PROJECT: CMOS CIRCUIT SPICE GENERATOR

Prof. Cherif Salama

Ali Moussa – 900160311

Marwan Eid – 900171885

Mohammed Abuelwafa – 900172603

The American University in Cairo
School of Sciences and Engineering
Department of Computer Science and Engineering

CSCE 3301 – Fundamental Microelectronics

Spring 2020

Table of Contents

1. Project Description

In this project we were asked to implement a program that generates a SPICE netlist that will describe the CMOS circuit that will be obtained from a Boolean function provided by the user. The SPICE deck and the aim of this project is to generate the circuit components giving each component a name. Each component is described by a single statement specifying its label, the labels of the nodes it is connected to, and any component specific parameters needed and how they are connected.
The expression will be provided as an infix expression and the first thing our code does is change that format into postfix using two functions “int precedence” which takes as input the characters and compares them to the known variables used in the Boolean expression and returns the number we choose to associate with either ‘and’, ‘or’ and ‘not’ this was bonus number 4 “Allowing parentheses to appear in input expressions to override original operator precedence. Implementing this feature would make the following input valid: y=((a|b)&(c|d)’)’”
We were also able to do the fifth bonus in the document “Allowing multiple semicolon-separated output symbols in the input expressions. Implementing this feature would make the following input valid: y=a&b|c’; x=y’|a|c’”. Which takes the input and we cut it at every semicolon into expressions, then taking this expression after the equal sign and this will be given to the function “infix_to_postfix” and this function iterates letter by letter and based on every input either it be ‘AND’ , ‘OR’ , ‘NOT’. This will call the function that relates to it.
We have three functions “process_not” , “process_or” , “process_and”. These functions are templates that generate the PUN and PDN for each gate. For example, if “and” is needed the function “process_and” is called and it will generate “NAND” gates and an inverter at the end to turn it into an “AND” CMOS circuit. The same happens for ‘OR’ where a ‘NOR’ is produced from the function ‘process_or’ and an inverter at the end. The code will produce the needed CMOS circuit for each input and when the expression is done we reinitialize the global counters to rerun the program for another expression until all expressions are dealt with.

2. Implementation

2.1 Data Structures

- **Strings:**
  Strings are used to form the entries of a netlist of a specific gate and a string contains the final netlist as well.
- **Vectors:**
  Vectors are mainly used as containers of elements that represent netlist entries to a specific gates and arguments for gates.
- **Stacks:**

Stacks are mainly used in the conversion from infix expression to the postfix expression and in the parsing of the expressions to get the arguments of the operators.

2.2 Algorithms

2.1.1 int precedence(char a)

The input for this function is a character which is the operators of the Boolean expression. Outputs for the algorithm are either (-1,0,1,2), each number is associated with an operator; if in the expression appears the "\" 'NOT' operator then the number "2" will be returned. If "&" 'AND' then the number "1" will be returned, the number "0" will be returned if "|" 'OR" operator and finally "-1" is if there is something wrong with the input. This algorithm is called within another algorithm "string infix_to_postfix" which is used inside to check if the input is actually a valid input and to be distributed accordingly.

2.1.2 bool isOperator(char a)

This algorithm is a simple Boolean algorithm which determines if the input is in fact an operator and it takes the character 'a' as input and output is either true or false.

2.1.3 string infix_to_postfix(string infix)

Here the function takes as input the string given by the user. The input is given as an infix expression and this algorithm is needed to convert it into postfix to ease the processing. Starting with initializing a new string called "postfix" to an empty string, then after getting the length of the original infix expression to be used in a while loop and go over each character inside this string. It then sends each character as input to function #2 "isOperator" to check if the input is valid and returns if it is true or false, if it is true it uses the precedence function to sort the expression into a postfix expression. The while loop goes for the length of the infix expression and adds the variables in the order of precedence to be a postfix expression. This algorithm is very accurate due to the two previous algorithms which check if the input is valid throughout the while loop.

2.1.4 string Process_not(string x, string& output_str = not_list)

This algorithm takes as input a string which has the "\" 'NOT' operator and processes the netlist of the "NOT GATE" which is given in the project description. The output is the netlist which is in the order <Mname> <drain> <gate> <source> <body> <type> in the case of the not gate it will be for example " M0 out a vdd vdd PMOS" this is in the case of a PMOS transistor that is connected by its drain to an NMOS transistor.

2.1.5 string process_or(vector<string>elements, bool invert)
string process_and(vector<string>elements, bool invert)

This algorithm takes as input a vector string which has the elements associated with an "OR GATE" and a bool variable to determine if it's a "NOR" gate or not simply it will invert in the end if it is a "NOR" and will dismiss this case if it is a normal "OR GATE" A for loop will go through each element and output the netlist in the order "Mname drain gate source body type" for example "M0 or0 a 0 0 NMOS" this is an NMOS connected from the drain to the PUN of the circuit and its gate is connected to input 'a' and finally the body and the source are connected to the ground and at the end the name of the transistor 'NMOS' in this example.
The exact same is for the "AND" where if we need a "NAND" the Boolean expression will give true and we will invert in the end else it will output in the same way except it will be for an AND.

2.1.6 string find_replace(string func, string sub, string str)

This algorithm is a string algorithm that will modify our netlist and is called in the function "produce_final_output" to reorganize our netlists.

2.1.7 string produce_final_output(string postfix , string output_label)

This is the final algorithm to produce the desired netlist as a whole it will take as input the postfix expression produced by function #3 "infix_to_postfix" creating the string netlist and in a for loop takes each character in the postfix expression and push it in the stack if it isn't one of our operators (\ , | , &) else it will enter a while loop with the operator and add it to the needed netlist to finally be output as a full netlist. This function can produce error messages if the postfix expression has any undesired inputs. Finally, it calls function #6 "find_replace" to remodify the netlist into the desired output and returns the full netlist.

3. User Guide

3.1 Compilation and Running

3.1.1 Ubuntu

1. Make sure GCC compiler is installed by running the following two commands in the terminal:
   i. sudo apt update
   ii. sudo apt install build-essential
2. Either using Visual Studio Code or using the terminal, navigate to the location containing the source code "main.cpp".
3. Run the following command to generate an executable file "main".
   i. g++ main.cpp -o main
4. Run the following command to run the executable file generated:
   i. ./main

3.1.2 Windows

1. Make sure GCC compiler is installed; details steps for installation could be followed from https://www.youtube.com/watch?v=hSAY50OL_J4.
2. Using Visual Studio Code, navigate to the location containing the source code "main.cpp".
3. Run the following command to generate an executable file "main".
    i. g++ main.cpp -o main
4. Run the following command to run the executable file generated:
    i. ./main

See Figure 1 for an example compiling and running the program using the previous method.

