PyGic
Digital Simulator

Documentation for PyGic Simulator.

Description

PyGic (stands for Python Logic) is a simple and lightweight Simulator, Written in python. using this software, you will be able to simulate digital circuits and monitor their outputs.
With this package, you can simulate *combinational* circuits.

PyGic, come with 7 standard gates which are AND, OR, XOR, NAND, NOR, XNOR, NOT. Later, I will discuss how to use these gates to describe your circuit. There is also, a built-in digital plotter (to visualize the waveform of the input and output signals.

This project is an open-source project, and more features will be added to it. Like Graphical User Interface, Website-based version, more complete real-world electronics parts, a genetic programming core to design circuits with AI, and so on. So if you want to contribute to this open-source project, contact me at ali.fele@gmail.com.

This document contains the following material:
1. Brief documentation of the code and utilities
2. Tutorial
3. examples

Documentation

Source Code

The first version of this package, contains two source code, main.py, and digiPlotter.py.
main.py contains the simulator kernel. Circuit class, is the only class defined inside the script, which can be used to describe your circuit with. Also, you can run your simulation and calculate the output of your circuit. We will go for that how to use this class later.
digiPlotter.py is a plotting utility, which uses matplotlib in the backend, and you can easily plot the waveform of the inputs and outputs of the circuit. Note that, you don't need to directly deal with this script. main.py uses this library to show the results.

Utilities

As I said before, there are built-in gates in this library, that you can use them to describe your circuit easily. PyGic uses a "gate approach" to define the circuit. you will see later, how you can define your circuit with this approach. The first version of this package only contains the simulator kernel which can only simulate the combinational circuits built with ideal gates.

Future works

AND, NOR, XNOR, NOT.

1. Standard Parts

Ideal gates and components are only useful when you want to learn the basics of logical circuits. But to design a real-world circuit, you will use real-world electronic components which are not Ideal any more. they have propagation delay, input and output limits, etc.
This package also has some of the well-known ICs. If you are using a component that there is no library for it in the software, you can easily add new ones (based on its characteristics discussed on the datasheet). we will get to that soon.

2. Signals (for Sequential Circuits)

every circuit board is designed to manipulate the inputs and generate desired outputs. the inputs can be as simple as the logical 0 and 1 or as complicated as an audio signal. In this package, there will be built-in signals that you can use to examine your circuit. Also, you will be able to add your own custom signal, which we will discuss later.
Here is the list of the future built-in signals:
Sine wave, square wave, triangle wave, sawtooth wave, digital clock, logical 0 and 1 and last but not least a custom sequence.

3. Probes

to figure out what is going on in your circuit (for evaluating and debugging purposes) you need a tool that can monitor the output of a specific node of your circuit. This is easily done

