THE AMERICAN UNIVERSITY IN CAIRO
الجامعة الأمريكية بالقاهرة

CSCE 2301

Section 2

Dr. Mohamed Shalan

Yasmina Mahdy - 900221083

Rana Taher - 900221430

Haya Shalaby - 900222031

First Project

Circuit Simulator - Project Report

Outline:

Introduction

This project simulates the output of a logic circuit. The program is given three files. Firstly, the library file, and it consists of all gate components (and their information) that could be needed by the circuit. The second file is the circuit file which contains a circuit's inputs, outputs, and operations executed. The third and last file, the stimuli file, has the external events applied to the circuit. Given these three files, the program is expected to read in and parse information as needed in order to be able to build the circuit, correctly evaluate the stimuli applied, produce accurate outputs, and document the results in the simulation file. This report will cover extensively all the data structures and algorithms used as well as discuss the obstacles faced while putting this project together.

Used Data Structures and Algorithms:

Structure Gate:

This structure has all the attributes of a gate (as read from the library file): name, number of inputs, operation expression, and gate delay.

Structure Signal:

We have created a structure called Signal to encapsulate the name and the value of each signal in out circuit, including inputs, outputs and wires.

Structure Component:

We have also used a structure to model gates in each circuit. The component structure includes the gate's name, type, delay, output signal and input signals.

Set (signal):

To keep track of the signals in a circuit to make sure that no gate has inputs which do not exist in the circuit.

Stacks:

The operStack function in the Library class uses two stack structures, one for values (input values) and one for (logic) operators. These are used for the correct evaluation of the logic operations in order to produce correct outputs.

Look Up Table (log):

We have utilized a LUT in our program to keep track of which inputs go into which gates, i.e. affect their outputs. This LUT is a vector of pairs of Signals and ints, created by looping over every component in the circuit, then creating pairs for every input to this component with its (the component's) index.

Waveform Map:

To implement our GUI we have used a map that maps a signal's name to a structure containing its behavior throughout the simulation (string) and the last time at which that signal had changed.

Priority queue (MinHeap):

We created a min heap called simOrder that is used to create the simulation file by tracking the inputs and outputs of each gate. The values read from the stimuli file are first inputted into simOrder so they can be used to figure out the outputs of each gate with its timelapse and then they are inputted into simOrder so each signal can get executed in increasing order of their timelapses. Each element in simOrder is a pair of Signals and ints in order to compare its elements with the elements in the log to be able to generate the simulation files.

Testing

We adapted a multi-phased testing procedure. Firstly, each part of the code was individually tested by the person responsible for it using unit and output tests. Secondly, once we put the three parts of the code together, we did complete testing and checked the output for its accurateness by comparing it with our expected output. Post our first functional code, we noticed we had to add some more code for better and further functionality which called for repeated testing. Once we finalized the code, and it was running correctly, it actually ended up correcting our own pre-written simulation files.

Time Complexity

n = the size of the min heap

m = number of gates in each circuit

k = number of inputs of a gate

t = the size of the log

r = the size of the stack used in operstack

j = size of the size of the simulation file
q = the size of the time interval

Complexity of the first while loop that reads the stimuli file = $\Theta(n^2)$
Complexity of the setlogic function: $\Theta(mk)$
Complexity of setting the correct initial values of the gate: $\Theta(m^2)$
Complexity of the operstack function: $\Theta(s^2)$
Complexity of logic change: $\Theta(ks^2)$
Complexity of the inner loop of the second while: $\Theta(nt+ntks^2)$
Complexity of the Json function:$\Theta(jn+jq)$

GUI

After our program outputs the simulation to a .sim file, we read that file to create a JSON file that can be then sent to the website WaveDrom, which graphically models the behavior of the circuit throughout the simulation. This is achieved by reading the contents of the .sim file, which is formatted as “time,” “Signal name,” and “value,” then for each line, we use the waveform map to get the last time the signal had changed, and use it with the current time to calculate the period for which the signal was constant, and add that to the string modeling the signal’s behavior. After the last line is read, the program loops over all the signals to set them as “unchanged” since the last time they were read and up till the end of the simulation. Finally, the strings storing the behavior of each signal are outputted according to the JSON format.