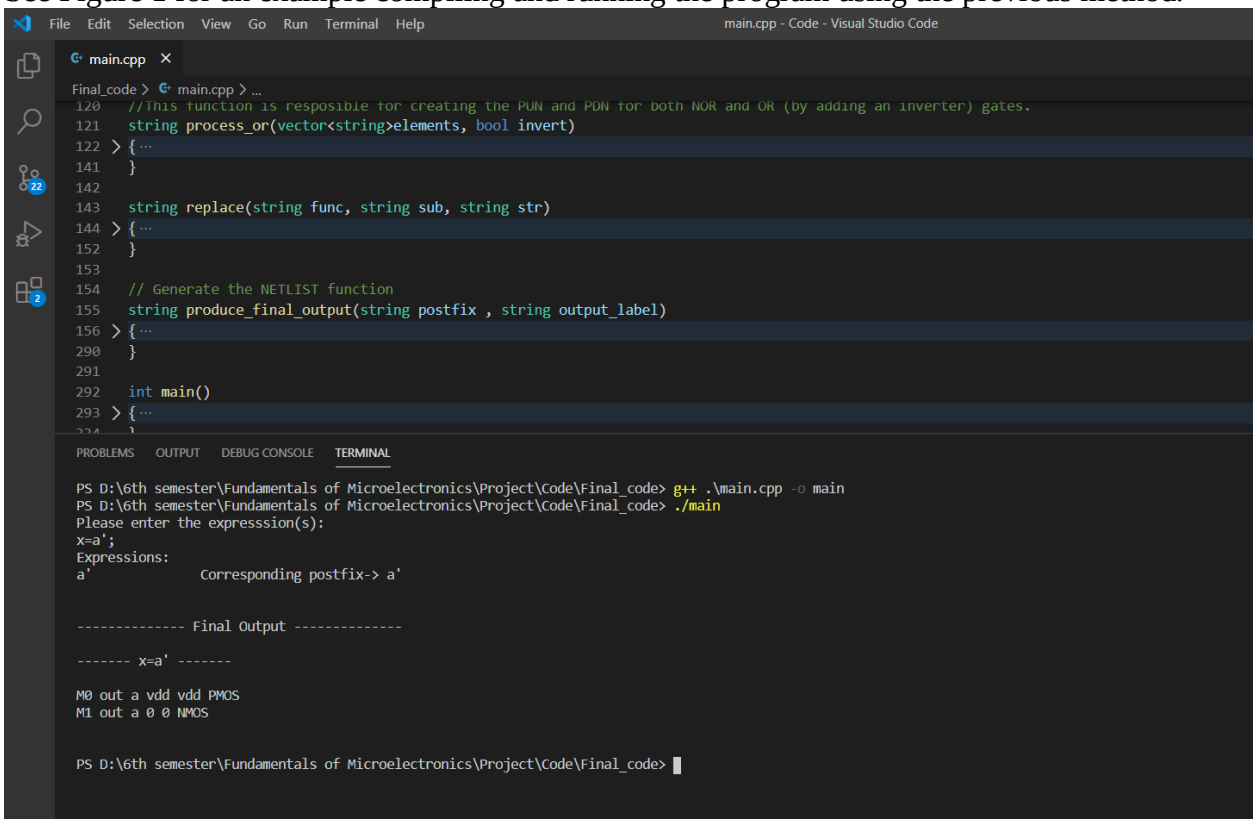

Figure 1

Alternatively,

Open Visual Studio, create a new c++ project, add an existing source file and select the source code file "main.cpp", and run the program using F5 or the Local Windows Debugger button.

3.1.3 MacOS

Either compile and run using xCode, or run the following two commands in the terminal:

- g++ main.cpp
- ./a.out

3.2 Usage

- The user is requested to enter the Boolean expression for which the data statement part aforementioned is needed to be generated.
- The user input should follow the following requirements:
    - Each expression should not have any spaces and should end with a semicolon.
    - Multiple expressions are allowed and should be separated by a space.
    - The Boolean expression entered should be valid.
- The user input for $n \geq 1$ expressions should be as follows:

    <output_name_1>=<Boolean_expression_1>;
    <output_name_2>=<Boolean_expression_2>; …
    <output_name_n>=<Boolean_expression_n>;
    , where for $1 \leq i \leq n$ output_name_i is the i^{th} name of the output variable, such as x or y, and Boolean_expression_i is the Boolean expression corresponding to the i^{th} output, such as a&b or c|d'.4.

See Figure 2.a and 2.b for an example of a valid user input containing 3 Boolean expressions; note that both Figures show one output of the program, but is split into two pictures, because of their size.

```
PS D:\6th semester\Fundamentals of Microelectronics\Project\Code\Final_code> ./main
Please enter the expresssion(s):
x=(a&b)|(c'&(d|e')); y=f'&g'; z=h&(i|(j&m'));
Expressions:
(a&b)|(c'&(d|e'))               Corresponding postfix-> ab&c'de'|&|
f'&g'           Corresponding postfix-> f'g'&
h&(i|(j&m'))            Corresponding postfix-> hijm'&|&


-------------- Final Output --------------

------- x=(a&b)|(c'&(d|e')) -------

M0 AND0 a vdd vdd  PMOS
M1 AND0 a M3NODE M3NODE NMOS
M2 AND0 b vdd vdd  PMOS
M3 M3NODE b 0 0  NMOS
M4 NOT0 AND0 vdd vdd PMOS
M5 NOT0 AND0 0 0 NMOS

M6 NOT1 c vdd vdd PMOS
M7 NOT1 c 0 0 NMOS

M8 NOT2 e vdd vdd PMOS
M9 NOT2 e 0 0 NMOS

M10 or0 d 0 0 NMOS
M11 M13NODE d vdd vdd PMOS
M12 or0 NOT2 0 0 NMOS
M13 or0 NOT2 M13NODE M13NODE PMOS
M14 NOT3 OR0 vdd vdd PMOS
M15 NOT3 OR0 0 0 NMOS

M16 AND1 NOT1 vdd vdd  PMOS
M17 AND1 NOT1 M19NODE M19NODE NMOS
M18 AND1 NOT3 vdd vdd  PMOS
M19 M19NODE NOT3 0 0  NMOS
M20 NOT4 AND1 vdd vdd PMOS
M21 NOT4 AND1 0 0 NMOS

M22 or1 NOT0 0 0 NMOS
M23 M25NODE NOT0 vdd vdd PMOS
M24 or1 NOT4 0 0 NMOS
M25 or1 NOT4 M25NODE M25NODE PMOS
M26 out OR1 vdd vdd PMOS
M27 out OR1 0 0 NMOS
```

