Mini Project

Prepared By:-

– Cherukumalli Pardha Krishna Mohan(2022CSB077)
– Abhilash Kumar(2022CSB105)
– Shivam Rai(2022CSB011)
– Souvik Majee(2022CSB028)

Determining Probability of outputs in a Digital Circuit

Introduction

- In this project, we explore a critical aspect of digital circuit analysis: calculating the probability of each gate's output.
- By assigning probabilities to the binary signals (0 or 1) that flow through these gates, we aim to understand how variations in input signals can impact the overall behavior of a circuit.
- This probabilistic approach has significant implications for circuit reliability, performance, and optimization.

Objectives

1. **Topological Sorting:**
   a. The initial objective is to arrange the digital circuit's gates in a topologically sorted order. This ensures that each gate's output depends only on those gates that precede it in the sorted sequence.
2. **Circuit Analysis:**
   a. After sorting, the next objective is to calculate the probability of the output for each gate. This requires considering the circuit's structure and the nature of each gate.
3. **Probability Calculation:**
   a. This objective involves computing the probabilities for each gate in the circuit. The probability of a gate's output depends on the type of gate, its inputs, and their associated probabilities.

Applications and Outcome

- **Reliability Analysis:** Assessing the reliability of digital circuits by
- how likely certain outputs are given varying input probabilities.
- **Circuit Design and Optimization:** Identifying critical gates or pathways to improve circuit efficiency.
- **Probabilistic Computing:** Exploring non-binary outputs and probabilistic logic in computing applications.

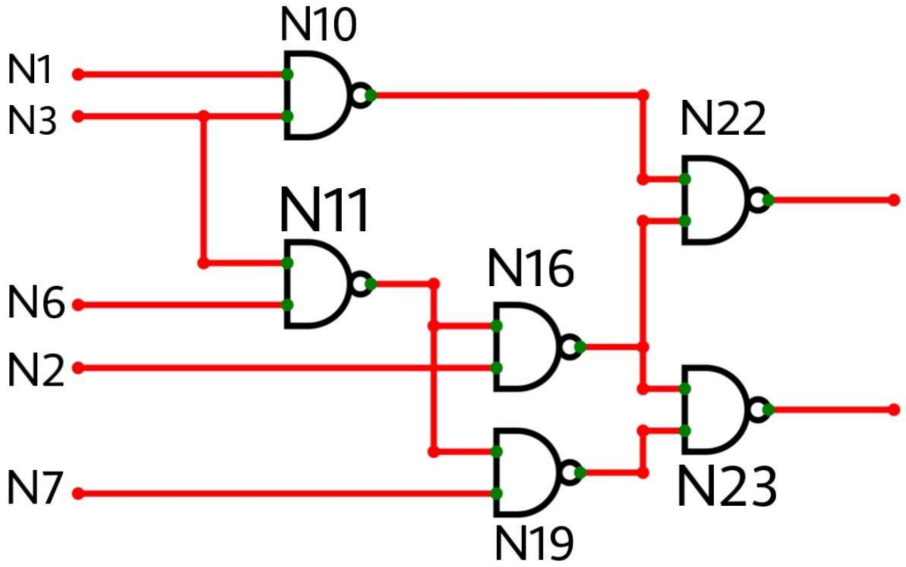
Detailed Approach of the Project

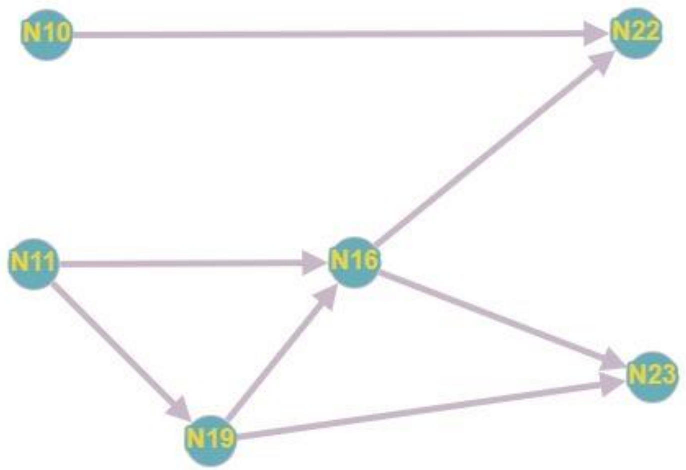
Using toposort for a digital circuit

1. Any digital circuit can be assumed as a graph.
2. And any gate can be named as its output(since any gate has only one output path)

