EE 633 - Computational Aspects of VLSI

Term Project

Mehmet Fatih Gülakar

January 2022

In the project, I have designed a program that receives a SPICE netlist, and placement data of the transistors inside it, then produce a *.asc* file, so that the netlist can be displayed on LTspice as a schematic. Such a program can also be called *"SPICE Netlist Visualizer"*.

First and foremost, a sub-program to parse a SPICE netlist is designed. In the project, we only have dealt with MOSFET devices. So parsing a resistor/capacitor is not an issue. In a SPICE netlist, MOSFETs are declared using the syntax below:

$$InstanceName\ \ ND\ \ NG\ \ NS\ \ NB\ \ ModelName\ \ Length\ \ Width\ \ Typename$$

where *InstanceName* is the name of the transistor, for example, M1 or M2. *ND, NG, NS & NB* are node names of the drain, gate, source, and bulk terminals of the transistor, respectively. *ModelName* is the model name of the transistor, e.g. nfet. *Length & Width* is the usual MOSFET parameters. Finally, *Typename* indicates this transistor is either NMOS or PMOS.

Moreover, there are external commands in the netlist. These commands do not exist in an ordinary netlist but are included so that placement and routing are handled easier. These commands start with "$* >>$", which is considered as a comment by the LTspice simulator, but not by our parser. Command types are explained below,

- $* >>$ **pairs Mn Mp**: Pairs Mn & Mp transistors, so that these transistors are placed together. Also, one MOSFET in the pair is mirrored with respect to the other, so routing becomes easier.
- $* >>$ **port input/output netName**: This command stores the fact that netName is a port. The ports are then displayed on the *.asc* file.

For example, if we analyze the following SPICE line,

$$M4\ \ N003\ \ N005\ \ N006\ \ 0\ \ nfet1\ \ l = 300n\ \ w = 4u$$

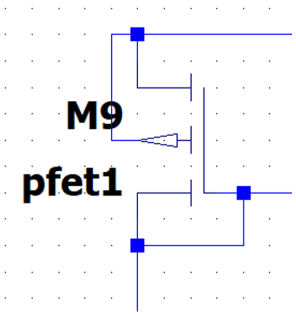
This line creates a MOSFET object called M4. Its drain/gate/source/bulk nodes are N003, N005, N006 and 0 (ground), respectively. So in the nets attribute of the netlist object, there will be an entry called N003, which has M4D, which means drain of M4. Similarly, the N006 entry will have M4S. Using such a technique eases the routing process since we have each net and name of the transistor terminal.

After designing the netlist class, I have started to design the Router class, which is the biggest and most complex class of the whole program. This class takes the netlist object created before as an attribute, so transistors and nets inside it are accessed more easily. It also needs the result of the placement program. The placement file also needs to be parsed. It has the following syntax,
(M9 M10) (M6)
(M7 M8) ()
(M4 M5) ()
(M2 M3) ()
(M0) (M1)
This file above indicates the following,

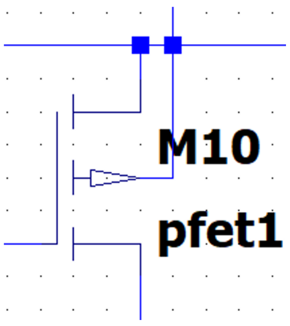
- M0 is left of M1 and below M2&M3.
- M9/M10, M7/M8, M4/M5, and M2/M3 are all pairs and placed together.
- Above M9, M10, and M6, there is the VDD power line.
- Below M0 and M1, there is a ground line.
- Empty parentheses indicate there is no transistor there. That means there is an empty area between M1 and M6.

During the parsing, we have the names of the MOSFETs in an NxM grid. We also have whether the transistor is left or right in a pair. For example, M9 is left and M10 is the right one. This helps us during the rotation of the MOSFETs.

