International Journal for Multidisciplinary Research(IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com● Email: editor@ijfmr.com

Modelling Logic Gates in Python

Yashasvini Raghuvanshi[1], Dhananjay R. Mishra[2], Pankaj Dumka[3]*

[1]Department of Computer Science and Engineering, Jaypee University of Engineering and Technology, A. B. Road, Raghogarh-473226, Guna
[2,3]Department of Mechanical Engineering, Jaypee University of Engineering and Technology, A. B. Road, Raghogarh-473226, Guna
(Email: p.dumka.ipec@gmail.com)

Abstract
In this paper, an attempt has been made to develop a Python module for different Logic Gates. The correctness of the codes has been checked against three numerical problems, and it has been observed that the program results match exactly with the results in the literature. As a result, the developed functions have shown high accuracy with the least effort and error in all the cases.

Keywords: Logic Gates, Python Programming, Truth Table

Introduction

Logic gates are basic elements of a digital system that constitute few inputs and a single output. Without Logic Gates, data storage and data transfer would be a difficult task. The concept of logic gates has been implemented in various devices, and we are unable to see them due to technological advancement. Still, its idea is embedded everywhere, like in alarm clocks.There is a total of 7 Logic Gates, and of out seven logic gates, there are two universal gates, which are NAND and NOR gates. The basic seven logic gates are AND, OR, XOR, NOT, XNOR, NOR, and NAND[1,2]. The logic gate receives input in binary format and returns output in binary format. There is an input-output table called the truth table for the tabular arrangement of a combination of inputs and outputs. The truth table represents binary format as logic levels,0 as low and 1 as high. The application of basic logic gates is a vast topic of discussion as it is used in microprocessors, microcontrollers, embedded applications, safety thermostats, and automatic watering systems. The purpose of logic gates is that when a sequence of correct logic has been applied in a particular project, it provides a distinguishing output when the Logic Gates are turned on. While implementing these in pen and paper mode, there are chances that the errors can occur due to some negligence or several logic-building steps. So, the better option is to implement the logic gates through programming.

Here comes the importance of Python language for programming. Python is effortless to understand, with lucid syntax and a rich programming community[3–6]. Modelling any problem in Python requires a few lines of code, backed up by modules like SymPy, NumPy, SciPy, Matplotlib etc. [7–12]. Several researchers have adopted this language for numerically and symbolically solving different engineering problems[13–15]. In this research article, the Logic Gates were modelled in Python to ease the computation and make the solution process error-free.

A Brief description of Logic Gates

AND Gate: AND Gate is an important digital logic gate. AND Gate is the logical multiplication of binary digits where 1 is called as high, and 0 is called as low. Two or more inputs are there with an output. When both inputs are in a high state, the output is in the high state, but if either of the input is in the low state or both are in the low state, then the output will be low. It also denotes that AND Gate may have any input probes, but there will be a single output probe. The schematic representation of AND Gate is shown in Fig. 1.

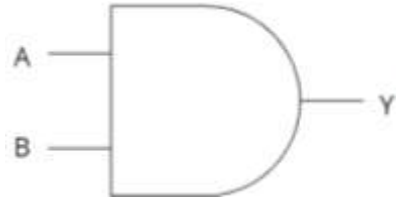

Fig. 1: Schematic of AND Gate

OR Gate: OR Gate is another digital logic gate. This Gate works on logical addition. At least two or more inputs are required to produce a single output. If two inputs are considered, and both are low, then the output will be low; if either of the input is in the high state,, then the output will be true. As we have considered two inputs $2^2 = 4$, then there will be four outputs for the following OR Gate, which is widely used to find out the maximum of between the inputs, that are binary in nature. Fig. 2 shows the schematic of the OR Gate.

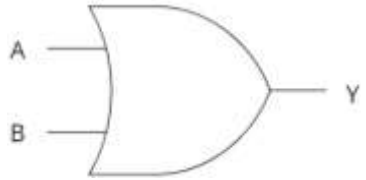

Fig. 2: Schematic of OR Gate

XOR Gate: Exclusive OR Gate is commonly known as the XOR gate (Fig. 3). This Gate comprises two inputs and a single output. This Gate is not basic; it is constructed with the help of other Logic Gates. For example, the high state condition or 1 is obtained only when both inputs are different, not when both are in the high state. The symbol for the XOR gate is an addition sign enclosed within a circle, i.e., $\oplus$.

$$A \oplus B = A.\bar{B} + \bar{A}.B = Y \quad (1)$$

The above equation can be simplified as follows:

When $A = 0$ and $B = 0$

$$A \oplus B = 0 \oplus 0 = 0.\bar{0} + \bar{0}.0 = 0.1 + 1.0 = 0 \quad (2)$$

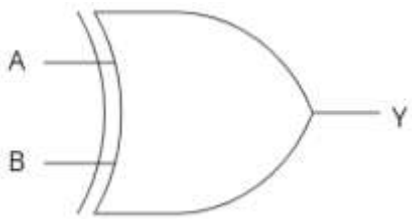

Fig. 3: Schematic of XOR Gate

NOT Gate: NOT Gate is the most basic Gate of all digital logic gates. NOT Gate has one input value and one output value. NOT Gate is also known as inverting buffer. NOT Gate comprises a small circle in the logic diagram called an inverted bubble. Input in NOT gate will complement the output, i.e., if the input is 1 then the output is 0 and vice versa. The schematic of NOT Gate is shown in Fig. 4.

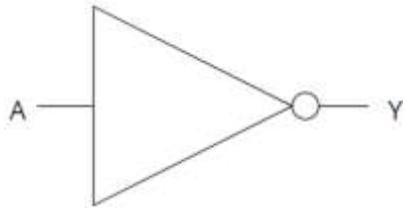

Fig. 4: Schematic representation of NOT Gate

NAND Gate: NAND gate is inverse of AND Gate, or we can call it "Not AND". NAND gate is a universal gate which means all basic gates like AND, OR, and NOT can be represented with the help of the NAND gate (Fig. 5). There can be two or more inputs that will produce a single output. This Gate is a logical complement of multiplication viz $(A.B)^{'} = Y$.

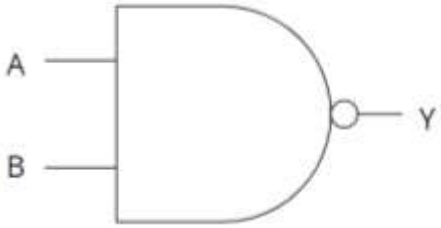

Fig. 5: Schematic representation of NAND Gate

NOR Gate: NOR Gate is the inverse of OR gate, or we can say it is 'not OR gate'. The input state of the NOR gate will be in a high state, while all input states will be in a low state. As it is the inverse of the OR Gate, it can have two or more inputs but a single output. Complement of logical addition is performed for the particular Gate to produce the correct logical output. It can be denoted by:

$$(A + B)^{'} = Y \qquad (3)$$

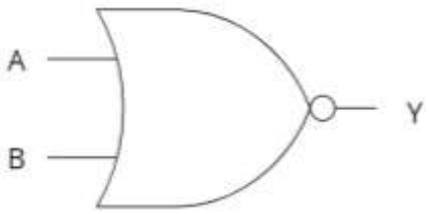

Fig. 6: Schematic of NOR Gate

XNOR Gate: XNOR gate is called the exclusive NOR Gate. XNOR is a hybrid Gate combining XOR and NOT (Fig. 7). It can have two or more inputs. XNOR gate can only produce a high state when both inputs are in a high state. The XNOR Gate is called an equivalence Gate because both inputs are treated the same when the output is for two inputs. For inputs A and B, the output can be written as:

$$Y = (A \oplus B) = ((AB)^{'} + AB) \qquad (4)$$

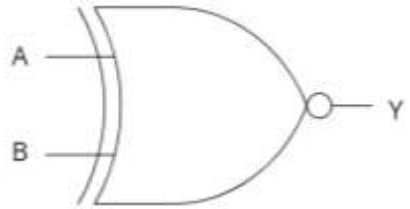

Fig. 7: Schematic of XNOR Gate

The truth table for different gatesis shown in Tab. 1.

Tab. 1: Truth Table for different gates

X	Y	AND	OR	XOR	NAND	NOR	XNOR
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

1. Implementation of logic Gates in Python

The Gates discussed above are converted into Python functions as mentioned in Tab. 2. The function has also been developed to print the truth table.

Tab. 2: Python functions for different Logic Gates

Logic Gate	Python Function

AND Gate	`defAND(a,b):`
	`if a==Trueand b==True:`
	`return1`
	`else:`
	`return0`

NAND Gate	`defNAND(a,b):`
	`if a==Trueand b==True:`
	`return0`
	`else:`
	`return1`

OR Gate	`defOR(a,b):`
	`if a==Trueor b==True:`
	`return1`
	`else:`
	`return0`

XOR Gate	`defXOR(a,b):`
	`if a!=b:`
	`return1`
	`else:`
	`return0`

NOT Gate	`defNOT(a):`
	`if a==True:`
	`return0`
	`else:`
	`return1`

NOR Gate	`defNOR(a,b):`
	`if a==b==False:`
	`return1`

	else: return0

XNOR Gate	defXNOR(a,b):
	if a==b== Trueor a==b== False:
	return1
	else:
	return0

Function for the printing of Truth table	defTruth_Table(GATE,name):
	print('--------------------')
	print(f'a\tb\t{name}')
	print('--------------------')
	for a in [0,1]:
	for b in [0,1]:
	print(f'{a:1} {b:7} {GATE(a,b):8}')
	print('--------------------')

Let us take some examples to demonstrate how these functions can be used.