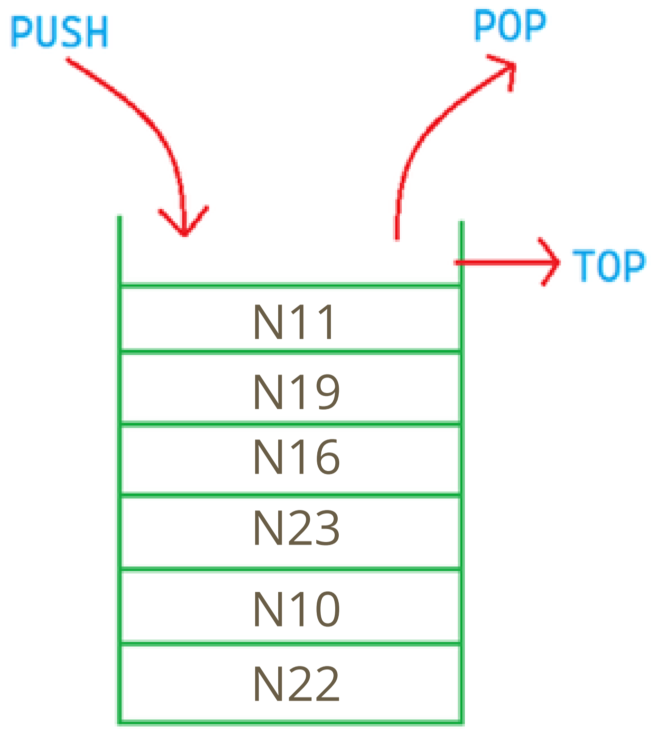
After toposort, we will be left with a stack whose top elements are gates that occur before gates in bottom section

C17.bench file - The analogy between a graph and a digital circuit





Stack after the toposort :



Programming approach :

Step 1 :

We need to be able to identify logic gates which aren’t dependent on the other logic gates (i.e. their inputs are given by external source)

How to identify them?

We create an array of external inputs.

Then in the further steps we use this array to identify independent logic gates.

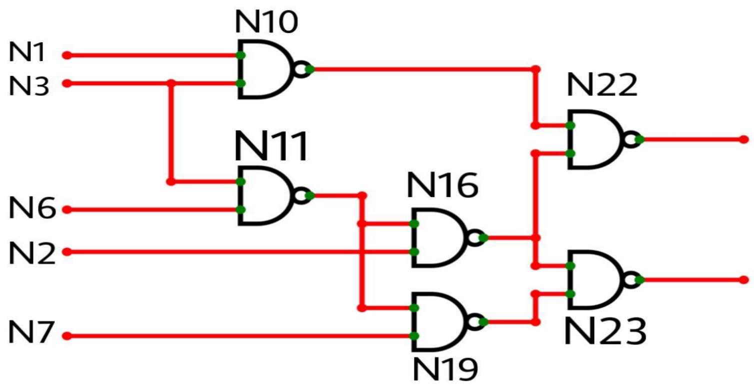
Example :

Array of external inputs :
N1,N2,N3,N6,N7

Independent gates will be :

N10
N11

For a logic gate to be classified as independent, all of its inputs must be of external.

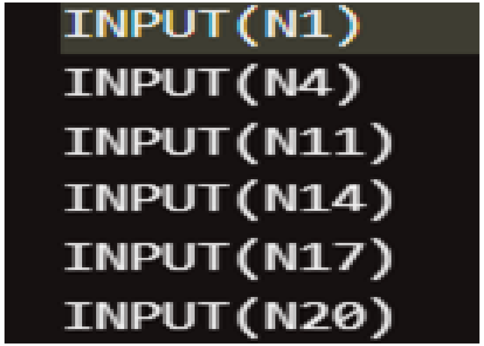


Pseudo code for step 1 :

```
char pIn[n];

char s = line 1 of the file

while(s contains “INPUT”)

        Add the gate to pIn[n]

        s = next line of the file

End while loop
```

C functions used :