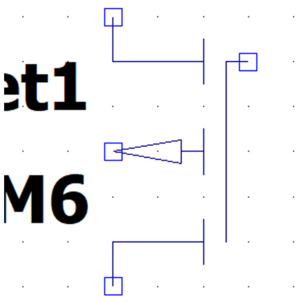
After parsing the placement file, we start to rotate the transistors. Rotating generates better routing results. Before explaining how it is done, it is better to explain LTspice's syntax about this. In LTspice, there are several configurations of rotation,



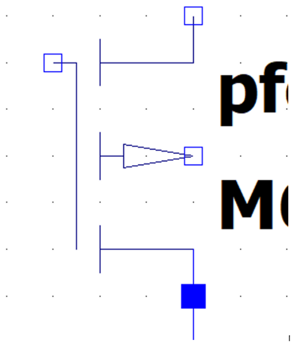
(a) M0 rotation



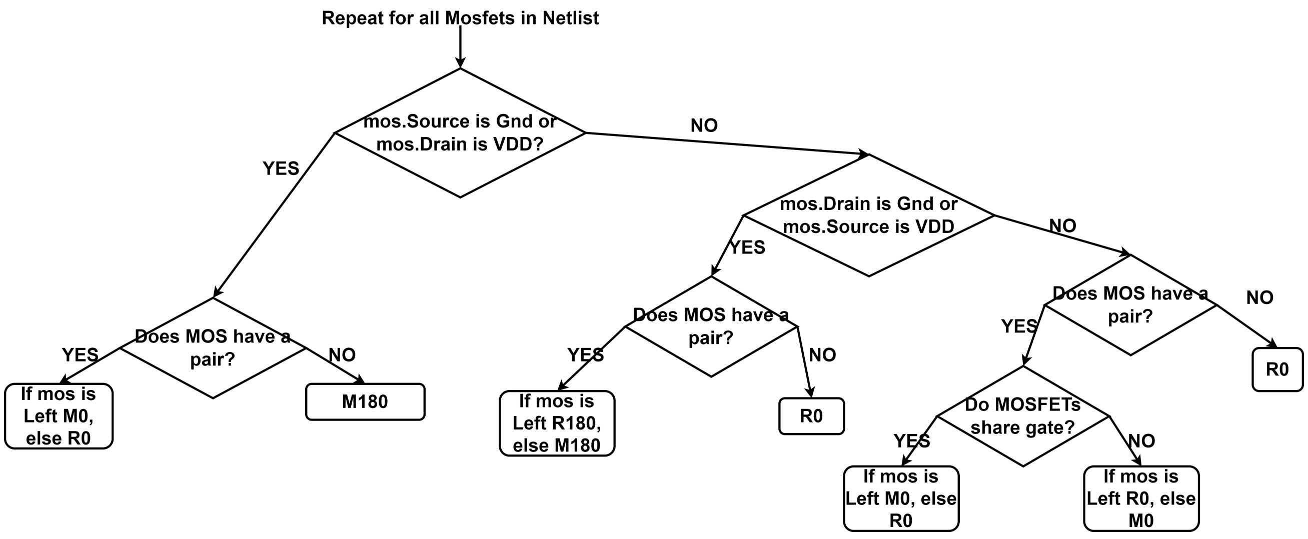
(b) R0 rotation



(c) R180 rotation


(d) M180 rotation

Determining the orientation of a transistor uses the flowchart in Figure 2:


Repeat for all Mosfets in Netlist
mos.Source is Gnd or mos.Drain is VDD?
YES
NO
Does MOS have a pair?
YES
NO
If mos is Left M0, else R0
M180
mos.Drain is Gnd or mos.Source is VDD
YES
NO
Does MOS have a pair?
YES
NO
If mos is Left R180, else M180
R0
Does MOS have a pair?
YES
NO
R0
Do MOSFETs share gate?
YES
NO
If mos is Left M0, else R0
If mos is Left R0, else M0
Figure 2: Flowchart to determine orientation of transistorss

Simply, terminals that are connected to VDD and ground became closer to top and bottom, respectively. We also mirror one of the transistors in a pair.

After rotation, we start to place MOSFETs in a 2D grid. In an *.asc* file, a MOS transistor has set of commands:

$$SYMBOL\ \ symbolName\ \ mos.y\ \ mos.x\ \ mos.rotation$$

where symbolName is whether pmos4 or nmos4, depending on transistor type, mos.y, mos.x, and mos.rotation are the y and x coordinates, and rotation of the transistor

$$SYMATTR\ \ InstName\ \ instanceName$$

instanceName is the name of the transistor as in SPICE netlist.

$$SYMATTR\ \ Value\ \ MOSmodel$$

MOSmodel is model name, like nfet.

$$SYMATTR\ \ Value2\ \ length\ \ width$$

assigns length and width, which can be easily obtained from the netlist. The only unknown in these commands here is the coordinates of the MOSFET. In the place_mos() method, we calculate coordinates of the transistor based on whether they have a pair and their location in the placement file. Rotation of them also affects the coordinates, so we take it into account. After placement, the program gives the output shown at Figure 3.

