CSCE2301 - Digital Design 1

Logic Circuit Simulator

Names: Jana Elfeky, Kareem Elnaghy, Mark Kyrollos

Dr. Mohamed Shalan

Project Overview

In our project, we had to build an event-driven logic circuit simulator that accepted three inputs consisting of: a library file, a circuit file and a stimuli file. Using these inputs, we were asked to output a simulation file which models the circuit.

Data Structures

To efficiently manage the components of the logic circuit, such as gates, wires, and input/output signals, we relied on four main data structures: priority queues, maps, vectors and structs.

Priority Queue: The main data structure used to implement the event-driven simulator was the priority queue. The queue here acts as an activity list that stores the active gates, which makes processing and outputting the gates very easy to implement and based on changing events.

Maps: We used maps to store operator precedence as well as to store our logic gates' expressions. We used maps here because they make it easy and efficient to access data given a specific key, which in our case, was the name of our component.

Vectors: We used vectors to store our inputs that we read from the circuit files, and to store the gates used for a given circuit. Moreover, they were used to store the input information that we read from the stimuli file which we used throughout the code. Also, vectors were used as temporary containers used for reading from the files. Vectors were used here instead of arrays because of their ability to be resized and for the ease with which elements can be added to or removed from them.

Struct: Created a struct in the gate class called DataStruct which can store three attributes, a name, value and timestamp. This struct is responsible for allowing us to store information on the events during our simulation such as changes in the input and changes in the gate outputs.

Our Process

The project had three deadlines and an extension to fix some bugs, each a week apart, so we will divide this section into four parts: week 1, week 2, week 3 and week 4.

Week 1

On week 1, we had to submit 5 test cases and commit our source code to our GitHub repo. We started by brainstorming circuits we can use to test our code that consisted of enough variation to challenge our simulator. We came up with test cases that had varying complexities, different gates of different numbers of inputs and circuits we learned about in class, like the 2x1 multiplexer. After we thought of the circuits we wanted to use, we drew their timing diagrams by calculating the expected delays of each circuit. For the code portion, we decided to start out by creating our library, circuit and stimuli files. Then we implemented a class which models the gate with its inputs, output, delay, and so on. We finished implementing this class with most of its member functions by March 7th and we committed our code on that day.

Week 2

On week 2, we were expected to submit a semi-functional code. During this week, we updated all of our circuit files and modified our gate class. Also, we created a class to model our circuit, including a vector of all the gates, the files corresponding to that circuit, some values read from the stimuli file and other data corresponding to our circuit. We implemented member functions in our class to aid in parsing the stimuli and circuit files and store their values so we can calculate outputs and delays and we have a function that is responsible for writing our final output to the simulation file. After week 2 was done, our code was creating a correct output that was only missing the time stamps and we had some issues with our function which translates the output expression of the gates to a valid boolean expression.

Week 3

By the end of week 3, we are asked to submit a fully functional code. During this week, we implemented the function which calculates the timestamps of each input, wire and output using the gate delays and the input timestamps read from the stimuli file. Also, we fixed the function which translates the output expression of the gates to a valid boolean expression. Most of the week was spent on cleaning up the code and perfecting our classes and their functions along with the output file. Moreover, we changed the test cases up a bit to test our code under different circumstances and modified things accordingly.

Week 4

We met with Dr. Shalan on the 24th of March, and he extended our deadline to the 30th of March so we could fix some issues with the simulator. When we met with the professor, he told us that our GitHub repository needed to be more organized by adding folders and raising more issues. Also, his insight helped us understand that our structure for the simulator needed to be modified in order to make it event-driven and to make it work with more generic test cases. We started this week by modeling how we were going to implement the event-driven logic simulator and reading more about it. After some research, we realized that using maps like we had done initially was not going to work if we were given an input that changes multiple times throughout the simulation. Therefore, we thought of using a priority queue which acts as an activity list that stores the active gates. This queue stores the active gates ascendingly based on their timestamps, executes the top gate, and pushes all its fanout gates to the queue before popping it out of the queue. Moreover, we worked more on error handling by reporting errors when we are unable to open files, there are different files being incorrectly formatted and provided, there are undefined components in the circuit description, or there are logical Errors in

the circuit i.e. an input also serving as an output. Lastly, we organized our GitHub repository by putting the files into meaningful folders and adding a detailed README which gives an overview about our project.

Challenges Faced

Throughout the project, we encountered several challenges that tested our problem-solving skills and technical abilities. One of the biggest challenges we faced was parsing the output expression of each gate and converting them into boolean expressions which could calculate the gate's output based on its input signals. We had to think outside of the box and we used stacks to translate the expression string into a boolean value. Another issue we faced was the timestamp calculation. Calculating time stamps for logic gates with different delays and input timestamps proved to be a bit difficult, but we figured it out in the end and got the code working as expected. After the extension given to us, we had to figure out how to model an event-driven logic simulator and which structures to use, and that was one of the big challenges we faced; but with some research, we figured out how to implement it. We also had to research how to run the program through the terminal which was an interesting learning experience as we've never done that before.

Contributions

Each team member made significant contributions to different aspects of the project:

Kareem: Implemented functions to read circuit, stimuli, library files and wrote the respective simulation files for each circuit. Also assisted in reading the expressions in order to evaluate the outputs. Worked on developing the functions responsible for event driven simulation using priority queues. Incorporated error handling into the program.

Jana: Handled translating output expressions of gates to boolean variables, implemented timestamp calculations, parsing library files, and modeled how priority queue was going to work and implemented helper functions and added more attributes to the circ and gate classes as needed.

Mark: Simulated the outputs by implementing a waveform generator on QT. Developed the gate class and its functions.

In conclusion, our Digital Design project provided valuable insights into the complexities of logic circuit simulation. Through collaboration and effective division of tasks, we developed a functional simulation tool that made us delve deeper into the intricacies of logic circuits.