Challenges

We faced a number of challenges when coming up with the algorithm for the program. The first of these were how we would check the circuit connections in order to create the simulation files. At the beginning we first came up with a recursive algorithm that checks the input then uses an algorithm similar to the depth first search to traverse through the circuit. Then checks how the change of a single input can affect the gates that it is connected to and we pick one of those gates to check how they affect one of the gates connected to the output of it until we reach the output then goes back and checks the other gates affected. This approach had a major issue. It did not take into account the time lapses of the outputs of the gates connected to the changed inputs. Due to that problem it was almost impossible for us to be able to create an accurate simulation file as we would not be able to distinguish which change comes first. After discussing all these issues we decided to implement a different approach which is to use a min heap. This way we would be able to check the time lapses of the inputs and the outputs of each gate and then give priority to the signal with the smallest time lapse. Then for that given signal the program checks the gates it directly affects and calculates their time lapses then inputs it into the min heap then it pops the signal that was used to make those changes and outputs it into the simulation file. This process is repeated until the min heap is empty and the simulation file has all the changes that occurred in the simulation.

Another challenge was returning an accurate log to the main function. This proved to be a challenge because we were unable to include the library file in the circuit file to avoid circular dependency, as the library file itself includes the circuit file. This resulted in the inputs and outputs of each component being set to a default of 0, even when the actual circuit being modeled would have had different values at the rest state (rest state taken to be all inputs set to 0). Not only that, but all the gate delays were set to zero because the circuit class had no access to the gate delays, which were stored in the library class. In order to fix that, we decided to add a function called setLogic to the library class that is called on every component in the circuit before the log is created. The setLogic function reads the component with its inputs, the way the function is called in the main guarantees that the inputs of the component the function is called on are all set to their correct values. LogicSet, then performs the operation specified by the gate type on the inputs, and tests whether the result is different from the output of the component, if that is the case, the value of the output is modified, then the function loops over all the components and changes the same signal if found in the inputs of any other gate.

Additionally, we faced some issues with the operStack function. We had first referenced GeeksForGeeks' code for Expression Evaluation which combines both the process of changing an Infix expression to a Postfix expression and, then, evaluating it for the result. However, afterwards, we realized many things would need to be changed in order to apply it to our specific case correctly. Even after the adjustments, we still observed incorrect operation of the function which led us to a long, line-by-line debugging session. The major two issues were substituting in (pushing) the correct input into the values stack and dealing with the differing position of the NOT (whether it's on an individual input or on a bracket). For the earlier, we used the number in the operation expression (e.g. the 1 in i1) as guidance to accessing and pushing the correct input. As for the latter, we opted to use a flag that will indicate the position of the NOT, and so we can choose the next step in expression evaluation accordingly.

Contributions of Each Member

Please note while each one of us had their own parts to individually model, we have all participated extensively in coming up with the algorithm, as well as the testing and debugging of the program as a whole, which constituted the most difficult and time-consuming part of the project.

Haya: Creating the library class.

Yasmina: Creating the circuit class.

Rana: Writing the main function.
All: Coming up with the algorithm, testing, debugging, and GUI.

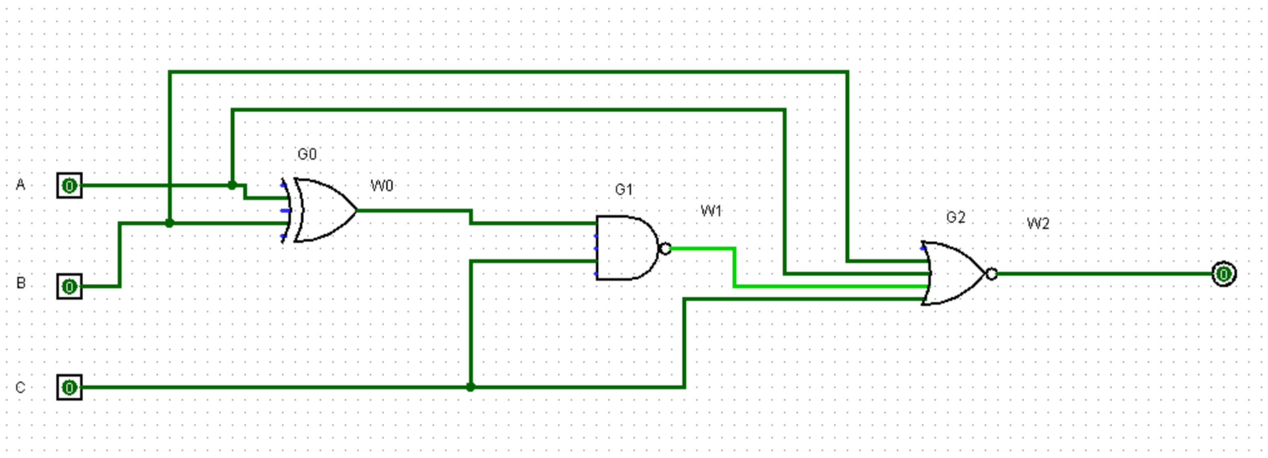
References

GeeksforGeeks. (n.d.). Expression Evaluation. Retrieved from https://www.geeksforgeeks.org/expression-evaluation/