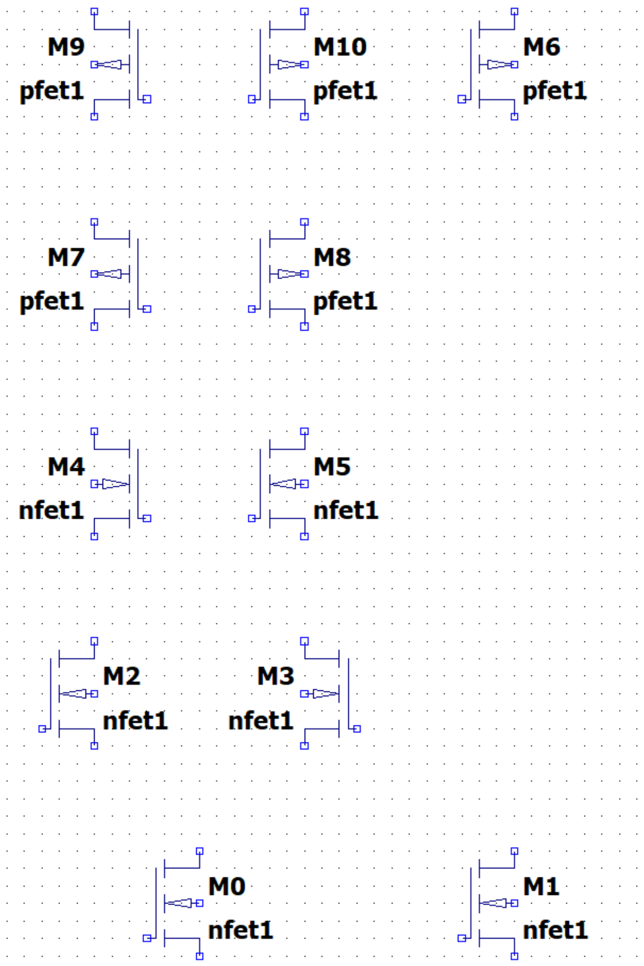


Figure 3: Schematic of Telescopic Two-Stage amplifier after rotation & placement

Then, we do an operation called set_all_terminal_coordinates(). Given the transistor coordinate, this method calculates the coordinates of its terminals. So, we know exact points we want to connect using wires. Figure 4 shows the symbol of the pmos4 model. The cross at top-left is the overall coordinate of the transistor. Drain coordinate is 3 grids left of this point. In LTspice, grid size is 16. So, we calculate drain coordinate by adding 48 to transistor's x position. Also, these offsets depend on rotation of the transistor.

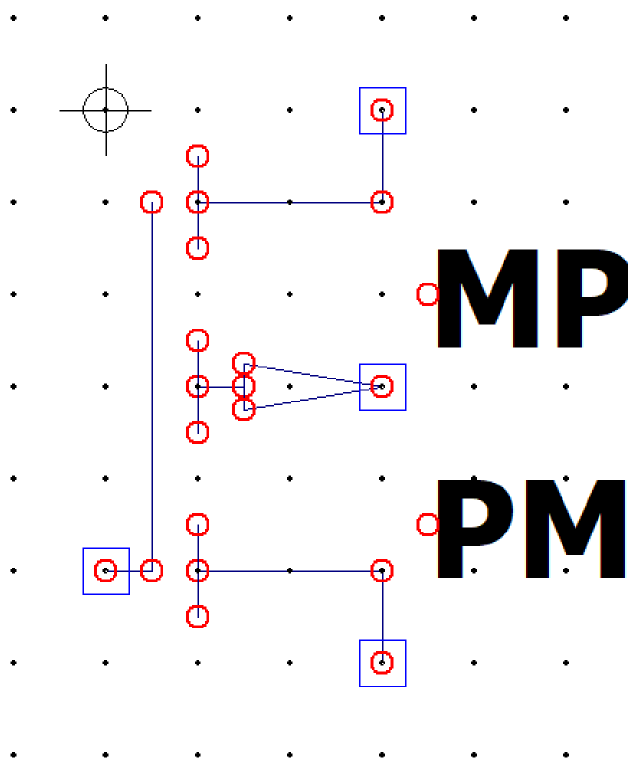


Figure 4: Detailed symbol of pmos4

Now we have the coordinates of each terminal. Before moving into the routing phase. We execute a small sub-program called extend_mos_terminals(). This sub-program extends the transistor terminals by putting a small wire segment on them. This process prevents some bad looking routing results, such as one shown in Figure 5. It is not clear to see whether wire is connected to gate of M1&M2.

