Dialog semiconductors

Solutions for Dialog tasks

Python program for Verilog equivalent netlist generation using NAND and NOR logic gates

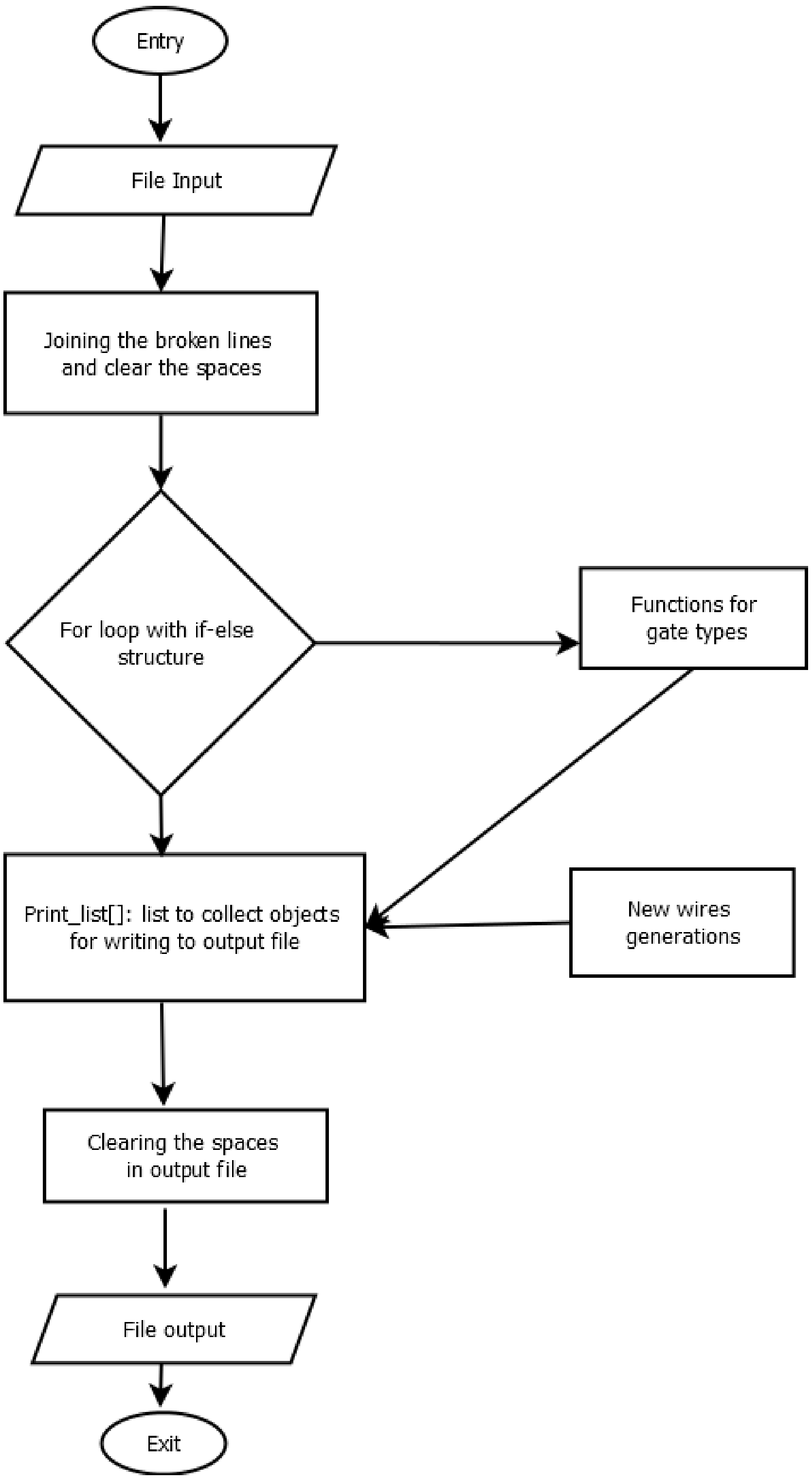
Venkata naga ravikiran bulusu

6/21/2015

VENKATA NAGA RAVIKIRAN BULUSU

As per the given task I developed the python code for equivalent Verilog netlist generation using NAND and NOR logic gates. Below flowchart is the basic idea for implementation and design the algorithm in python

Flowchart:


Entry
File Input
Joining the broken lines and clear the spaces
For loop with if-else structure
Functions for gate types
Print_list[]: list to collect objects for writing to output file
New wires generations
Clearing the spaces in output file
File output
Exit

Usage of main.py code:

Input the file name in the lines 10 and 13 as shown below

```
num_lines = sum (1 for item in open ('block32 .v'))          # Input the filename

with open("block32 .v", "r") as f:                           # Input the filename
```

DEVELOPMENT OF PROGRAM:

Based on the flowchart the program is divided into following blocks and generated the output file.

