CSCE330302 - Fundamental Microelectronics

Spring 21

Professor:

Dr. Dalia Selim

CMOS Circuit SPICE Generator

Program objective:

This program is a basic CMOS circuit SPICE generator which takes Boolean expression and then generate the SPICE of its circuit which shows the details of the transistors which are used to make that expression.

Summary:

Since inverters are the basic gates for making any function, we can use the combination of PUN and PDN (Pull Up Network and Pull Down Network) to design any function. Therefore, the Boolean expression that is taken from the user could be implemented by constructing the PUN and PDN; furthermore, the concept of the duality between both networks makes the design process easier by implementing one of them and infer the other by changing the way the transistors are connected. In our case, we used the Boolean function to generate the PUN and then, by duality, we obtained the PDN easily.

Program description:

The program starts by taking the Boolean expression from the user and validating it for any violation, then it constructs the PUN by making the PMOS transistors along with any inverters needed to obtain a complemented input from an uncomplemented input. Using the concept of duality, the program constructs the PDN by making the NMOS transistors. In other words, transistors that are connected in series in the PUN are connected in parallel in the PDN and vice versa.

How it works:

The user writes the full Boolean expression (including the output variable) without any space or parentheses. ‘&’, ‘|’, and ‘’’ are used for AND, OR, and NOT respectively. In case of multiple functions, semicolons are used. For examples, these inputs are valid:

y=a&b

x=A’|c’

y=a’&b; x=c|d’

Code documentation:

Transistors are stores in a construct *transistor* which includes the information required for the SPICE: name, drain, gate, source, body, and type. Transistors are stored as a vector of *transistor.* The functions are *generatePunAndInverters, generageNmos, generateSpice, nOfInverters*, and *validated.* From their names, the first two functions are used to design the transistors, the *generateSpice* is used as the control function that directs the flow of the program, and the *nOfInverters* and *validated* are used as utility function that handle the number of nodes and validate the Boolean expression, respectively.

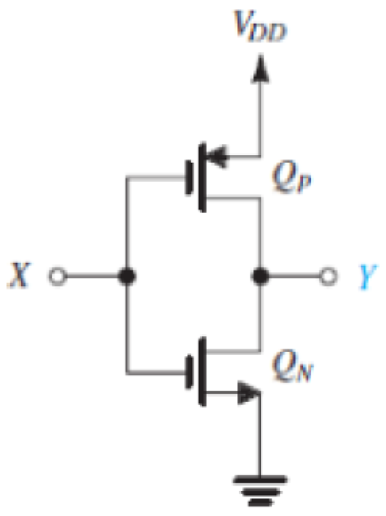
1) validated

Since the user can enter multiple semicolon-separated functions, the program at first validates the expression for any violations as well as for splitting the input into functions in case of multiple functions. Therefore, the function takes the direct input the user entered and works on it, then it returns true if the input was validated without any violations and false otherwise. It receives a vector of strings **by reference** as an approach to obtain the function/s from the input, and this is like a technique that helps to return information more than the Boolean variable that shows the validation of the input. Violations could be not starting with a variable, putting some character rather than the '=', or using variables names with something else than the letters, '|', '`', '&', or ';'. If such a violation happens, the program asks the user to reinput a correct expression.

2) *nOfInverters*

It is a utility function added to do a specific task. Since transistors used are not only PUN and PDN, but rather inverters are used. And in order to make the numbers of nodes count properly, we do the following:

Calculate the number of inverters used, then this number will represent the number of nodes used just for the inverters before making any PUN or PDN. For example, in the inverter shown in the following figure, we can see that X is the input and Y is the output, but when it is a larger circuit, Y then will just be an **intermediate** output. Therefore, we give such intermediate outputs a node (wire) numbered starting from 1 and incrementing with the different nodes. Therefore, in order to organize the PUN and the inverters, it was important to know how many nodes will be used for the inverters, then the first intermediate node in the PUN will start from the following number of the number of inverters. The number of transistors is calculated using a global variable as a counter which increments once it detects an uncomplemented input (in order to obtain its complement and use it directly in the PUN). An uncomplemented input is detected if there is any letter in the expression (English letter which represents a variable) NOT followed by '``'. And '``' here has the ASCII code of 39.