Figure 2.a

```
------- y=f'&g' -------

M0 NOT0 f vdd vdd PMOS
M1 NOT0 f 0 0 NMOS

M2 NOT1 g vdd vdd PMOS
M3 NOT1 g 0 0 NMOS

M4 AND0 NOT0 vdd vdd  PMOS
M5 AND0 NOT0 M7NODE M7NODE NMOS
M6 AND0 NOT1 vdd vdd  PMOS
M7 M7NODE NOT1 0 0  NMOS
M8 out AND0 vdd vdd PMOS
M9 out AND0 0 0 NMOS


------- z=h&(i|(j&m')) -------

M0 NOT0 m vdd vdd PMOS
M1 NOT0 m 0 0 NMOS

M2 AND0 j vdd vdd  PMOS
M3 AND0 j M5NODE M5NODE NMOS
M4 AND0 NOT0 vdd vdd  PMOS
M5 M5NODE NOT0 0 0  NMOS
M6 NOT1 AND0 vdd vdd PMOS
M7 NOT1 AND0 0 0 NMOS

M8 or0 i 0 0 NMOS
M9 M11NODE i vdd vdd PMOS
M10 or0 NOT1 0 0 NMOS
M11 or0 NOT1 M11NODE M11NODE PMOS
M12 NOT2 OR0 vdd vdd PMOS
M13 NOT2 OR0 0 0 NMOS

M14 AND1 h vdd vdd  PMOS
M15 AND1 h M17NODE M17NODE NMOS
M16 AND1 NOT2 vdd vdd  PMOS
M17 M17NODE NOT2 0 0  NMOS
M18 out AND1 vdd vdd PMOS
M19 out AND1 0 0 NMOS
```

Figure 2.b

See Figure 3 for 2 examples of invalid user input.

```
PS D:\6th semester\Fundamentals of Microelectronics\Project\Code\Final_code> ./main
Please enter the expresssion(s):
x=a
Expressions:


-------------- Final Output --------------

PS D:\6th semester\Fundamentals of Microelectronics\Project\Code\Final_code> ./main
Please enter the expresssion(s):
x=a&b;y=c|d;
Expressions:
a&b             Corresponding postfix-> ab&
|d              Corresponding postfix-> d|


-------------- Final Output --------------

------- x=a&b -------

M0 AND0 a vdd vdd  PMOS
M1 AND0 a M3NODE M3NODE NMOS
M2 AND0 b vdd vdd  PMOS
M3 M3NODE b 0 0  NMOS
M4 out AND0 vdd vdd PMOS
M5 out AND0 0 0 NMOS


------- =c|d -------

Error in the counter of successor operands, post expression is probably wrong
M0 or0 d 0 0 NMOS
M1 M3NODE d vdd vdd PMOS
M2 out OR0 vdd vdd PMOS
M3 out OR0 0 0 NMOS
```

Figure 3

4. Test Cases

Table 1. Test Cases

	Boolean Expression	**Corresponding SPICE Statements <Name> <D> <G> <S> <B> <type>**	**Corresponding Circuit**
1	x=a&b;	M0 AND0 a vdd vdd PMOS M1 AND0 a M3NODE M3NODE NMOS M2 AND0 b vdd vdd PMOS M3 M3NODE b 0 0 NMOS M4 out AND0 vdd vdd PMOS M5 out AND0 0 0 NMOS	See Figure 4.
2	x=a|b;	M0 or0 a 0 0 NMOS M1 M3NODE a vdd vdd PMOS M2 or0 b 0 0 NMOS M3 or0 b M3NODE M3NODE PMOS M4 out OR0 vdd vdd PMOS M5 out OR0 0 0 NMOS	See Figure 5.
3	y=a’;	M0 out a vdd vdd PMOS M1 out a 0 0 NMOS	See Figure 6.
4	y=(a&b)’;	M0 out a vdd vdd PMOS M1 out a M3NODE M3NODE NMOS M2 out b vdd vdd PMOS M3 M3NODE b 0 0 NMOS	See Figure 7.
5	y=(a|b)’;	M0 or0 a 0 0 NMOS M1 M3NODE a vdd vdd PMOS M2 or0 b 0 0 NMOS M3 or0 b M3NODE M3NODE PMOS	See Figure 8.
6	y=f'&g';	M0 NOT0 f vdd vdd PMOS M1 NOT0 f 0 0 NMOS M2 NOT1 g vdd vdd PMOS M3 NOT1 g 0 0 NMOS M4 AND0 NOT0 vdd vdd PMOS M5 AND0 NOT0 M7NODE M7NODE NMOS M6 AND0 NOT1 vdd vdd PMOS M7 M7NODE NOT1 0 0 NMOS M8 out AND0 vdd vdd PMOS M9 out AND0 0 0 NMOS	See Figure 9.