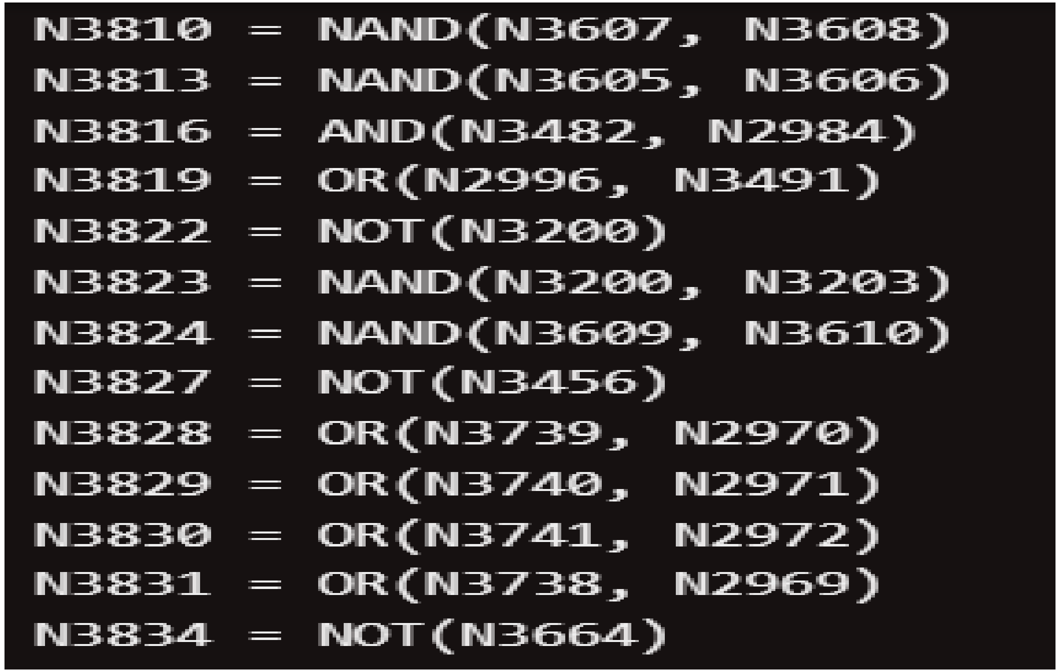
1. fgets() - To read the each line of given bench file(Predefined function)
2. takein() - A self defined function(Discussed later)
3. strstr() - To search for substring in a given string(Predefined Function)



```
INPUT(N1)
INPUT(N4)
INPUT(N11)
INPUT(N14)
INPUT(N17)
INPUT(N20)
```

Programming approach :

Step 2 :
Printing all the relation section to another file for easy access



```
N3810 = NAND(N3607, N3608)
N3813 = NAND(N3605, N3606)
N3816 = AND(N3482, N2984)
N3819 = OR(N2996, N3491)
N3822 = NOT(N3200)
N3823 = NAND(N3200, N3203)
N3824 = NAND(N3609, N3610)
N3827 = NOT(N3456)
N3828 = OR(N3739, N2970)
N3829 = OR(N3740, N2971)
N3830 = OR(N3741, N2972)
N3831 = OR(N3738, N2969)
N3834 = NOT(N3664)
```

Programming approach :

Step 3 :

Using the relation file, we proceed to read each line from it and check for independent gates in the main() function.

Then we start the DFS() of each of these gates

Pseudo code for step 3 :

```
s = line 1 of relation.txt file
while(s != NULL)
        p = read logic gate in s
        if(p == independent gate && notVisited)
          DFS(p)
          PUSH P
          ADD p to Visited list
          s = line 1 of the relation.txt file
          continue;
         End for condition
         s = next line of relation.txt file
End while loop
```

New C Function used :

1. GateExt() - Self defined function(Discussed later)
2. DFS() - Self defined function used to toposort,
   DFS = Depth First Search(Discussed later)
1. vis() - A self defined function used to check if the given logic gate is already visited.
2. AddToVis() - A self defined function used to add the visited logic gates into an array. vis() uses the same array for the visited logic gates' list.

Programming approach :

Step 4 :

After step 3, we will be left with a stack that contains all the logic gates topologically sorted.

So we just POP from the stack and calculate probabilities of each logic gate popped from the stack.

Before we start popping, we make sure that all external inputs as set to a probability of [0.5,0.5]

A structure to store logic gates and their probabilities :

```
struct GateInfo{
        char g[15];
         float p;
};
```

This structure type is used to get the logic gate and its probability together.

