Digital Design Project
Circuit simulator

Table of Contents:

Introduction

Logic circuits are the fundamentals of all devices and chips manufactured in today's world of ever expanding technology. Technology has been growing at an exceedingly rapid rate. One proof of that theory is Moore's law, it states that the number of transistors in a circuit that make up logic gates doubles approximately every two years. It is truly a remarkable thing that humanity has been able to achieve with advancements made by companies such as Nvidia, AMD, Intel every single day. Nvidia has been doing advancements in circuits with their recent new Nvidia Blackwell GPU that has 208 Billion transistors! This project will not go to such a large scale, but will go in depth to the core of it all. The logic behind simple circuitry will be written in C++ code that keeps in mind scalability. The code provided makes it possible to simulate a digital space of logic gates connected together to form circuits. The simulation also contains every single step and detail of the inputs and outputs and intermediaries. The data that gets displayed is the wire/input/output name, delay, and logic value. The program also offers adaptability alongside scalability meaning any gate can be used on the circuit as long as an expression is provided.

Data structures and algorithms

In order for the data to be stored in a smart and efficient manner we had to take advantage of the STL library provided by C++. A wide range of data structures were used and they are as follows:

- We used vectors to store the gates themselves. The vector type was made as a struct called Gate that stores all the necessary information for a gate specified. The .lib file is read and all the information is broken down and stored accordingly for gate definition
- For the .circ file it was a big file that has many variables that need to be stored. So a mix of an unordered map data structure and vector were used. For the vector the type of it was another struct called component and this struct would store the name of the gate, the output and the input names too. As for the map implementation we store as the documentation specified the values of all the inputs as zero for it to be later modified by the stim file. This is done using a string as a key and the values stored are a pair of a bool and an int which sets both to false and 0. One component for the logic value and the other for the delay. The stim file is then read which changes both the logic values and delay from the same map.
- We then have a priority queue of outputs and an unordered map. These three methods where used to properly and neatly sort the output, they were also used for some logical issues of the code. Firstly we have an unordered map called previous values. This is responsible for checking whether the values in any of the wires experienced a change or not. It stores similar values to the map discussed in reading the files without a key this time as it only acts as a placeholder for the pair or the data part of the previously defined map.The priority queue has a simple application of arranging all the data based on delays.
- Finally a set was used in the postfix function to ensure uniqueness of the delay and avoid replication.

Algorithms:

As for the star of the show we have the algorithm that was the most time consuming and makes the whole program function. This algorithm is the Postfix calculator. The best part about this algorithm is the adaptability it has to add new operands and the fact that it can manage any expression and return the output correctly. It does not matter the number of inputs in the expression it will still manage it adding to the scalability. We start off the .h file with a function called precedence that determines the precedence of all operators that can show up in the expression. And then there is the function that converts a character to a string of size one. This was done because some of the algorithms used and data structures only used strings as parameters. Then we have the infixtoPostfix function which is a very textbook function that converts from infix to post fix by removing the parentheses and smartly setting up the output of post fix to be later processed.

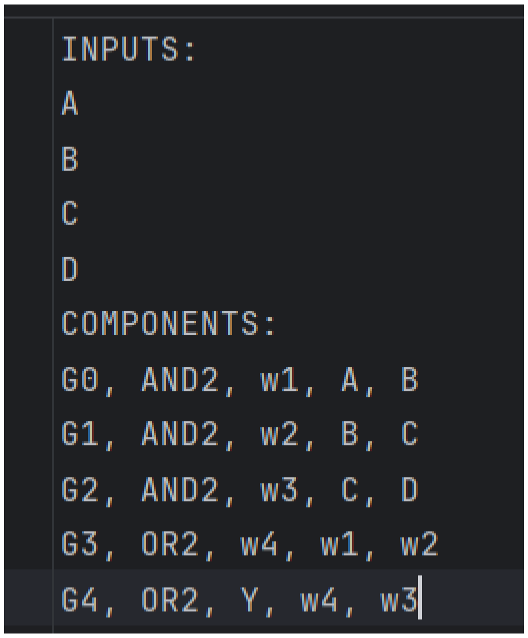
Now for the evaluation function of the postfix this is where the real functionality of the program begins. Let’s talk about the parameters of the function because this function was made to take many parameters. Firstly the postfix that was processed by the function previously discussed is the first parameter. And Then we have the map with the input names and their delay and logic values from the stim file. And then the third parameter is to pass the components to be used in the function to check for delays, input and output names etc… Then we have an integer i passed so that we can know which gate we are on now to adjust it. And then the delay of the gate is also passed so that it can be added and not reseted from the start. And finally we have the priority queue to store the outputs. The first step is that we extract the index if a letter is found indicating that we have an input name. And using that index we can start pushing in the operands stack of booleans we created inside the function to start doing the arithmetics. We also add the delays of the inputs in a vector of input times by accessing the right aspects of the map we passed. We keep adding the inputs until two operands are added and the operand size is two; this will generally happen when we find a symbol that is not alphanumeric. And then we check if we have a non-symbol. If we do then we only pop once invert then push back into the stack the logic value stored in the boolean stack. As for any other logical operator we will check and do its calculations accordingly. In this example we have the AND and the OR we pop twice rather than once. This keeps happening until the postfix is finished and when that happens the operand size will be less than 2. When This happens the delay is calculated and is checked for uniqueness and an output object is made and everything is stored both on the map and priority queue.

