EECE 7353: Spring 2024

FINAL PROJECT

32-Bit ALU

Dr. Yong-Bin Kim

Electrical and Computer Engineering

Northeastern University

Bradley Perritt

Aryan Mehrotra

April 19th, 2024

Implementation Summary:

Aryan:

- Basic logic gates: NOT, OR, AND, XOR
- Repeaters of varying bus sizes
- 32-Bit Carry Select Adder
- 32-Bit Adder/Subtractor
- All the layouts with DRC and LVS passed

Bradley:

- Basic logic gates: NOT, NOR, NAND, XOR
- Repeater with scaling functionality
- 2-1 Mux
- Transmission Gate Block
- Shifters
- Encoder and Decoder
- Top Level Integration of ALU functionality
- Simulations and testing for delay and power

.

Summary of CMOS Implementation:

32-bit Adder/Subtractor:

The adder implemented is the Carry Select Adder (CSA). It consists of three smaller 16-bit CSAs, a 1-bit 2-1 mux, a 16-bit 2-1 mux, and a 16-bit buffer. The initial adder design being considered was the Carry Lookahead Adder (CLA) due to its reduced propagation delay compared to ripple carry adders, and its modular design. However, there were concerns with the area overhead due to the CLA requiring additional specialized logic for look-ahead generation. Additionally, they tend to have larger than normal static and dynamic power consumption due to the large number of transistors. The Ripple Carry Adder wasn't considered viable due to its propagation delay. In the end, the Carry Select was chosen since it provided a good compromise between speed, area efficiency, and delay. Its structure was hierarchical as well, which made it easy to implement. As seen in the schematics below, once the basic logic gates were created, implementing a Full-Adder was straightforward. The 1-bit full adder was scaled up to 4-bits by adjusting its properties. The three copies of 4-bit adder were used along with two multiplexers and a repeater to create a 8-bit CSA. The 8-bit CSA was used to create the 16-bit version, and the 16-bit version was used to create the 32-bit CSA. This building-block approach makes it easier to troubleshoot any issues, and demonstrates that the adder is easily scalable.

The subtraction operation was implemented by finding the two's complement of input B, and adding it to input A. To find a two's complement, the XOR operation was used along with the Cin. It was decided that all subtraction operations would be performed when the Cin was high. Each bit of input B would be XOR'd with Cin to find the complement, and the Cin would be sent to the adder to act as the "plus one" which would complete the two's complement. The two's complement version of input B would be added to input A thereby completing the subtraction. In order to do the bitwise XOR of input bits, a special unit was created called the Subtractor-XOR. The base version consists of 8 XOR gates which each take 1 bit and XOR them with Cin. The full 32-bit version of the XOR unit for the subtractor consists of 4 of these units which can be seen in the schematic section below.

32 Bit AND:

For the AND operation, the design was the standard CMOS implementation due to its simplicity and speed.

32 Bit OR:

For the OR operation, the design was the standard CMOS implementation due to its simplicity and speed.

32 Bit XOR:

The XOR gate was implemented using the standard inverter, and a pair of transmission gates. While the CMOS implementation of XOR was considered, it was eventually disregarded

due to its higher power consumption, overall area and delay. The transmission gate XOR minimized the transistor count and gave consistently low propagation delay. Since the XOR gate was heavily used in the Adder/Subtractor and it was a primary ALU operation, favoring a design with the mentioned properties was necessary.

Logical Shifter:

The basis of the logical shifter is the transmission gate, which allows us to pass through a strong 0 or 1 value with rail-to-rail swing while utilizing the absolute minimal number of transistors. The shifter is composed of a large number of 2 to 1 multiplexers built using these gates, enabling us to use a total of only 6 transistors per multiplexer. In addition to reduction of the size lending to more area efficiency, transmission gates offer lower power dissipation due to the lack of short-circuit paths. Couple this with the fact that they offer faster speeds than traditional cmos for the purpose of shifting data and this was an obvious choice for a shifter.

The multiplexers are cascaded down in a total of 6 stages for each shifter, allowing us to define different shift offsets for each bit of input B. For each bit that is high in B one of the stages will be turned on, causing the input A to be shifted by some exponent of 2 matching the position of the bit in B. This is how one defines a shift in the fewest possible number of stages, and is the hallmark of the barrel shifter as the most efficient shifter.

One weakness of the transmission gate is that the delay can become severe as they are chained together, proportional to n^2 where n is the number of transmission gates in the chain. To combat this we utilize repeaters as buffers to break up the chain, which effectively limits the number of consecutive transmission gates to one that will not cause the delay to rapidly balloon. We found that the optimal number of gates to include between each buffer (m_{opt}) was 3. However, despite the fact that we had to insert an even number of buffers, we could not use only a single inverter for each buffer stage, and had to opt to use a repeater instead. This is because the barrel shifter can introduce new 0s to the pipeline at any stage where a shift occurs. If this happens after the first inverter, a second, lone inverter at the end could cause one of those 0s to be inverted only once and thus incorrectly. Because each stage was driving a similar load and there was no fan out, we opted for a simple scaling of S=1 for both inverters in the repeater stages. Although we experimented with various scaling factors, we found 1 simply worked best. Our implementations for the repeater and inverter include a pParam that allows us to easily define inverter scaling factors so that larger capacitive loads can be driven if necessary.

Top Level:

The implementation of the top level contained some surprising complexity, contrary to what more introductory level courses might lead one to believe about the process of "simply connecting the logical units together". In practice, it wasn't as simple. First, we did simply need to lay out all the logical units for the operations we wanted to perform. Then, we needed a way to select an operation from one of the seven available, and we chose a decoder in order to perform the job. For the decoder, we went with a relatively standard inverter into 3 input AND gate implementation for a few reasons; the load of the 3 input AND gate would be relatively easy to drive as the signal was coming directly from the input, and the inverter at the end of the AND would act as a buffer to boost the signal before it was used as an enable. This is important because each enable signal drives the input of 3 other buffers throughout the circuit, meaning that a strong signal capable of quickly charging the gate capacitances is important.

The other, and much more significant challenge of the top level, was figuring out how to handle the portions of the pipeline that were going to remain unused. While we could have simply transmitted each bit to every one of the seven operations in the ALU, this not only would have led to a larger delay due to the fan in, but also a significantly higher power draw due to enabling gates for operations that were going to remain unused. The output was an even more challenging problem. The ALU had to drive a final load of 20 fF, which would take substantial time to charge from the output of a normal gate, but we realized there was a problem with using repeaters: if the outputs were all connected into a single channel, the disabled repeaters would short the output to ground, destroying the integrity of the valid signal. Thus, we needed some way of preventing the other unused operations from interfering with the input and output, while driving a substantial enough load to charge the output capacitor. We settled on tri-state buffers, in particular the NAND and NOR gate implementation that ends in an inverter. This is a unique buffer that has a near infinite output impedance when not enabled, meaning that the signal would only travel down desired paths at both the input and output. In addition to this, the fact that it resolves in an inverter structure means it's well equipped to drive an output load. These buffers ended up being the perfect solution to balancing our high impedance and signal driving needs.

Transistor Sizing:

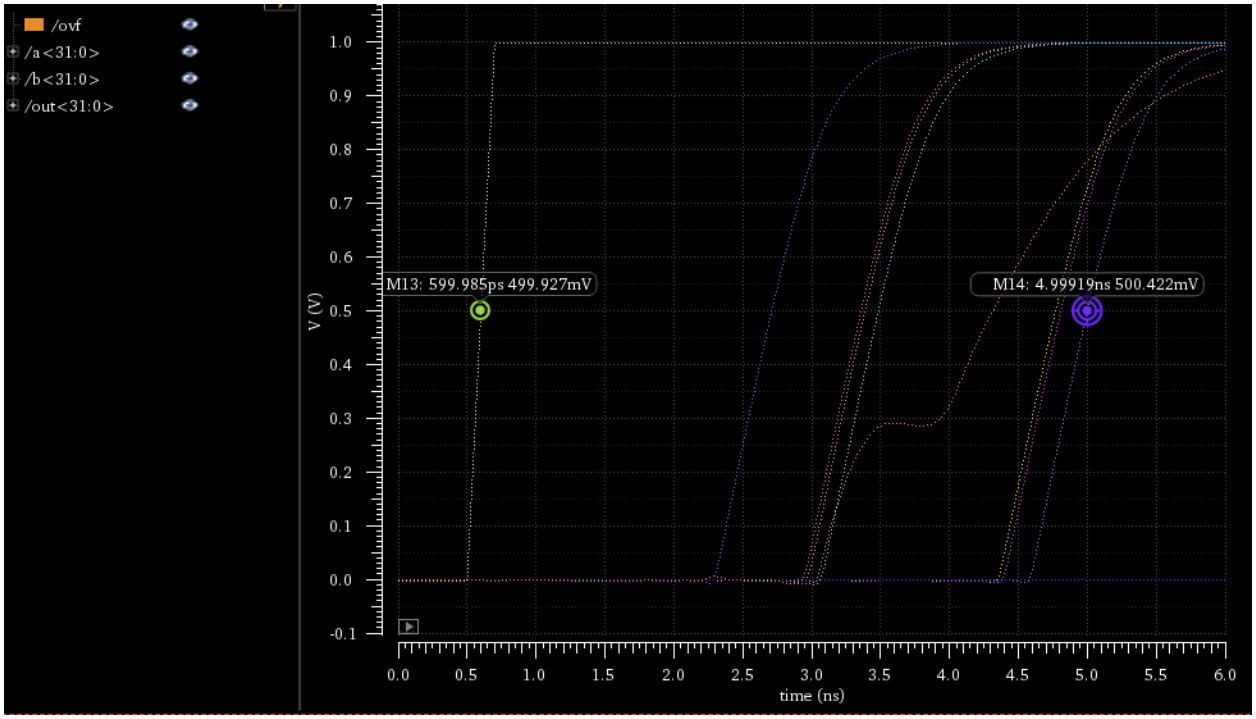
The transistor sizes used were slightly different depending on the component being implemented. The length of the transistor stayed constant at 180 nm. However, the width was adjusted depending on if it was a PMOS or NMOS device. We aimed for a W/L ratio of 2 in our NMOS devices for a good balance of speed and power dissipation (the width was rounded up to 400nm by Cadence), and a W/L ratio of 5 for PMOS to account for the lower hole mobility compared to NMOS. In instances like the AND gate, we increased the W/L of the series-connected NMOS transistors to reduce their equivalent resistance and enhance their drive capability, while for the OR gate we did the same for PMOS.

Simulation Results:

For our simulations, we tested the propagation delay and power draw of the entire datapath from decoding to the output for each operation to most realistically capture the performance of our circuit. Thus, all of our simulations include decoding, buffering, operations themselves, and any parasitic elements introduced by the surrounding structure or operations as part of the simulation.

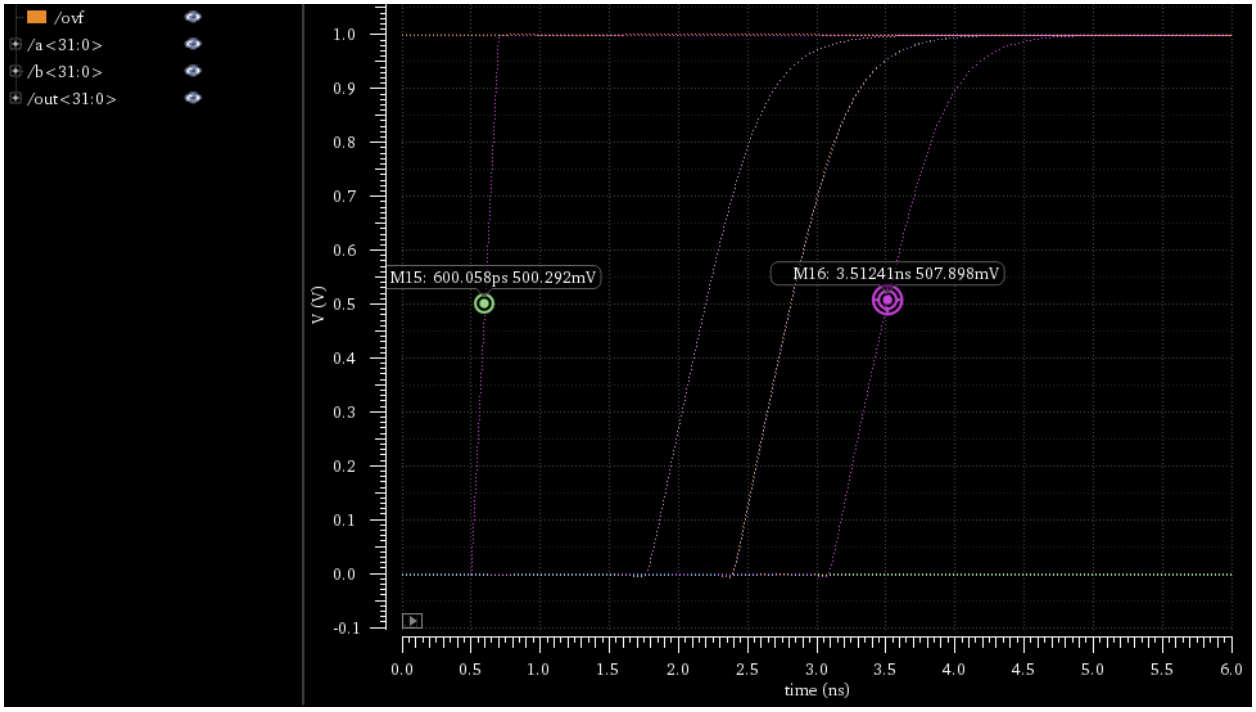
Delay Tests:

32 Bit Adder



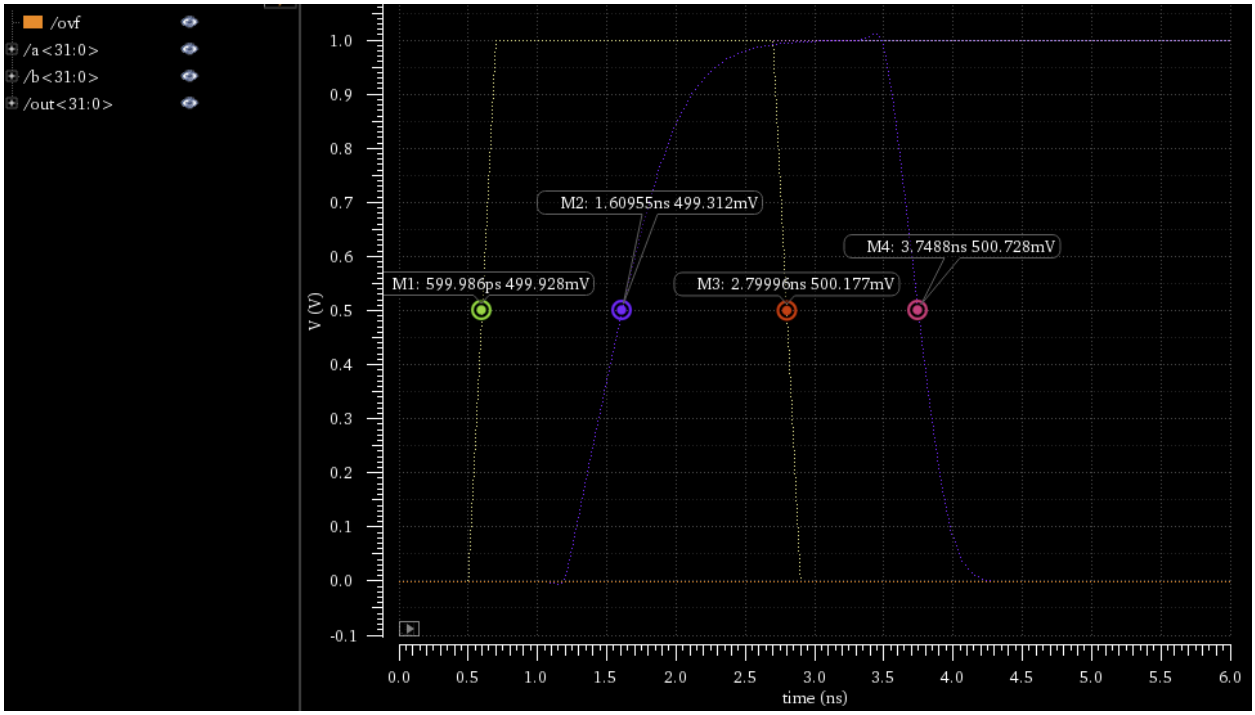
This is the delay result for the addition of a full set of 1s with another full set of 1s, resulting in the maximum length datapath and the overflow bit being set high. With a total delay of 4.4 ns, this represents the critical path of our ALU and the longest propagation delay in our circuit.