Pseudo code for step 4 :

```
GateInfo GI[k];
Add pIn[n] to GI[k] and GI[k] = 0.5
s = line 1 of relation.txt file
while(stack is not empty)
    pop(data)
    if(data in s)
        calculate probab(data) using  relation in s
        store probab(data) in GI[k]
        s  =  line 1 of relation.txt file
    End if condition
    s = next line of relation.txt file
End while loop
```

New C functions used :
Dprobab - A self defined function used to determine probability of a given logic gate using its relation line(Discussed later)

And finally we print the GI[] array in a file

This is the end.

Lets us know about some useful functions used in this process :

1. char* takein(char *s,char *p)

1. char* GateExt(char *s,char *p)

1. void DFS(char *s,FILE* relation,Mystack * stack)

1. Dprobab(char *s, GateInfo *GI,int n)

1. GateDeter(char *s,float f1,float f2) - Used in Dprobab()

char * takein(char *s,char *p)

What the function does :

Given a string, it reads the first logic gate it encounters after ‘(‘

Can be used to read logic gates from lines such as :

1. INPUT(N11) - reads “N11” and stores it in p
2. N22 = NAND(N2, N5, N6, N7) - reads “N2”
3. N4 = NOT(N5) - reads (N5)

Uses ‘)’(closed bracket) and ‘,’(comma) as termination of read

Returns the pointer to the rest of the part to read and if the line actually ends(Incase of 1,3) , it returns NULL

Pseudo code for takein :

```
takein(s,p)

   Increment s until *s==’(‘          N22 = NAND(N2, N5, N6, N7)      N4 = NOT(N5)

   Increment s once more              N22 = NAND(N2, N5, N6, N7)      N4 = NOT(N5)

   while(*s != ‘,’ && *s != ‘)’ )

        copy value of s into p        N22 = NAND(N2, N5, N6, N7)      N4 = NOT(N5)

   End while loop

   if(*s==’)’ )

     return NULL                      N22 = NAND(N2, N5, N6, N7)      N4 = NOT(N5)

   End if condition

return s
```

char * GateExt(char *s, char *p)

What the function does :

Reads the first logic gate it encounter in the string s

Uses ‘,’(comma),’ ‘(space),’)’(closed bracket) as terminators for read

Can used to read :

1. N22 = NAND(N2, N5, N6, N7) - “N22” from this string
2. , N5, N6, N7) - “N5” from this string

Returns the pointer to the rest of the part to read and if the line actually ends , it returns NULL

Pseudo code for GateExt :

```
GateExt(s,p)
    Increment s if it is comma or space     N22 = NAND(N2, N5, N6, N7)     , N5, N6, N7)
    while(*s != ‘,’ && *s != ‘)’  && s != ‘ ‘)
          copy value of s into p            N22 = NAND(N2, N5, N6, N7)     , N5, N6, N7)
     End while loop
     if(*s==’)’ )
       return NULL                          N22|= NAND(N2, N5, N6, N7)     , N5, N6, N7)
     End if condition
return s
```

Combining takein and GateExt

Using both of these functions any relation can be completely read.

Examples :

N22 = NAND(N1, N3, N6, N11, N77)

takein() reads - N1

After reading N1, pointer is handed to GateExt. Using GateExt multiple times, we can read the rest of the line

GateExt can also read N22

void DFS(char *g,FILE *relation,MyStack *stack)

