Logic Simulator Project

Team Members:

Ahmed Fawzy Abdelkader, Ahmed Abdeen, Tony Gerges

Introduction and Problem Description

The goal of this project is to create a circuit simulator that receives inputs .cir, .stim, and .lib and return a .sim file with the simulation details. We decided to achieve this goal through the use of two classes Logic_Gate and circuit where we would create all logic gates present in the .lib file then take the ones needed for the circuit object in the usedGates vector. We split the program into input/output and evaluating the circuit.

Used data structures:

Logic_Gate.h:

- vector<pair<string,int>> cir_Input_Names: contains the inputs of the gate

Circuit.h:

- vector <tuple<string, bool, int>> cirInputs: vector with stim file inputs and any new inputs

- map <string, tuple <string, bool, int>> current_Inputs_Map: Map of inputs that are equal or less than current_time
- map <string, Logic_Gate> All_gates: stores library gates in generic objects
- vector<vector<string>> components: used to store contents of circuit file

Complexity

When possible if we were to choose between a map and vector we would use a map for its constant lookup time. The highest time complexity of application is O(n^2*T) where n is the length of cirInputs and T is the largest time in the .stim file.

Used Algorithms:

- Evaluating Logical Expressions
    - Evaluates logical expressions that define the behavior of logic gates within the circuit.
    - Uses a combination of stacks to manage operators and operands within the expression. Supports basic logical operations (AND, OR, NOT) and handles precedence and unary operations.
- Sorting Circuit inputs

    - Sorts the inputs of a logic circuit based on their timestamps to ensure that they are processed in the correct chronological order.
    - Utilizes the `std::sort` algorithm from the C++ Standard Library, comparing elements based on the third element of the tuple, which represents the timestamp.
    - Time complexity: O(nlogn) according to STL sort.
- Run Operator Test:
    - Evaluates gates based on inputs in the current_Inputs_Map using and updates the circuit's state.
- Finding Index of Used Gates (find_usedGates_Index)
    - Finds the index of a gate within the vector of used gates based on its component name.
    - Linearly searches the vector of used gates for a gate with the matching component name.
    - Time complexity: O(n) where n is the number of elements in usedGates.
- Shunting Yard Algorithm:
    - Used in operator function
    - Sends numbers to the stack
    - Handles operators differently
    - If the stack is empty or contains a left parenthesis on top, or the operator has higher precedence than the operator on the top of the stack, the algorithm pushes it onto the stack.
    - Otherwise, the algorithm pops operators from the stack to the output queue until it sees an operator with a lower precedence or a left parenthesis on top of the stack, and then it pushes the current operator onto the stack.

- Time complicity: O(n) where n is the number of elements in cir_Inputs_Name

Testing and Limitations:

We approached testing by allowing each member to finish their part of the code then write test cases first to ensure that their part of the code was working. Despite this we still faced challenges integrating the code together due to the complexity of the program; however, we were able to do it through debugging and checking the outputs for the 6 test circuits.

While testing we realized some limitations of our program. The most significant limitation is that if a user enters a file that is logically incorrect they may receive an unexpected output. To safeguard our program we dealt with the most common errors, for example not entering enough inputs into a gate.

Challenges and Improvements:

Ahmed Fawzy Abdelkader:

- Finding if an input exists

When iterating through each gate in used gates, where we saved all gates used in a circuit, finding if an input existed in cirInputs, the .stim file data structure, was a challenge. This was because every .stim can be different therefore the input_exists function had to handle incomplete inputs, repeated inputs, and incorrect inputs. The cir_Input_Names data structure was used to track inputs that can be present in the gate, and then if an input was present it was updated through input_exists as well by finding the corresponding value from cirInputs. Ultimately, the best approach was to use hashing since it provided a unique identifier for every permutation of inputs therefore repetitions were avoided.

- 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.

- How do we model the time?

As a group we had several meetings about which method we would use for tracking time in our application: counter, discrete time intervals, use known values from .stim. We decided to use a counter with the knowledge that it would cost more time. We made this decision because we realized that for any .stim and .lib file the user can provide values that are not uniformly distributed therefore if we used discrete time intervals some inputs will be ignored. Initially we wanted to use the known values of time from the .stim, but we found that this is not time based but index based and this caused issues in finding correct propagation delays. We do, however, believe that in a newer version of this program we would find a way to save the known values before and account for new inputs without requiring it to be time based since we think this will enhance the time complexity.

Tony Gerges:

- Issue pulling files where merging did not work so I opted to copy the required files manually.

- In the preoperator section: 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.

- 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.

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.
- The code was not modular in the march 14th milestone, where 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.

The contributions of each member:

- Classes and algorithm design (all)
    - Through our zoom meeting
- Ahmed Abdeen
    - Reading the lib file
    - Reading the stim file

    - Writing the output into sim file

- Tony
    - Pre-evaluation of the operator
    - Circuit reading file(input)

- Ahmed Fawzy Abdelkader:
    - Evaluation function and its helper functions
    - Debugging input/output files

Bonus

Waveforms:

https://github.com/Ahmed-Fawzy14/Logic-Circuit-Simulator/blob/main/Test%20Circuits/Circuit%201/waveform%20circuit1.svg

https://github.com/Ahmed-Fawzy14/Logic-Circuit-Simulator/blob/main/Test%20Circuits/Circuit%205/5circuit5.svg

https://github.com/Ahmed-Fawzy14/Logic-Circuit-Simulator/blob/main/Test%20Circuits/Circuit%206/waveform%20circuit%206.svg

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.

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 in insertValuesInExpression
    - Prompt: This function removes the parentheses but I'm not sure why fix it.
- General syntax questions

References:

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