Example 1: For XNOR and NAND, generate the truth table using the functions developed.

Simply the function `Truth_Table()` will be called with the arguments: Function name and Gate name as follows:

```
Truth_Table(XNOR,'XNOR')
Truth_Table(NAND,'NAND')
```

The code output is shown in Fig. 8.

```
--------------------
a       b       XNOR
--------------------
0       0        1
0       1        0
1       0        0
1       1        1
--------------------
--------------------
a       b       NAND
--------------------
0       0        1
0       1        1
1       0        1
1       1        0
--------------------
```

Fig. 8: Program output for Example 1

Example 2: For different possible combinations of inputs (A, B, C, D) evaluate the final output from the multiple logic Gates as shown in Fig. 9.

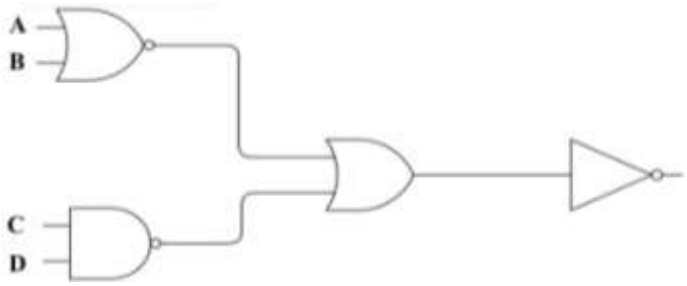

Fig. 9: Multiple logic Gates (Example 2)

The first Gate to start is NORand the second is NAND. Then the output of these will be transferred to OR, and finally, the output of OR will be given to NOT. The outcomeof the multiple logic Gates will be the output from the NOT Gate. In Python, we can simply implement this by first making a list of inputs A, B, C, and D and then looping them to print the truth table as follows:

```
a=[0,1]
b=[0,1]
c=a.copy()
d=b.copy()
print('--------------------------------------')
print('a\tb\tc\td\tOUTPUT')
print('--------------------------------------')
for i in a:
for j in b:
for k in c:
for l in d:
print(f'{i:1} {j:7} {k:7} {l:7}
{NOT(OR(NOR(i,j),NAND(k,l))):7}')
print('--------------------------------------')
```

The code output is presented in Fig. 9.

```
--------------------------------------
a       b       c       d       OUTPUT
--------------------------------------
0       0       0       0       0
0       0       0       1       0
0       0       1       0       0
0       0       1       1       0
0       1       0       0       0
0       1       0       1       0
0       1       1       0       0
0       1       1       1       1
1       0       0       0       0
1       0       0       1       0
1       0       1       0       0
1       0       1       1       1
1       1       0       0       0
1       1       0       1       0
1       1       1       0       0
1       1       1       1       1
--------------------------------------
```

Fig. 9: Program output for Example 2

Example 3: For different possible combinations of inputs (A and B) evaluate the final output from the multiple logic Gates as shown in Fig. 10.

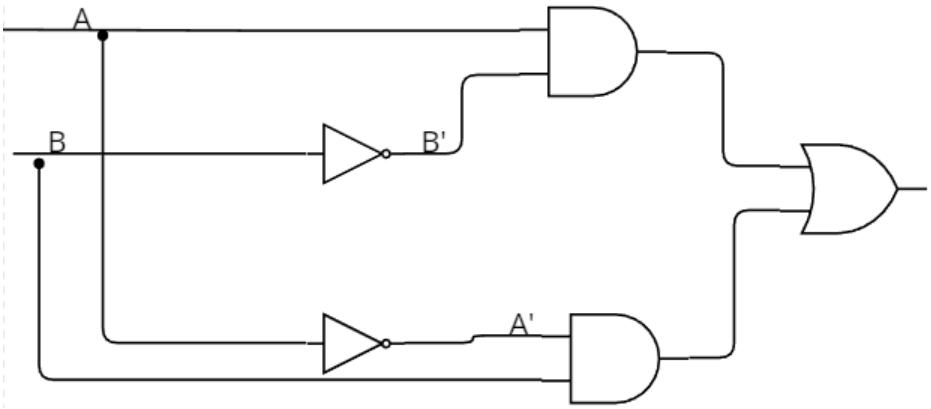

Fig. 9: Multiple logic Gates (Example 3)