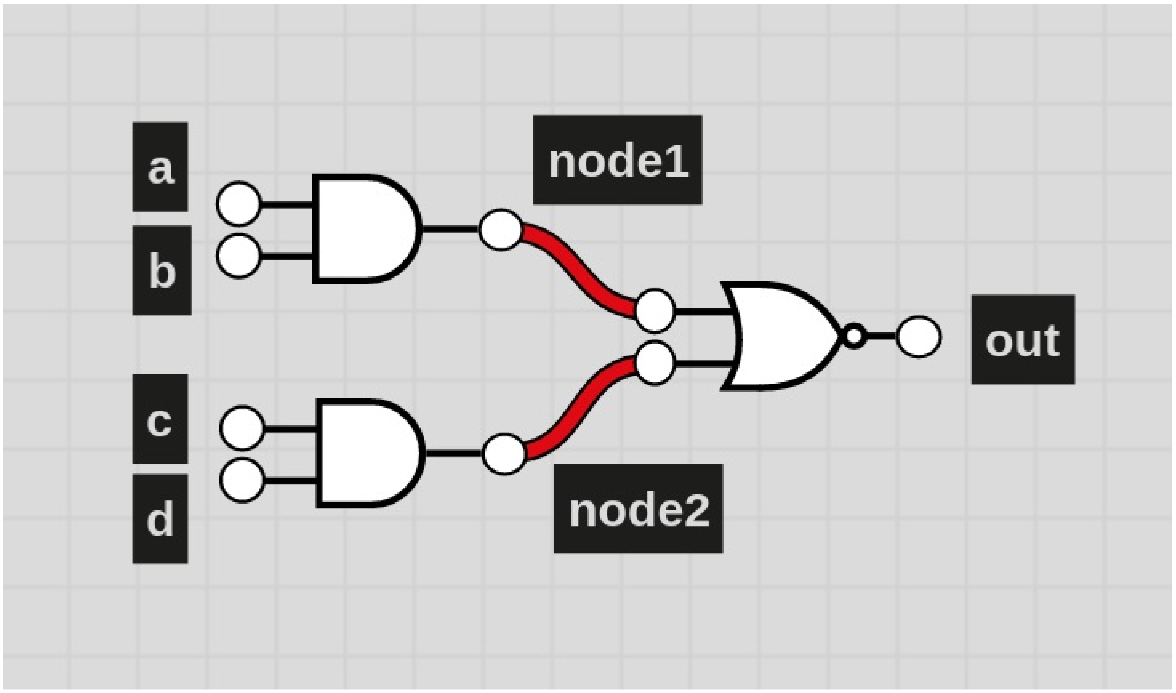
Tutorial

Using this package is very straightforward. As the first step, you need to draw the sketch of your circuit. This is required, because you need the nodes of your circuit, in order to use this package. Now, to define the circuit in the software, you need to make an instance of the Circuit class. This class gets the inputs and outputs of the circuit in its input. you need to
simply use the python built-in dictionary for this purpose. This Dictionary must have only the following keys :

key	Value	Description
inputs	[‘a’,’b’]	‘a’ and ‘b’ are the binary inputs
outputs	[‘out’]	‘out’ is the binary output

when you are creating the new instance of the Circuit class, you need to pass this dictionary as its input. You can check out the source code to figure out what is happening inside this class.

Now suppose that we want to simulate the following circuit in the package[1]:



```
# The first way to define the inputs and outputs
inout1 = {
    'inputs' : ['a','b','c','d'] ,
    'outputs' : ['out']
}



myCircuit = Circuit( inout1 );
```

Architecture Design

Now you have to design the architecture of the circuit.
All of the built-in gates are the methods of the Circuit class.
So to put a gate to the design of your circuit, you need just simply call the appropriate method.
the GATE methods will accept three arguments, in which the first argument is the output and the others are the inputs (exactly like Verilog). Here are the possible gates:

Gate	Method
AND	addANDgate
OR	addORgate
XOR	addXORgate
NAND	addNANDgate
NOR	addNORgate
XNOR	addXNORgate

NOT	addNOTgate

for example, to design the sample circuit you should write

```
myCircuit.ANDgate('node1', ['a', 'b'])
myCircuit.ANDgate('node2', ['c', 'd'])
myCircuit.NORgate('out', ['node1', 'node2'])
```

applying the signal stimulus

You can apply input signals to the circuit and get the output of your system. To do this, you should make an array that contains arrays in it. each of the inner arrays contains the inputs for the input nodes. you can apply more inputs over time by putting more inner arrays. For example, the following arrays put 5 inputs at different times.

```
inputs = [[1,1,1,1],[1,0,0,1],[0,0,1,1],[1,0,1,0],[1,0,0,0]]
```

Monitoring the nodes using probes

Probes are built-in tools in the Circuit class that you can use them to monitor the outputs in different nodes of your circuit. These are something like an oscilloscope or a logic analyzer. The probes will be added to the output node and you don’t need to do anything.

Starting the Simulation

To start the simulation, you need to call the Run method of the Circuit class. For example see the following code:

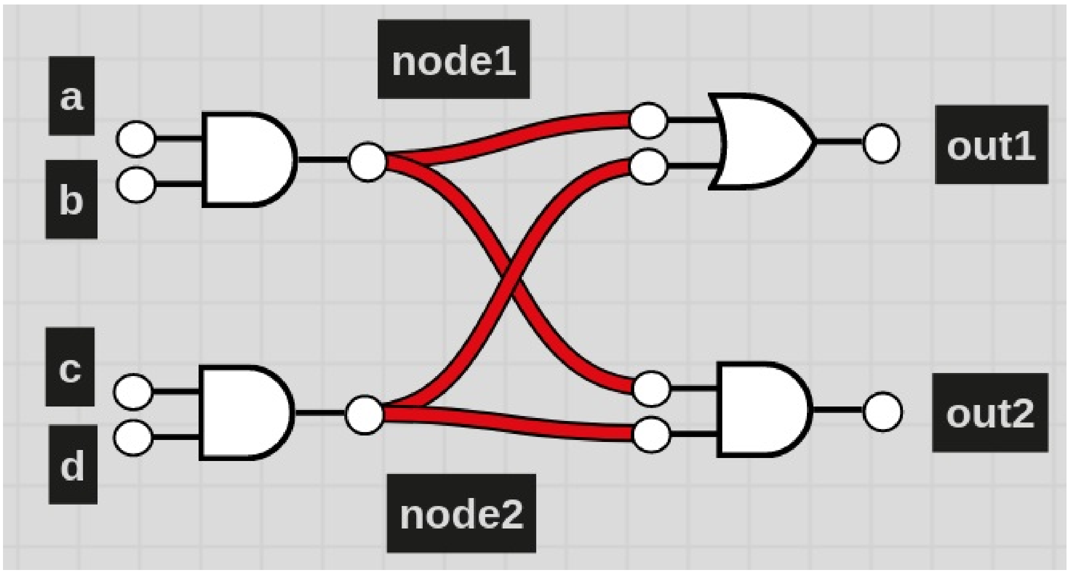
```
result = myCirc.Run(inputs, ['out'], time_step = 5, order=['a','b','c','d'])
```

As the first argument, you need to pass the inputs array we defined before. the second argument is the outputs that you want to monitor. time_step argument determines the duration of each input to the circuit. And finally, the order argument specifies the order of the input node at the ‘inputs’ argument.
This method returns the result, which you can examine it yourself. But the better way to evaluate your circuit, it is better to call the plot method of the Circuit class.