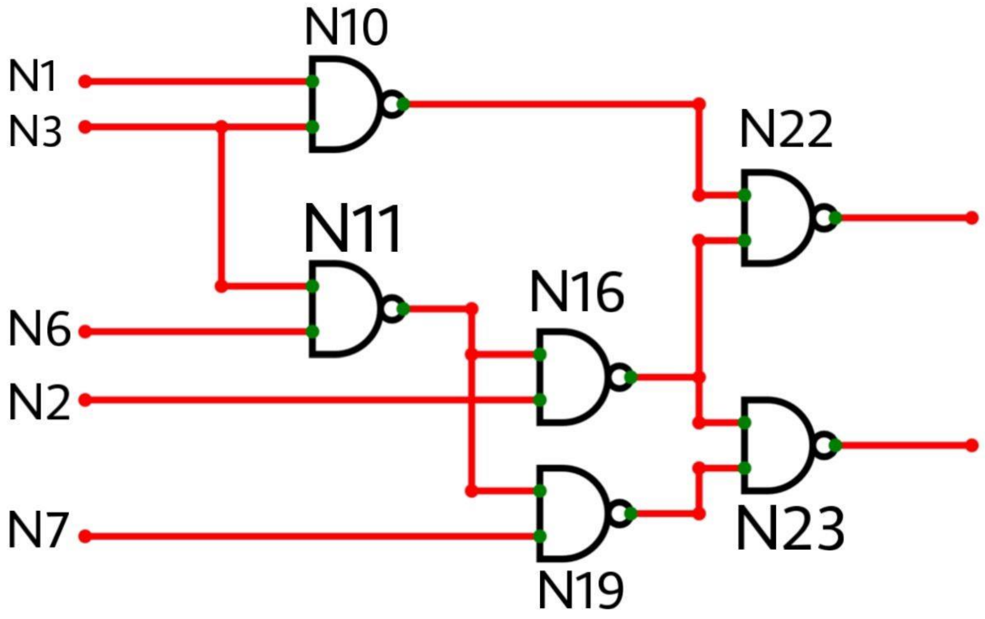
Does the DFS of given gate 'g' nd pushes elements into stack in topological order

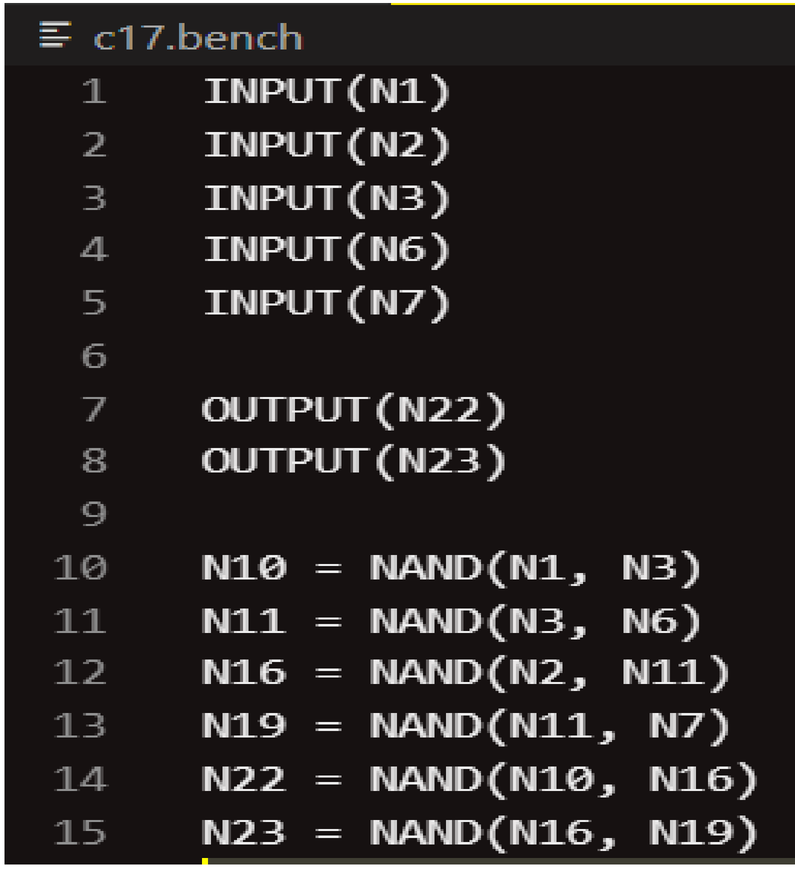
Uses GateExt() and takein() to read a relation perfectly

Runs on recursion

Pseudo code for DFS :

```
DFS(g,rf,stack)                                 N7 Undergoing DFS

    s = read relation file                      s = "N22 = NAND(N4, N6, N7)"

    while(all the relations are not read)

         GateExt(s,g1)                          g1 = "N22"

        char *ptr =  takein(s,g2)               g2 = "N4"

         if(ptr == NULL and g2 == g && g1 notVis)   compare "N4" and "N7"

           DFS(g1)

         else

           ptr = GateExt(ptr,g2)                g2 = "N6"                  g2 = "N7"

           while(ptr != NULL)

                If(g2 == g && g1 notVis)        compare "N6" and "N7"      compare "N7" and "N7"

                  DFS(g1,rf,stack)               DFS(N22)

             End while loop
```




c17.bench

```
INPUT(N1)
INPUT(N2)
INPUT(N3)
INPUT(N6)
INPUT(N7)

OUTPUT(N22)
OUTPUT(N23)

N10 = NAND(N1, N3)
N11 = NAND(N3, N6)
N16 = NAND(N2, N11)
N19 = NAND(N11, N7)
N22 = NAND(N10, N16)
N23 = NAND(N16, N19)
```

float gateDeter(char *s,float f1,float f2)

