CSCE 2301 Spring 2024

Project Report

Dr. Mohamed Shalan

Logic Circuits Simulator

Ahmed Yasser Ahmed Fawzy Abdelkader 900222336

Tony Yohana Kamal Asaad Gerges 900222981

Ahmed Mohamed Abdeen 900225815

Department of Computer Science and Engineering, AUC

Abstract :

In the course this semester we have used several logic circuit simulators: Logisim and Vivado. With these we were able to represent and test complex circuits without needing to build their physical representations. However, building our own event driven logic circuits simulator allowed us to gain a deeper understanding and appreciation for these tools. To do this we used various data structures and algorithms; most notably the EventDriven typedef (See Appendix …) and the shunting yard algorithm [1]. In this report, provide an in-depth recollection of the development process, talk in detail about the data structures and algorithm used, specify the input data and user experience and how we overcame challenges related to that, and finally reflect on our application and methodologies.

Outline:

1. Introduction and Problem Description

The goal of this project is to create an event driven logic circuits simulator that receives inputs .cir, .stim, and .lib and returns a .sim file with the simulation outputs. Given the different types of data being sent and the requirements of an event driven simulation we decided to use two classes Logic_Gate and circuit. Through these classes and the EventDriven typedef we are able to provide a correct event driven simulation. The used data structures and algorithms will be discussed in more depth later.

2. Methodology and Used Data Structures and Algorithms

In the beginning of this project we were unsure how to start but from the start we had several zoom and in person meetings where we would brainstorm how we wanted to proceed. Further we would highlight the flaws in our thinking and work together as a team to solve them. Most of our planning was done on Google Drawings ([2], *Google Drawings Plans*).

To start the program we first prompt the user to enter the file paths for their desired .lib, .cir, .stim, and .sim. From this we use the functions in the circuit class ([2], *Source Files/circuit.cpp*) to read the files and fill our data structures. We use the Logic_Gate class ([2], *Source Files/Logic_Gate.h*) to first create all logic gates provided in the .lib file as generic objects in the map<string, Logic_Gate> allGates. Then the vector<vector<string>> components are filled with the data of each gate from the .cir file. Then the vector<Logic_Gate> usedGates identifies which gates are present in both the allGates and components. This is done in the objectModification function using a brute force method ([2], *Source Files/circuit.cpp*). Here we note the error handling present: if there is a gate mentioned in the .cir file that is not present in the .lib file an error is sent to the user and the program stops. Further, Finally the vector<tuple<string, bool, int>> cirInputs ([2], *Source Files/Logic_Gate.h*) is filled with the .stim data using the format Input Name, Logic Value, Time. Consequently, any input or output is a tuple<string, bool, int>.

After the initial reading of files and filling of data structures we start to interact with the main algorithms and the EventDriven typedef. EvenDriven is the key data structure in our simulation since it combines all other smaller structures. EventDriven is defined as a map<int, pair<vector<tuple<string, bool, int>>, queue<shared_ptr<Logic_Gate>>>>. The format of EventDriven is a map which has the time as the key (this is the time stack), and a pair where the first element is the schedule events vector and the second is the activity list queue. Notably we use shared pointers of logic gates not logic gates since we use and update elements in the activity list which we need updated in all necessary functions. We talk more about how we realized this in the challenges.

The main algorithm is split into three functions all called in runAlgorithm: fillAcitvityList, update_cirInputs_names, and addOutputsToScheduledEvents ([2], *Source Files/main.cpp*). All these functions are reasonably complex so we will discuss them simply and provide their respective time complexities. For the fillAcitvityList function we traverse all of the scheduledEvents at the current time, if any event is present in the logic gate member vector<pair<string,int>> cir_Input_Names we add it to the activity list. The time complexity of this function is O(n*m) where n is the size of the elements in the scheduled events vector and m is the number of gates in used gates.

Then since we know which gates need to be evaluated we update the second element in cir_Input_Names with the value of the input at the current time. Note that we faced some challenges doing this since if for example A became 1 at time 2 it wouldn't stay this way when needed later. To fix this we needed careful making of our data structures: we talk more about this in the challenges section. The time complexity of this function is O(n*m*k) where n and m are the same as above, and k is the number of elements in cir_Inputs_names.

Finally, the addOutputsToScheduledEvents function calls the evaluate function which applies an adapted shunting yard algorithm to find the logic value of the gate and push the evaluated gate into scheduled events at the corresponding time. The shunting yard algorithm we adapted was referenced from [1], but edited to abide by the rules of boolean algebra. The time complexity of this function is O(n*a) where n is the length of the boolean expression and a is the number of elements in the activity list.

Finally, we display the output for the user using EventDriven allStructures and write to the .sim file the user provided.

3. Testing and Limitations:

In the early stages of development so we could work in parallel and not need each other's code to move forward we used sample data to test the functionality of each function in the program. Then once we started combining our code we found that we needed to abandon samples, and we checked our final outputs using Logisim. For all our test circuits we have correct outputs.

Further, we used Github Actions, our first time using it,to automate testing the validity of new code pushed to the repository. We created our own workflows and used community workflows, but we found that our own custom workflows worked best since we could shape them to test what we desired. We tested the valley of our code by checking if the program was able to build and run on github actions after pushing to the develop branch. Then whoever pushed the code would issue a pull request, if the code passed the Github Actions test it would be pushed to the main branch for manual testing. It is important to note that we always kept a Backup-of-main-branch so that if we faced any large errors we could recover what we had.

Our testing method highlighted the errors in our implementation and allowed us to iteratively fix these bugs; however, we believe that with more experience in Github Actions we can reach a higher level of automation in our testing which would remove a lot of the rote debugging and testing.