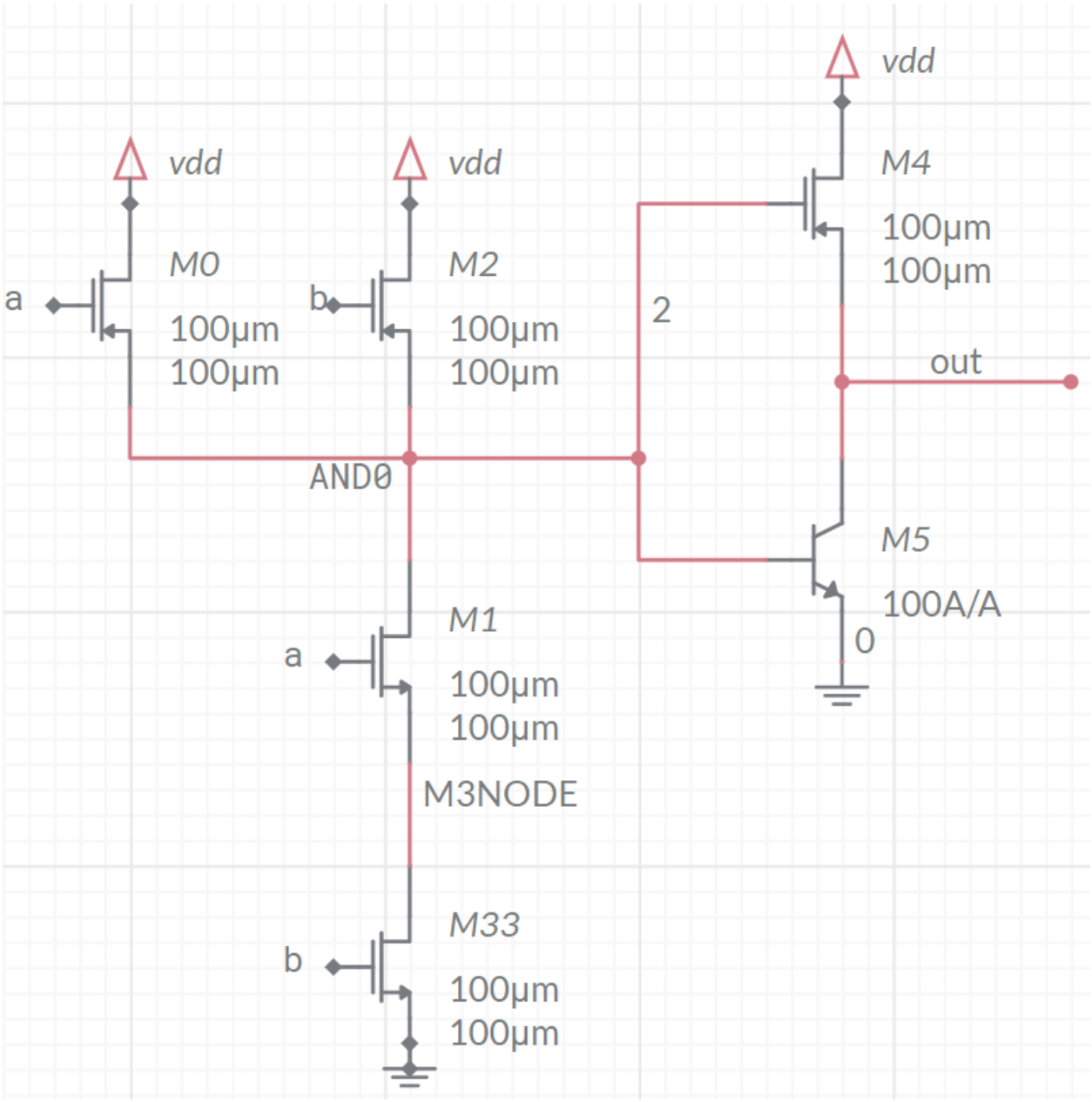
vdd
vdd
vdd
M0
M2
M4
a
b
100µm
100µm
100µm
100µm
100µm
100µm
2
out
AND0
M5
100A/A
0
M1
a
100µm
100µm
M3NODE
M33
b
100µm
100µm

Figure 4

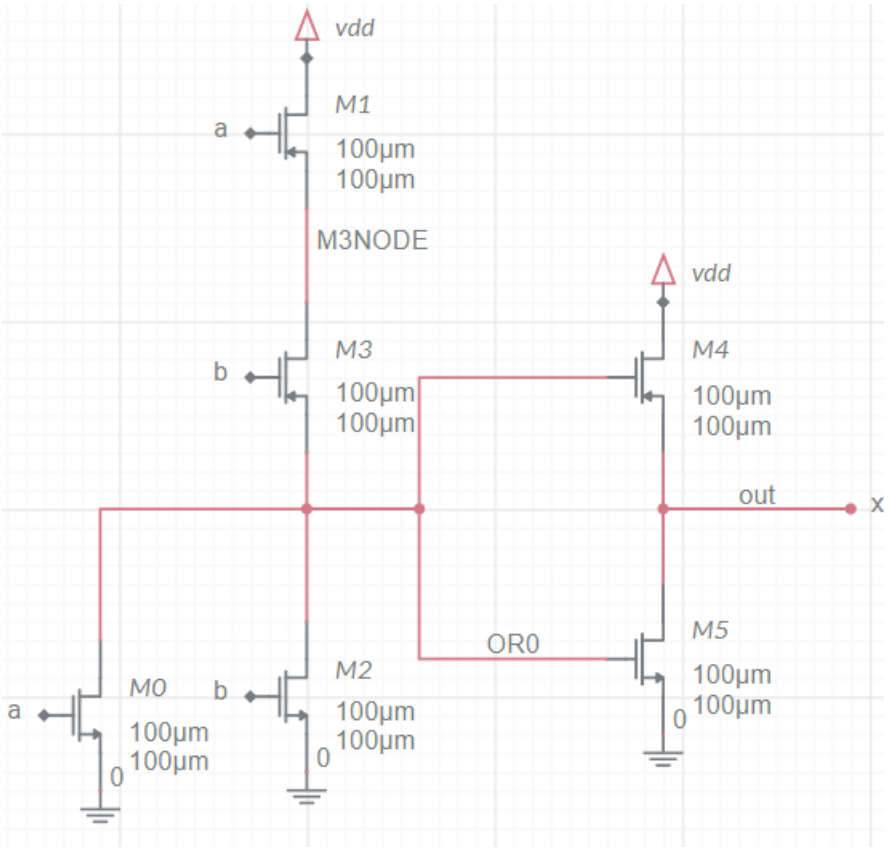
vdd
M1
a
100µm
100µm
M3NODE
vdd
M3
b
100µm
100µm
M4
100µm
100µm
out
x
OR0
M5
100µm
100µm
M0
a
100µm
100µm
0
M2
b
100µm
100µm
0
0