1. Block for joining lines in the input file

a. In real time the lines in the Verilog netlist are filled with spaces and breaks in the lines. This block is developed to join the broken lines and remove the spaces for processing the file in python.

b. When the Input file is given the program generates a new file after clearing the spaces and broken lines and this file is used for processing in the next steps of the program.

c. Example : Input file: '**block32.v**' and program generated Input : '**ingenerated.v**'

d. '**ingenerated.v**' file is used in the program to generate the output file '**block32_equivalent.v'**.

e. The input file '**block32.v'** is not manipulated /changed for file processing, so the input file is preserved.

2. Block for finding the start and end of wires used

a. Using string operations in python the program will append the lines **starting with 'wires' or 'n'** and stored them in a list.

b. By string strip, split, join operations on the above stored lines the **maximum (zmax)** and **minimum numbers (zmin)** in the wires from the given input file '**ingenerated.v**' is collected.

c. The maximum number (zmax) +1 is set for new wire generation ('r') in the program.

d. Example: if 'zmax' is 495, 'r' is set to 496.

3. Block for main functionality

As per the given netlist there are 22 gate types which represent 22 different patterns of data. Analysed the patterns and designed the algorithm which is generic and flexible to add new gates in the future without using pyparsing and verilogparse.py methodologies. Based on line-to-line execution of python program the algorithm is designed and implemented.

ALGORITHM DESIGN AND IMPLEMENTATION:

a. At first the Verilog gate instantiation is studied and realised the logic gates using NAND and NOR for the given gate types (refer glossary for realizations).

b. Implemented the gate types in the program using a 'for' loop with 'if-else' structure and functions to call each gate from the input file '**ingenerated.v**' as shown below.

c. Python code example:

```
def NAND3XL():
        print_list.append()       #List for writing to output file
       return;
def NAND2X1():
        print_list.append()       # List for writing to output file
         return;
for line in f:                       # for loop for gate types in the input file
      if line.find("NAND3XL")> -1:
       line = line.replace(" ","")
      input_list=[]
      NAND3XL()                    # Function call for NAND3XL
       r+=2                        #Counter for adding new wires
      elif line.find("NAND2X1")> -1:
       NAND2X1()                    #Function call for NAND2X1
```

d. As shown in the example the 'for' loop is developed for 22 gate types and is repeated for all gates in the input file.
e. As shown in the below example Verilog gate instantiation for OAI22XL.

```
OAI22XL U289 ( .A0(b_var[10]), .A1(var_code[0]), .B0(n245), .B1(var_code[1]), .Y(n246) );
```

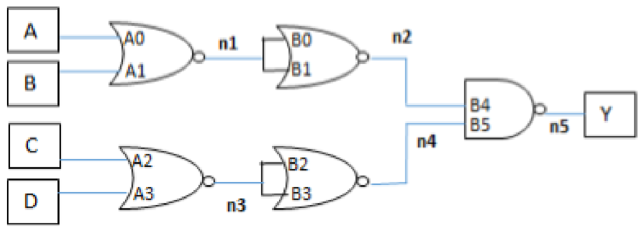
f. The gate module number : U289, pins : A0, A1, B0, B1, wires/Inputs : b_var[10], var_code[0], n245, var_code[1], n246, output : Y.
g. Appended the line to an input_list[] from the given file and used string operations to remove the spaces between each element.

```
OAI22XLU289(.A0(b_var[10]), .A1(var_code[0]), .B0(n245), .B1(var_code[1]), .Y(n246) );
```

h. Using string operations the 'U', 'A0', 'A1', 'B0', 'B1', 'Y' position values are known in a line to collect the gate name, module name, wires and stored as strings in input_list[].
i. Python code example:

```
r0 = (line.find('U'))
input_list.append(line[0:r0])          # To store the gate name
r1 = (line.find('.A0'))
input_list.append(line[r0:r1-1])       # To store the module name
r2 = (line.find('.A1'))
input_list.append(line[r1+3:r2])      # To store Input b_var[10]
r3 = (line.find('.B0'))
input_list.append(line[r2+3:r3])      # To store Input var_code[0]
r4 = (line.find('.B1'))
input_list.append(line[r3+3:r4])     # To store wire n245
r5 = (line.find('.Y'))
input_list.append(line[r4+3:r5])     #To store Input var_code[1]
r6 = (line.find(';'))
input_list.append(line[r5+2:r6-1]) #To store wire n246
```