4. Challenges and Critiques:

Ahmed Fawzy Abdelkader:

- Tracking values across functions

When we first modeled the EventDriven data structure we were pushing Logic_Gate objects into the queue; however, when debugging runProgram we had an issue that the outputs were all the same which meant that cir_Input_Names was not being updated along with other members. We used shared pointers to fix most of the tracking issues. Nonetheless, cir_Input_Names was not tracking yet. Ultimately, I was able to fix this using a brute force approach, but I believe that I could find a better solution for this since initially cir_Input_Names should have been updated by reference from the start

- Order of operations for logic gates and evaluating logic gates

Deciding to use the shunting yard algorithm was because of its capable abilities in order of operations, and O(n) time complexity. I changed the standard mathematical operator to boolean operators, and created the insertValuesInExpression which replaced the values in the expression with the corresponding truth values based on the order of the inputs in the .cir file similarly to verilog. The algorithm was adapted from [1] with the addition of ~ operator which required its own helper function since it is a unary operator.

- Debugging and Implementation

While this is a challenge in all programming projects due to the complexity of this simulator we found extra difficulties. We had to deal with hardware and compiler issues and midterms while working on the project so this section was a burden. However, personally this stress made me a better programmer since I had to learn more about how to use debugging tools. I also learned how to better split the work between the team members to best utilize efficacy. I would however like to use Github Actions more to help with automation.

Tony Gerges:.

- Initially, we were going to use the time values in the cirInputs vector and update the current_time but this caused multiple logic issues. After that, we were going to use discrete steps by finding the smallest interval between two inputs in the cirInputs, yet this caused problems where multiple inputs with different timestamps could be accidentally pushed into the map since the new inputs pushed into the vector will not be known at the time of writing the code. Finally, we opted to push all stim file inputs in initially and work using the event driven simulation.

- In the circuit reading file, I wanted to be able to use the data from the library file. However, the issue of having the same type of gate in several instances in the circuit file was an issue. As a result, I opted to make the Allgates a map which stores the gates with generic objects. In other words, every type of gate already has the common attributes filled, so in the circuit class, the gate is searched for from the allgates map and a new object is instantiated from the generic object in order to fill it with the gate specific information.

- replaceOperands function produced false outputs during the presence of brackets, and brackets are necessary for the shunting yard algorithm to work. This was fixed using regex

Ahmed Abdeen:-

- Issue in reading the boolean expression in each gate in the lib file. At first we stored it in a vector, but we realized that it would cause difficulties in the operator function. Thus, we opted to store the expression in a string.

- In the read_stim_file function, we first stored the contents of the stim file in a map so that it can be sorted automatically during the running of the program. However, it introduced the problem of making the simulator less general, as it will not accept duplicates. Therefore, we opted for a vector of tuples (timestamp, name, value) and in the main function we will sort the vector in each iteration.

After that we decided to construct a bigger data structure that stores a pair of a vector of tuples (scheduled events) and a queue of logic gates (activity list), and this pair is mapped to a key (timestamp). This facilitated the simulation, as all the data were connected to each other.

- The code was not modular in the march 14th milestone, as the reading function of the files were included in the main. To solve this problem, we included member functions in the circuit class, including lib, cir and stim files reading functions. Each function takes the path of the file as a parameter, which makes the code more modular. In addition, the write_to_sim function which writes the output in a sim file has been modified to be a member function in the circuit class so that it can be called for each circuit to write its output in a file in the same run without any unnecessary messiness. However, we replaced this function with another function because we changed the data structures, so the function objective was still the same, but with a different name and different parameters.

5. The contributions of each member:

Logic_Gate Class:

- Planning - Ahmed Abdelkader, Ahmed Abdeen, Tony Gerges
- Writing Class - Ahmed Abdelkader, Tony Gerges

circuit Class:

- Planning - Ahmed Abdelkader, Ahmed Abdeen, Tony Gerges
- Writing Class - Ahmed Abdelkader
- Reading .lib File - Ahmed Abdeen
- Reading .cir File - Tony Gerges
- Error Handling - Ahmed Abdeen, Tony Gerges
- Input/Output debugging - Ahmed Abdelkader

testOperator:

- Planning - Ahmed Abdelkader, Ahmed Abdeen, Tony Gerges
- replaceOperands - Tony Gerges
- Evaluate - Ahmed Abdelkader
- Helper Functions - Ahmed Abdeen

main/runProgram:

- Planning - Ahmed Abdelkader, Ahmed Abdeen, Tony Gerges
- fillActivityList - Ahmed Abdelkader
- addOutputsToScheduledEvents - Ahmed Abdelkader
- update_cirInputs_names - Ahmed Abdeen
- Input_exists - Ahmed Abdeen
- Debugging and Integration - Ahmed Abdelkader, Tony Gerges

Bonus:

- Github Actions - Ahmed Abdelkader
- WaveForms GUI - Ahmed Abdeen, Tony Gerges

6. Bonus: GitHub Actions

From the creation of the repo we utilized github actions for continuous integration. We used one workflow from the github actions community, testCI01.yml, and another we wrote, test_build.yml. We found that the community actions rams successfully while our own one did not run successfully frequently. We believe this is due to our inexperience in github actions, but will be using it in all coming projects in order to make better use of its capabilities.

7. Use of Chat GPT

- Used in waveforms to change between C++ language and json.

    - Prompt: given this sim file and the original signal Wavedrom, make a similar structure using json.
- Used to create printConsolidatedallStructures
    - Prompt: create a function that will print EventDriven
- General syntax questions
- Regex
    - Prompt: I need to find the operand without altering the rest of the expression

8. References:

https://www.geeksforgeeks.org/expression-evaluation/ [1]

https://github.com/Ahmed-Fawzy14/Logic-Circuit-Simulator [2]