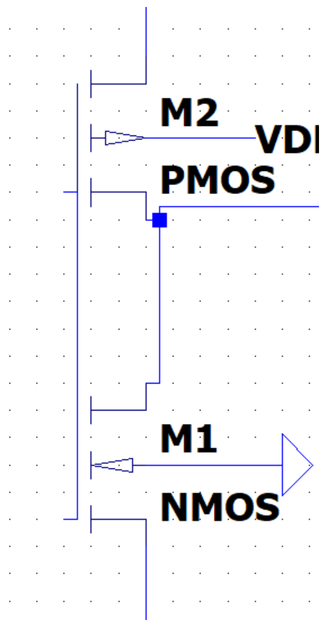


Figure 5: Not-good looking routing

We extend the terminals so that wires don't overlap transistor symbol, like in Figure 5. So after the extension, the schematic will look like Figure 6.

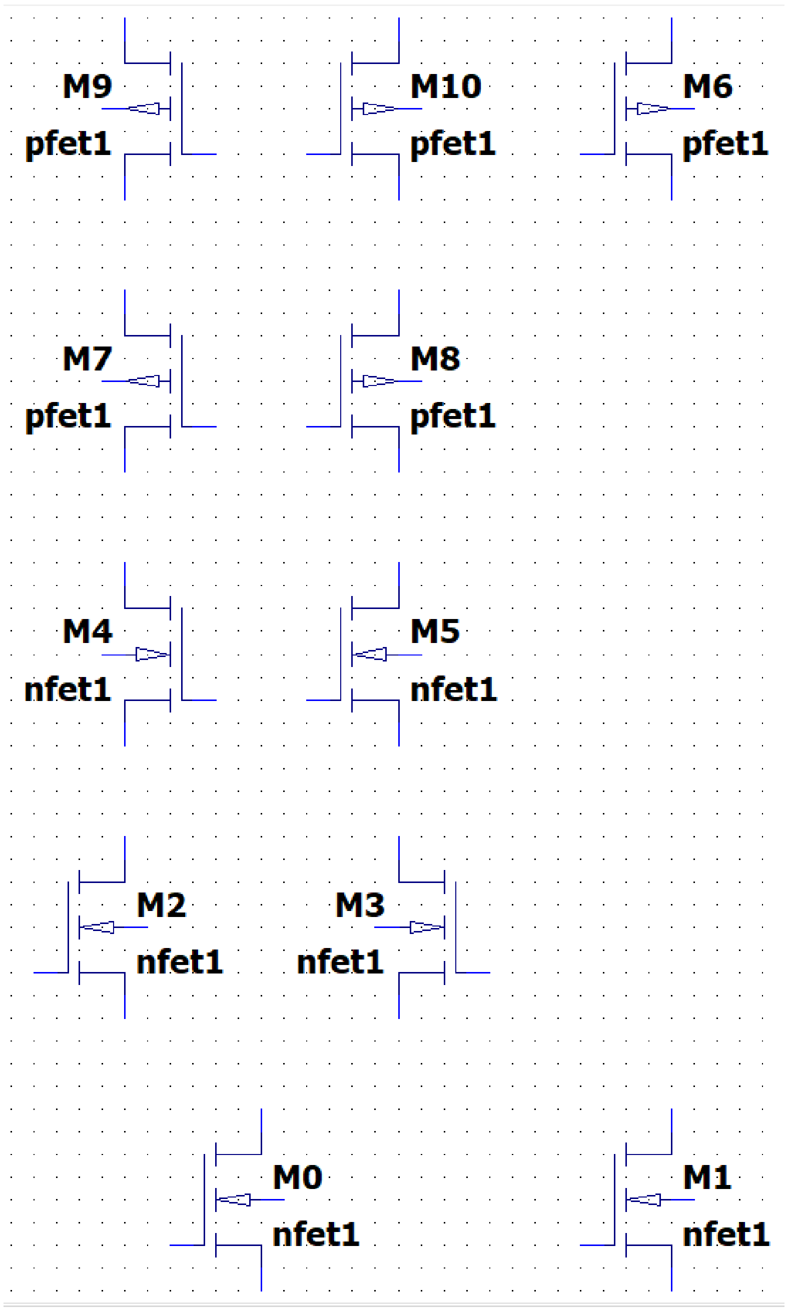


Figure 6: Schematic after wire extension

Then, we can start routing, the process has several steps:

- **For all points in a net, append it to a list**:
  There are several shortcuts I followed while determining whether a point should be appended or not. First of all, input nets are not routed. Most of the time inputs are connected to only one terminal. For inputs with more than one endpoint, flags are used instead of connections so that there is no functional error in the schematic. Also, the bulk terminals of the transistor in immediate rows are not routed. A VDD or ground flag is put on them. The routing algorithm can surely route these points, but doing so will not result in good-looking output (Wires will go through transistors).

- **Build a graph using the list**:
  In the list, we hold a set of points with their names and coordinates. The graph's vertices are terminal indices, and edges are the distance between them. As a function, I have used the Euclidean distance function.

- **Build Minimum Spannig Tree (MST)**:
  Now we have a graph, holding the points and distances between them. We just need to calculate MST. To do that, I have used Kruskal's algorithm. Kruskal's algorithm is a greedy algorithm. Its steps are as follows:

  Sort all edges

  Starting from the shortest edge, add the edge to MST, if adding the edge creates a cycle, reject it.

  Keep adding edges until all vertices are added to MST.

- **Draw each wire segment (edges of MST)**:
  Kruskal algorithm gives us a set of coordinate pairs we should connect. I have written a function called add_wire() that receives two points. If these points share x or y dimensions, we directly put a wire that intercepts these points. For other cases, we need a temporary point. We can calculate this point's coordinate using the algorithm below,

Algorithm 1 Wire drawing algorithm

```
if ($p1.y < p2.y$ and $lower$) or ($p1.y > p2.y$ and upper) then
    $Temp.y \leftarrow p2.y$
    $Temp.x \leftarrow p1.x$
else
    $Temp.y \leftarrow p1.y$
    $Temp.x \leftarrow p2.x$
end if
Draw a wire from p1 to Temp, then draw another from temp to p2
```

We also need to determine whether the path should be lower or upper L-shape. I simply choose the upper for transistors connected to V_{DD}, and the bottom for transistors connected to the ground. For everything else, the lower L-shape is chosen. Obviously, this is not optimal for wire length. However, it gives good results for all netlists tested. In Figures 7, 8, 9 & 10, output schematic for Telescopic Two-Stage amplifier, buffer, Five Transistor OTA & NAND2 netlists are shown, respectively.