(ChatGPT, personal communication, March 16, 2024)

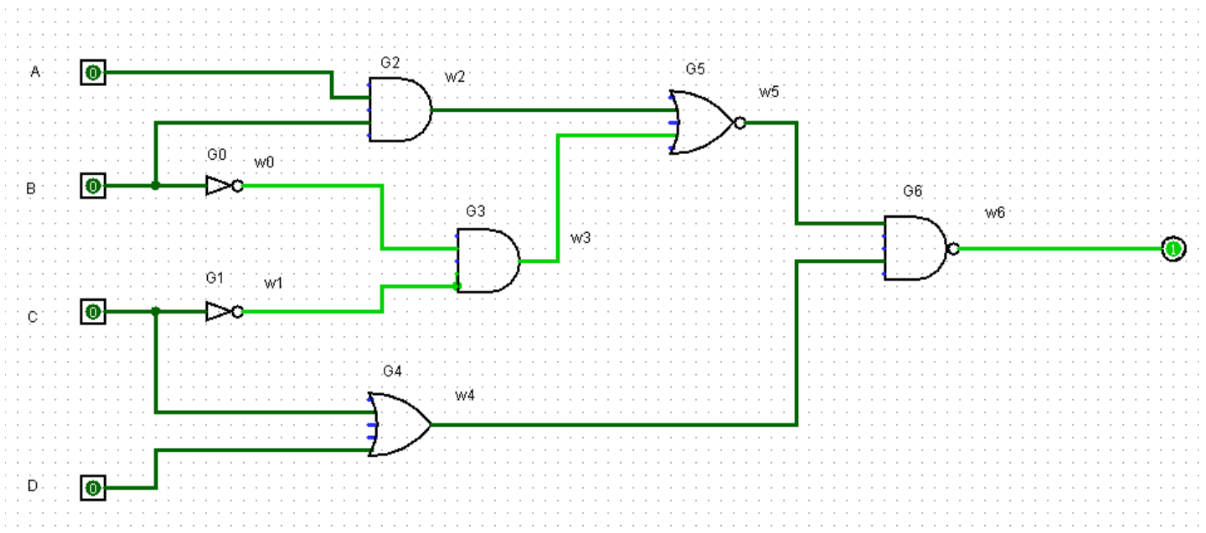
Visualization of circuits

Circuit 1



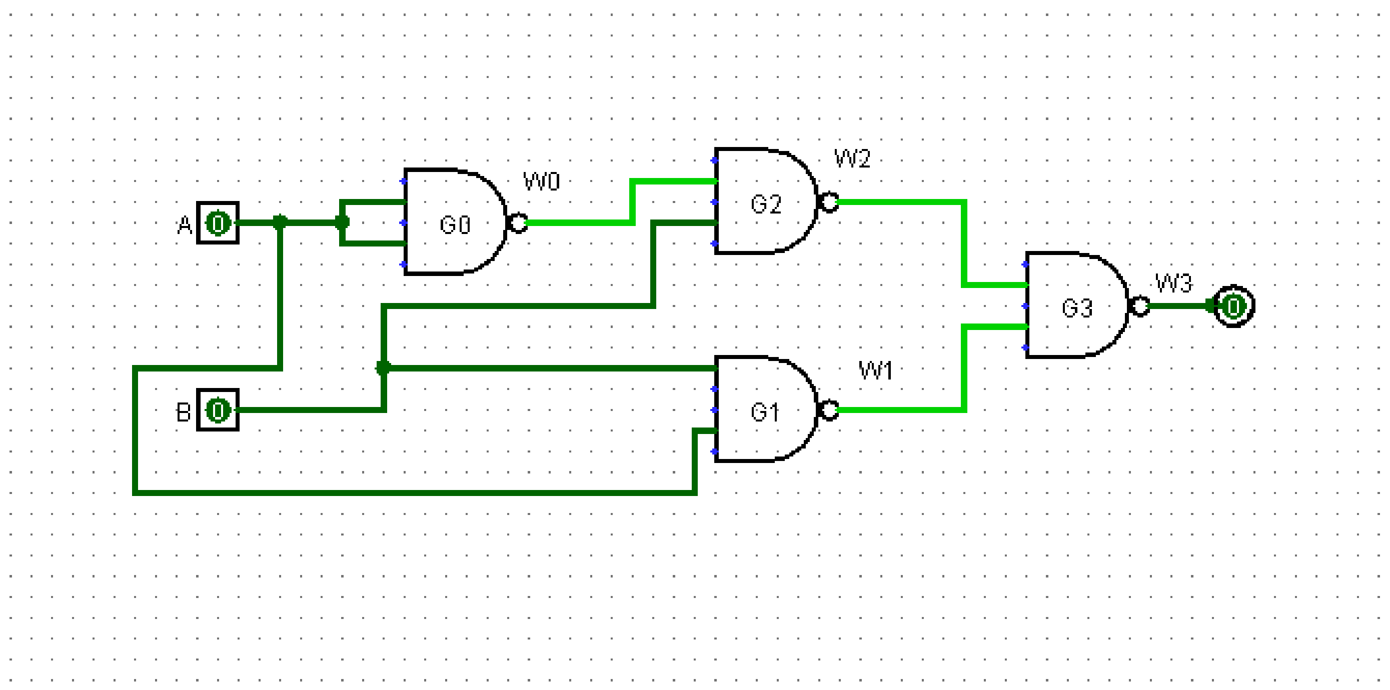
A	B	C	W0	W1	W2
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	1	0
1	1	1	0	1	0