32 Bit Subtractor



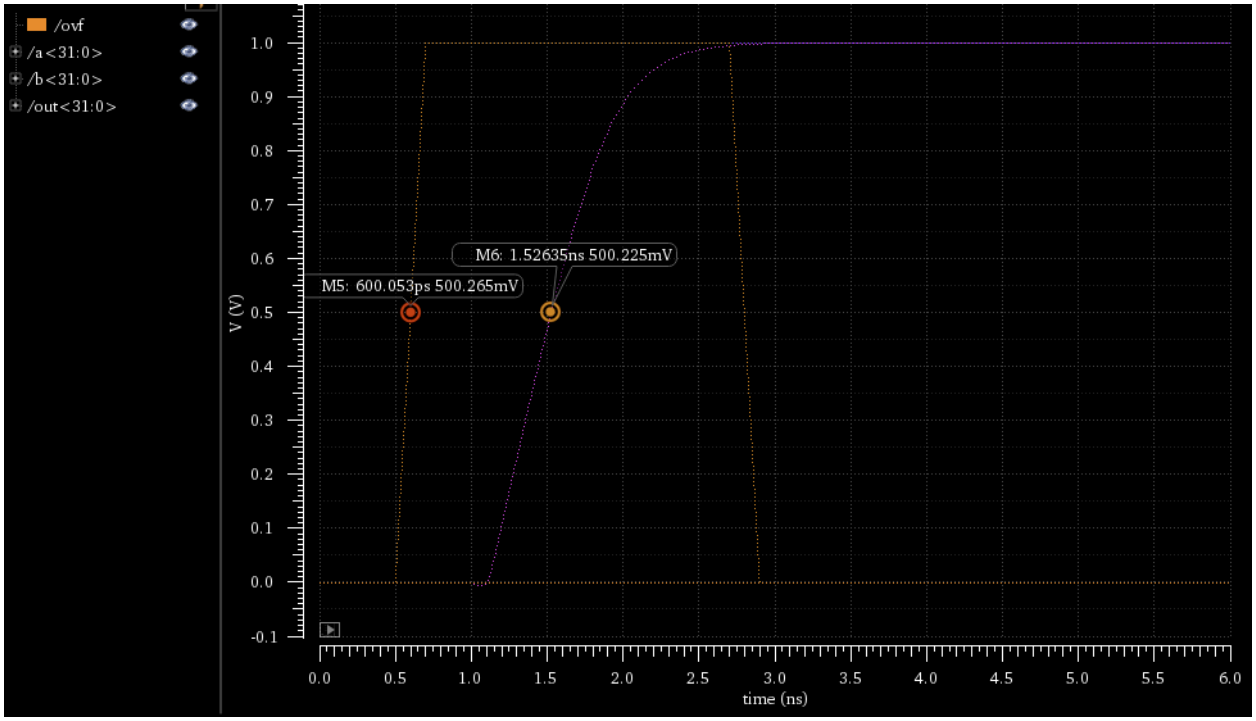
The subtractor has a delay of only 2.9 ns when subtracting an input of half ones and half zeroes from an input of all ones, likely due to the reduced number of inputs that must swing high compared to with an adder.

32 Bit AND



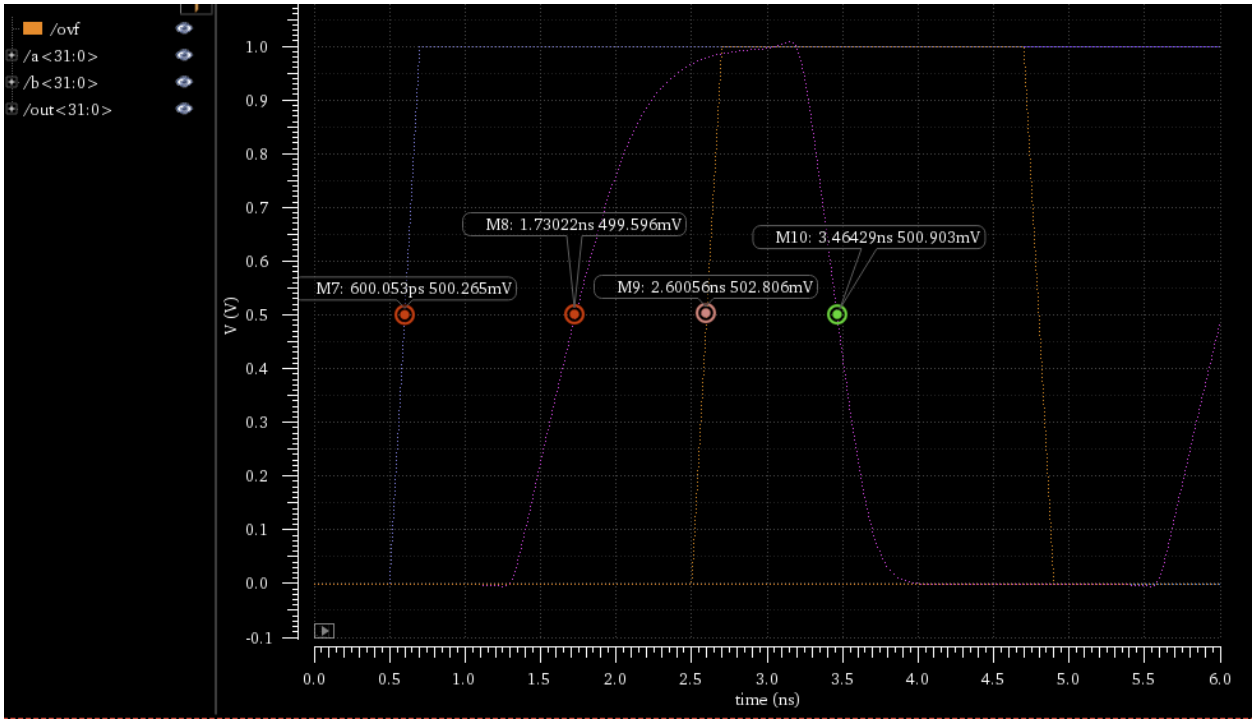
This has a relatively low propagation delay of 1 ns even for every input swinging high. We also tested the high to low rail switching speed for the case where an input goes from 1->0, which gives an even lower delay of 950 ps.

32 Bit OR



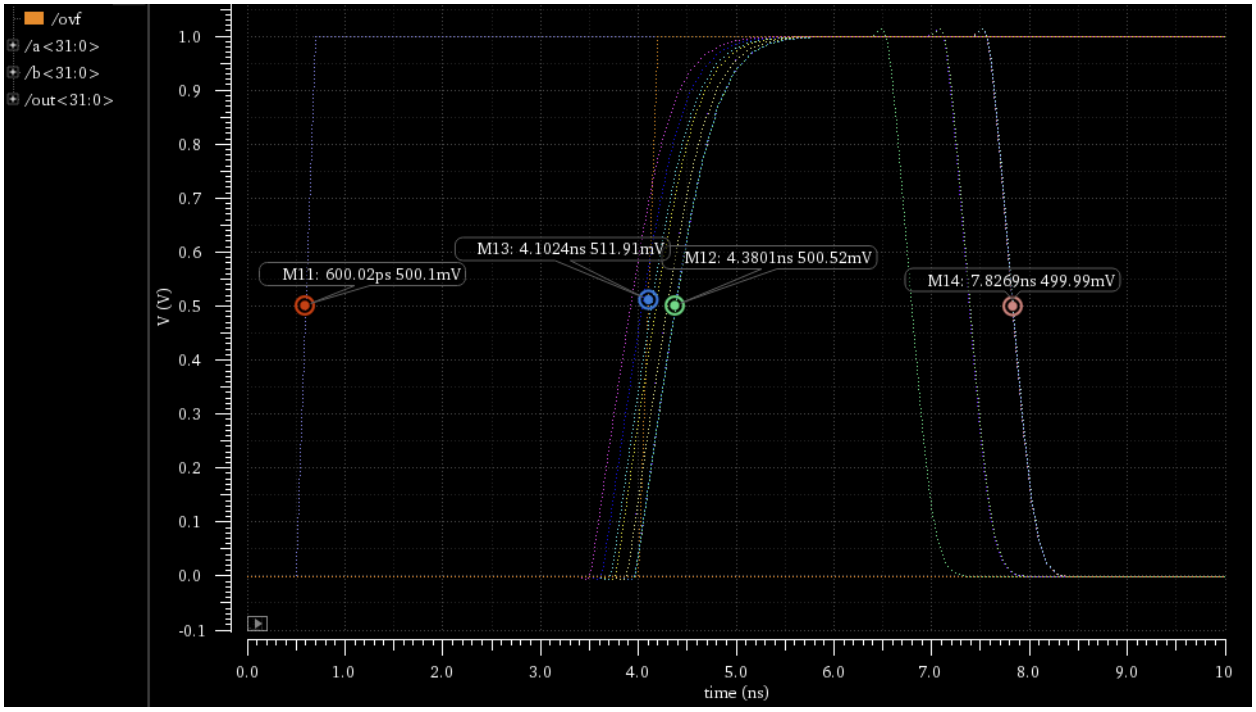
A similar situation to the AND, with a rise time of 920 ps

32 Bit XOR



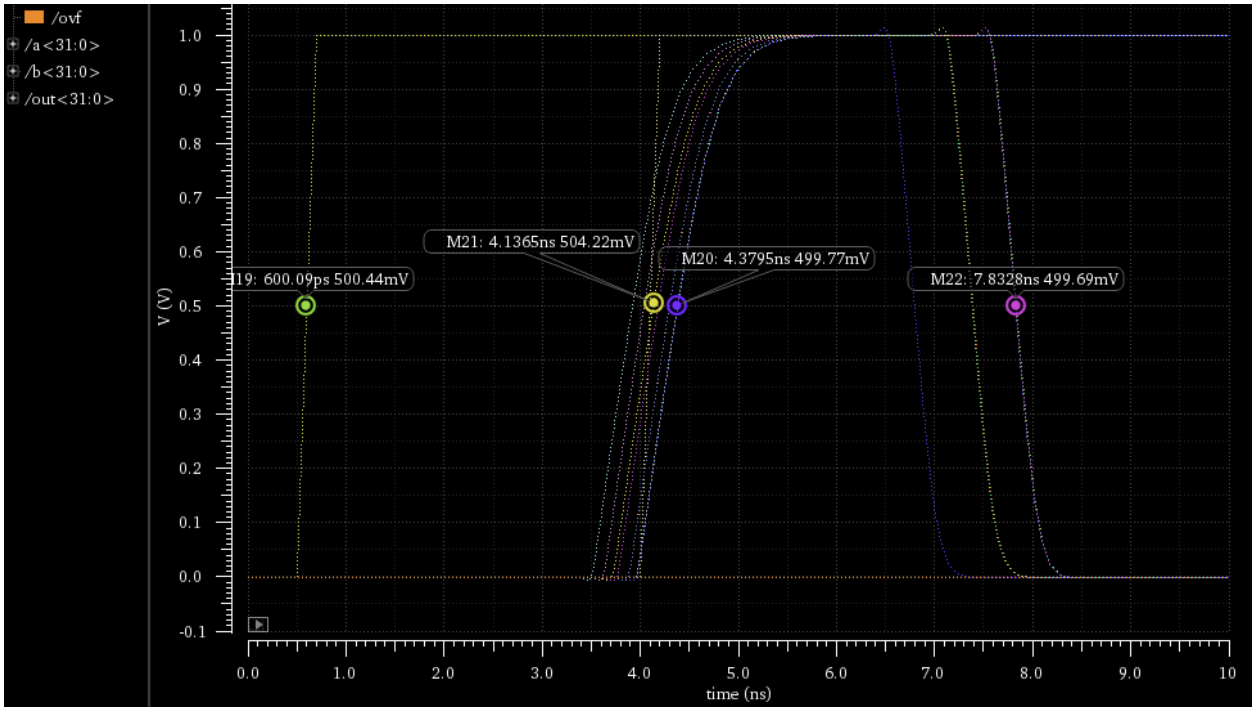
Slightly longer than AND and OR but still in the same ballpark, with a slightly better fall time of 860 ps.