Figure 5

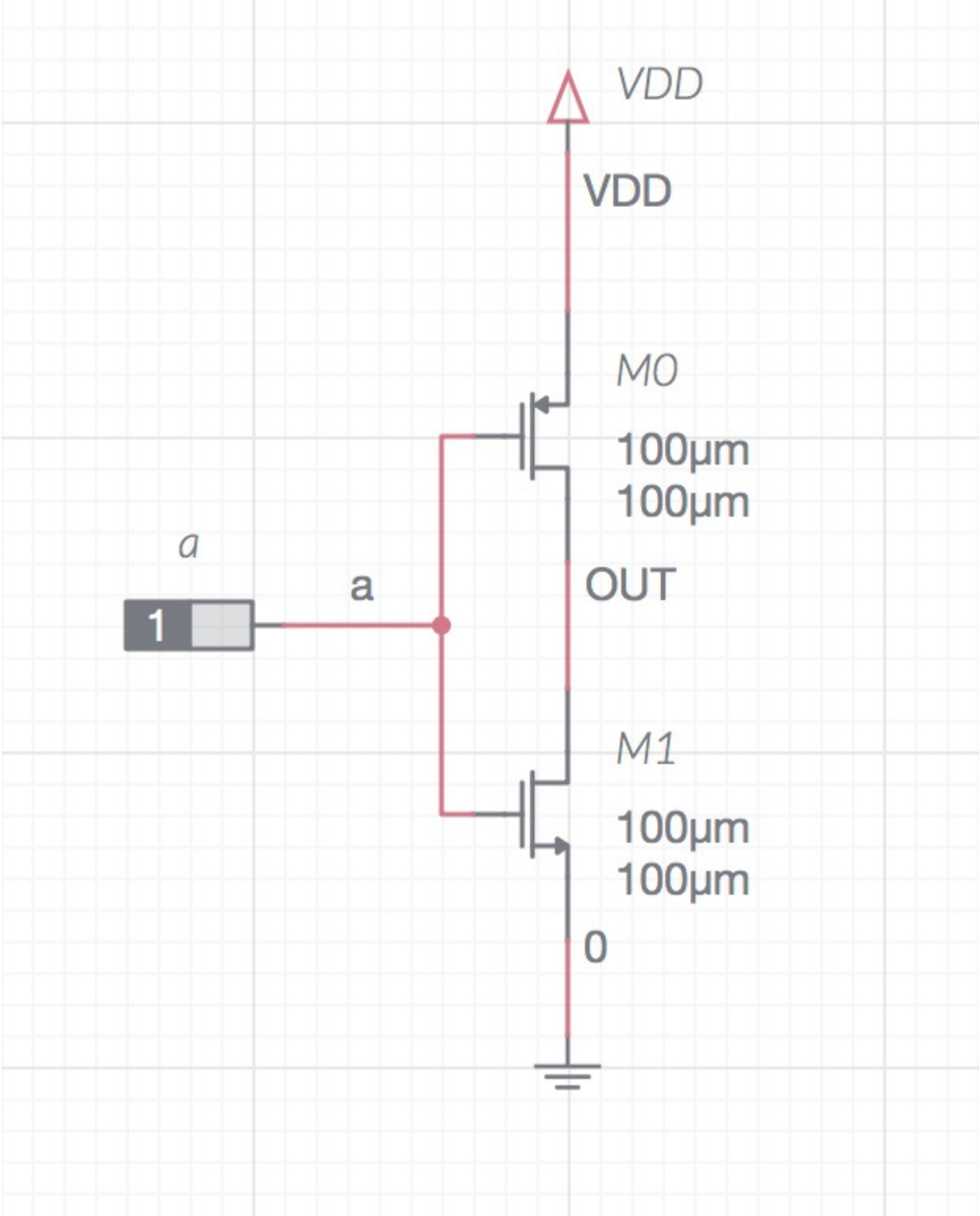

Figure 6

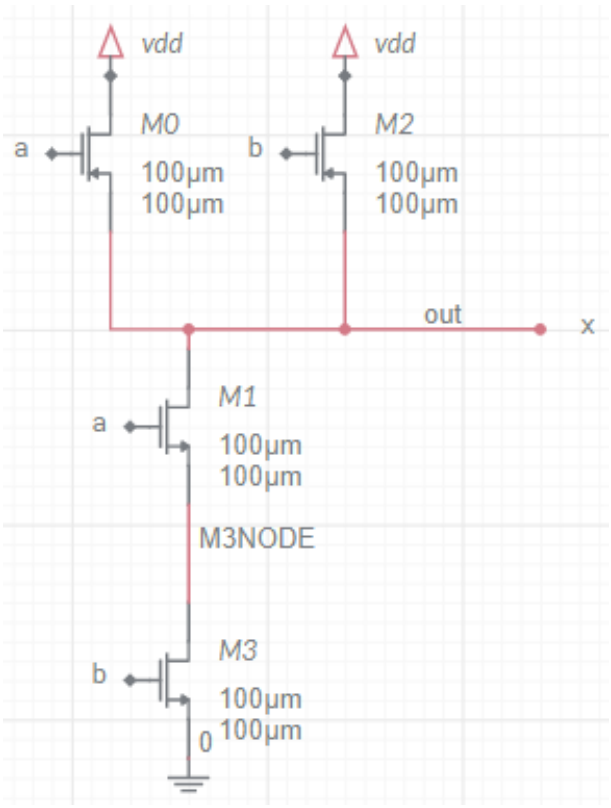
vdd
vdd
M0
M2
a
b
100µm
100µm
100µm
100µm
out
x
M1
a
100µm
100µm
M3NODE
M3
b
100µm
100µm
0