3) *generatePunAndInverters*

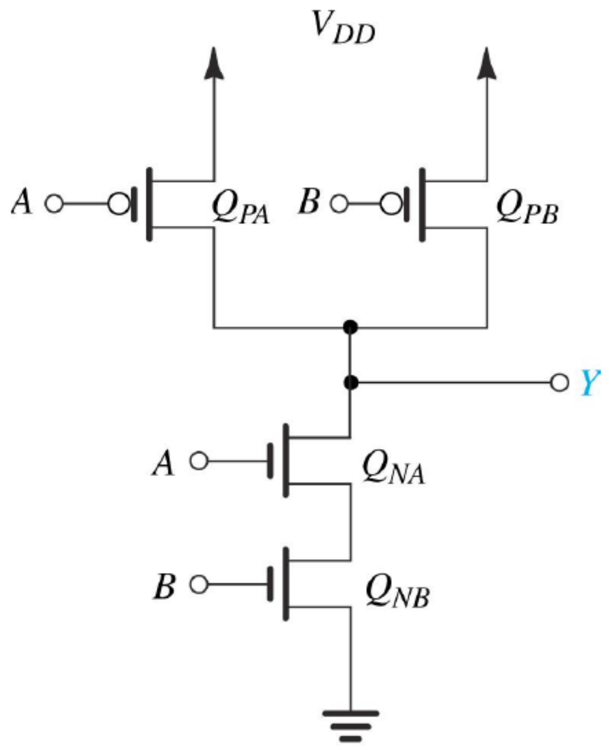
This is the actual starting point of generating the SPICE. From its name, it generates the PUN and the inverters by returning the PUN and making the inverters through a called by reference vector. The algorithm was thought of as following:

> We loop over the Boolean expression for the function. Then if we met an English letter (not the '&' or '|' characters) followed by '``', then this is an input that does NOT need an inverter to be used. Else if it is an English letter not followed by '``', then it is an uncomplemented input that needs to be inverted in order to be used in the PUN. In the first case when no inverters are needed, the logic goes the following:
>
> > If it is the starting of the expression or exactly after an '|', then the source of the PMOS is connected to the Vdd. If the expression approaches either the end or an '|', then the drain is connected to the output of function. And since no parentheses are allowed by default, then the process will keep going without any limitations. When we meet an '&', then we have a series connection which requires an intermediate outputs which gets assigned by the global variable *nodeCount* which is incremented with every new node. Also, in our case, the transistors have a common terminal between the source and body.
> >
> > Input gates are the complement of the input variables, so if the input was originally complemented, we just use the uncomplemented version of it (without making any inverters). Otherwise, when the input is originally uncomplemented, then we construct an inverter at first with whose input is that uncomplemented input and its output will be the one used in the gate of the PMOS transistor.
> >
> > Names are initialized in an ascending order starting from M0 by using a global variable that gets incremented whenever a new transistor is created. Types are initialized as PMOS in case of the pmos vector and as PMOS and NMOS in the case of the inverters vector.

At the end of the function, the pmos vector is returned and the inverter are initialized back to the *generateSpice* since it is a call by reference.

4) generageNmos

This function is basically implemented using the concept of duality. It takes the PMOS transistors (with their connections) and accordingly generate the PDN. As shown in the following figure, by duality, the PDN takes the same inputs of the pmos transistors (which are basically inverted from the Boolean expression) then change the connection: the series becomes parallel and vise versa. As the example states that the PMOS transistors in PUN have inputs A and B which are the same as the NMOS transistors in PDN.