As for the inputs we needed to take into consideration that there could be the same input with different values at different delay times. This is done through a priority queue in the main. The main takes in the file names and runs the program on them to store data from several different files. It then enters a loop where it processes each component in the circuit. For each component, it checks the inputs and calculates the output based on the logic of the gate. This output is then pushed onto the inputs priority queue. To avoid duplicate outputs and only display the values that change from the last time, the program uses an unordered_set to keep track of the outputs that have already been printed and an unordered_map to keep track of the last value for each name. Finally, the program writes the simulation results to an output file named simulation.sim. The results include the time stamp, the name of the component, and the value of the output. The program ensures that only the outputs that have changed from the last time are written to the file. If the output file is successfully written, a message is displayed to the user. The program then closes the output file and ends.

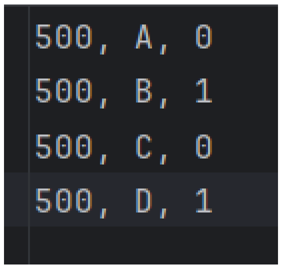
Testing

In order to check the validity of our program test cases were made and were simulated on logicism to check if the outputs matched. These were the results:

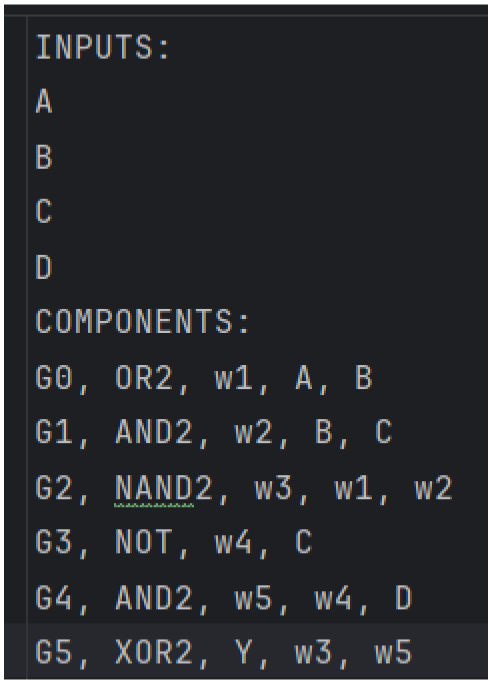
Test 1:

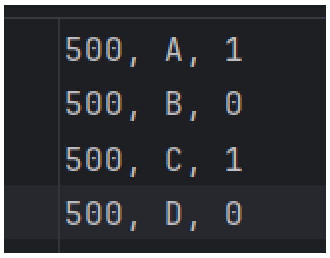


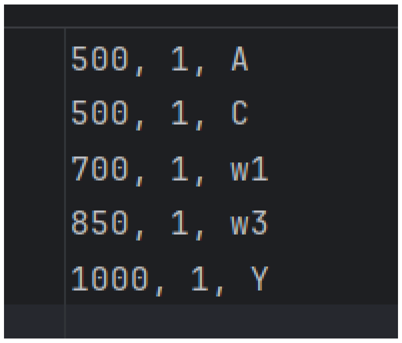
```
INPUTS:
A
B
C
D
COMPONENTS:
G0, AND2, w1, A, B
G1, AND2, w2, B, C
G2, AND2, w3, C, D
G3, OR2, w4, w1, w2
G4, OR2, Y, w4, w3
```


```
500, A, 0
500, B, 1
500, C, 0
500, D, 1
```

Test 2:


```
INPUTS:
A
B
C
D
COMPONENTS:
G0, OR2, w1, A, B
G1, AND2, w2, B, C
G2, NAND2, w3, w1, w2
G3, NOT, w4, C
G4, AND2, w5, w4, D
G5, XOR2, Y, w3, w5
```


```
500, A, 1
500, B, 0
500, C, 1
500, D, 0
```



```
500, 1, A
500, 1, C
700, 1, w1
850, 1, w3
1000, 1, Y
```

These are some of the few test cases the rest is on the github repository

Challenges

We faced many challenges when undertaking this project. One of the main difficulties was problem definition. We struggled with problem definition because there are so many approaches that could be taken to solve this problem of creating digital circuits. But in the end the approach that was chosen is very satisfactory. We also ran across many logical errors especially with the postfix evaluation function as there were many hurdles to be able to interact with the files and manipulate them easily in the operands stack. Another challenge we faced was making the code look nice and organized in order for the user to understand it well. As for the terminal interface it is pretty straightforward to either input a file path or default to the standard file path. Another issue that was faced was the output sim file there were many conflicts caused due to not fully grasping what the output sim file should be composed of. At the end we settled for that everything is initialized as 0 as stated and if any wire or input is changed to a 1 it is outputted in the sim file.

Individual contributions

Both Marwan and Mazin worked fully on the code together with full functions implementation and algorithm designs. It was overall a 50/50 split in the code.

Marwan worked on the report and edited all errors out of in the test files on github

Khalid worked on all the test cases and the time diagrams and everything outside the code

Conclusion

In conclusion, this proved to be a very difficult project to undertake but with teamwork and high efforts it was managed and done within a concise time frame. The circuit simulator experience was filled with many challenges and errors to debug but overall it bettered our skills as thinkers and programmers. we successfully navigated through complexities, emerging with a functional tool that showcases our dedication to innovation and excellence. This project not only deepened our knowledge of data structures, algorithms, and testing methodologies but also underscored the importance of teamwork and resilience in overcoming obstacles.