Circuit 2



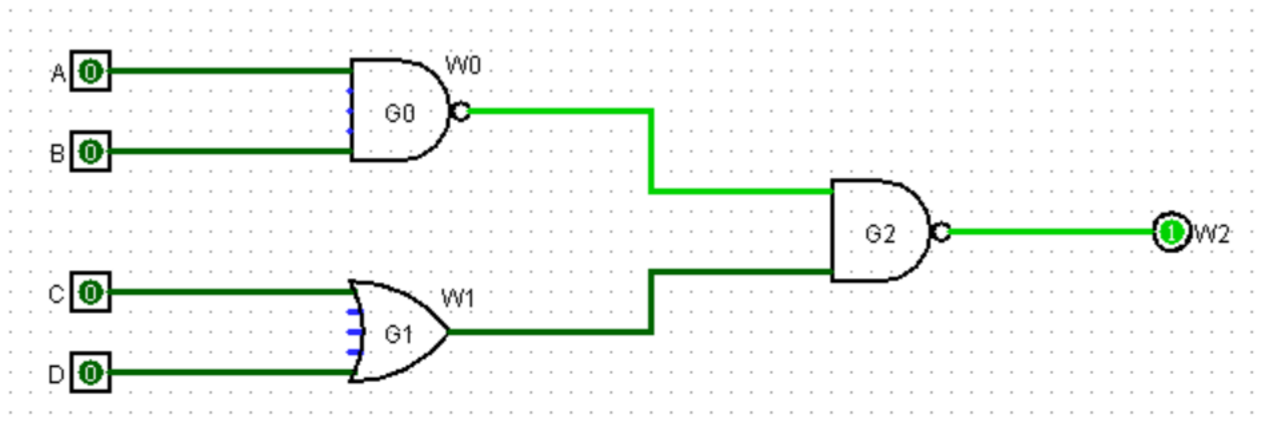
A	B	C	D	W0	W1	W2	W3	W4	W5	W6
0	0	0	0	1	1	0	1	0	0	1
0	0	0	1	1	1	0	1	1	0	1
0	0	1	0	1	0	0	0	1	1	0
0	0	1	1	1	0	0	0	1	1	0
0	1	0	0	0	1	0	0	0	1	1
0	1	0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	0	0	1	1	0
0	1	1	1	0	0	0	0	1	1	0
1	0	0	0	1	1	0	1	0	0	1
1	0	0	1	1	1	0	1	1	0	1
1	0	1	0	1	0	0	0	1	1	0
1	0	1	1	1	0	0	0	1	1	0
1	1	0	0	0	1	1	0	0	0	1
1	1	0	1	0	1	1	0	1	0	1
1	1	1	0	0	0	1	0	1	0	1
1	1	1	1	0	0	1	0	1	0	1