Therefore, with the information we have from the PUN, we can easily obtain the NMOS transistors as following:

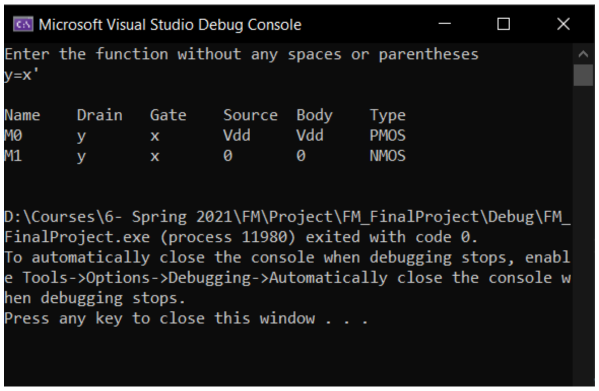
The leftmost branch in the PUN (connected in series) will be the lowest branch in the PDN (connected in parallel), therefore, the source will be grounded. The rightmost branch in the PUN will be the top branch in the PDN, therefore, the drain will be connected to the output function. When the PUN reads a new transistor connected to Vdd, then this is a new branch, which will be translated into a new node in the PDN as long as it is not the rightmost branch.

The function returns the vector of nmos transistors.

5) generateSpice

This function organizes the flow of the program. After validating the expression and splitting it into different functions in case of multiple functions separated by semicolons, then we need to generate SPICE for the function/s. And as mentioned, in order to generate SPICE, we need to construct the PUN along with any required inverters and the PDN. Therefore, the vectors pmos, inverters, and nmos are used to obtain those transistors from the other functions (generatePunAndInverters and generageNmos).

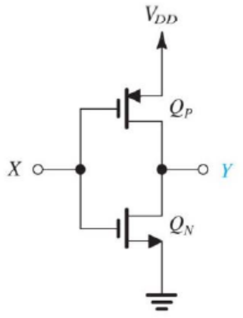
Samples:

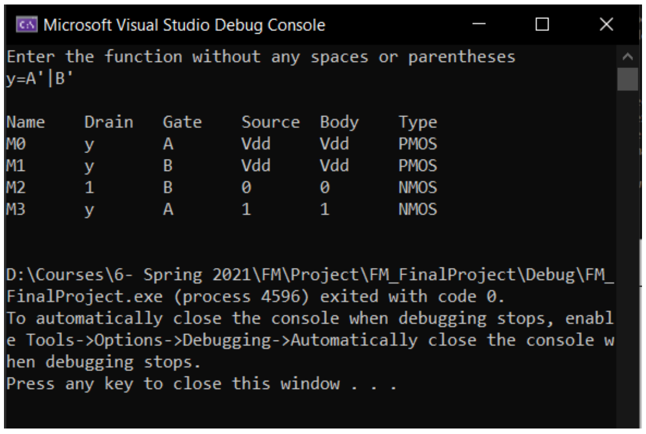


```
Enter the function without any spaces or parentheses
y=x'

Name    Drain   Gate    Source  Body    Type
M0      y       x       Vdd     Vdd     PMOS
M1      y       x       0       0       NMOS


D:\Courses\6- Spring 2021\FM\Project\FM_FinalProject\Debug\FM_
FinalProject.exe (process 11980) exited with code 0.
To automatically close the console when debugging stops, enabl
e Tools->Options->Debugging->Automatically close the console w
hen debugging stops.
Press any key to close this window . . .
```

