Logic Circuit Simulator

Project Description:

Your group has been tasked with making a simplified backend code for a GUI that simulates digital logic systems (like LogicWorks.) Note the task at hand is not the GUI itself, rather this project focuses on programming the functionality behind the system. In other words, you will have to program how the different logic components behave and how these interact with each other. This means your team will have to plan, communicate, and distribute the workload if you want to do the project in an efficient and timely manner.

Project Goals:

1. Get a firm grasp of Object-Oriented Programming (OOP) practices.
2. Testing code, so it complies with project demands.
3. DOCUMENTING THE CODE.
4. Using data structures to solve problems.
5. Reading other people’s code.
6. Collaborating and understanding each other’s code.
7. Planning and structuring the code, so it does not look like a total mess.

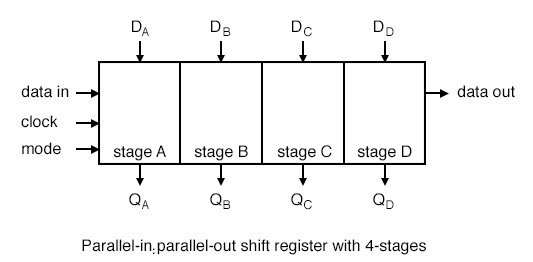
Project tasks:

1. Planning:
   1. Your team should first figure out what the given tasks are and how you should go about doing them.
   2. Outline what structure the code will have. This includes classes, methods, and functions the code should have to perform the tasks. Note this will probably change as you go along, but this will give you an idea of the tasks that you need to perform.
   3. Distribute the workload accordingly. This may also change as you start to realize some of the tasks are not as simple as they seemed to be.
   4. Read any necessary additional material that is required to do the part that you were assigned.
2. Functionality
3. Components: Your program must have the following components:

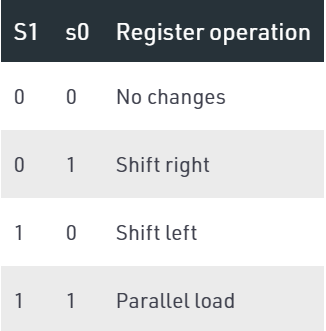
* And gates
* Or gates
* Xor gates
* Nand gates
* Nor gates
* Universal Shift Registers (USR)
* Inverters
* Constant outputs (a component that constantly outputs either 1 or 0)
* Clock
* Switch
* MUX

Requirements:

1. The And, Or, and Xor gates must have the capacity to receive an arbitrary number of inputs, not just 2 inputs.
2. The Nand and Nor gates cannot receive more than 2 inputs.
3. The USR should be able to perform right shifts, left shifts, parallel loads, and remain idle with the correct inputs. In addition to that, your shift register must use the same inputs as a normal USR.
4. The clock should be able to control the behavior of clock controllable components. In this case, only the USR is clock controllable.
5. The USR you create for the project should be based on the figures:

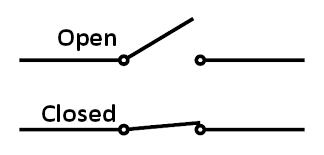


where the D’s are the parallel inputs, Q’s are the parallel outputs, data in is the serial input, data out is the serial output, and mode chooses how the USR operates. The mode of the USR is determined by the inputs s0 and s1 in the USR component and the possible operations are:

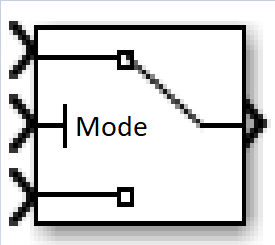


For s0 = 0 and s1 = 1, data\_out = data\_in and data\_in = data\_out for the USR.

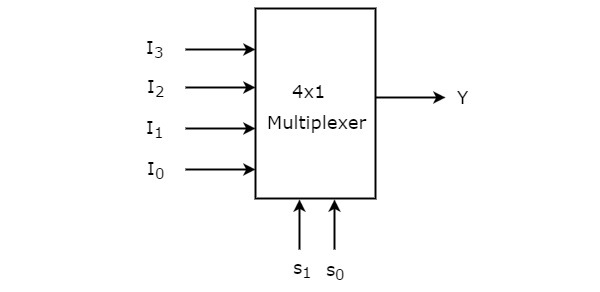
1. The switch component also has a mode input. This determines what it should do with its inputs (just like the USR.)
2. Depending on the size of the input, the switch should do one of the following:
   1. If the switch receives 1 input, it should act as a “disconnected line”. That is, if the line is closed (short circuited), it passes the current value of the input. Otherwise, if the line is opened (open circuit), it does not read the value of the input and instead it uses the last value of the input for when the switch was closed. Mode = 0 should open the switch and Mode = 1 should close the switch.



* 1. If the switch receives 2 inputs, it should behave like a 2x1 MUX. The mode input should determine which of the two inputs is used as the output.



1. The MUX component must be 4 x 1. In other words, it chooses from 4 inputs and saves only one of them as its output. When creating your component, base yourself on the following image:



1. System

Requirements:

1. Since your team is designing the backend for a logic circuit simulator, your program must be able to simulate **any** digital network with the logic components you are tasked to program. This is different from your run-of-the-mill script that can simulate a specific logic circuit.
2. Your system must have a “run” method to start the simulation. When the simulation finishes, it must generate a .txt file that shows the state of each component for each iteration. The .txt file that the code above generated was sent as an email attachment along with this document. In addition to that, you need to show what the register contains, which is also shown in the .txt file. Note that if the register shifts right then the text file must show this happening. The same goes for left shifts. Furthermore, the .txt file should have some indicator to where the components are located. This is also shown the code generated file I sent.
3. (Optional) You must implement a way to stop the simulator from doing unnecessary runs in the simulation. For example, if the registers are empty, you really cannot do much else and the system will stay doing nothing for the rest of the simulation if it is not stopped.
4. You will need to implement a way of determining in what order you need to execute the components.
5. Miscellaneous
6. Try to make the code as clean as possible. This means your group should trim and remove any redundancies in the code.
7. Debugging
   1. There is a high chance that when you finish the code, your code does not run. Your team needs to check and see what is wrong with the code. This requires that you have full knowledge of what is happening with the part you were assigned for. If needed, you should try and check your teammate’s code to see if you can spot the error.
   2. You will need to use pytest to write scripts to test if your code is giving the correct output.
   3. After you finish testing, I will give you some logic circuits you should simulate along with some of the respective .txt files that my code generated, so you can verify if your code is working properly.
8. Writing code guidelines
   1. For writing the code, you will follow “the rule of 30”:
      1. All methods within a class must be less than or equal to 30 lines. This does not include the initial docstring and any spaces in between the lines.
      2. Each class must contain less than or equal to 30 methods. (Although, you should not reach this point for this project)
   2. As a rule of thumb for writing code, if a method is too long then there is a part of that method that can be put as another method.
   3. You should comment on what a tiny chunk of code does within a function/method.
   4. When writing names for variables, write a name that tells you what the variable is used for. In other words, write variable names in such a way that if you leave the code for a while and comeback to read it again, it should not take you long to know what your code did. The same goes for classes, functions, scripts, etc.
9. Reading solution code
   1. The solution to this project is in the following repo:

<https://github.com/YousefSalaman/LogicCircuitSimulator>

Treat this link in the same manner that you would with a solution manual. Only read this if you have finished with a part of the code and want to compare your results with the “project solution” or if you’re really stuck on a part of the project. **Do not copy of this. I will most likely know that you copied of this.**

* 1. After you finish the code, you should read the repo and see how a more optimal solution would be for the code.