The basic inputs are A and B and A' and B' input from NOT Gate. AB' and A'B are input from (NOT+AND) or NAND Gates. A'B+AB' is a combination of NOT, AND, and OR Gates. In Python, we can simply implement this by first making a list of inputs A and B, then looping them to print the truth table as follows:

```
a=[0,1]
b=[0,1]

print('----------------------------')
print('a\tb\tOUTPUT')
print('----------------------------')
for i in a:
for j in b:
print(f'{i:1} {j:7} {OR(AND(i,NOT(j)),AND(j,NOT(i))):9}')

print('----------------------------')
```

The code output is presented in Fig. 10.

```
----------------------------
a       b       OUTPUT
----------------------------
0       0          0
0       1          1
1       0          1
1       1          0
----------------------------
```

Fig. 10: Program output for Example 3

Conclusion

In this research article, an attempt has been made to understand the basic Logic Gates and their implementation in the Python language. The functions were developed for the NOR, AND, OR, XOR, NOT, NAND, and XNOR Gates, along with the code for printing the truth table. The robustness of the functions has been demonstrated with the help of three problems. Furthermore, the developed functions

have given correct results in each case. The article will be of great help to undergraduate students in understanding the Logic Gates along with its programming aspects.

References

1. Martin KW. Digital integrated circuit design. Oxford University Press (New York); 2000.
2. Boylestad RL, Nashelsky L. Electronic devices and circuit theory 11th ed 2018.
3. Dumka P, Sharma S, Gautam H, Mishra DR. Finite Volume Modelling of an Axisymmetric Cylindrical Fin using Python. Res Appl Therm Eng 2021;4:1–11.
4. Huei YC. Benefits and introduction to python programming for freshmore students using inexpensive robots. Proc. IEEE Int. Conf. Teaching, Assess. Learn. Eng. Learn. Futur. Now, TALE 2014, 2015, p. 12–7. doi:10.1109/TALE.2014.7062611.
5. Moruzzi G. Python Basics and the Interactive Mode. Essent. Python Phys., Cham: Springer International Publishing; 2020, p. 1–39. doi:10.1007/978-3-030-45027-4_1.
6. Dumka P, Pawar PS, Sauda A, Shukla G, Mishra DR. Application of He's homotopy and perturbation method to solve heat transfer equations: A python approach. Adv Eng Softw 2022;170:103160. doi:10.1016/j.advengsoft.2022.103160.
7. Cywiak M, Cywiak D. SymPy. Multi-Platform Graph. Program. with Kivy Basic Anal. Program. 2D, 3D, Stereosc. Des., Berkeley, CA: Apress; 2021, p. 173–90. doi:10.1007/978-1-4842-7113-1_11.
8. Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, et al. SymPy: Symbolic computing in python. PeerJ Comput Sci 2017;2017:1–27. doi:10.7717/peerj-cs.103.
9. Johansson R. Numerical python: Scientific computing and data science applications with numpy, SciPy and matplotlib, Second edition. Apress, Berkeley, CA; 2018. doi:10.1007/978-1-4842-4246-9.
10. Dumka P, Chauhan R, Singh A, Singh G, Mishra D. Implementation of Buckingham ' s Pi theorem using Python. Adv Eng Softw 2022;173:103232. doi:10.1016/j.advengsoft.2022.103232.
11. Dumka P, Rana K, Pratap S, Tomar S, Pawar PS, Mishra DR. Modelling air standard thermodynamic cycles using Python. Adv Eng Softw 2022;172:103186. doi:10.1016/j.advengsoft.2022.103186.
12. Pawar PS, Mishra DR, Dumka P, Pradesh M. OBTAINING EXACT SOLUTIONS OF VISCO-INCOMPRESSIBLE PARALLEL FLOWS USING PYTHON. Int J Eng Appl Sci Technol 2022;6:213–7.
13. Huang C. Python Solver for Stochastic Differential Equations 2011;34:1–13.
14. Pawar PS, Mishra DR, Dumka P. Solving First Order Ordinary Differential Equations using Least Square Method : A comparative study. Int J Innov Sci Res Technol 2022;7:857–64.
15. Dumka P, Deo A, Gajula K, Sharma V, Chauhan R, Mishra DR. Load and Load Duration Curves Using Python. Int J All Res Educ Sci Methods 2022;10:2127–34.