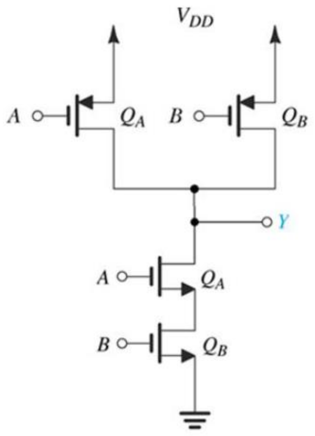


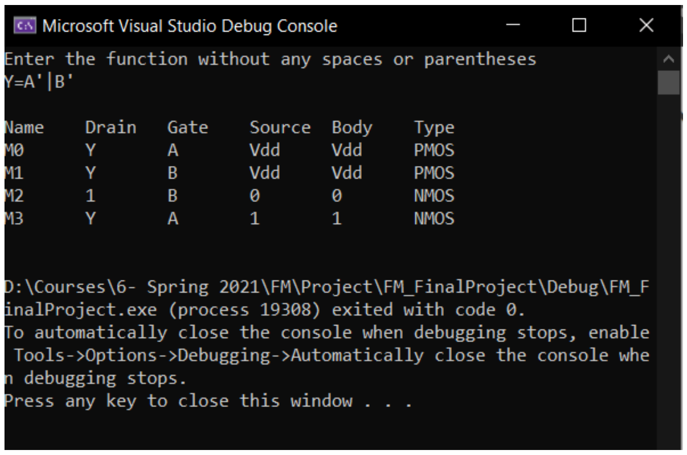


```
Enter the function without any spaces or parentheses
y=A'|B'

Name    Drain   Gate    Source  Body    Type
M0      y       A       Vdd     Vdd     PMOS
M1      y       B       Vdd     Vdd     PMOS
M2      1       B       0       0       NMOS
M3      y       A       1       1       NMOS


D:\Courses\6- Spring 2021\FM\Project\FM_FinalProject\Debug\FM_
FinalProject.exe (process 4596) exited with code 0.
To automatically close the console when debugging stops, enabl
e Tools->Options->Debugging->Automatically close the console w
hen debugging stops.
Press any key to close this window . . .
```

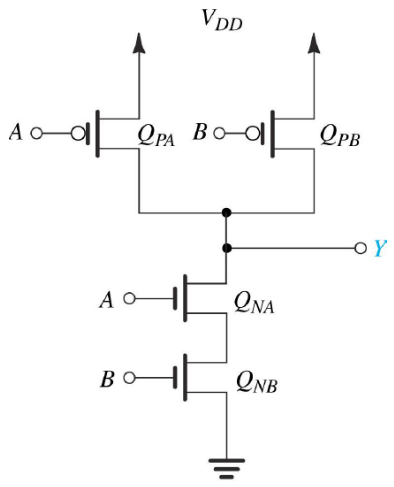


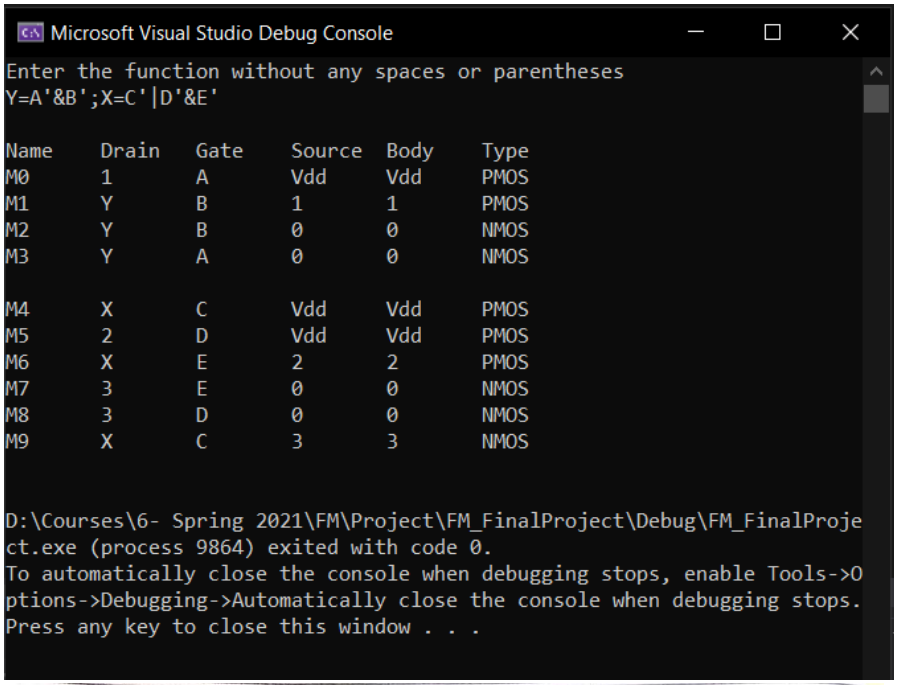


```
Enter the function without any spaces or parentheses
Y=A'|B'

Name    Drain   Gate    Source  Body    Type
M0      Y       A       Vdd     Vdd     PMOS
M1      Y       B       Vdd     Vdd     PMOS
M2      1       B       0       0       NMOS
M3      Y       A       1       1       NMOS


D:\Courses\6- Spring 2021\FM\Project\FM_FinalProject\Debug\FM_F
inalProject.exe (process 19308) exited with code 0.
To automatically close the console when debugging stops, enable
 Tools->Options->Debugging->Automatically close the console whe
n debugging stops.
Press any key to close this window . . .
```