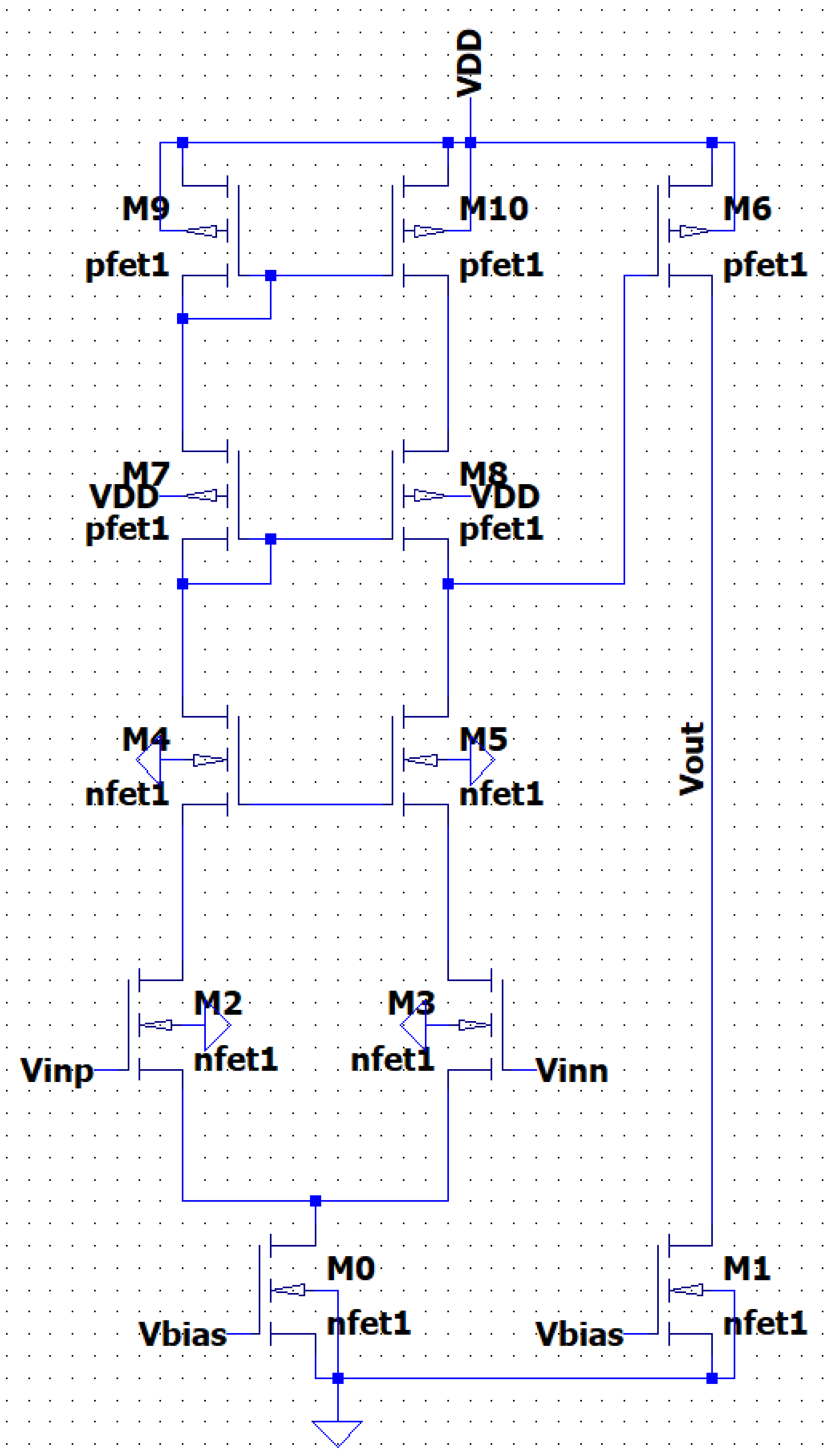


Figure 7: Telescopic Two-Stage amplifier output

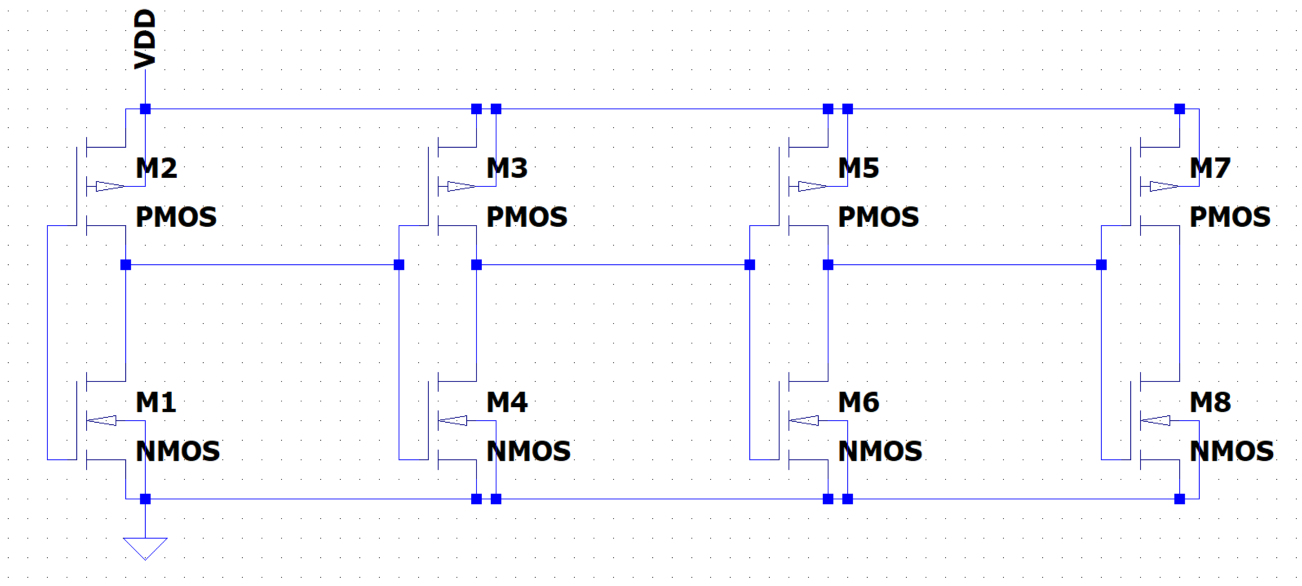


Figure 8: Buffer output

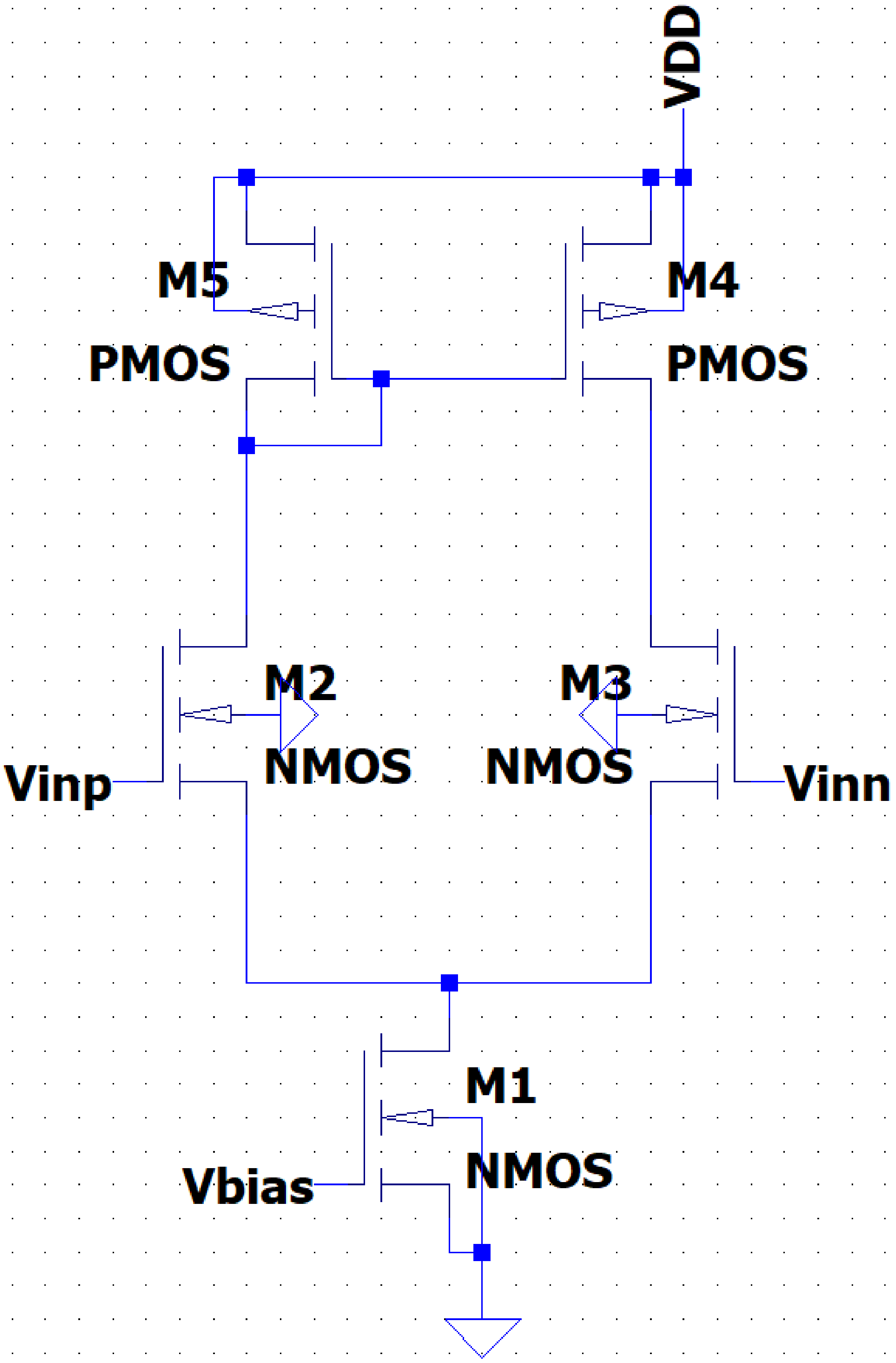


Figure 9: Five transistor OTA output

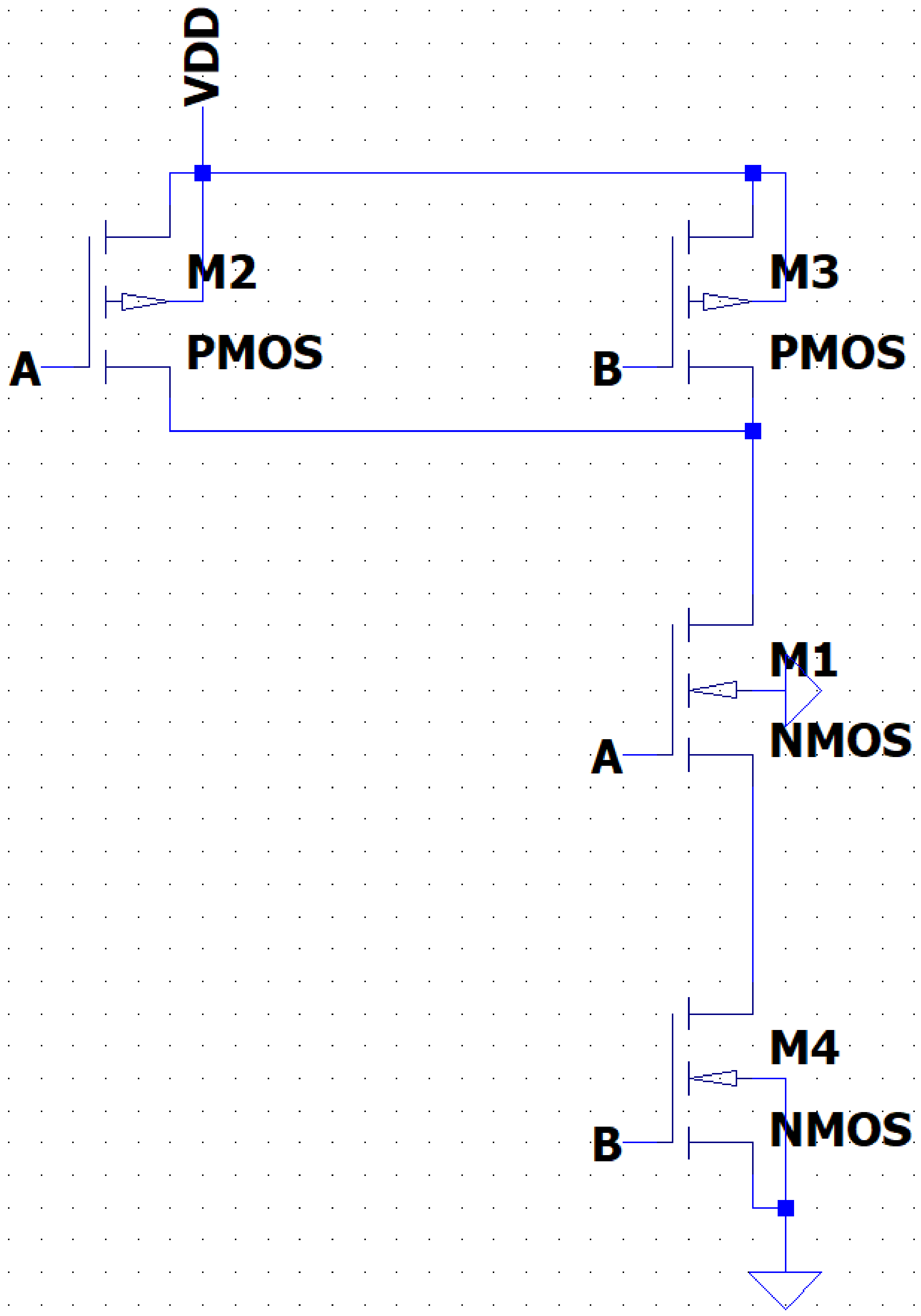


Figure 10: NAND2 gate output