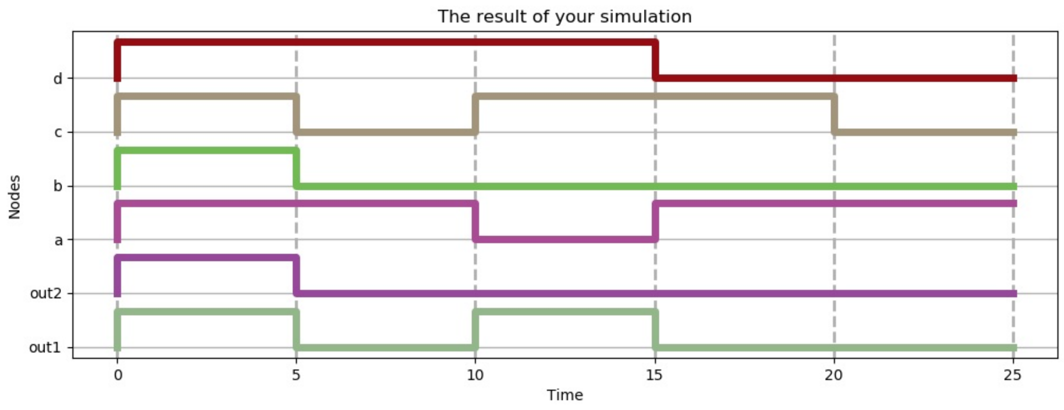
```
myCirc.plot()
```

Examples

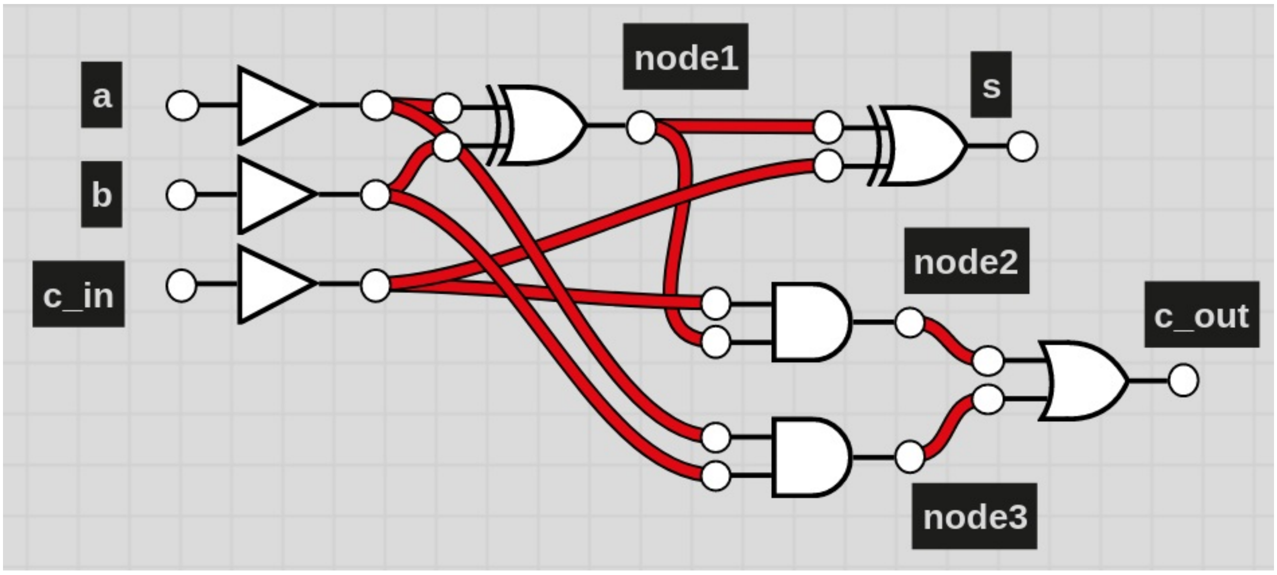
1. Sample Circuit



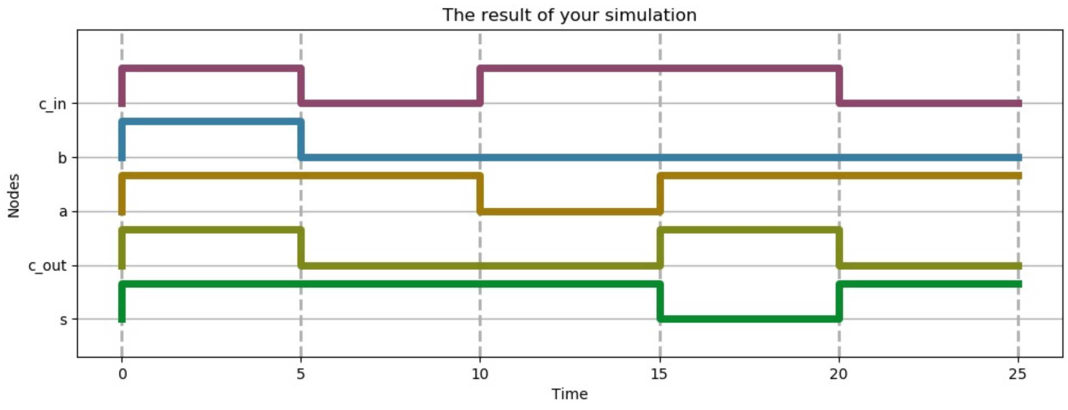
```
from main import Circuit

inout = {
    'inputs': ['a','b','c','d'],
    'outputs': ['out', 'out1']
}

myCirc = Circuit(inout)
myCirc.addANDgate('node1', ['a','b'])
myCirc.addANDgate('node2', ['c','d'])
myCirc.addORgate('out1', ['node1', 'node2'])
myCirc.addANDgate('out2', ['node1', 'node2'])


inputs = [[1,1,1,1],[1,0,0,1],[0,0,1,1],[1,0,1,0],[1,0,0,0]]

result = myCirc.Run(inputs, ['out1','out2'], time_step = 5, order=['a','b','c','d'])
myCirc.plot()
```



2. Full adder



```
from main import Circuit

inout = {
    'inputs': ['a','b', 'c_in'],
    'outputs': ['s', 'c_out']
}

myCirc = Circuit(inout)
myCirc.addXORgate('node1', ['a','b'])
myCirc.addXORgate('s', ['node1','c_in'])
myCirc.addANDgate('node2', ['node1','c_in'])
myCirc.addANDgate('node3', ['a','b'])
myCirc.addORgate('c_out', ['node2', 'node3'])


inputs = [[1,1,1],[1,0,0],[0,0,1],[1,0,1],[1,0,0]]

result = myCirc.Run(inputs, ['s','c_out'], time_step = 5, order=['a','b','c_in'])
#print(result)
#print(myCirc.result)
myCirc.plot()
```



Notes:
1. I have used https://logic.ly/demo website for the drawings.

You will need the following packages to use this library
python 3.7 (or above)
numpy
* matplotlib

Ali Fele Paranj

Physics student at Sharif university of Tech.

Summer 2020