Logical Shift Right



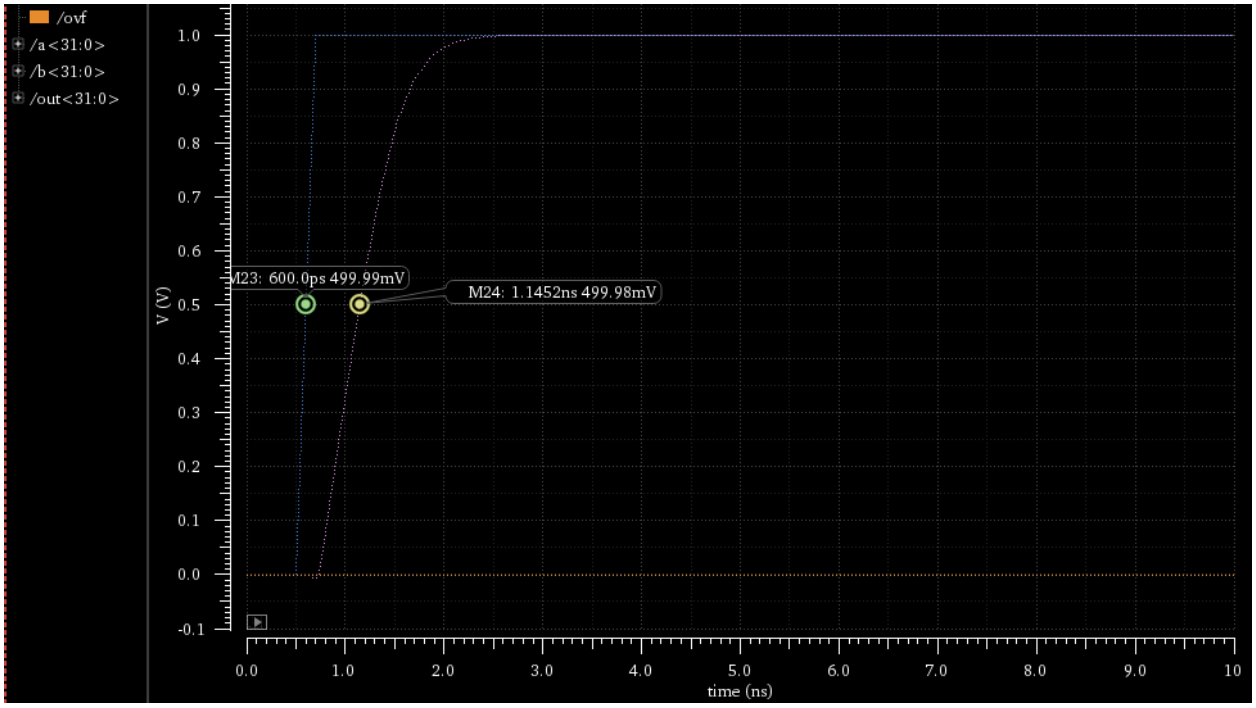
Closest to the adder in terms of propagation delay, which is expected as it operates on 6 stages of pass transistor multiplexers. Ironically, the highest delay path is one where no shift happens at all, with a delay of 3.78 ns. If a shift is triggered, even while the input is currently in the process of rising, it still takes less time to swing back down at only 3.7 ns.

Logical Shift Left



Same situation as the right shift, just with the order of the inputs and outputs reversed.

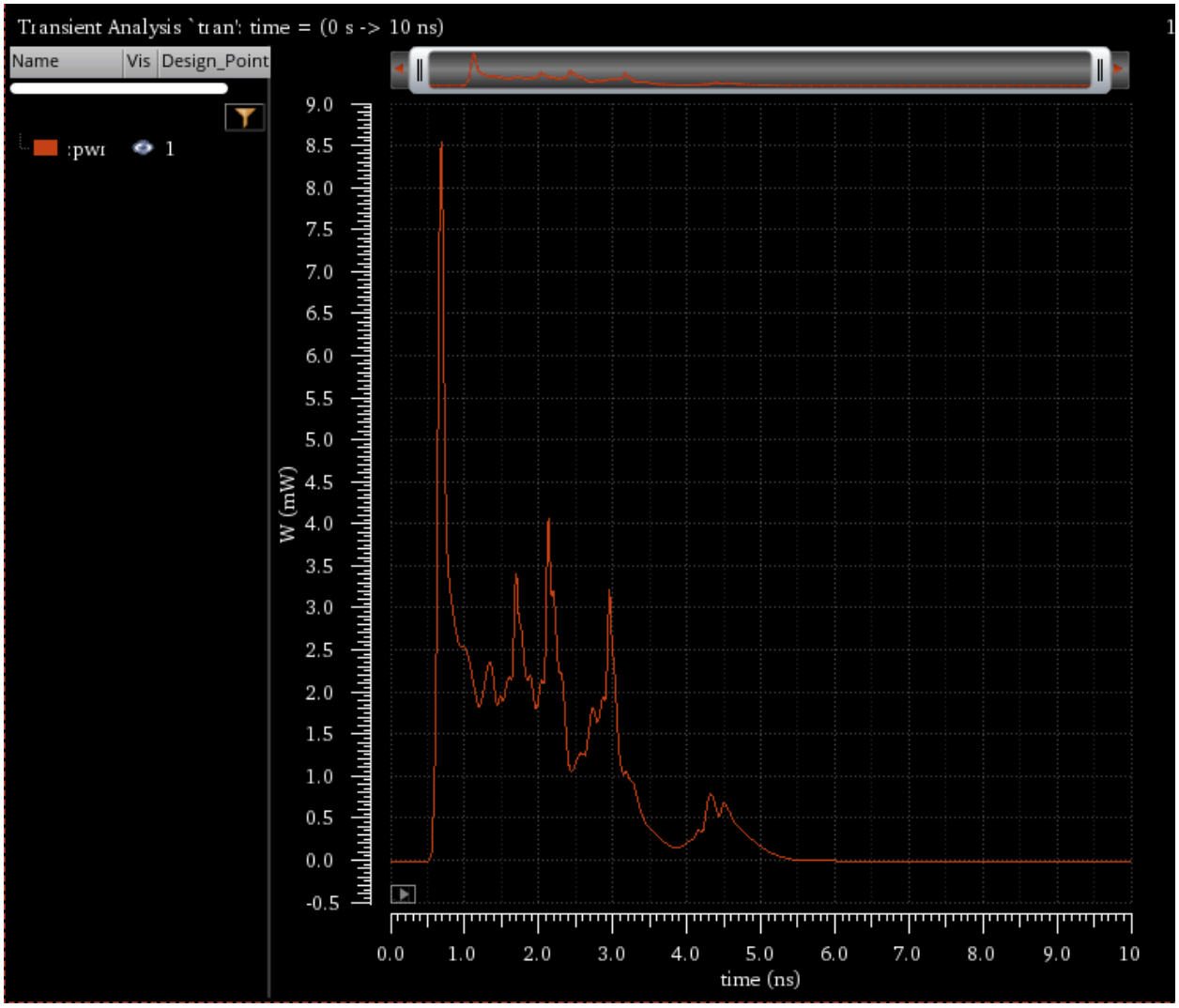
Passthrough



This represents the delay for sending a signal unchanged through the pipeline without applying any operation. The delay is 550 ps, which represents the “base delay” of the ALU.

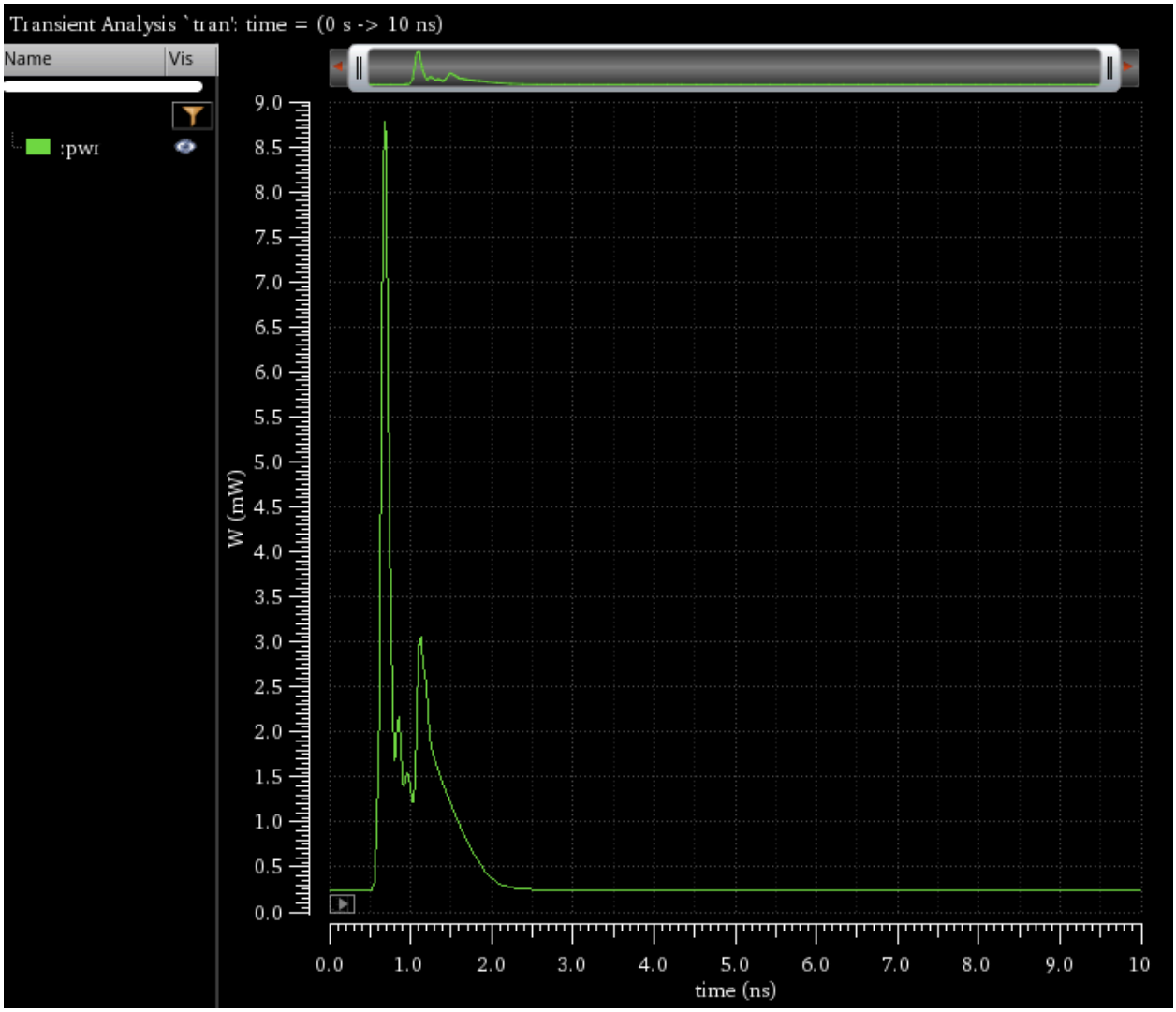
Power measurements:

Adder/Subtractor Pipeline



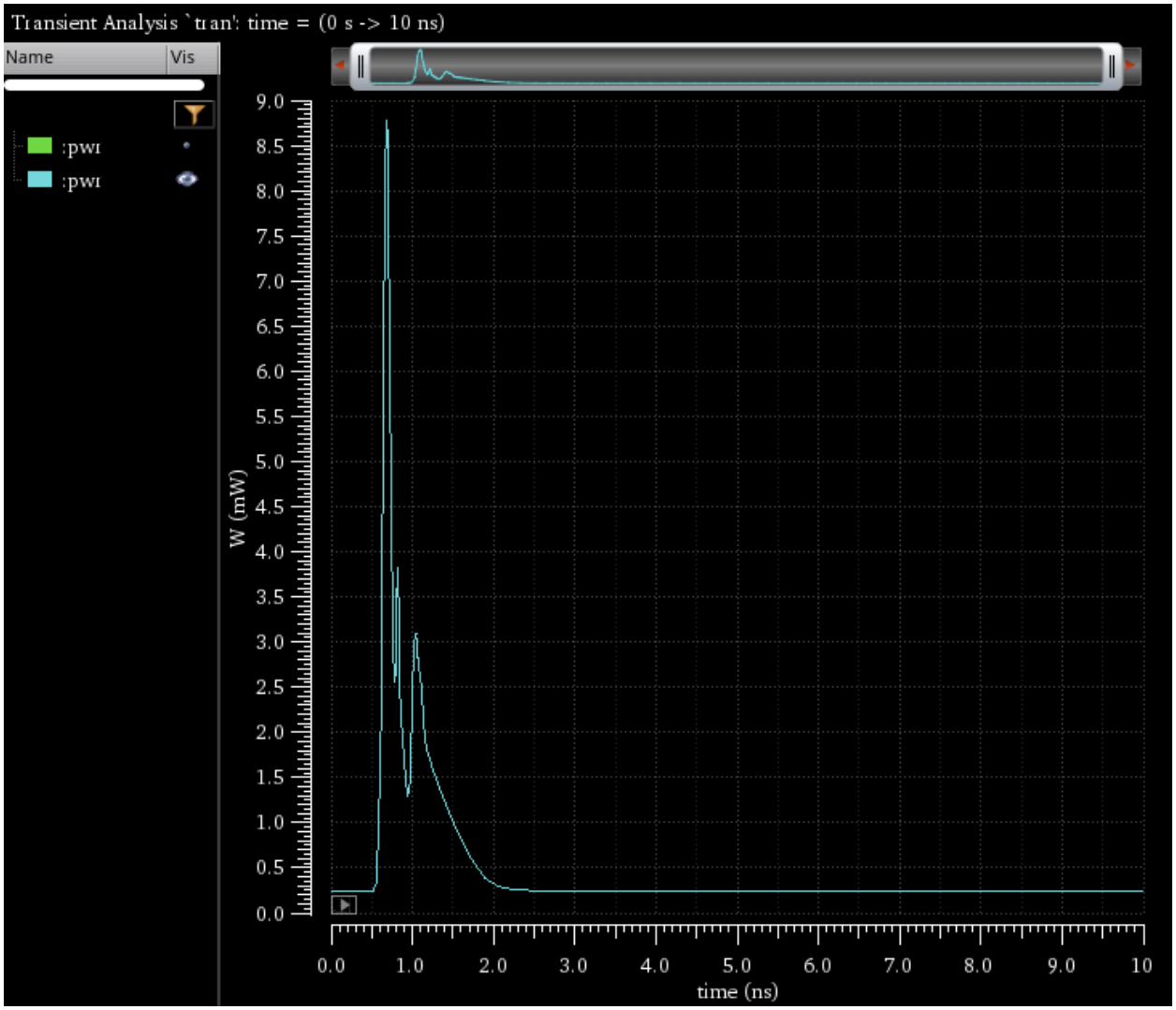
Relatively high in power with more prolonged use due to the length of the operation.