Figure 7

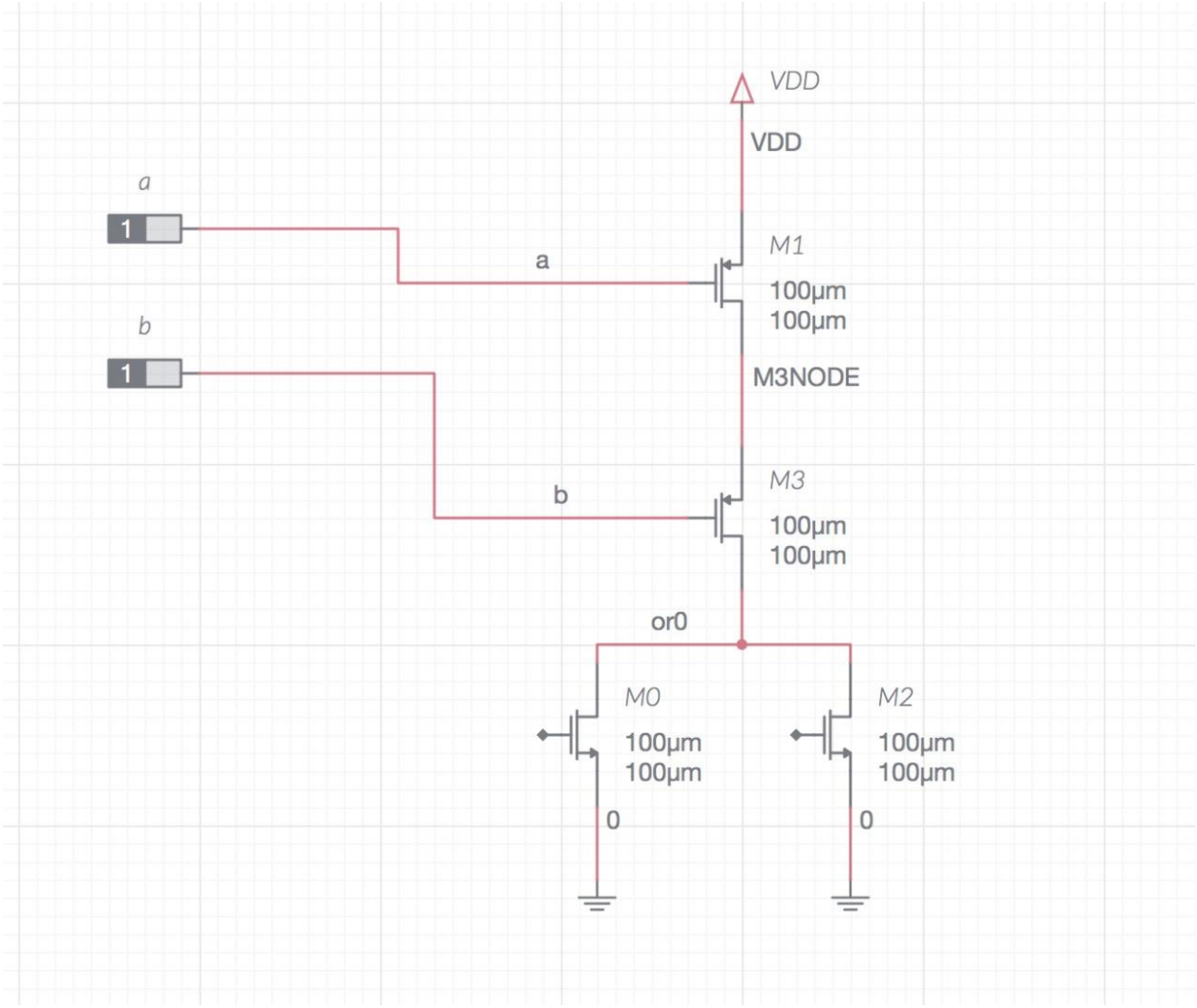
VDD
VDD
a
1
a
M1
100µm
100µm
b
1
M3NODE
b
M3
100µm
100µm
or0
M0
100µm
100µm
0
M2
100µm
100µm
0

Figure 8

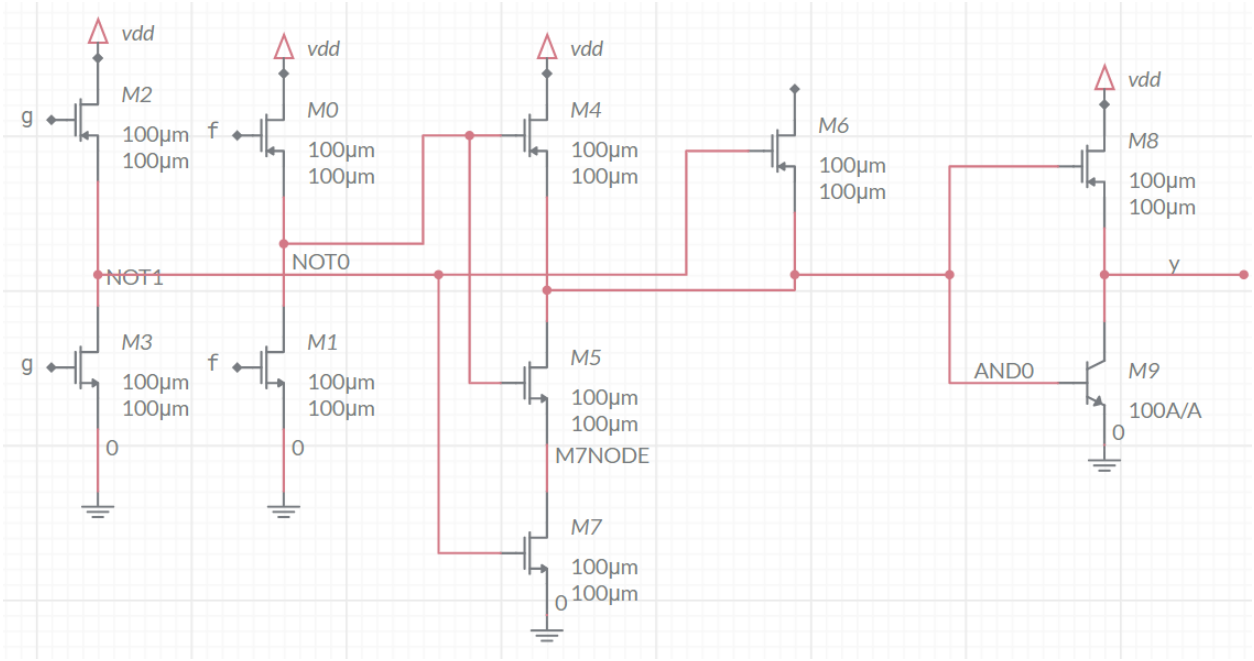

Figure 9

5. Limitations

We realize that our code does not provide us with the most optimized circuit since we are using some sort of templates for the gates (AND, OR, NOT, NAND, and NOR). Hence, our generated netlist has high cost in terms of area and propagation delays. One proposed solution was to attach to the project (Quine McCluskey logic minimizer c++ source code implemented by Mohammed Abuelwafa) in order to get a minimized logic expression from each expression and then insert it to our code, but unfortunately we were short in time in terms of implementation. A comparison between 2 NOR gates is shown Figure 8 and Figure 9. Figure 8 shows an optimized NOR gate with 4 transistors only, while Figure 9 shows a NOR gate that is not optimized, i.e. contain double inversion, and thus has 6 transistor; notice that the total number of transistors needed for the Boolean expression is 10, because, besides the NOR gate requiring 6 transistors, there are 2 input inversions needed, each of which requiring 2 transistors: a total of 10 transistors.

6. Contributions

We split the work of the project and worked on it through zoom meetings and WhatsApp conversations. Such work includes discussion of the problem and approaches to the solution, writing and debugging the code, showing the corresponding circuits for test cases, and writing the report. The aforementioned work was split evenly among us. For instance, for the code, we discussed the algorithms needed for the solution and each one started writing some functions before we integrated the work and debugged it. Other than the easy and other supporting functions, the main function Ali focused on was main(); Marwan was infix_to_postfix(); and Mohammed, was produce_final_output(). For the circuits, each one produced a circuit for two

out of the six test cases mentioned. For the report, each one contributed to interleaving sections; Ali focused on the description and the algorithm; Marwan focused on the user guide and part of the implementation; Mohammed focused on the data structures and of the limitations.

Appendix: Source Code

```
#include<iostream>
#include<string>
#include<vector>
#include<stack>
using namespace std;

// Global variables
int mos_count = 0; // Index used for naming
string not_list = "" , or_list="" , and_list =""; // string for each gate that re
cords its netlist
int not_nodes = 0 , or_nodes = 0 , and_nodes = 0 ; // counters to name the nodes
that represents the output of each gate

// This function associates precedence with operators
int precedence(char a)
{
    if (a == '\'')
        return 2;
    else if (a == '&')
        return 1;
    else if (a == '|')
        return 0;
    else
        return -1;
}

// This function determines whether a character is an operator or not
bool isOperator(char a)
{
    if (a == '&' || a == '|' || a == '\'')
        return true;
    return false;
}

// This function converts the infix expression to postfix expression to ease its
processing
string infix_to_postfix(string infix)
{
```

```
    string postfix = "";
    int i = 0, n = infix.length();
    char e='%', c='-', current;
    stack<char> conversion;
    while (i < n)
    {
        current = infix[i];
        if (!isOperator(current) && current != '(' && current != ')')
            postfix += current;
        else if (current == '(')
            conversion.push(current);
        else if (current == ')')
        {
            e = conversion.top();
            while ((e != '(') && (!conversion.empty()))
            {
                e = conversion.top();
                conversion.pop();
                postfix += e;
                e = conversion.top();
            }
            e = conversion.top();
            conversion.pop();
        }
        else
        {
            if (!conversion.empty())
                e = conversion.top();
            while ((!conversion.empty()) && (precedence(infix[i]) <= precedence(e
)))
            {
                c = conversion.top();
                conversion.pop();
                postfix += c;
                if (!conversion.empty())
                    e = conversion.top();
            }
            conversion.push(infix[i]);
        }
        i++;
    }
        while (!conversion.empty())
        {
            e = conversion.top();
            conversion.pop();
```

```
            postfix += e;
        }
    return postfix;
}

// This function is responsible for creating the inverter PUN and PDN
string Process_not(string x , string & output_str = not_list)
{
    output_str = output_str + "M" + to_string(mos_count) + ' ' + "NOT" + to_strin
g(not_nodes) + ' ' + x + ' ' + "vdd " + "vdd " + "PMOS" + '\n';
    mos_count++;
    output_str = output_str + "M" + to_string(mos_count) + ' ' + "NOT" + to_strin
g(not_nodes) + ' ' + x + ' ' + '0' + ' ' + '0' + ' ' + "NMOS" + '\n';
    mos_count++;
    not_nodes++;
    return "NOT" + to_string(not_nodes - 1);
}

// This function is resposible for creating the PUN and PDN for both NAND and AND
 (by adding an inverter) gates.
string process_and(vector<string>elements, bool invert)
{
    for (int i = 0; i < elements.size(); i++)
    {
        and_list = and_list + "M" + to_string(mos_count) + ' ' + "AND" + to_strin
g(and_nodes) + ' ' + elements[i] + ' ' + "vdd vdd " + " PMOS" + '\n';
        mos_count++;
        if (i == 0)
            and_list = and_list + "M" + to_string(mos_count) + ' ' + "AND" + to_s
tring(and_nodes) + ' ' + elements[i] + ' ' + "M" + to_string(mos_count + 2) + "NO
DE " + "M" + to_string(mos_count + 2) + "NODE " + "NMOS" + '\n';
        else
            if (elements[i] == elements.back())
                and_list = and_list + "M" + to_string(mos_count) + ' ' + "M" + to
_string(mos_count) + "NODE" + ' ' + elements[i] + ' ' + "0 0 " + " NMOS" + '\n';
            else
                and_list = and_list + "M" + to_string(mos_count) + ' ' + "M" + to
_string(mos_count) + "NODE" + ' ' + elements[i] + ' ' + "M" + to_string(mos_count
 + 2) + "NODE" + "M" + to_string(mos_count + 2) + "NODE" + " NMOS" + '\n';
        mos_count++;
    }
    string out = "AND" + to_string(and_nodes);
    if (invert)
        out = Process_not(out, and_list); // flag to invert the NAND into AND to
avoid double inversion
```

```
    and_nodes++;
    return out;
}

//This function is resposible for creating the PUN and PDN for both NOR and OR (b
y adding an inverter) gates.
string process_or(vector<string>elements, bool invert)
{
    for (int i = 0; i < elements.size(); i++)
    {
        or_list = or_list + "M" + to_string(mos_count) + ' ' + "or" + to_string(o
r_nodes) + ' ' + elements[i] + ' ' + "0 0" + " NMOS" + '\n';
        mos_count++;
        if (i == 0)
            or_list = or_list + "M" + to_string(mos_count) + ' ' + "M" + to_strin
g(mos_count + 2) + "NODE" + ' ' + elements[i] + ' ' + "vdd vdd " + "PMOS" + '\n';
        else
            if (elements[i] == elements.back())
                or_list = or_list + "M" + to_string(mos_count) + ' ' + "or" + to_
string(or_nodes) + ' ' + elements[i] + ' ' + "M" + to_string(mos_count) + "NODE "
 + "M" + to_string(mos_count) + "NODE" + " PMOS" + '\n';
            else
                or_list = or_list + "M" + to_string(mos_count) + ' ' + "M" + to_s
tring(mos_count + 2) + "NODE" + ' ' + elements[i] + ' ' + "M" + to_string(mos_cou
nt) + "NODE " + "M" + to_string(mos_count) + "NODE" + " PMOS" + '\n';
        mos_count++;
    }
    string out = "OR" + to_string(or_nodes);
    if (invert)
        out = Process_not(out, or_list); //flag to invert the NOR into OR to avoi
d double inversion
    or_nodes++;
    return out;
}

string replace(string func, string sub, string str)
{
    std::size_t pos = func.find(sub);
    while (pos != std::string::npos)
    {
        func.replace(pos, sub.length(), str);
        pos = func.find(sub);
    }
    return func;
}
```

```
// Generate the NETLIST function
string produce_final_output(string postfix , string output_label)
{
    // check-> x=(a&b)|(c'&(d|e')); y=f'&g'; z=h&(i|(j&m'));
    string netlist;
    stack<string> s;
    string temp = "";
    int successor_operands = 0;
    bool add_inverter = false;
    vector <string> arguments;
    //cout << "postfix " << postfix <<endl;
    //Parsing the postfix expression
    for ( int i = 0 ; i < postfix.length(); i++)
    {
        if (postfix[i] != '\'' && postfix[i] != '&' && postfix[i] != '|')
        {
            temp = postfix[i];
            s.push(temp);
        }
        else if (postfix[i] == '&')
        {
            if(!s.empty())
            {
                add_inverter=true;
                successor_operands=2;
                while (i < postfix.length() - 1 && postfix[i] == postfix[i + 1])
                {
                    i++;
                    successor_operands++;
                }
                if (i < postfix.length() - 1 && postfix[i + 1] == '\'')
                {
                    i++;
                    add_inverter = false; // NAND Gate
                }
                while (successor_operands != 0)
                {
                    if (s.empty())
                    {
                        cout << "Error in the counter of successor operands, post
 expression is not correct.\n";
                        break;
                    }
                    else
```

```
                    {
                        temp = s.top();
                        s.pop();
                        arguments.insert(arguments.begin(), temp);
                    }
                    successor_operands--;
                }
                temp = process_and(arguments, add_inverter);
                // for ( int i = 0 ; i < arguments.size () ; i++)
                // cout << "argument" << arguments[i] <<endl;
                arguments.clear();
                netlist += and_list;
                and_list = "";
                netlist += "\n";
                s.push(temp);
            }
            else
            {
                cout << "There is an error in the postfix string before an -
&- \n";
                break;
            }
        }
        else if (postfix[i] == '|')
        {
            if (!s.empty())
            {
                add_inverter=true;
                successor_operands = 2 ;
                while (i < postfix.length() - 1 && postfix[i] == postfix[i + 1])
                {
                    i++;
                    successor_operands++;
                }
                if (i < postfix.length() - 1 && postfix[i + 1] == '\'')
                {
                    i++;
                    add_inverter = false; // NOR Gate
                }
                while (successor_operands != 0)
                {
                    if (s.empty())
                    {
                        cout << "Error in the counter of successor operands, post
 expression is probably wrong\n";
```

```
                        break;
                    }
                    else
                    {
                        temp = s.top();
                        s.pop();
                        arguments.insert(arguments.begin(), temp);
                    }
                    successor_operands--;
                }
                temp = process_or(arguments, add_inverter);
                arguments.clear();
                netlist += or_list;
                or_list = "";
                netlist += "\n";
                s.push(temp);
            }
            else
            {
                cout << "There is an error in the postfix string before an -
|- \n";
                break;
            }
        }
        else if (postfix[i] == '\'')
        {
            if (!s.empty())
            {
                temp = s.top();
                s.pop();
                temp = Process_not(temp);
                netlist += not_list;
                not_list = "";
                netlist += "\n";
                s.push(temp);
            }
            else
            {
                cout << "There is an error in the postfix string before an -
'- \n";
                break;
            }
        }
    }
    if (!s.empty())
```

```
    {
        temp = s.top();
        s.pop();
        netlist = replace(netlist, temp, output_label); // net list for a single
operation ( &,/, '). Hence, the output lable is passed to append all the netlists
 and replace the last with the output label
    }
    return netlist;
}

int main()
{
    string user_input, temp, output_label="out";
    cout << "Please enter the expresssion(s):\n";
    /*
    The input should strictly follow the following format: <output1>=<expression1
>; <output2>=<expression2>; ... <outputn>=<expressionn>;
    where each expression:
    1) has no spaces within
    2) followed by exactly one ; and one "␣"
    3) the last expression should be terminated by ';'
    Example of valid input: y=a&b; x=c|d;
    Example of invalid input: a&b;
    Example of invalid input: x=a&b y=c';
    Example of invalid input: x=a& b;
    */
    getline(cin, user_input);
    int n = user_input.length(), expression_starting_index = 0;
    vector<string> expressions, cropped_expressions;
    for (int i = 0; i < n; i++)
        if (user_input[i] == ';')
        {
            temp = user_input.substr(expression_starting_index, i - expression_st
arting_index);
            expressions.push_back(temp);
            cropped_expressions.push_back(temp.substr(2, temp.size() - 2));
            expression_starting_index = i + 2;
        }
    int m = cropped_expressions.size();
    cout << "Expressions:\n";
    for (int i = 0; i < m; i++)
        cout << cropped_expressions[i] << "\t\tCorresponding postfix-
>\t" << infix_to_postfix(cropped_expressions[i]) << endl;
    cout << "\n\n-------------- Final Output --------------\n\n";
    for (int i = 0; i < m; i++)
```

```
    {
        string final_netlist;
        not_list = "" ; or_list = "" ; and_list = "";
        mos_count = 0;
        not_nodes = 0; or_nodes = 0; and_nodes = 0; //reinitialize
        cout << "------- " << expressions[i] << " -------\n\n";
       final_netlist = produce_final_output(infix_to_postfix(cropped_expressions[
i]), output_label);
       cout << final_netlist << endl;
    }
    return 0;
}
```