j. Verilog equivalent netlist using NAND and NOR logic gates for OAI22X1



```
NOR2X1 U1 (.A (A), .B (B), .Y (n1));
NOR2X1 U2 (.A (n1), .B (n1), .Y (n2));
NOR2X1 U3 (.A(C), .B (D), .Y (n3));
NOR2X1 U4 (.A (n3), .B (n3), .Y (n4));
NAND2X1 U5 (.A (n2), .B (n4), .Y (n5));
```

k. Python code example:
Appended each line to print_list[] and by file operations the program writes them to output file: 'block32_equivalent.v'.

```
print_list.append('NOR2X1'+input_list[1].rstrip()+'00'+' ( .A'+input_list[2].rstrip()+' .B'+input_list[3].rstrip()+' .Y('+"".join([str('n')+str(r)])+') );')
```

l. As shown in the above implementation the Verilog gate instantiation is prepared for 22 gate types and program writes them to output file: 'block32_equivalent.v'.
m. Counter 'r' will be incremented for each gate execution from the 'for' loop.
n. Other lines in the file are appended directly to print_list[] and program writes them to output file : 'block32_equivalent.v'.
o. Example: "module", "endmodule","output", "Input".

4. Block for adding new wires generated
   a. **A new list of wires is prepared by program from minimum number (zmin) to last incremented new wire value ('r').**
   b. **Wrapped the text to a fixed width of '130' using textwrap.fill () imported from textwrap module and writes them to output file: 'block32_equivalent.v'.**
5. Output file generation and dashboard
   a. After the output file: '**block32_equivalent.v**' is prepared the following script is used to clear the spaces in the output file.

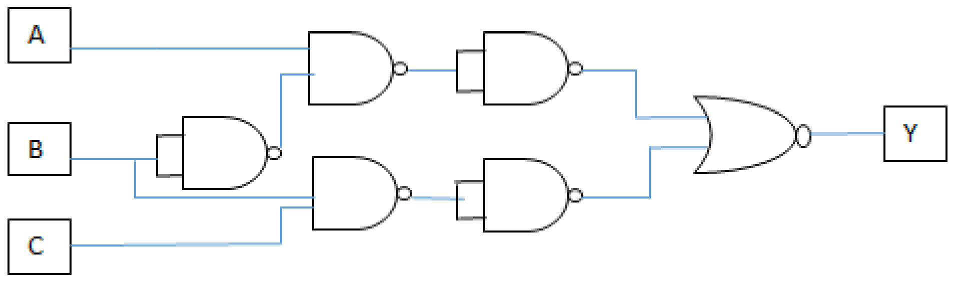
```
clean_lines = []
with open("block32_equivalent.v", "r") as file_output:
    lines = file_output.readlines()
    clean_lines = [l.strip() for l in lines if l.strip()]

with open("block32_equivalent.v", "w") as file_output:
    file_output.writelines('\n'.join(clean_lines))
```

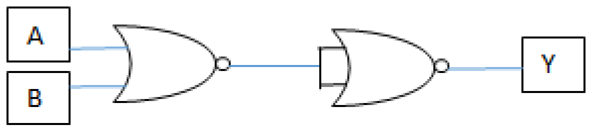
b. Dashboard is printed to the user in the python shell with following details: python version, total time conversion, total source lines, total number of new wires added and average lines/sec for file conversion.

Glossary for logic gates realization using NAND and NOR gates:

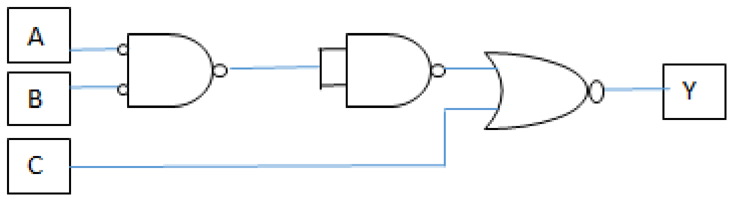
1. **MXI2 realization with 5 NAND and 1 NOR gates**



