A group ~~torture~~ project designed by Yousef Salaman

Project Description:

Your group has been tasked with making a simplified "framework" 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.
8. Not killing Yousef when this is all over.

Project tasks:

1. Planning:
    a. Your team should first figure out what the given tasks are and how you should go about doing them.

b. 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.

c. 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.

d. Read any necessary material that is required to do the part that you were assigned.

2. Functionality

a. 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 switch component must have a parameter that tells it which input it should choose to route. An additional requirement is that the switch must be able to simulate a “disconnected line”. That is, it only has one input, but you can “open”

the line, so that input cannot be read anymore, and the switch outputs the last input it read before opening. When the switch is “closed”, it passes the current value of the input.

6. 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.

b. System

Requirements:

1. Since your team is designing a framework 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. An example of a flexible network architecture is given in the figure below.

```
from LogicCirSim import *

a = ConstOut('Cst1', 0)
b = ConstOut('Cst2', 1)
c = Clock('Clk')

d = UniversalReg('Reg1',4,[0,0,0,0])
e = UniversalReg('Reg2',4,[1,1,1,0])

f = AndGate('And1')
g = AndGate('And2')

h = OrGate('Or1')

i = UniversalReg('Reg3', 4)

Test_Sys = DigitalSystem('TestSys', [redacted], [redacted], num_of_runs = 10)

Test_Sys.Run()
```

I will show you the logic circuit diagram that this simulates when we get together.

By importing from my module and “connecting” the components, I can generate any network with the components that were programmed.

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. 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.

c. Miscellaneous

1. Try to make the code as clean as possible. This means your group should trim and remove any redundancies in the code.

3. Debugging

a. 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.

b. 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.

c. You will need to use Pytest to write scripts to test if your code is giving the correct output.

4. Documenting code

a. For documentation, you will follow "the rule of 30":

i. All methods within a class must be less than or equal to 30 lines. This does not include the initial docstring.

ii. Each class must contain less than or equal to 30 methods.

b. As a rule of thumb for documenting, if a method is too long then there is a part of that code that can be grouped together and put to a separate method.

c. You should comment on what a tiny chunk of code does within a function/method.

5. Reading code

a. When your team is done with the project, I will pass my code to the group, so you guys can check it. Try and go through the code and see what steps I took to achieve the goals here. This may seem trivial but remember that we need to be able to read other people's code to better implement any code our group makes.