Given a relation S, it determines the logic gate from it

N22 = NAND(N1,N66,N77)

Using the logic gate and two given inputs’ probabilities, it establishes the suitable mathematical function and calculates suitable probabilities

```
float pOfNAND(float p1,float p2)

{

        return 1.0f - (pi1 * pi2);

}
```

float Dprobab(char *s,GateInfo GI[],int n)

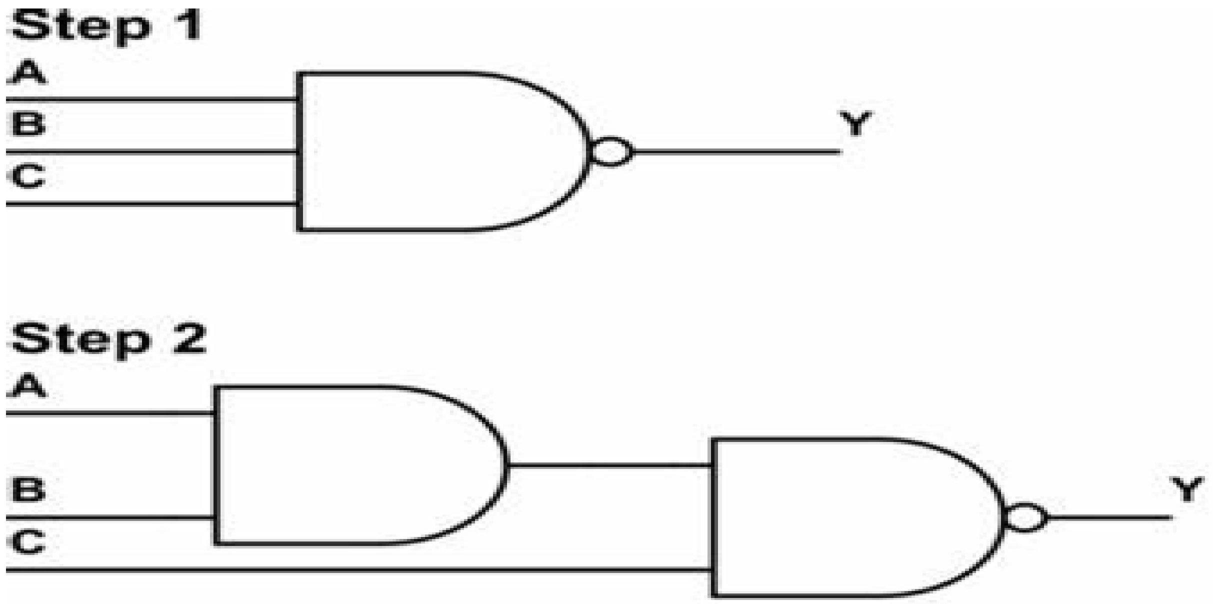
Given the relation s and probabilities' data, it determines the probability of given gate in the relation.

GateDeter() can only calculate probability for a given two output gate. What if we have multiple logic gates as inputs in the relation line?

As in :