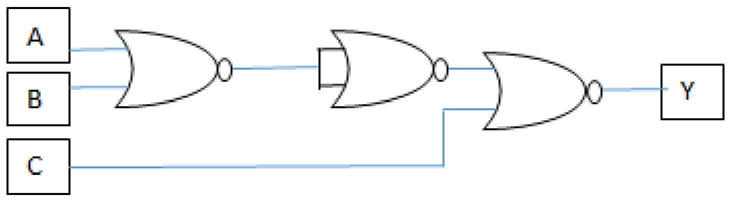
2. **OR2X1 realization with 2 NOR gates**



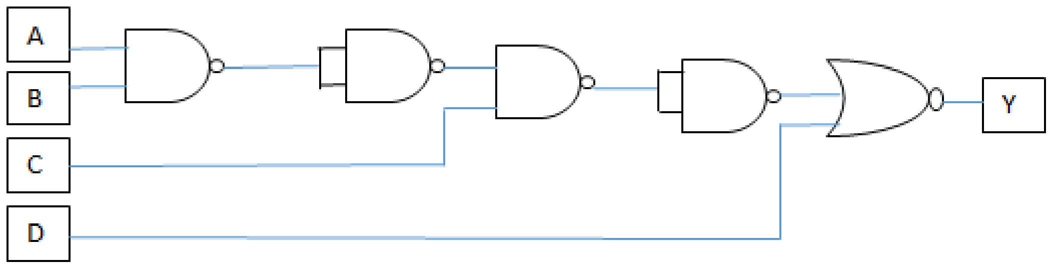
3. **AOI2BB1X1 realization with 4 NAND and 1 NOR gates**



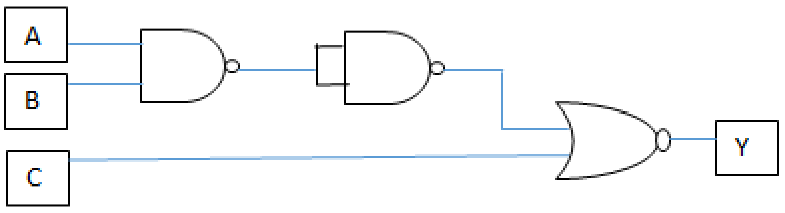
4. **NOR3X1 realization with 3 NOR gates**



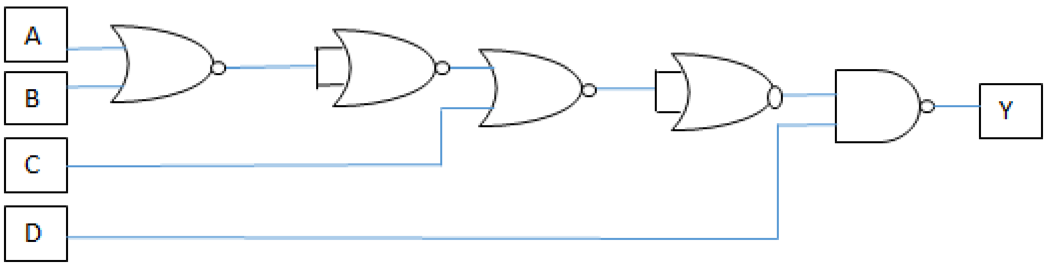
5. **AOI3X1 realization with 4 NAND and 1 NOR gates**



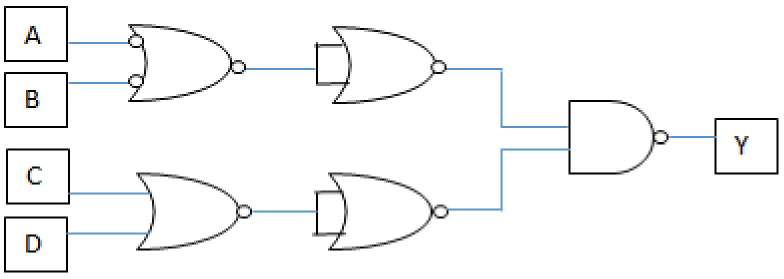
6. **AOI21X1 realization with 2 NAND and 1 Nor gates**



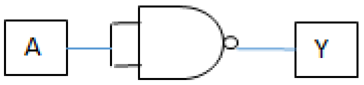
7. **OAI31X1 realization with 4 NOR and 1 NAND gates**