32 Bit AND Pipeline



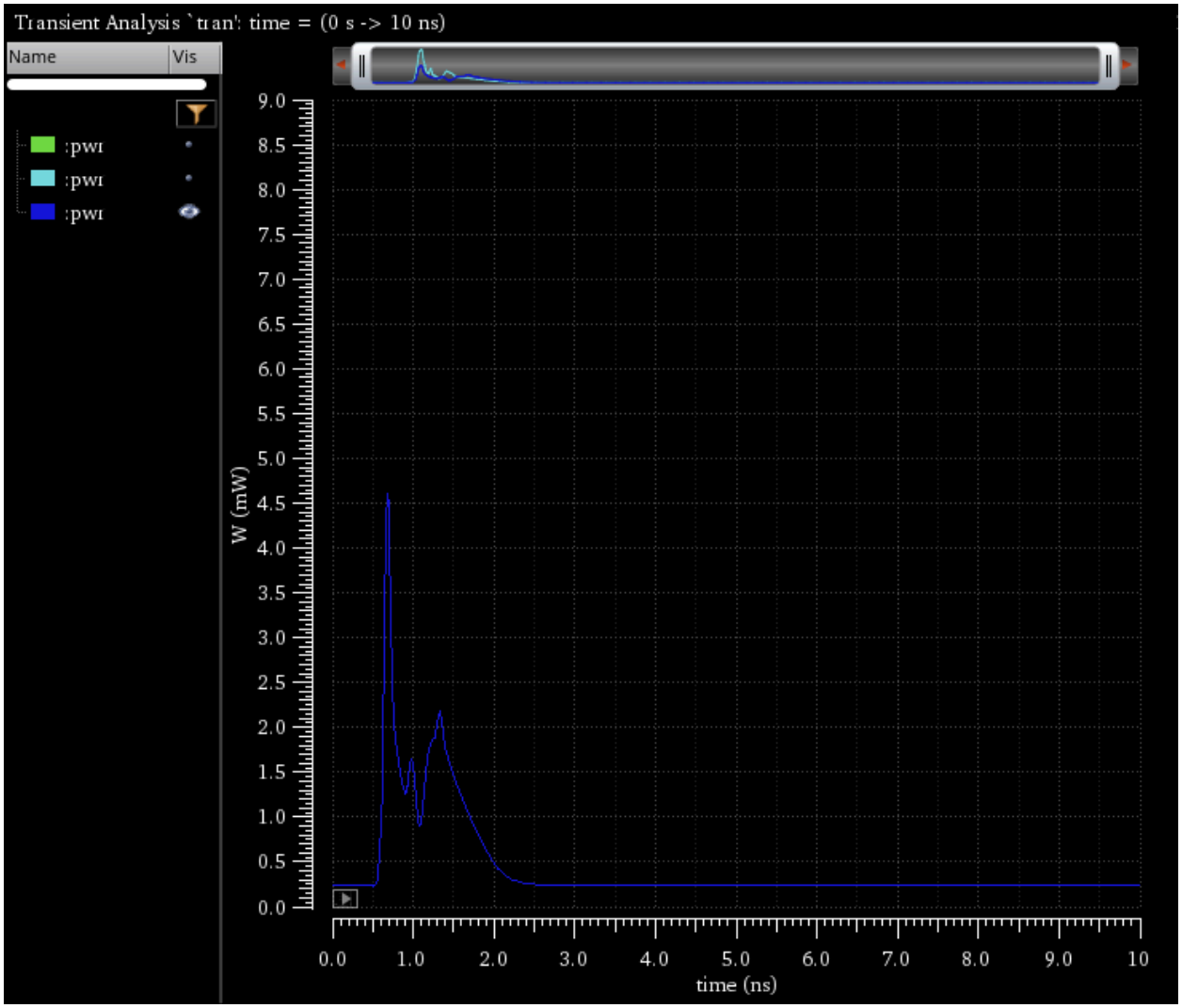
Quick spike in power, similar maximum to adder but less overall usage.

32 Bit OR Pipeline



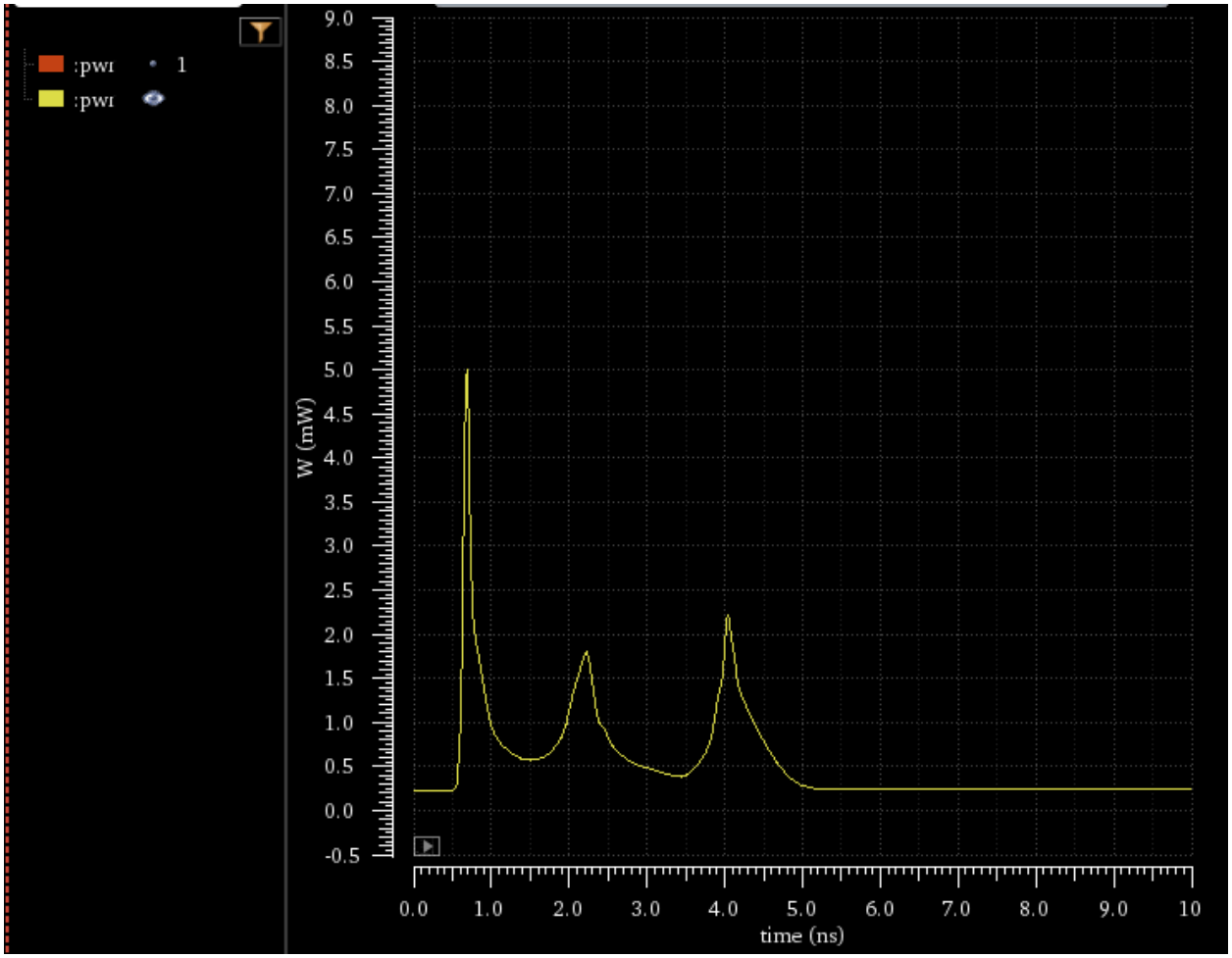
Similar case to AND.

32 Bit XOR Pipeline



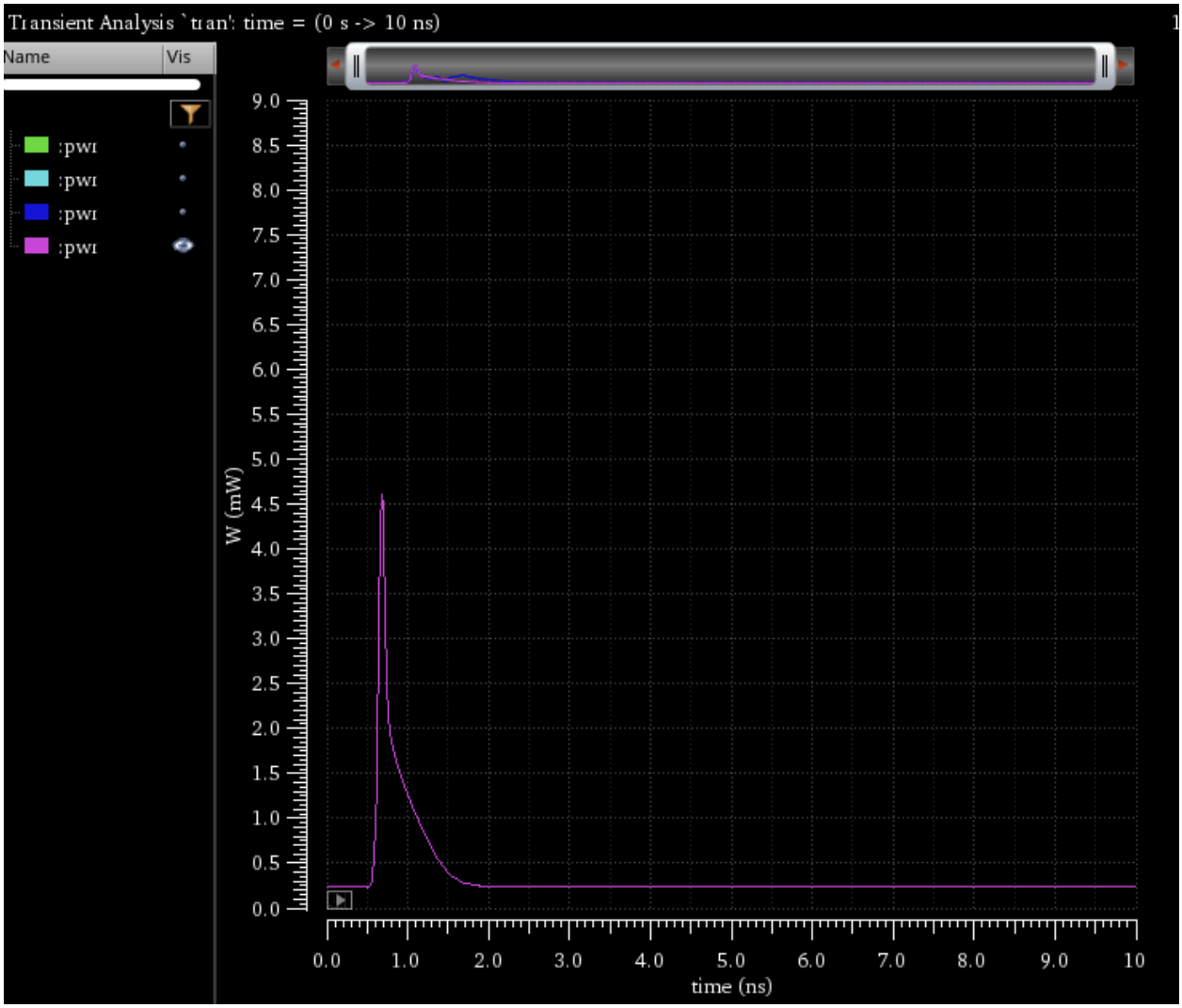
Substantially less power consumption due to transmission gate implementation.

Shifter pipeline



Despite the more prolonged usage, low average power draw due to transmission gate implementation.

Passthrough



Short, low power draw.

Technical Difficulties with Cadence:

- The most egregious and difficult problem we faced through this project was with the Globalprotect VPN. Globalprotect had severe connectivity issues and would drop out unexpectedly, sometimes multiple times within an hour and sometimes as frequently as every 10 minutes, for periods as long as 20 minutes. This led to work being lost several times, which severely affected productivity. In addition to any lost work, there was also the overhead of having to re-instantiate the entire working environment, only for it to sometimes crash immediately again in an extremely frustrating manner. Additionally, the crashes locked our schematics because of the cdslck, which we didn't know how to remove until well into the project (please at least provide a tutorial on this).
- It was challenging to share libraries with each other and collaborate designing larger components. As far as we could tell, there was no swap space on the Northeastern lab machine. We had to scp the files to our own PC, zip them, and send them to each other through some other means. Being able to simply share a working environment would be far more ideal.
- There were multiple instances where we got extremely weird simulation results because of wires that were named the same thing as each other. Even if you specifically select a given wire as the output, if there's another wire with the same name in the schematic it can take precedence in the output of the simulation, making it appear as if the circuit is broken. Cadence provides no warning for this naming overlap.
- The component and wire grid snapping would be randomly disabled not due to any input from the user (as far as we could tell) and the only way to restore it was by restarting Cadence.
- Undo is unreliable, and nothing that is done can be undone once saved. However, it's a requirement to save in order to simulate or try anything new.
- As far as we could tell there's no way to copy paste across different schematics, and the copy paste within a schematic is clunky and often messes up wires.
- It was surprisingly difficult to set up custom pParams to define things like modular scaling factors and have them bubble up through the design. For example, to use a pParam as a field for another pParam, it must accept a string as the input for its field rather than the type of value that would replace the string.
- Most of Cadence's support forums/articles were locked and required an account for access, which made any kind of troubleshooting very difficult with the already limited resources and support online.

Differences between expectations and simulation:

- We expected that it would be much easier to select different operations for the ALU, and disable the ones that weren't being used. Originally, we were using a transmission gate enabler, but we had to change the design as it wasn't powerful enough to drive the output capacitance. When we tried to boost the signal with a repeater, it caused the entire output

to be shorted to the ground of whatever repeaters were off. Thus, we had to research and implement a very particular kind of buffer (tri-state) to get the circuit to work.

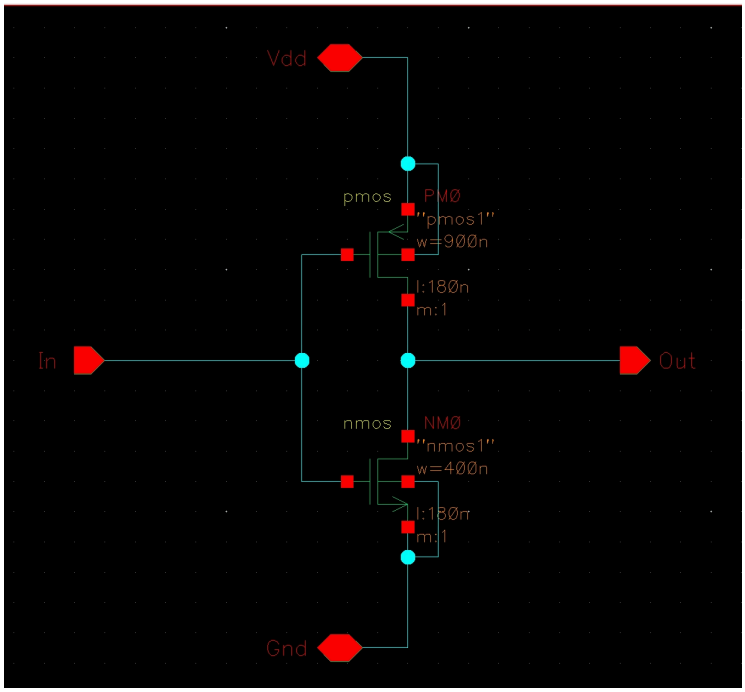
- We were surprised to see that the CMOS AND and OR gates were consuming a similar amount of power to the adder/subtractor. In comparison, XOR and Shifter were much more power efficient.
- We don't use the transistor body connections in homework or class problems, and we sometimes misremembered how they should be connected. Misconnected transistors led to inaccurate results that required time-consuming troubleshooting.
- We expected the shifter to have the highest delay based on the number of stages, resulting in us focusing on optimizing it more than necessary when more time should have been spent optimizing the adder/subtractor.

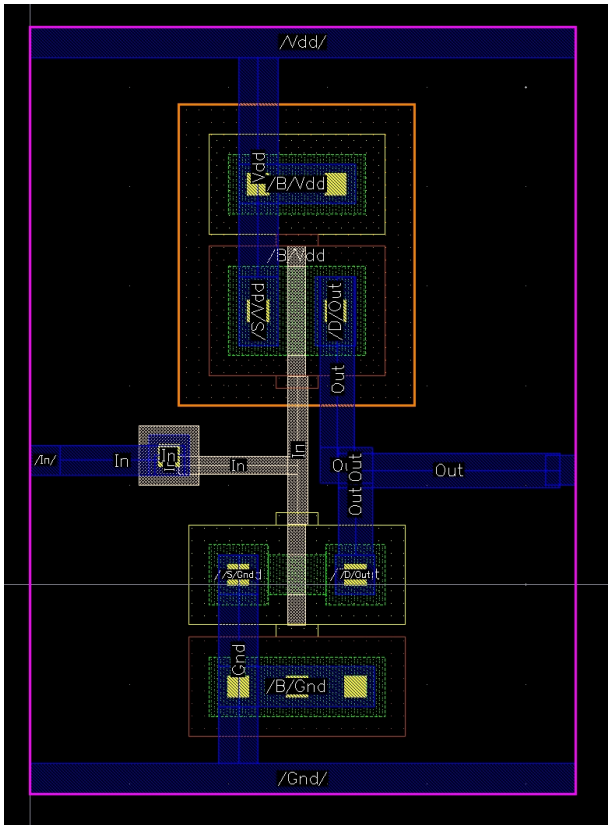
Conclusion:

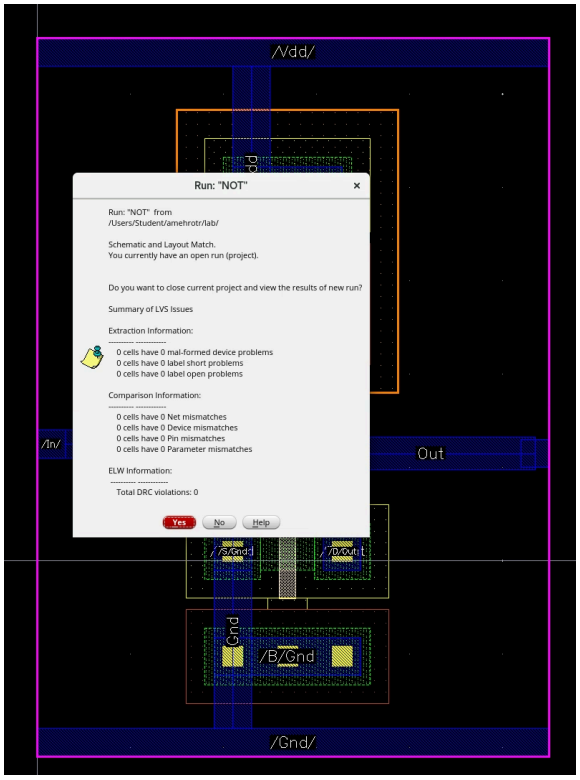
As seen in the simulation results, all ALU operations worked successfully. Despite using standard CMOS logic for implementing the AND and OR logic gates, the power consumption and the propagation delay were in-line with the rest of the system. The transmission gate logic in the XOR gate had very low power consumption and delay. The reduction in the amount of transistors had an observable effect compared to other operations. It was expected that the shifter would have the highest propagation delay due to the fact that it needs to go through 6 stages of multiplexers before reaching the output. However, the adder/subtractor had the highest delay with 4.4 ns for the case of adding when both inputs were high. In terms of power consumption, while the adder/subtractor unit did use close to 9 mW of power, it wasn't an outlier as the AND and OR operations used a similar amount. For next steps, it would be a good idea to test out the 32-bit AND gate and OR gate using either pseudo-nmos or the pass transistor technique to see if it reduces the power consumption and delay. We could also experiment with structural or architectural changes to the adder/subtractor unit to reduce the delay of the critical path, ideally below that of the shifter.

Schematic and Layouts:

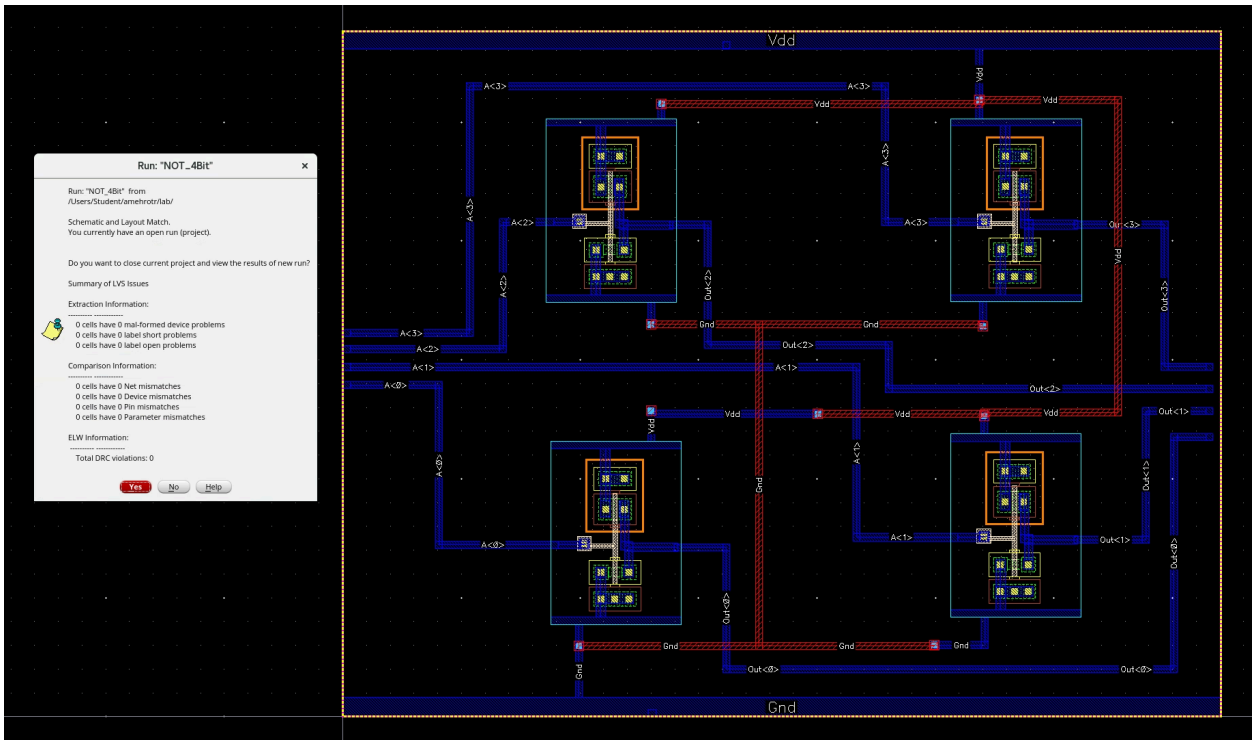
NOT Gate:



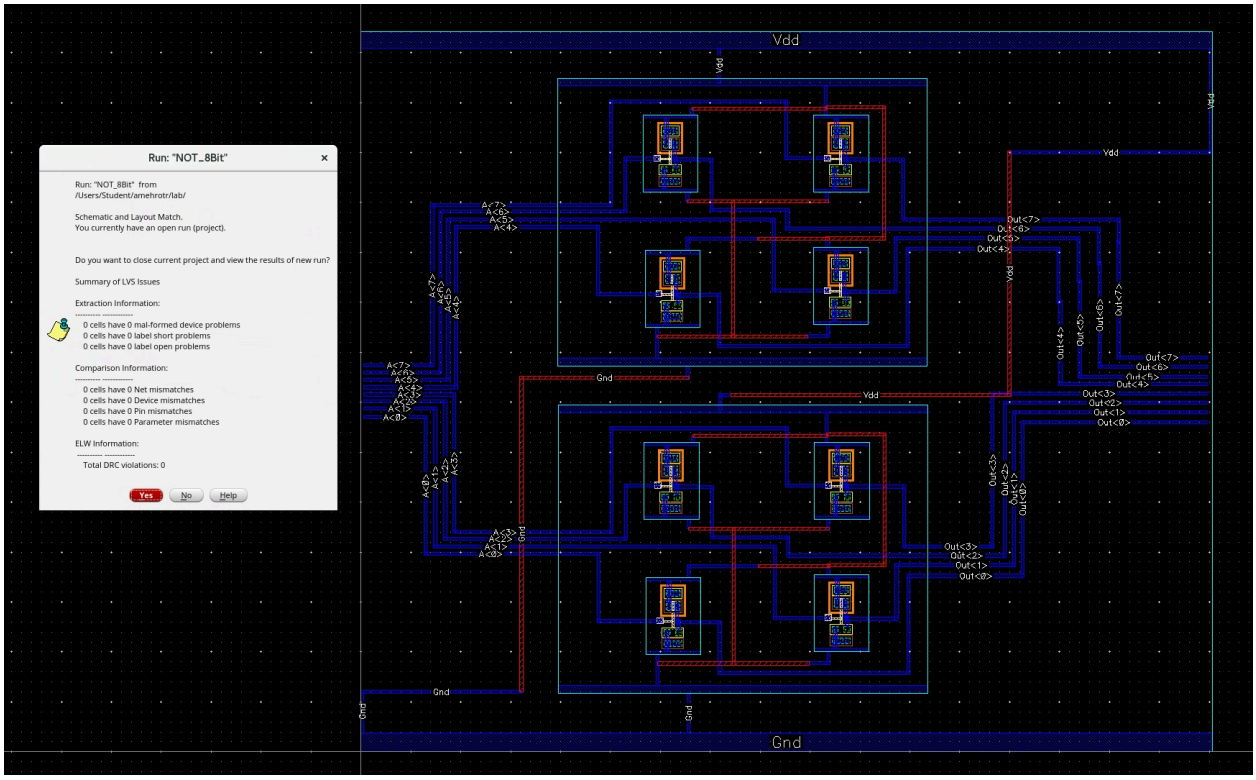




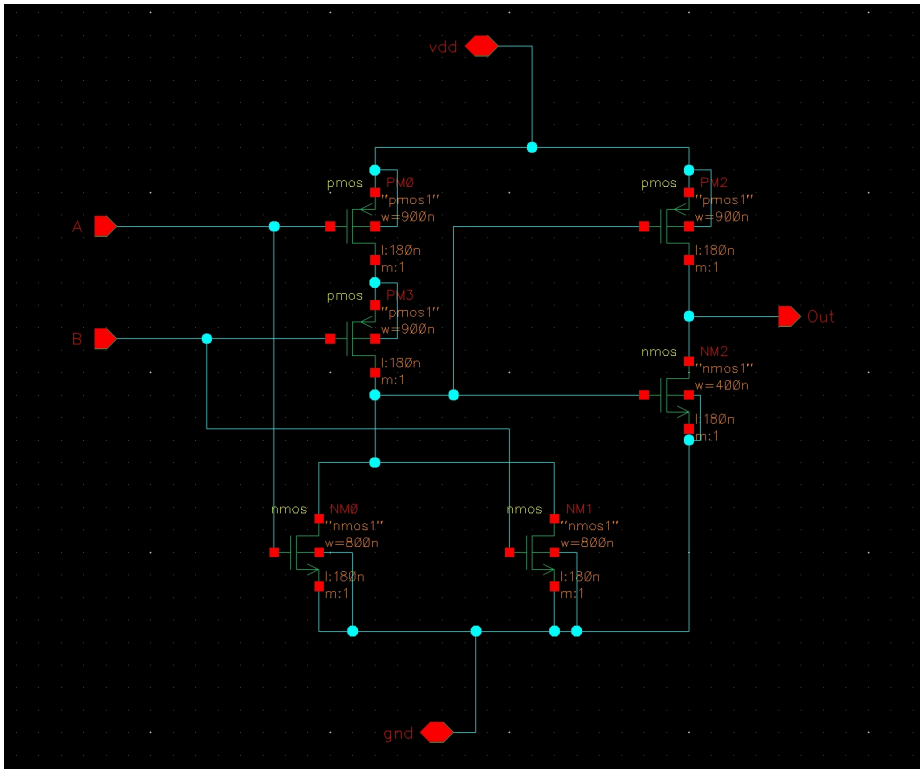
4-Bit NOT Gate Layout:

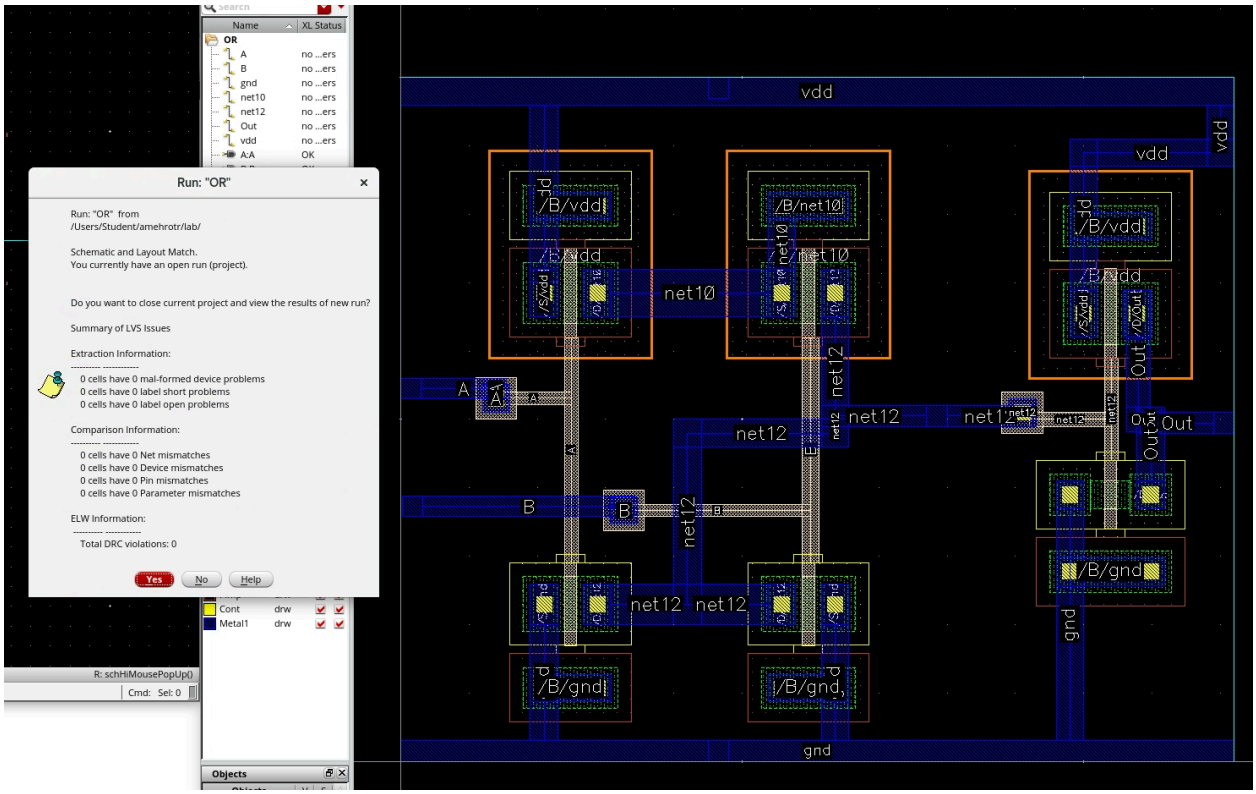


8-Bit NOT Gate Layout:

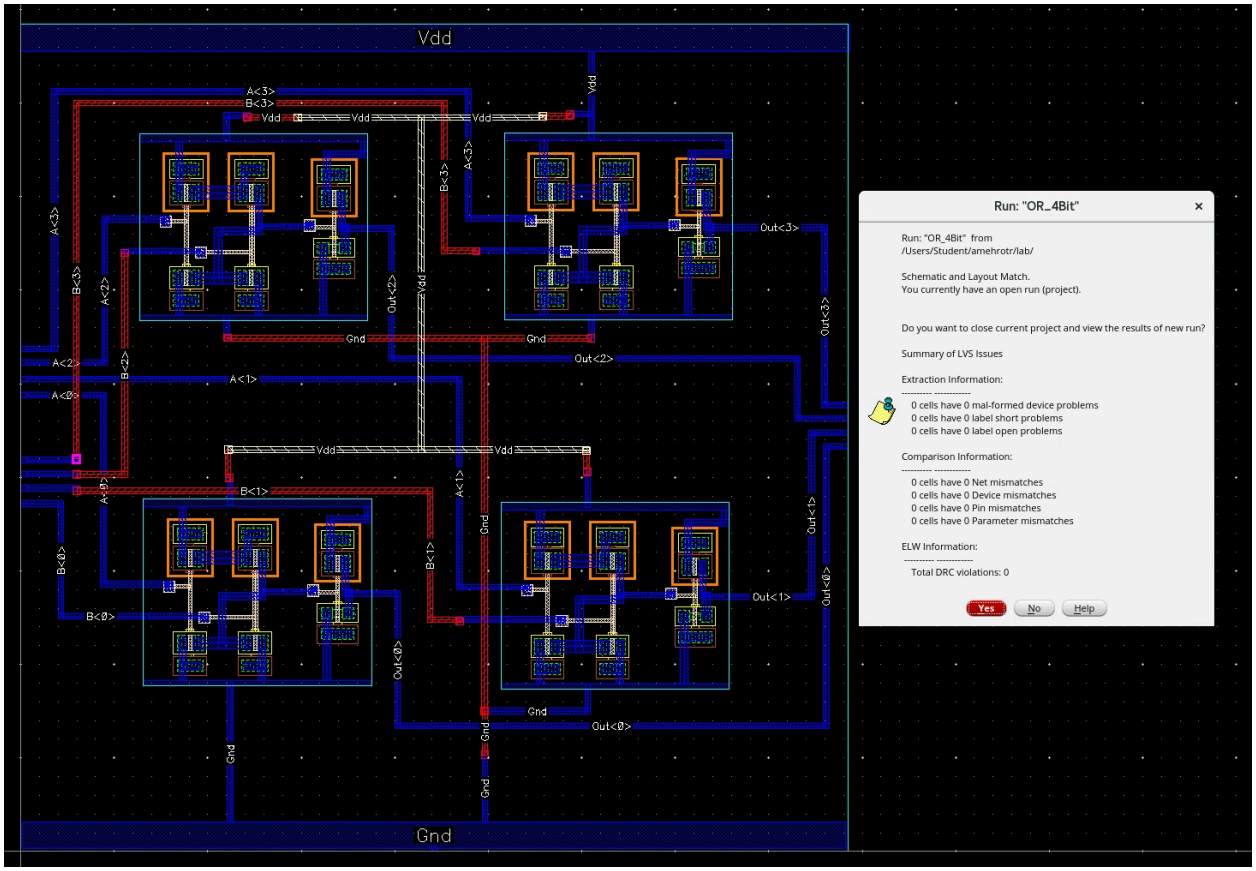


OR Gate:

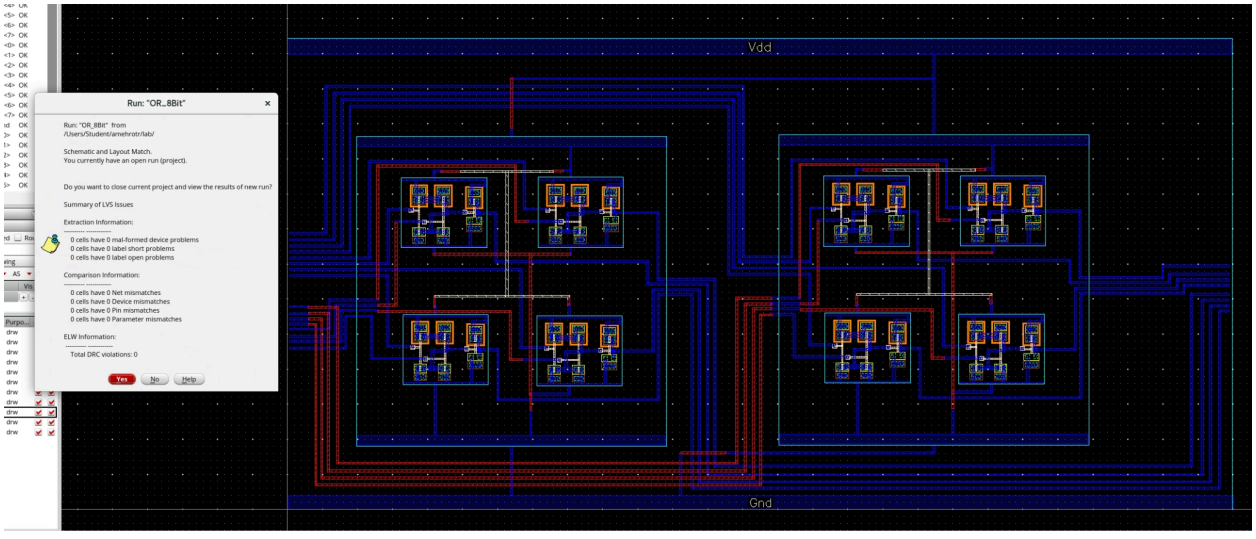

vdd
pmos
PM0
"pmos1"
w=900n
l:180n
m:1
pmos
PM2
"pmos1"
w=900n
l:180n
m:1
A
pmos
PM3
"pmos1"
w=900n
l:180n
m:1
B
Out
nmos
NM2
"nmos1"
w=400n
l:180n
m:1
nmos
NM0
"nmos1"
w=800n
l:180n
m:1
nmos
NM1
"nmos1"
w=800n
l:180n
m:1
gnd


Run: "OR"
Run: "OR" from
/Users/Student/amehrotr/lab/
Schematic and Layout Match.
You currently have an open run (project).
Do you want to close current project and view the results of new run?
Summary of LVS Issues
Extraction Information:
0 cells have 0 mal-formed device problems
0 cells have 0 label short problems
0 cells have 0 label open problems
Comparison Information:
0 cells have 0 Net mismatches
0 cells have 0 Device mismatches
0 cells have 0 Pin mismatches
0 cells have 0 Parameter mismatches
ELW Information:
Total DRC violations: 0
Yes
No
Help
vdd
gnd
net10
net12
A
B
Out

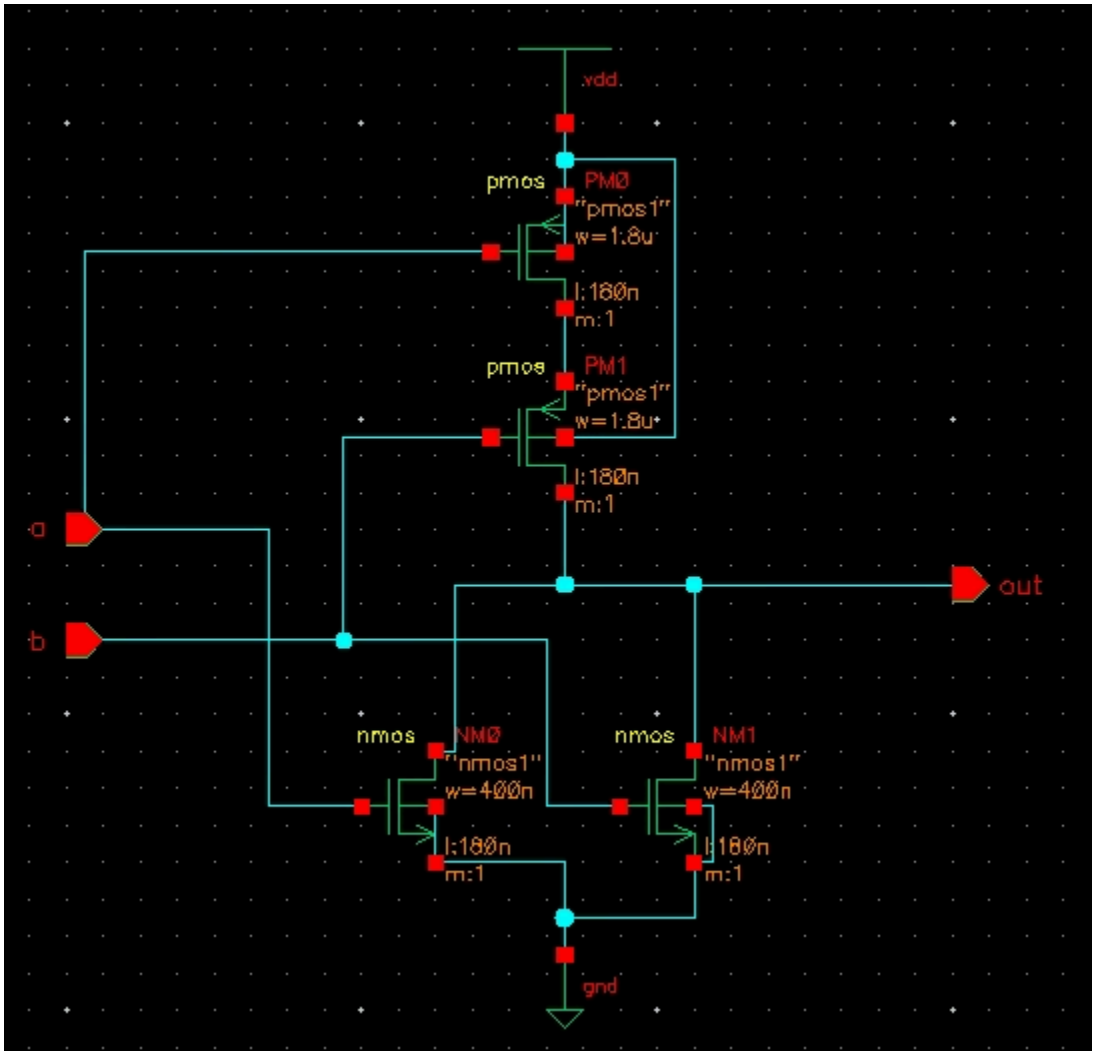
4-Bit OR Gate Layout:

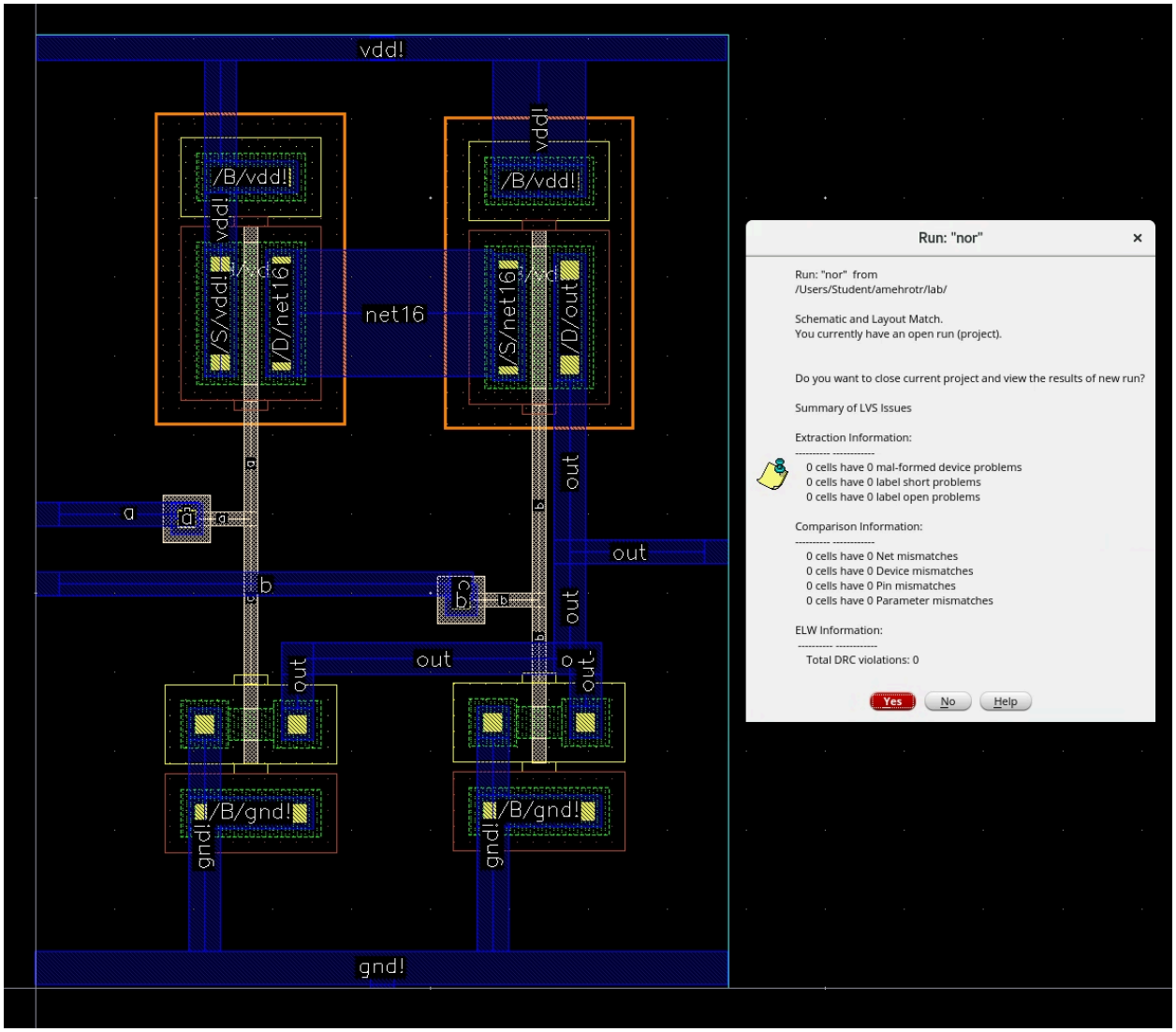


8-Bit OR Gate Layout:

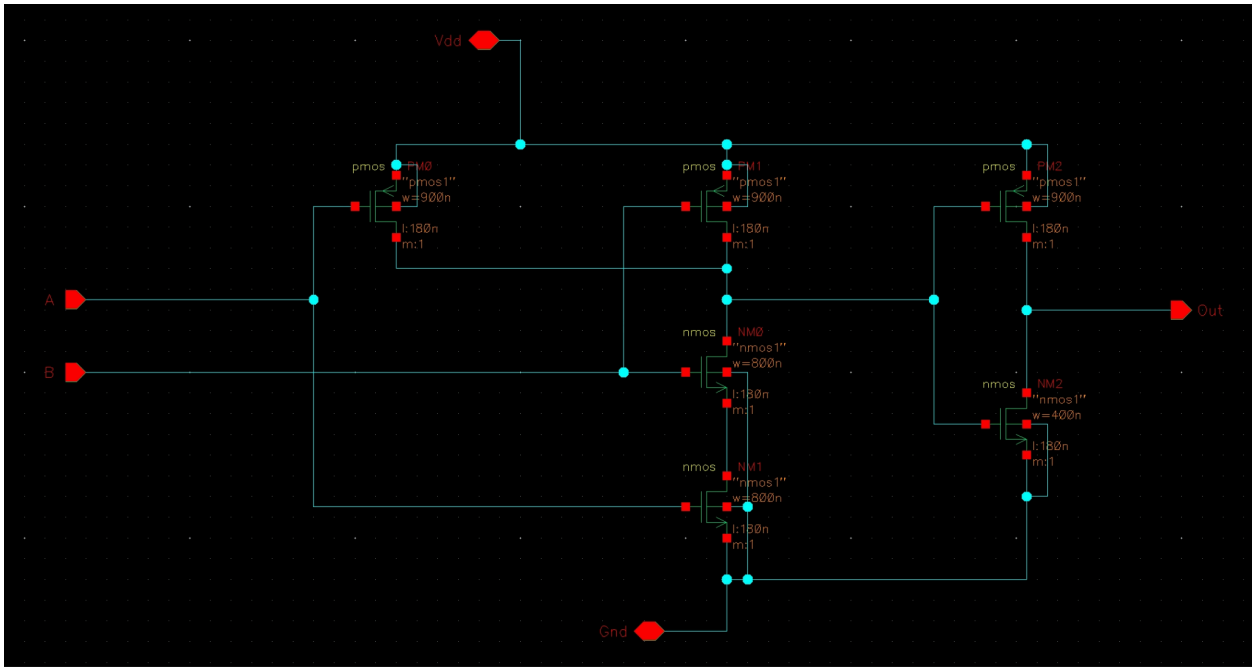


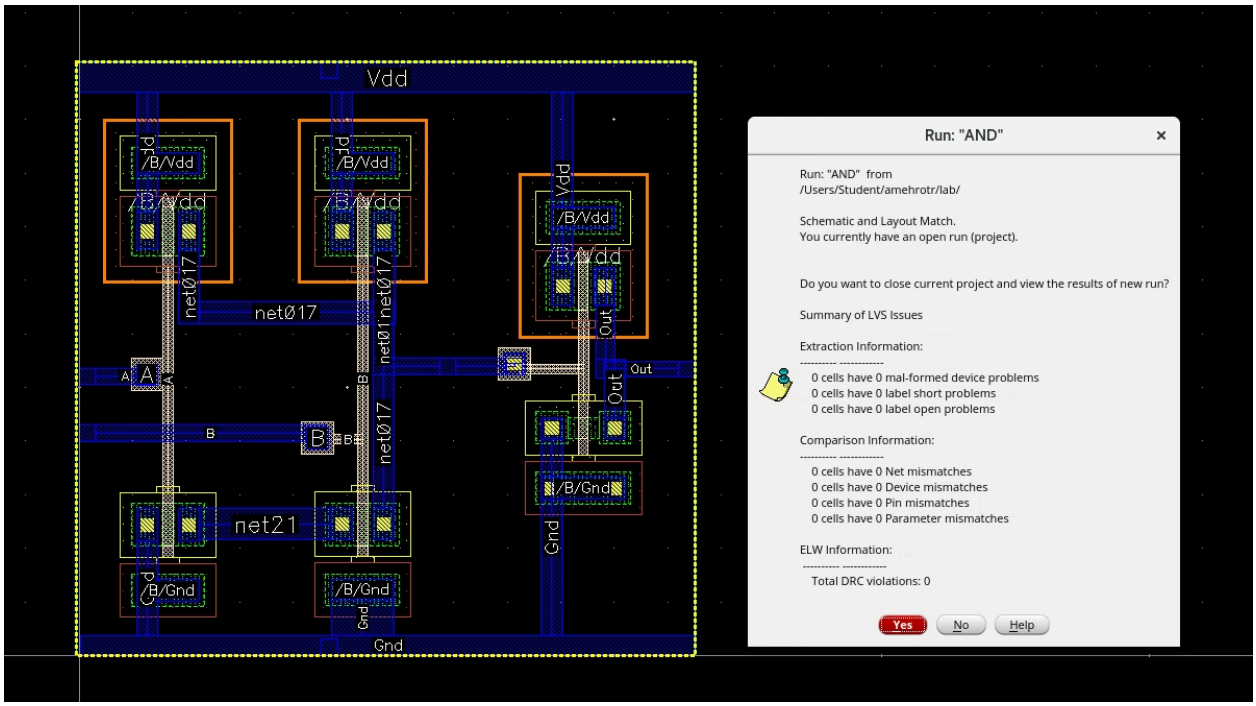
NOR Gate:


vdd
pmos
PM0
"pmos1"
w=1.8u
l:180n
m:1
pmos
PM1
"pmos1"
w=1.8u
l:180n
m:1
a
b
out
nmos
NM0
"nmos1"
w=400n
l:180n
m:1
nmos
NM1
"nmos1"
w=400n
l:180n
m:1
gnd

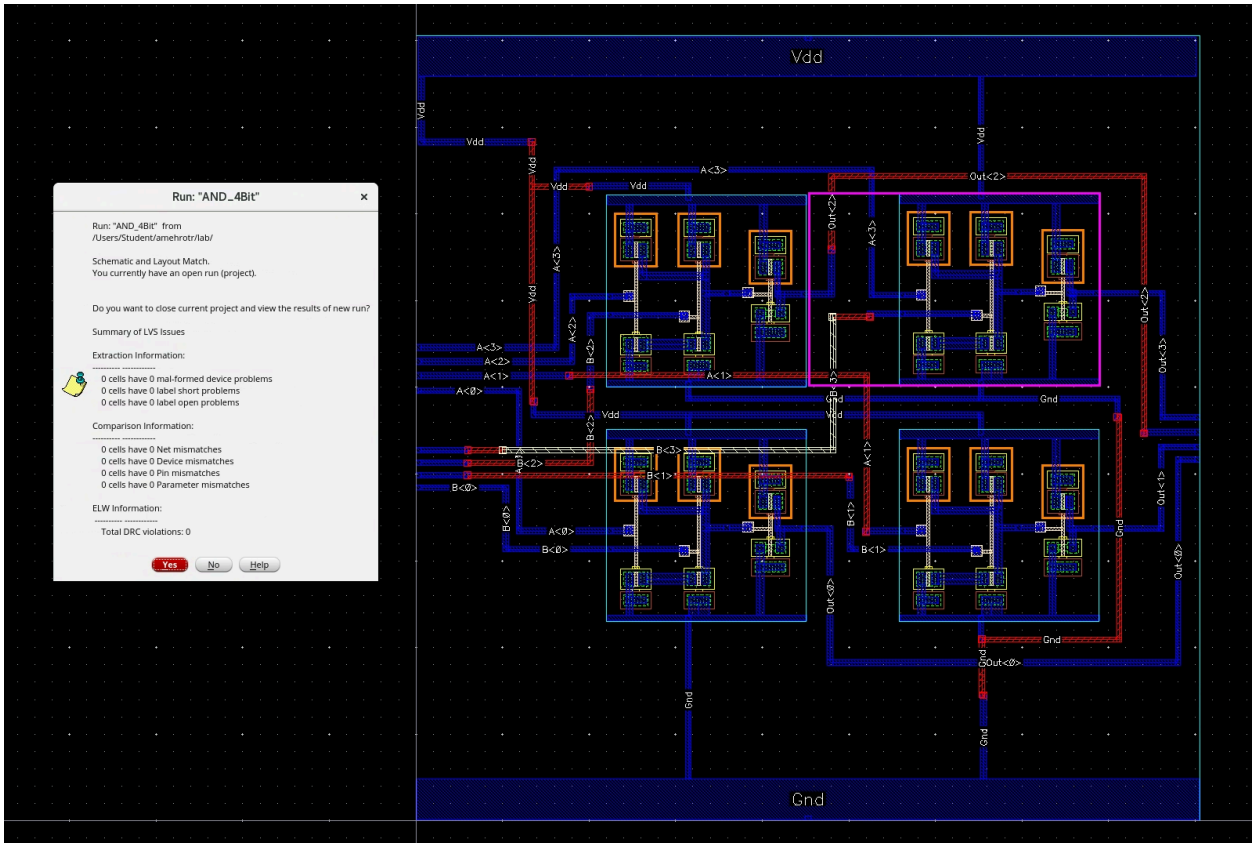

vdd!
/B/vdd!
/S/vdd!
/D/net16
net16
/S/net16
/D/out
out
a
b
out
out
out
/B/gnd!
gnd!
gnd!
Run: "nor"
Run: "nor" from
/Users/Student/amehrotr/lab/
Schematic and Layout Match.
You currently have an open run (project).
Do you want to close current project and view the results of new run?
Summary of LVS Issues
Extraction Information:
0 cells have 0 mal-formed device problems
0 cells have 0 label short problems
0 cells have 0 label open problems
Comparison Information:
0 cells have 0 Net mismatches
0 cells have 0 Device mismatches
0 cells have 0 Pin mismatches
0 cells have 0 Parameter mismatches
ELW Information:
Total DRC violations: 0
Yes
No
Help

AND Gate:

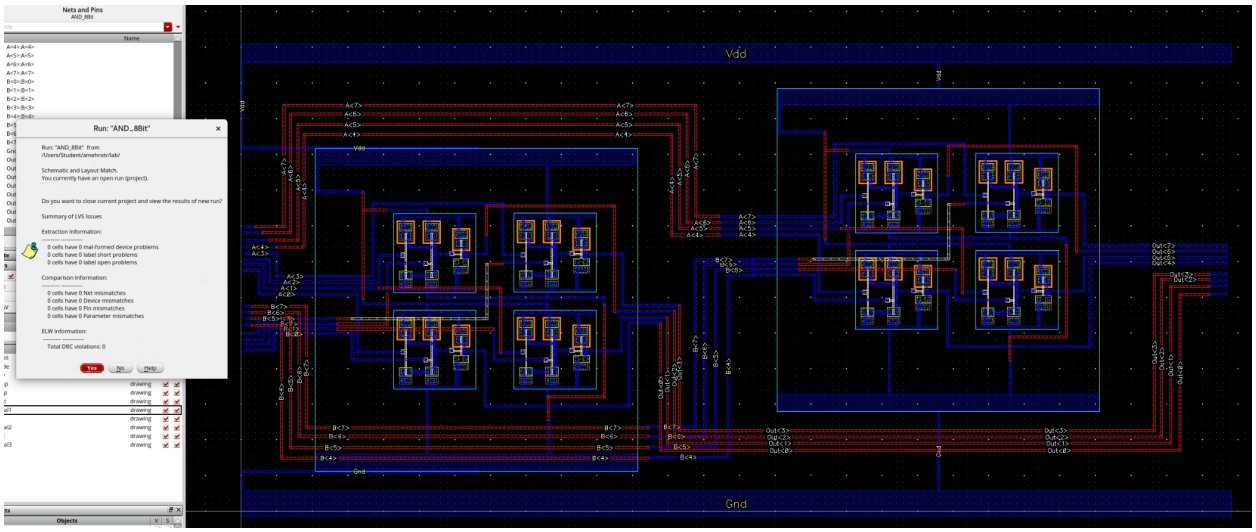

Vdd
pmos
PM0
"pmos1"
w=900n
l:180n
m:1
pmos
PM1
"pmos1"
w=900n
l:180n
m:1
pmos
PM2
"pmos1"
w=900n
l:180n
m:1
A
B
Out
nmos
NM0
"nmos1"
w=800n
l:180n
m:1
nmos
NM2
"nmos1"
w=400n
l:180n
m:1
nmos
NM1
"nmos1"
w=800n
l:180n
m:1
Gnd