Circuit 3



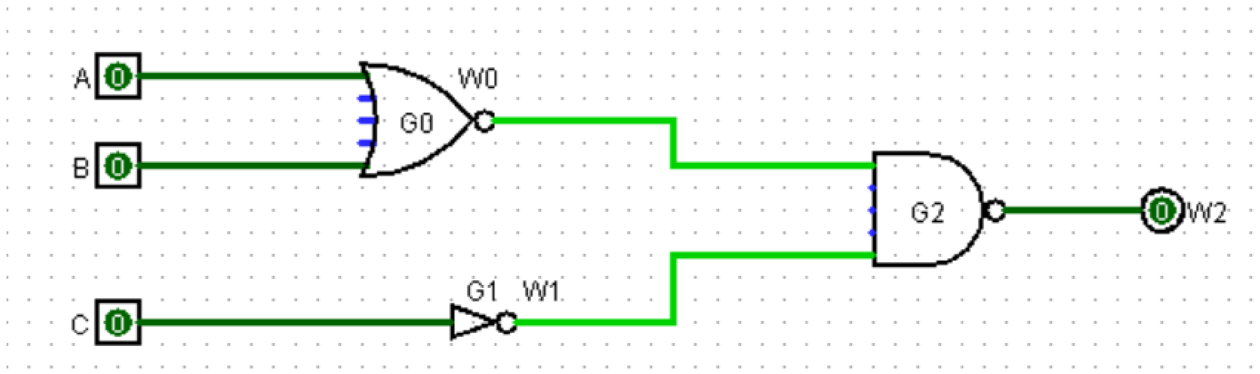
A	B	W0	W1	W2	W3
0	0	1	1	1	0
0	1	1	1	0	1
1	0	0	1	1	0
1	1	0	0	1	1

Circuit 4



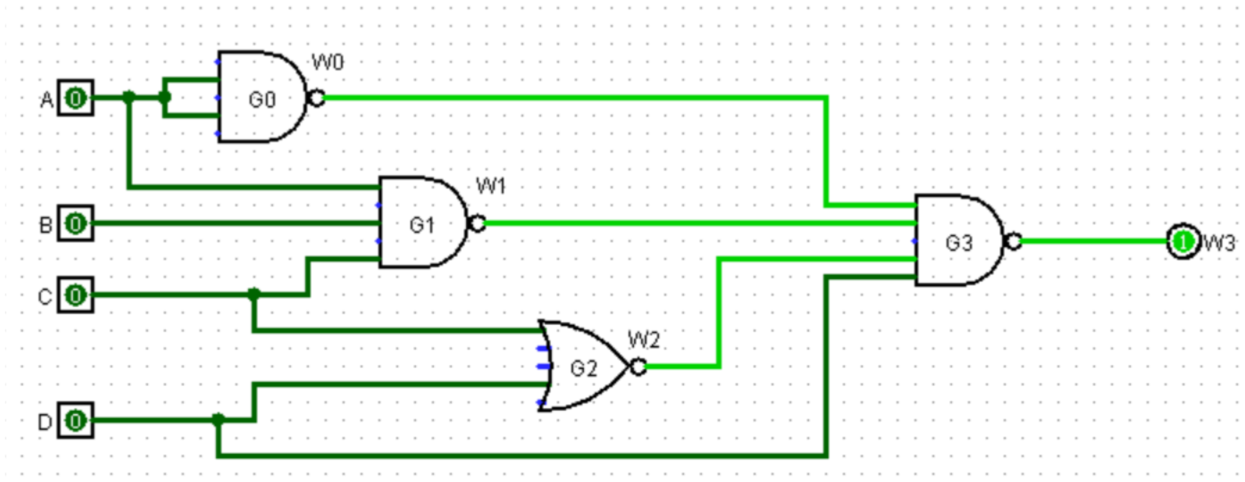
A	B	C	D	W0	W1	W2
0	0	0	0	1	0	1
0	0	0	1	1	1	0
0	0	1	0	1	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	1	0
1	1	0	0	0	0	1
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	0	1	1

Circuit 5



A	B	C	W0	W1	W2
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	0	0	1

Circuit 6



A	B	C	D	W0	W1	W2	W3
0	0	0	0	1	1	1	1
0	0	0	1	1	1	0	1
0	0	1	0	1	1	0	1
0	0	1	1	1	1	0	1
0	1	0	0	1	1	1	1
0	1	0	1	1	1	0	1
0	1	1	0	1	1	0	1
0	1	1	1	1	1	0	1
1	0	0	0	0	1	1	1
1	0	0	1	0	1	0	1
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	1
1	1	0	0	0	1	1	1
1	1	0	1	0	1	0	1
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	1