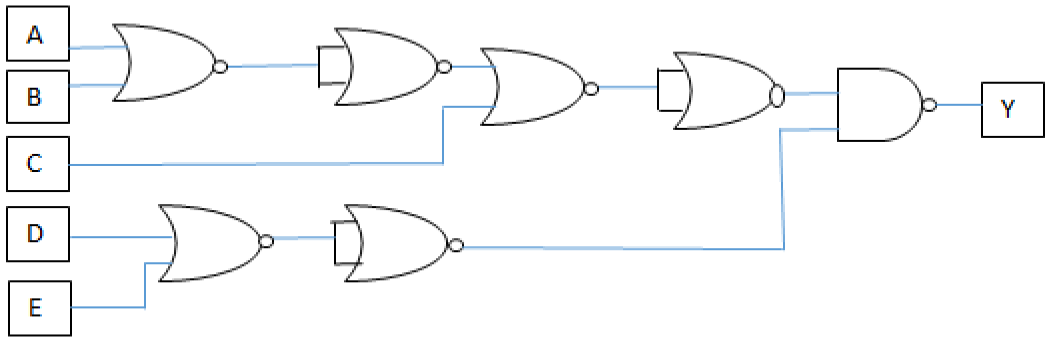
8. **OAI2BB2X1 realization with 6 NOR and 1 NAND gates**



9. **INVX1 realization with 1 NAND gate**

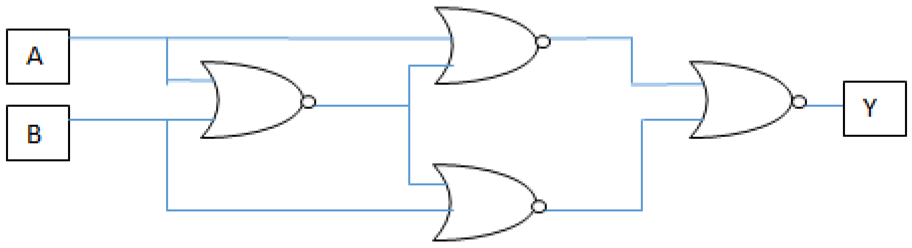


10. **OAI32X1 realization with 6 NOR and 1 NAND gates**

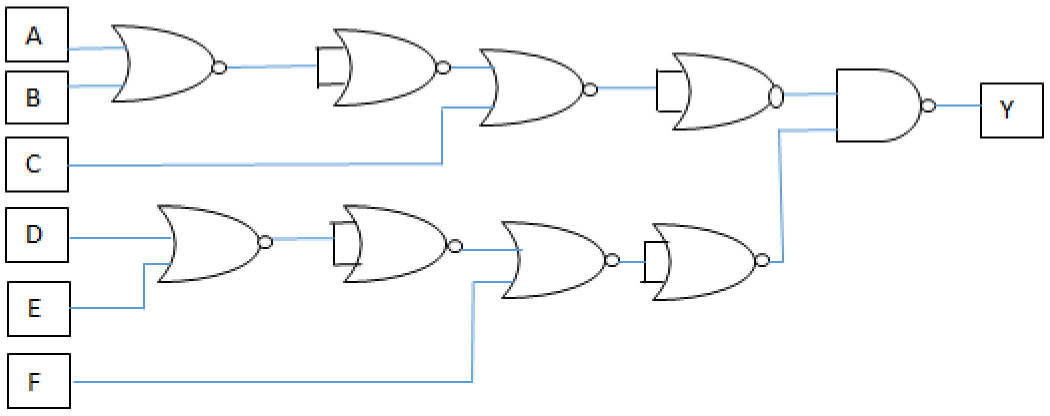


11. **AOI32X1 realization with 6 NAND and 1 NOR gates by replacing them in above figure.**

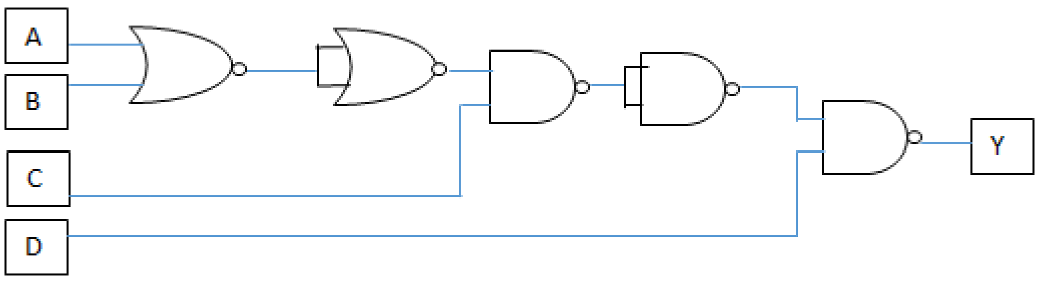
12. **XNOR2X1 realization with 4 NOR gates**



