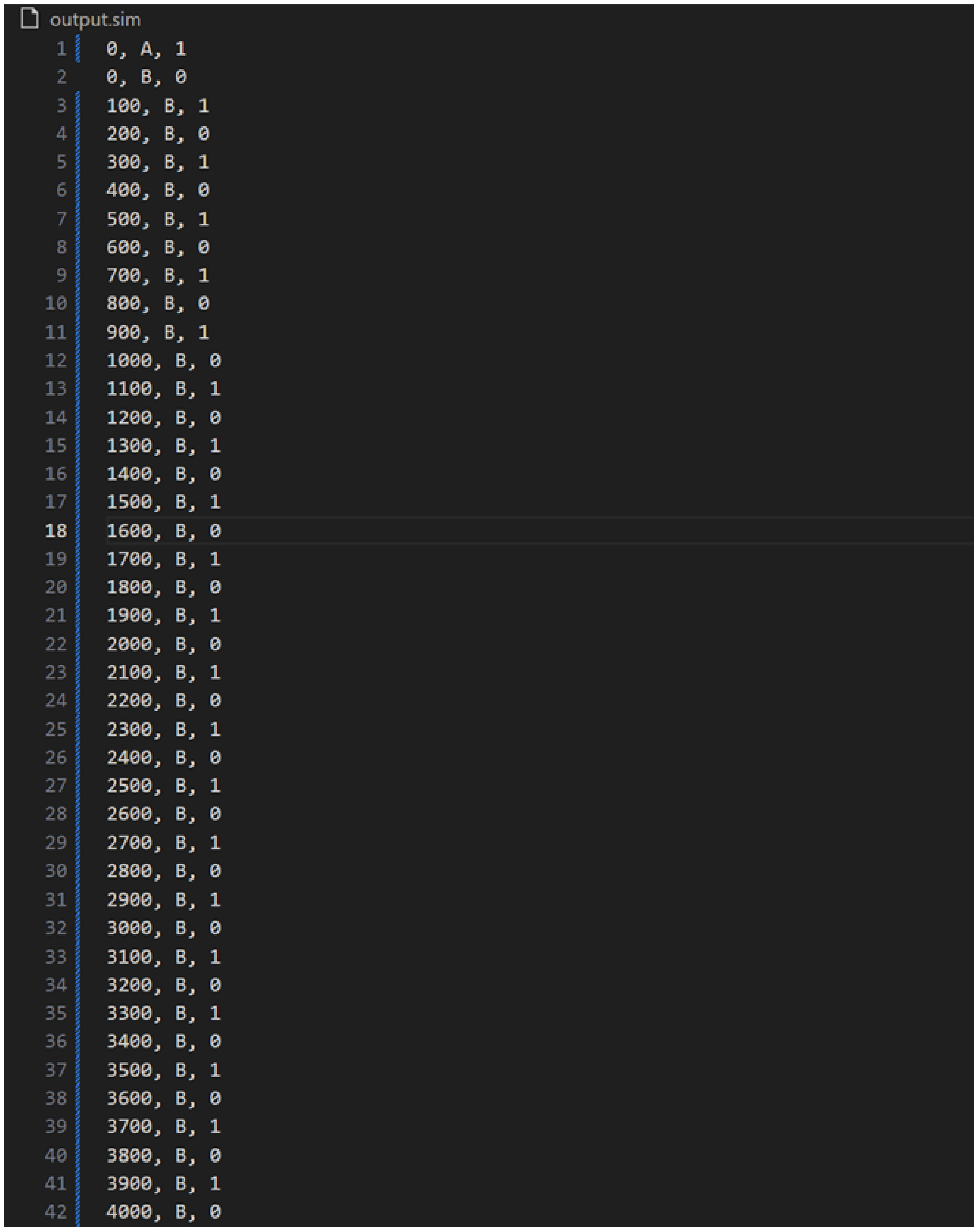


Figure 6 waveform for test circuit 2



```
output.sim
0, A, 1
0, B, 0
100, B, 1
200, B, 0
300, B, 1
400, B, 0
500, B, 1
600, B, 0
700, B, 1
800, B, 0
900, B, 1
1000, B, 0
1100, B, 1
1200, B, 0
1300, B, 1
1400, B, 0
1500, B, 1
1600, B, 0
1700, B, 1
1800, B, 0
1900, B, 1
2000, B, 0
2100, B, 1
2200, B, 0
2300, B, 1
2400, B, 0
2500, B, 1
2600, B, 0
2700, B, 1
2800, B, 0
2900, B, 1
3000, B, 0
3100, B, 1
3200, B, 0
3300, B, 1
3400, B, 0
3500, B, 1
3600, B, 0
3700, B, 1
3800, B, 0
3900, B, 1
4000, B, 0
```

Figure 7 output.sim for test circuit 6

Challenges

1) A challenge we faced in the beginning was how to process the events that come first in time even if they were not added first to the queue. We solved this by using a priority queue with a comparison function that prioritizes the event whose time is more recent

2) We wanted to have access to the current value of all bits/signals throughout the program and in different classes. We used a global map that links the logic state of the signal to its name – we preferred this approach instead of having a complete class for bits/signals to simplify the coding structure.

Additionally, we assigned an arbitrary value of -999 to all bits/signals initially to refer to the undefined/high impedance state.

3) After processing an event, we wanted to insert the new events triggered by it. We first thought about assigning to each bit a vector of the bits/signals that may change when the bit changes. However, we agreed to refresh all gates after an event occurs. Although it is not the most optimized solution, it makes the coding process easier, and the performance difference is not huge. When refreshing the gates, we call the function evaluate(), which returns the current value of the gate's output – which does not have to be equal to its current value, and that is the case when we insert a new event.

4) Although the Verilog file format was specified, we preferred to focus on reading the line of declaration of Verilog primitives gates only because it contains all the used bits/signals, gate operation, delay, and all necessary data for the program to operate. This approach enabled us to read and handle different formats of Verilog files/syntax if needed.

5) Because Verilog neglects whitespaces, we needed to be able to extract the information we needed from the Verilog primitive's line whatever whitespaces were there. To resolve this, we used regex expressions to extract the gate's delay and output even if there was any number of whitespaces before or after the extracted data. We used ChatGPT here to provide us with suitable regex expressions that fulfill our desires. We used the following prompt: write a regex expression that extracts text/number after '(' and before ','. The extracted text/number can have any number of whitespaces before or after it.