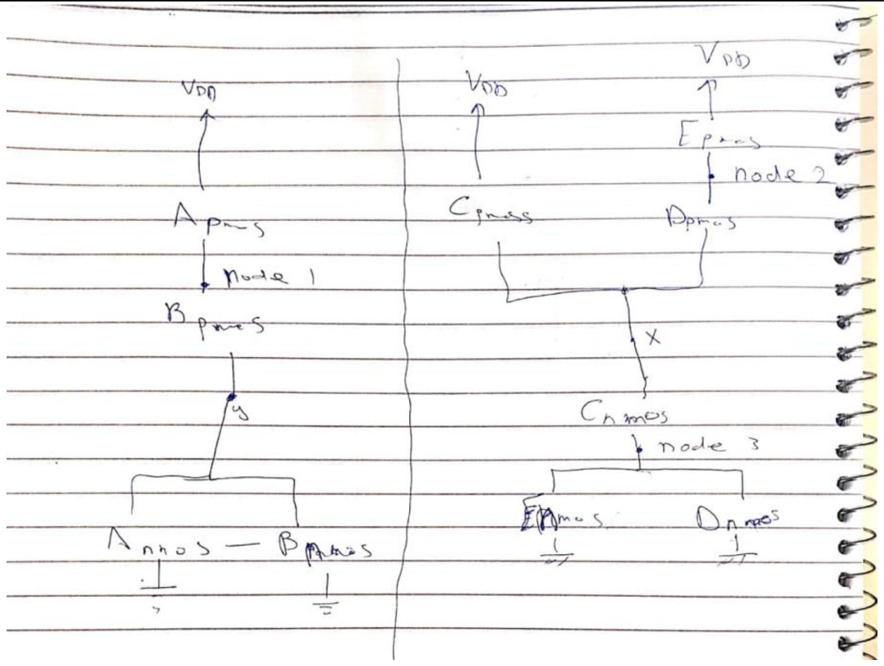


```
Microsoft Visual Studio Debug Console
Enter the function without any spaces or parentheses
Y=A'&B';X=C'|D'&E'

Name    Drain   Gate    Source  Body    Type
M0      1       A       Vdd     Vdd     PMOS
M1      Y       B       1       1       PMOS
M2      Y       B       0       0       NMOS
M3      Y       A       0       0       NMOS

M4      X       C       Vdd     Vdd     PMOS
M5      2       D       Vdd     Vdd     PMOS
M6      X       E       2       2       PMOS
M7      3       E       0       0       NMOS
M8      3       D       0       0       NMOS
M9      X       C       3       3       NMOS


D:\Courses\6- Spring 2021\FM\Project\FM_FinalProject\Debug\FM_FinalProje
ct.exe (process 9864) exited with code 0.
To automatically close the console when debugging stops, enable Tools->O
ptions->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```



VDD
A pmos
node 1
B pmos
y
A nmos
B nmos
VDD
VDD
E pmos
node 2
C pmos
D pmos
X
C nmos
node 3
E nmos
D nmos