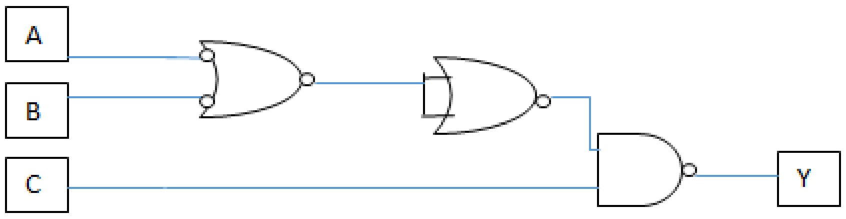
13. **OA133X1 realization with 8 NOR and 1 NAND gates**



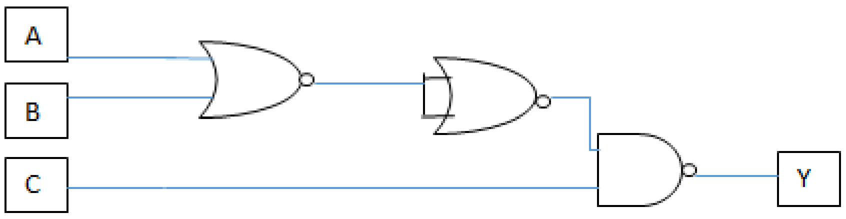
14. **OAI211X1 realization with 4 NOR and 1 NAND gates**



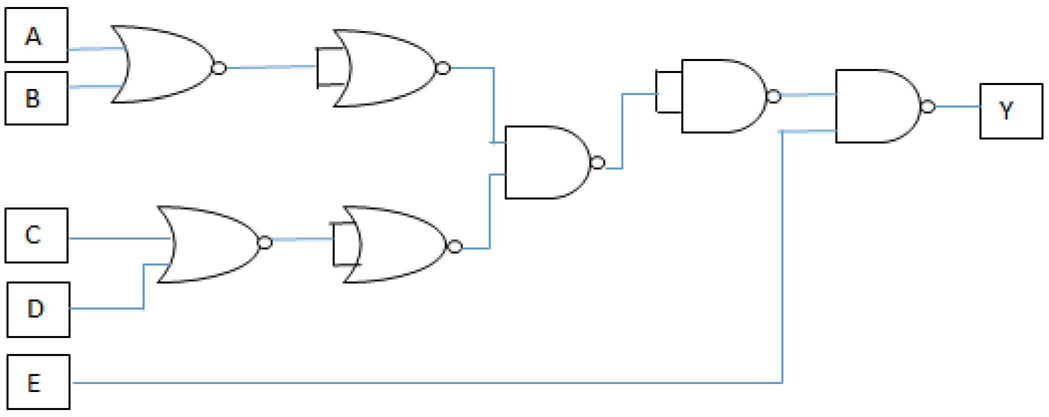
15. **OAI2BB1X1 realization with 4 NOR and 1 NAND gates**



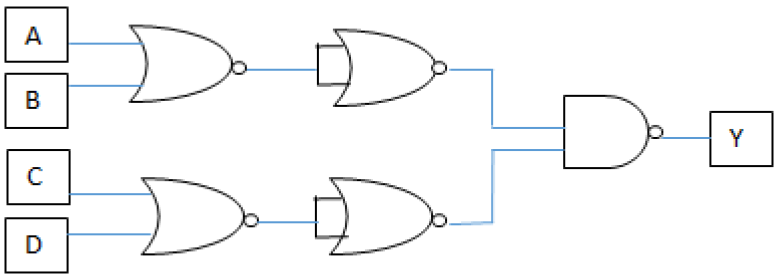
16. **OAI21X1 realization with 2 NOR and 1 NAND gates**



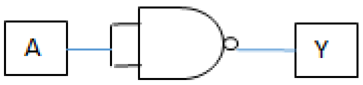
17. **OAI221X1 realization with 4 NOR and 3 NAND gates**



18. OAI22X1 realization with 4 NOR and 1 NAND gates



19. CLKINVX1 realization with 1 NAND gate



20. NAND3X1 realization with 3 NAND gates