Digital Design I

Logic Gates Simulator

Mohamed El-Refai 900222334
Hana Shalaby 900223042
Mohamed Azouz 900225230

Introduction

This project presents a digital circuit simulator that models the behavior of logic circuits based on an event-driven simulation approach. This simulator reads a circuit from a verilog file and processes a sequence of input changes from a stimuli file. It then outputs the resulting signal transitions over time. By simulating each logic gate, the simulator will provide a realistic view of the signal within the circuit. This generated output can be visualized using the waveform tool which helps in the analysis and validation of circuit functionality.

Data Structures and Algorithms

Data Structures

- **Module** struct represents the entire circuit, storing vectors of inputs, outputs, wires and logic gates and it allows each component to be accessed and updated easily.
- **ioVar** struct stores the name, value, index (in the module), and type (input, output or wire) of each signal in the circuit which allows organization of circuit signals and enables quick access during simulations. It also stores the index of any gate they affect in the module.
- **event** struct represents the change in the circuit over time, with attributes such as the time of the event, the affected variables, and new values which allows for event driven simulation as it records and tracks changes that need to be processed.
- **Priority queue of events** to store events that are instantiated when parsing the .stim file. Allows for the circuit events to be processed in chronological order. It acts as a min heap, making sure events in the queue are in chronological order.

Algorithms

- **Event Driven Simulation**
    - The simulator triggers a sequence of events when a change happens in the input signals, where each event contains a time attribute, allowing the simulator to process these changes in a time ordered sequence. While the priority queue serves as the backbone of this simulation as it ensures that each event is processed at the correct time and in the right order. As they are being processed, they may trigger further events if a signal change occurs dynamically simulating the behavior of logic gates over time.

- **Logic Gate Evaluation**
    - Each gate (AND, OR, NOT, etc.) is represented by a **logicGate** class, which includes methods for evaluating the gate's output based on its inputs. Also, the **evaluate** function within each gate uses conditional logic to compute the gate's output. For instance, an AND gate iterates through its inputs, setting the output to true only if all inputs are true. This way it efficiently models the behavior of each gate in the circuit.
- **Parsing and initialization**
    - Parsing algorithms are used to read the verilog and stimuli files, extracting information about inputs, outputs, wires and gates which is done by string manipulation and regular expressions techniques transforming the files into structured data that the simulator can process.

By combining these data structures and algorithms, the circuit simulator can model the behavior of complex digital circuits, accurately capturing both logical relationships and timing dependencies. This approach allows for scalable simulation, capable of handling circuits with multiple gates and various signal delays.

Testing

- **Unit testing of components**
    - Each Logic gate was tested independently to ensure correct output values based on different combinations of input signals. We also tested the **event** struct and queue to ensure that events were processed correctly in order. Besides that, the verilog and stimuli file parsers were tested with multiple circuit configurations to confirm that the inputs, output, wires and gates were correctly read and put in memory. All of this was done by debugging messages after finishing the program to check everything is running as expected.
- **Integration testing**
    - After we verified each logic gate is working correctly, we performed integration testing to make sure they work together correctly which involved running the entire simulation process with small, manually defined circuits and stimuli files like some of the example circuits we have in our test circuits in GitHub, verifying that each gate's output changed as expected over time. Integration testing helped us identify and resolve issues that showed up when components interacted with each other. From this step we tested more complex circuits like Full adders and 4 to 16 bit decoder as well as other more complex combinations of logic gates to ensure that the simulator can work as expected.

- **Output Validation**
    - We used python script **WaveFormGenerator.py** to visualize the simulation results, comparing the waveform results to expected behavior which allowed us to see if signals moved correctly over time and helped us to identify any issues easier. Also, we created reference outputs for known test cases to automatically compare against the simulation results and detect any differences.
- **AI assisted debugging**
    - During our testing we used ChatGPT to help us identify and resolve issues in code logic and data handling as it provided us insights into error patterns, suggested fixes, and guided us through the best practice especially for complex scenarios. It also helped us with understanding the regex library.

Challenges

One of the primary challenges we encountered during the project was implementing the parsing functionality in C++ as parsing required converting text descriptions into structured data that the simulator could interpret, which is a complex and error-prone task.
At first we explored several approaches to parsing without any external libraries, and tried experimenting with different solutions including basic string manipulation techniques. However, each attempt led to new issues like syntax errors, out of bound errors, as well as incorrect data extraction. During this process, we were introduced to regex which is a powerful tool for pattern matching and text parsing, which none of us had previously used or heard about. So learning regex added an extra layer of complexity, as it required understanding a new syntax and methodology for extracting information from strings. To overcome these challenges, we had some help from AI tools like ChatGPT. It was a good source to debug our code and implement regex effectively, as it helped us identify syntax errors, check out of bound expressions, and suggest regex patterns for specific tasks. With this support we refined our method, leading us to developing a solution that combined regex with structured parsing logic to accurately interpret the verilog structure and identify inputs, outputs, wires and gates. Finally, the program has some glitches in the output, occasionally printing signal transitions at the same timestamp. We haven't yet identified the root cause, and while the waveform generator can filter out these anomalies, it does not fully resolve the underlying issue.

Contributions

At first, we had a couple of meetings on how to start on the logic. That was one of the challenges as mentioned earlier, as we tried different methods and solutions until we came up with the event queue system together. We first planned it by sketching our ideas on a whiteboard, while Hana Shalaby searched for a library to use for parsing the verilog and stimuli files in C++. We found the library <regex>, which we tried to use, but failed to understand it. However, we still had to find a way to parse our files, so Hana came up with a method to do this without the regex library. Then, we started applying the logic to actual code. For the milestone requiring “semi-functional implementation of a simulator,” we submitted the code we wrote on the first draft of our event queue system. Unfortunately, it had some errors including accessing “out of bounds” memory locations, which made the code fail.
After that, Mohamed Azouz started most of the code from scratch with a clearer understanding of the logic we wanted to implement, and a plan to remove the errors we encountered on the first draft. He spent some time understanding the syntax of the regex library, as parsing without it proved to be difficult in the first draft. Furthermore, we still had some issues with the output, like incorrect output and incorrect assignment of wires and outputs. Then Mohamed El-Refai worked on these problems which led us to having a correct output and we used some AI support for debugging the issues in this part. Mohamed El Refai also with some help from ChatGPT made a waveform generator file that took the output of the program we made using C++ and converted this output to a waveform using Python and matplotlib library and stored it’s result as a “.png” in the file directory of the program.

Conclusion

The development of this simulator showed a comprehensive approach to modeling logic circuits using event driven simulation. By implementing data structures and algorithms, we created a tool capable of accurately processing and visualizing circuit behavior based on input stimuli and the use of priority queues allowed for accurate event handling, ensuring that all signal transitions were processed in their correct order. Testing efforts including unit and integration testing that were essential for validating the accuracy of individual components and their interactions. Visualization with waveform tool and AI-assisted debugging further improved our ability to verify and refine the simulator.
On the other hand, we had a lot of challenges especially in parsing complex circuit descriptions, which required the development of a customized approach using regex and overcoming these challenges made us understand more concepts of C++ and regex. Also, the contributions of each team member with AI support enabled us to build a functional, scalable simulator that provides valuable insights into digital circuit behavior. Overall, this simulator not only offers an educational tool for studying circuits but also a foundation for future enhancements in circuit simulation.