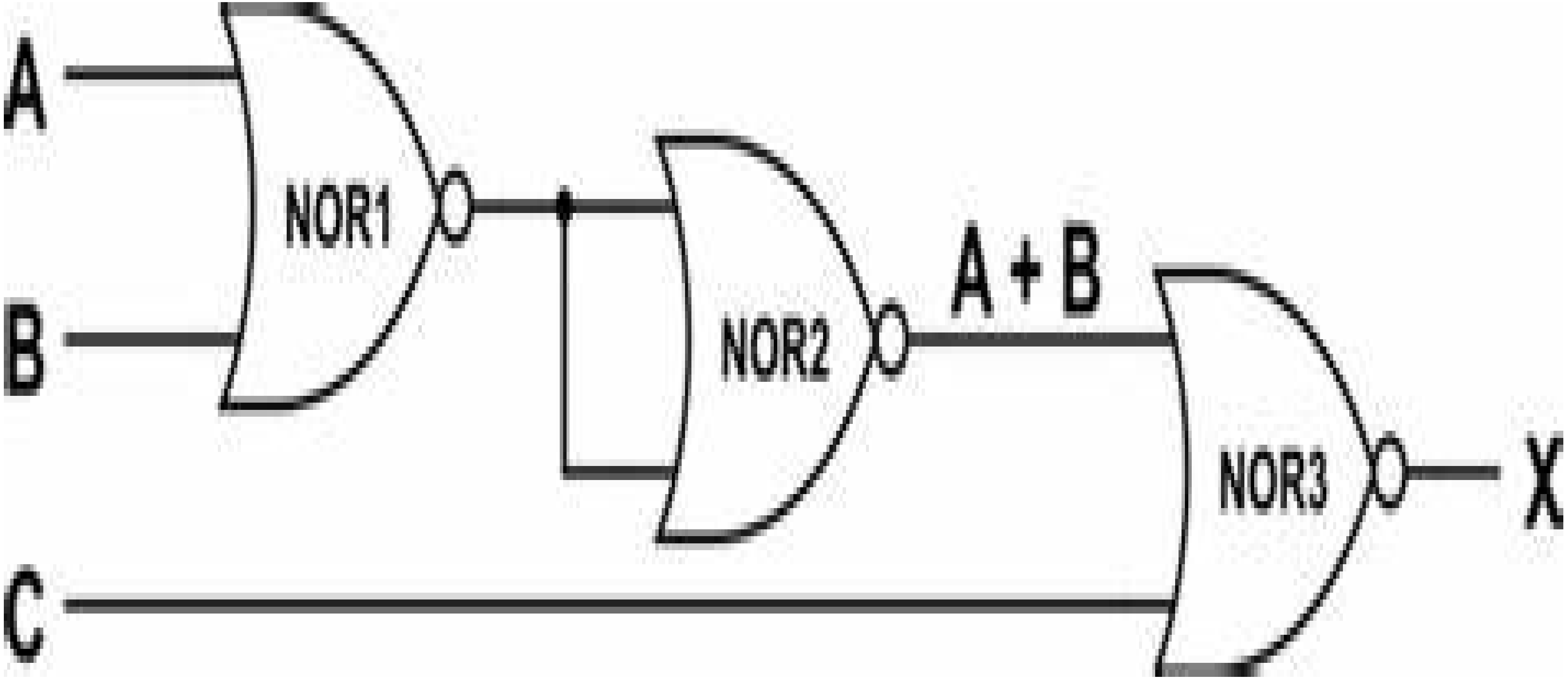
N23 = NAND(N67, N88, N55, N21, N33)

Logic for multiple input logic gates :



Such logics can be implemented for several input logic gates and probabilities can be determined.

Another Example :



Formula for calculating Probabilities of each gate

Let the probability of one input be p1 and the other be p2;

- For **AND** gate:-
  P(o)= p1 * p2;
- For **NAND** gate:-
  P(o)= 1-(p1*p2);
- For **OR** gate:-
  P(o)= (1-p1)*(1-p2);
- For **NOR** gate:-
  P(o)= 1-(1-p1)*(1-p2);
- For **XOR** gate:-
  P(o)=(1-p1)*p2+p1*(1-p2);
- For **XNOR** gate:-
  P(o)=1-(1-p1)*p2+p1*(1-p2);

Conclusion

It includes extension of this project.
For large scale computation, computational efficiency becomes a key concern.

1) **Scalability and Performance Optimization:-**

- **Optimizing Algorithms:** Develop more efficient topological sorting and probability computation algorithms to handle large-scale circuits with thousands of gates.
- **Parallel Processing:** Implement parallel or distributed computing techniques to speed up the analysis. This can be achieved by distributing computations across multiple processors or cloud-based platforms.
- **Memory Management:** Explore techniques to optimize memory usage for large circuits, such as sparse data structures or memory-mapped files.

1) **Integration with Circuit Design Tools:-**To bring the project into a larger ecosystem, consider integrating it with existing electronic design automation (EDA) tools, allowing engineers to analyze circuit designs within their familiar environments: