

**Microelectronics Project.**  
**The American University In Cairo (AUC)**

Group Members:

Abdulahkim Badawi 900171087

Omer Hassan 900171920

Yahya Saad 900171786

**Project Summary:**

The program is generating the data part of the SPICE netlist of a given Boolean Expressions with NO spaces in between. The input expression must NOT include any parenthesis, spaces, or feedback (i.e. the output cannot be included in the equation of the input. E.g. we don't allow inputs in the form  $y=a\&y|c$ ). In addition, the inputs are case sensitive. The program respects the guidelines and the assumptions outlined in the project description. The output statements describing the transistors are in the format:

Name Drain Source Body Type.

The naming of the wires connecting the drains and the sources of the transistors are in the form OP\_ for PMOS and ON\_ for NMOS except for ground which is denoted by '0', vdd, and the output of course.

The Gate inputs are named the same after the inputs by the user.

The transistors are named by assigning unique numbers to them (i.e. M1 M2 etc.)

The naming we follow is that each inverter is called by its input name to give uniqueness to the inverter and to easily distinguish it. For example, if we want to invert A, the statement will look like **M\_A1 A' A 0 0 NMOS** and **M\_A2 A A' vdd vdd PMOS**, where the output is A'.

**Implementation Details:**

**Data structures used:**

- Vectors: used as a dynamic data structure storing values.
- Structs: making entities to help deal with the problems.

**Structs List:**

- mosfet: Struct for mosfet: specifies drain, source, and gate of a given MOSFET
- network: Struct for network: specifies every network connected together, Contains a vector of mosfets connected together and a count for the number of the transistors in the network.

**Assumptions:**

The inputs and the output are alphabets. We allow them only to be followed by “ ’ ” to indicate the negation. This assumption is done for simplicity; the other cases can be handled easily by looping over the string and store every distinct string of alphabets in an input array.

#### Program Flow: -

1. The program takes an input string from the user with no spaces.
2. The output is considered to be the string before “=”.
3.
  - 3.1. If the output is complemented, we modify the input string to satisfy the conditions of the Pull Down Network by inverting variables that are complemented and adjust the correct inputs. Then we call the PDN function to perform the circuit.
  - 3.2. If the output is not complemented, we modify the input string to satisfy the conditions of the Pull UP Network by inverting variables that are complemented and adjust the correct inputs and adjust the correct inputs. Then we call the PUN function to perform the circuit.
4. Invert function is called whenever needed since the inversion circuit is fixed.
5.     **PDN**
  - 5.1. the PDN function receives a string and construct a vector of networks to store the and networks since ANDing has the priority over oring. After completing the ANDing networks, it ores the networks together if needed any or operations.
  - 5.2. The network is constructed by connecting the drain of the current transistor to the source of the previous transistor, and connecting the source of the current transistor by a new wire , and storing the whole entity in the network vectors.
  - 5.3. Then, we add the or transistors.
  - 5.4. The first drain at all of the PDN is connected to the output variable and the last source of the whole PDN is connected to the ground.
  - 5.5. We loop over the networks and output them on the screen by the naming convention mentioned before for the NMOS.
6.     **PUN**
  - 6.1. the PUN function receives a input string and construct a vector of networks to store the and networks since ANDing has the priority over oring. After completing the ANDing networks, it ores the networks together if needed any or operations.
  - 6.2. The network is constructed by connecting the **source** of the current transistor to the **drain** of the previous transistor, and connecting the **drain** of the current transistor by a new wire output (e.g. OP-1) , and storing the whole entity in the network vectors.
  - 6.3. Then, we add the or transistors if needed.

- 6.4. The last **drain** at all of the PUN is connected to the output variable and the first **source** of the whole PUN is connected to the voltage source (VDD).
  - 6.5. We loop over the networks and output them on the screen by the naming convention mentioned before for the PMOS.
7. After completing either PDN and PUN, the complement function is applied to the string to apply De Morgan's law to the expression so that we can construct the other network (PDN/PUN). Then either PUN2 or PDN2 are called.

## 8. PDN2

- 8.1. It's applying the same logic as PDN, but this time it's giving the constructing the or networks first because it's called after we apply DE morgen's laws.
- 8.2. After the or networks are done, we add the and transistors if needed.
- 8.3. We loop over the vector and output the networks correctly according to the naming convention.

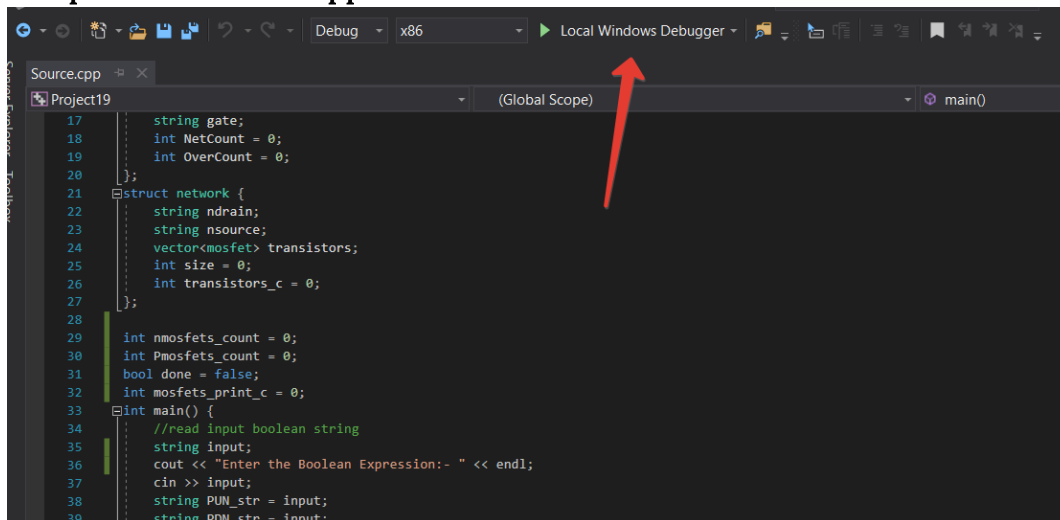
## 9. PUN2

- 9.1. It's applying the same logic as PUN, but this time it's giving the constructing the or networks first because it's called after we apply DE morgen's laws.
- 9.2. After the or networks are done, we add the and transistors if needed.
- 9.3. We loop over the vector and output the networks correctly according to the naming convention.

**After the completion of these steps the data segment of the netlist is displayed on the screen.**

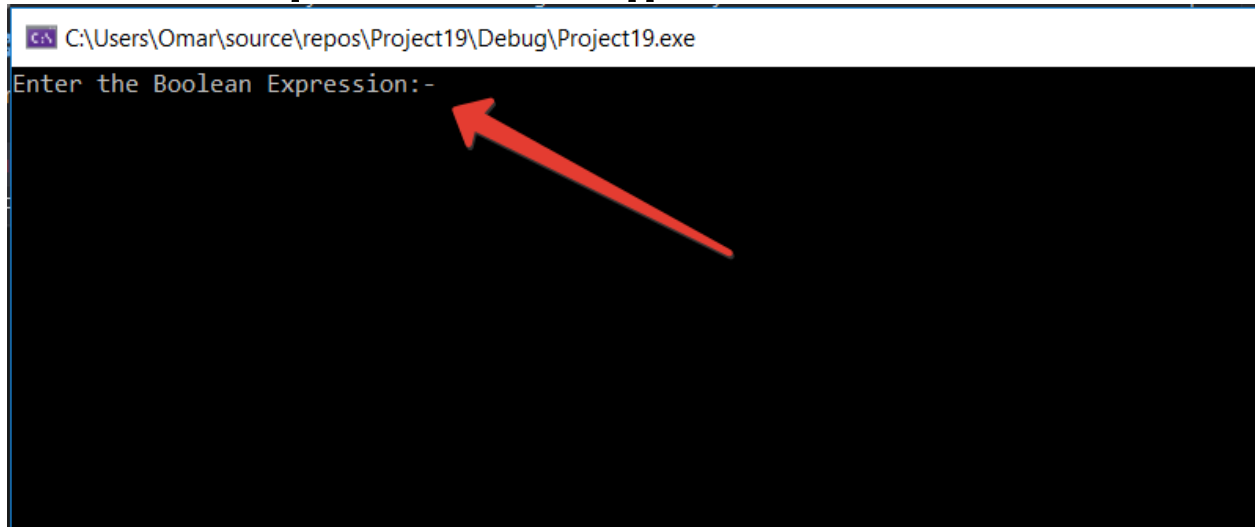
How to run the program:

1. Compile and build the cpp file.



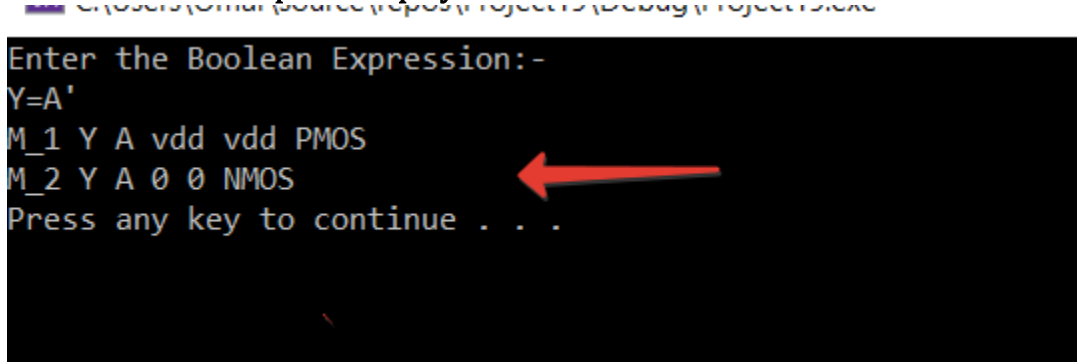
```
Source.cpp | Project19 | (Global Scope) | main()
17  string gate;
18  int NetCount = 0;
19  int OverCount = 0;
20  };
21  struct network {
22  string ndrain;
23  string nsource;
24  vector<mosfet> transistors;
25  int size = 0;
26  int transistors_c = 0;
27  };
28
29  int nmosfets_count = 0;
30  int Pmosfets_count = 0;
31  bool done = false;
32  int mosfets_print_c = 0;
33  int main() {
34  //read input boolean string
35  string input;
36  cout << "Enter the Boolean Expression:- " << endl;
37  cin >> input;
38  string PUN_str = input;
39  string PDN_str = input;
```

2. Enter the Boolean expression in the console application.



```
C:\Users\Omar\source\repos\Project19\Debug\Project19.exe
Enter the Boolean Expression:-
```

3. Then the netlist output is displayed on the screen.



```
Enter the Boolean Expression:-
Y=A'
M_1 Y A vdd vdd PMOS
M_2 Y A 0 0 NMOS
Press any key to continue . . .
```

- **Note:** Step 1 can be replaced by running the .exe directly.

## Test Cases:

### Test 1:

Input:  $Y = A \& B | C \& D'$

```
Enter the Boolean Expression:-
Y=A&B|C&D'
M_A_1 A' A vdd vdd PMOS
M_A_2 A' A 0 0 NMOS
M_B_1 B' B vdd vdd PMOS
M_B_2 B' B 0 0 NMOS
M_C_1 C' C vdd vdd PMOS
M_C_2 C' C 0 0 NMOS
M_1 OP_1 A' vdd vdd PMOS
M_2 Y B' OP_1 OP_1 PMOS
M_3 OP_2 C' vdd vdd PMOS
M_4 Y D OP_2 OP_2 PMOS
M_5 Y A' ON_1 ON_1 NMOS
M_6 Y B' ON_1 ON_1 NMOS
M_7 ON_1 C' 0 0 NMOS
M_8 ON_1 D 0 0 NMOS
Press any key to continue . . .
```

### Test 2: AND Gate:

$Y = A \& B;$

```
Enter the Boolean Expression:-
Y=A&B
M_A_1 A' A vdd vdd PMOS
M_A_2 A' A 0 0 NMOS
M_B_1 B' B vdd vdd PMOS
M_B_2 B' B 0 0 NMOS
M_1 OP_1 A' vdd vdd PMOS
M_2 Y B' OP_1 OP_1 PMOS
M_3 Y A' 0 0 NMOS
M_4 Y B' 0 0 NMOS
Press any key to continue . . .
```

### Test 3: NAND Gate:

$$Y' = A \& B;$$

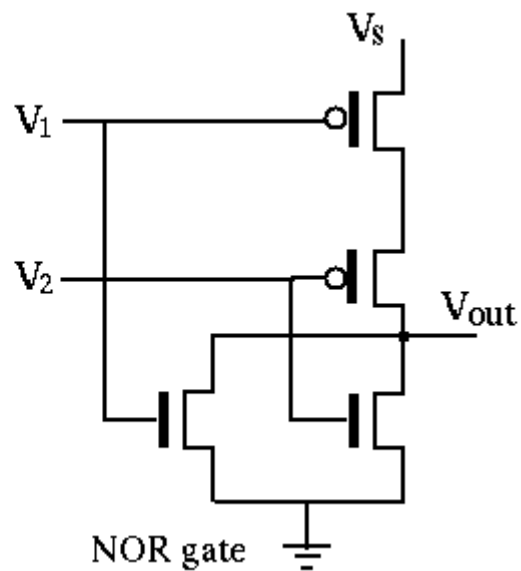
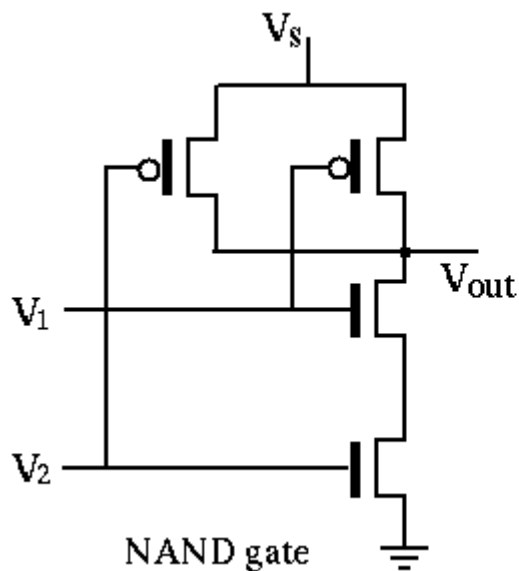
```
Enter the Boolean Expression:-  
Y'=A&B  
M_1 Y A ON_1 ON_1 NMOS  
M_2 ON_1 B 0 0 NMOS  
M_3 Y A vdd vdd PMOS  
M_4 Y B vdd vdd PMOS  
Press any key to continue . . .
```

### Test 4: NOR Gate:

$$Y' = A | B$$

```
Enter the Boolean Expression:-  
Y'=A|B  
M_1 Y A 0 0 NMOS  
M_2 Y B 0 0 NMOS  
M_3 OP_1 A vdd vdd PMOS  
M_4 Y B OP_1 OP_1 PMOS  
Press any key to continue . .
```

Note: NOR and NAND are identical to the correct output studied before.



### Test 5: XOR Gate:

$$Y=A\&B'|A'\&B$$

Enter the Boolean Expression:-

$$Y=A\&B'|A'\&B$$

M\_A\_1 A' A vdd vdd PMOS

M\_A\_2 A' A 0 0 NMOS

M\_B\_1 B' B vdd vdd PMOS

M\_B\_2 B' B 0 0 NMOS

M\_1 OP\_1 A' vdd vdd PMOS

M\_2 Y B OP\_1 OP\_1 PMOS

M\_3 OP\_2 A vdd vdd PMOS

M\_4 Y B' OP\_2 OP\_2 PMOS

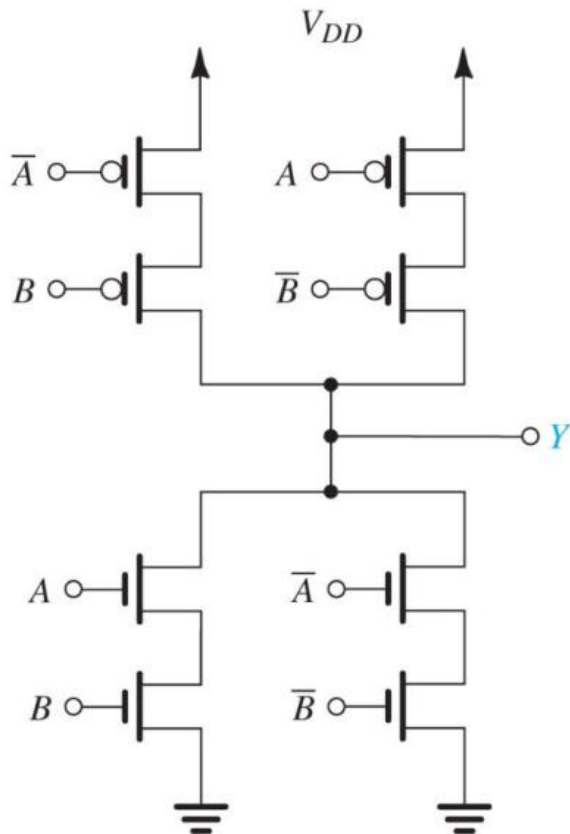
M\_5 Y A' ON\_1 ON\_1 NMOS

M\_6 Y B ON\_1 ON\_1 NMOS

M\_7 ON\_1 A 0 0 NMOS

M\_8 ON\_1 B' 0 0 NMOS

Note: the result is correct according to the circuit description studies before.



Test 6:

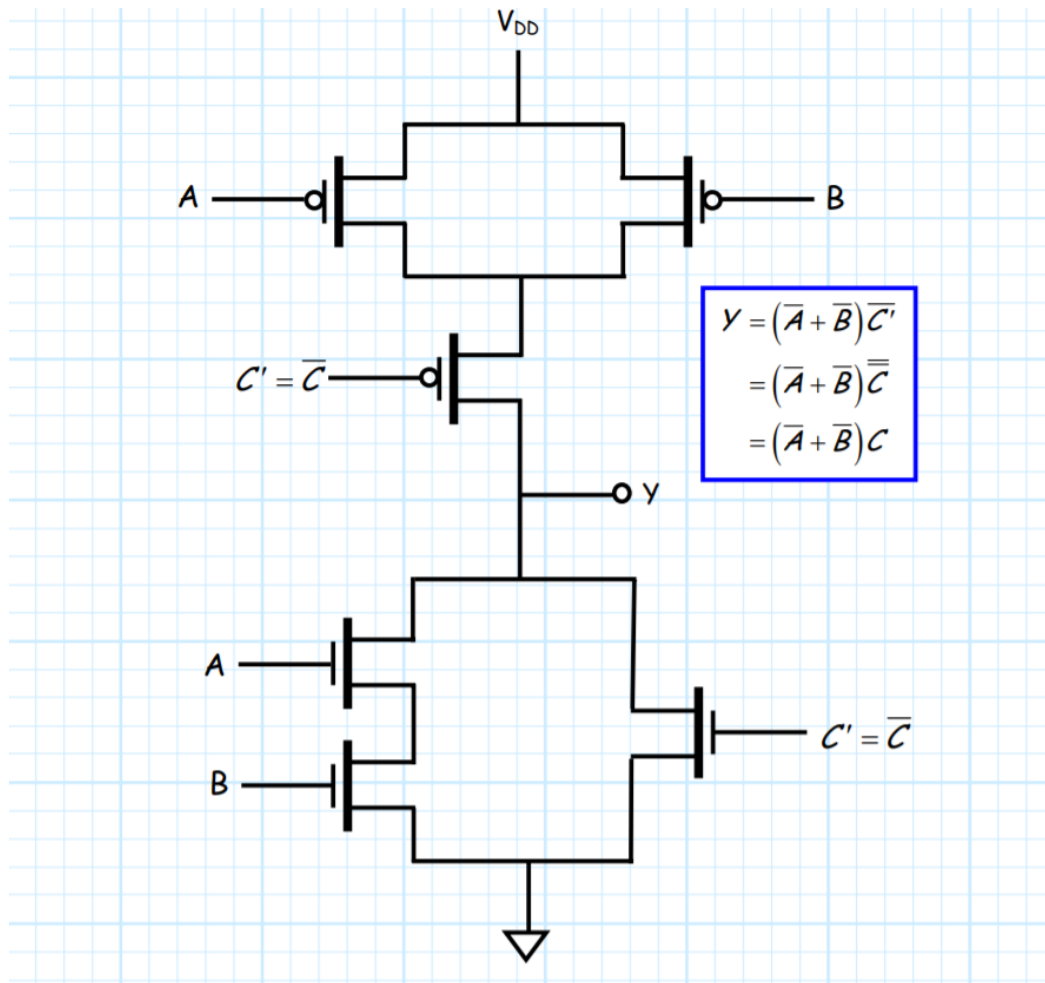
$$Y = A \& B | C'$$

```

Enter the Boolean Expression:-
Y'=A&B|C'
M_C_1 C' C vdd vdd PMOS
M_C_2 C' C 0 0 NMOS
M_1 Y A ON_1 ON_1 NMOS
M_2 ON_1 B 0 0 NMOS
M_3 Y C' 0 0 NMOS
M_4 OP_1 A vdd vdd PMOS
M_5 OP_1 B vdd vdd PMOS
M_6 Y C' OP_1 OP_1 PMOS
  
```

Note: the result is correct according to online sources.





Test 7: A generic Expression:

Enter the Boolean Expression:-

$Y = A' \& B | C' \& D | K \& M$

M\_B\_1 B' B vdd vdd PMOS

M\_B\_2 B' B 0 0 NMOS

M\_D\_1 D' D vdd vdd PMOS

M\_D\_2 D' D 0 0 NMOS

M\_K\_1 K' K vdd vdd PMOS

M\_K\_2 K' K 0 0 NMOS

M\_M\_1 M' M vdd vdd PMOS

M\_M\_2 M' M 0 0 NMOS

M\_1 OP\_1 A vdd vdd PMOS

M\_2 Y B' OP\_1 OP\_1 PMOS

M\_3 OP\_2 C vdd vdd PMOS

M\_4 Y D' OP\_2 OP\_2 PMOS

M\_5 OP\_3 K' vdd vdd PMOS

M\_6 Y M' OP\_3 OP\_3 PMOS

M\_7 Y A ON\_1 ON\_1 NMOS

M\_8 Y B' ON\_1 ON\_1 NMOS

M\_9 ON\_1 C ON\_2 ON\_2 NMOS

M\_10 ON\_1 D' ON\_2 ON\_2 NMOS

M\_11 ON\_2 K' 0 0 NMOS

M\_12 ON\_2 M' 0 0 NMOS

Sources:

[http://www.ittc.ku.edu/~jstiles/312/handouts/section\\_10\\_3\\_CMOS\\_Logic\\_Gate\\_Circuits\\_package.pdf](http://www.ittc.ku.edu/~jstiles/312/handouts/section_10_3_CMOS_Logic_Gate_Circuits_package.pdf)

<http://turner.faculty.swau.edu/mathematics/materialslibrary/truth/>

[http://bwrce.eecs.berkeley.edu/Courses/ic541ca/ic541ca\\_f01/Notes/chapter6.pdf](http://bwrce.eecs.berkeley.edu/Courses/ic541ca/ic541ca_f01/Notes/chapter6.pdf)