Vdd
/B/Vdd
net017
net01
A
B
net21
/B/Gnd
Out
Gnd
Run: "AND"
Run: "AND" from
/Users/Student/amehrotr/lab/
Schematic and Layout Match.
You currently have an open run (project).
Do you want to close current project and view the results of new run?
Summary of LVS Issues
Extraction Information:
0 cells have 0 mal-formed device problems
0 cells have 0 label short problems
0 cells have 0 label open problems
Comparison Information:
0 cells have 0 Net mismatches
0 cells have 0 Device mismatches
0 cells have 0 Pin mismatches
0 cells have 0 Parameter mismatches
ELW Information:
Total DRC violations: 0
Yes
No
Help

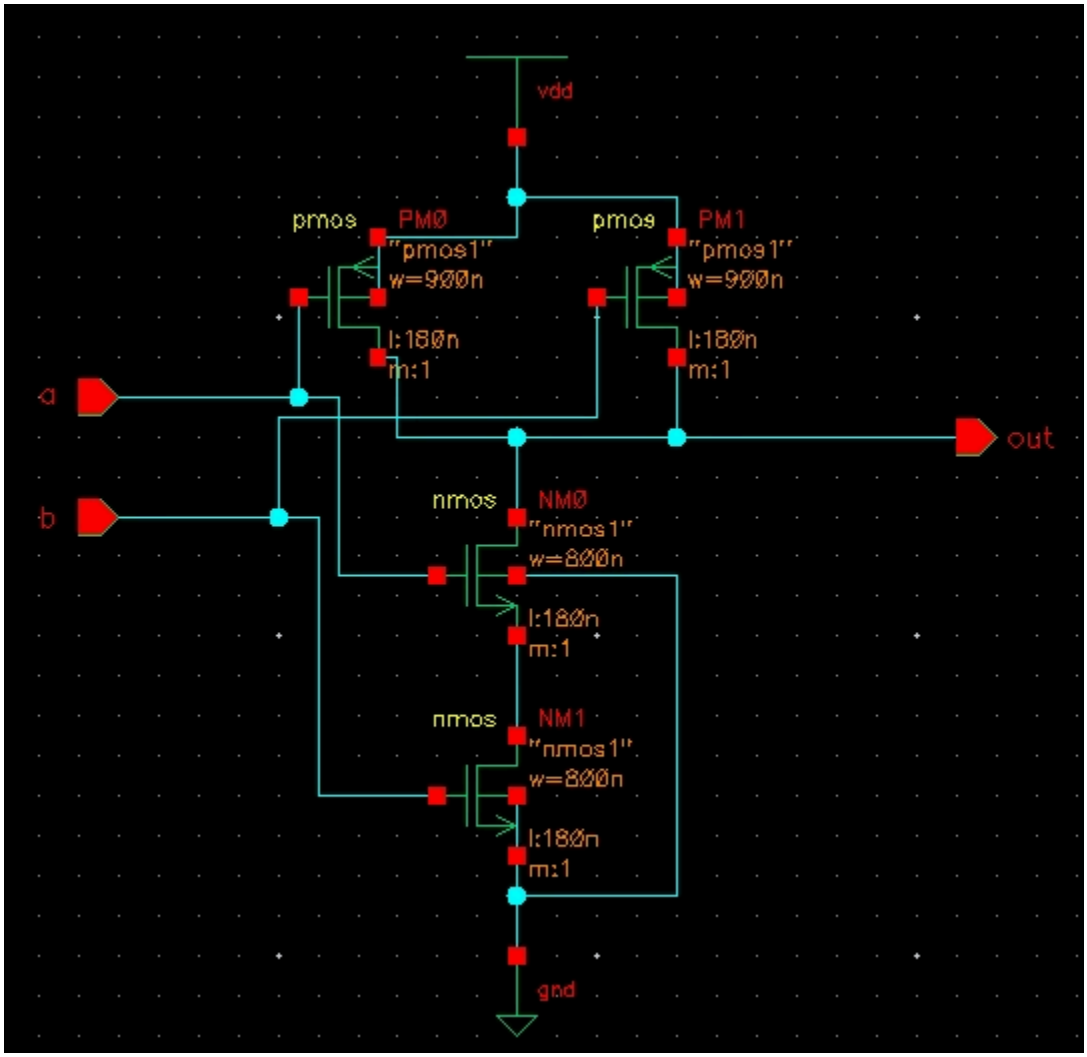
4-Bit AND Gate Layout:

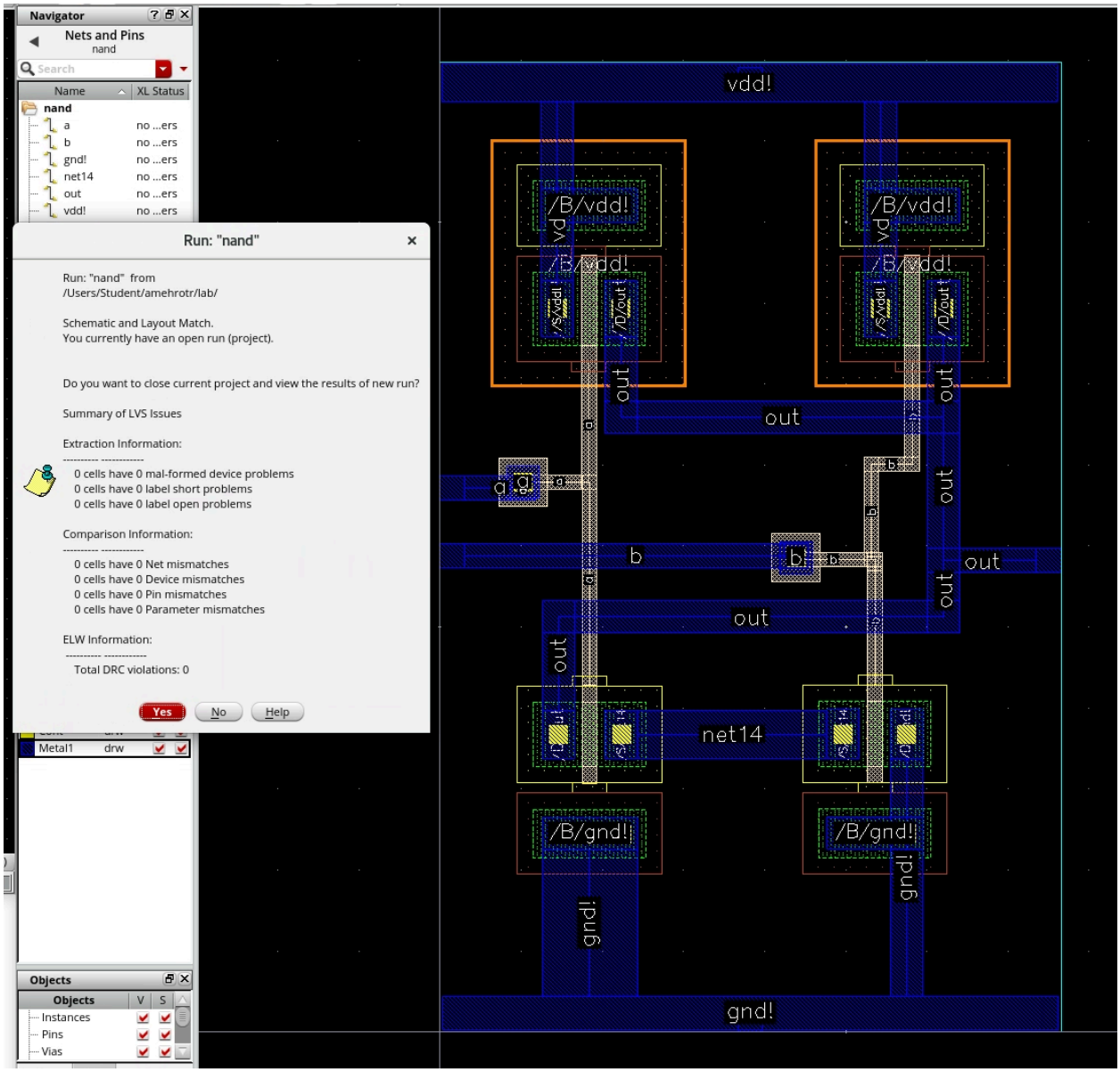


8-Bit AND Gate Layout:

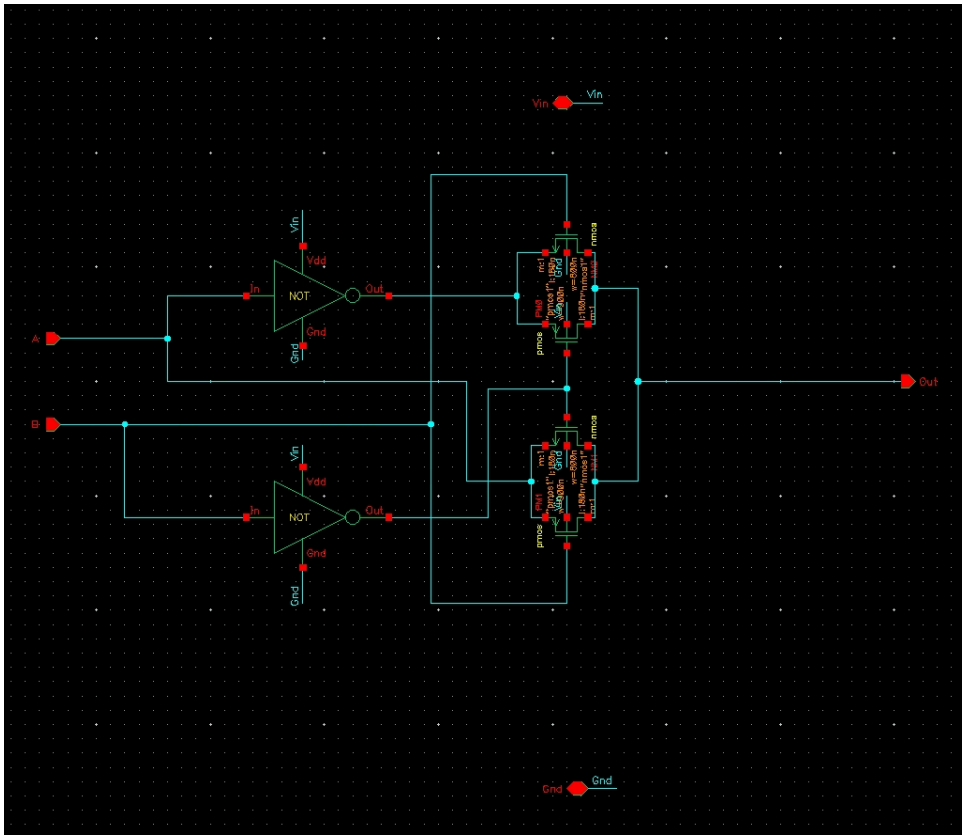


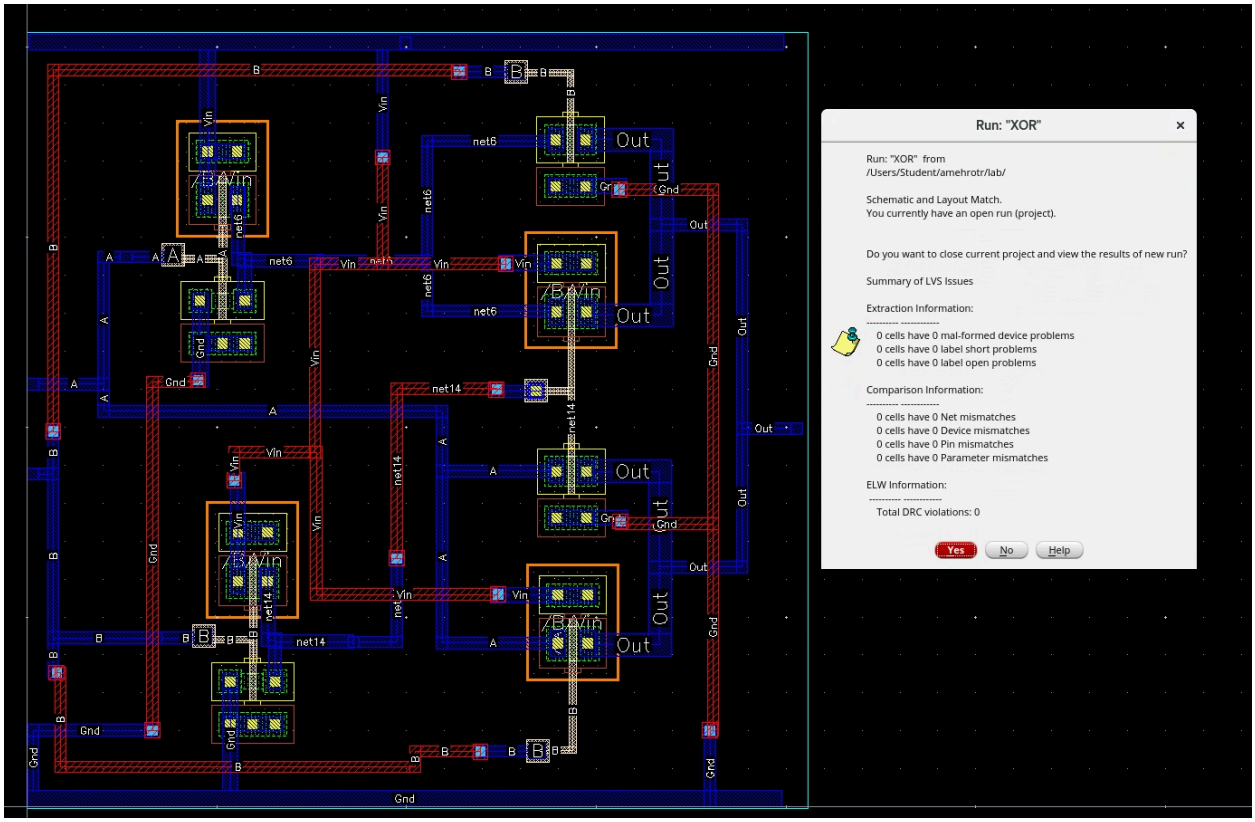
NAND Gate:


vdd
pmos
PM0
"pmos1"
w=900n
l:180n
m:1
pmos
PM1
"pmos1"
w=900n
l:180n
m:1
a
b
out
nmos
NM0
"nmos1"
w=800n
l:180n
m:1
nmos
NM1
"nmos1"
w=800n
l:180n
m:1
gnd

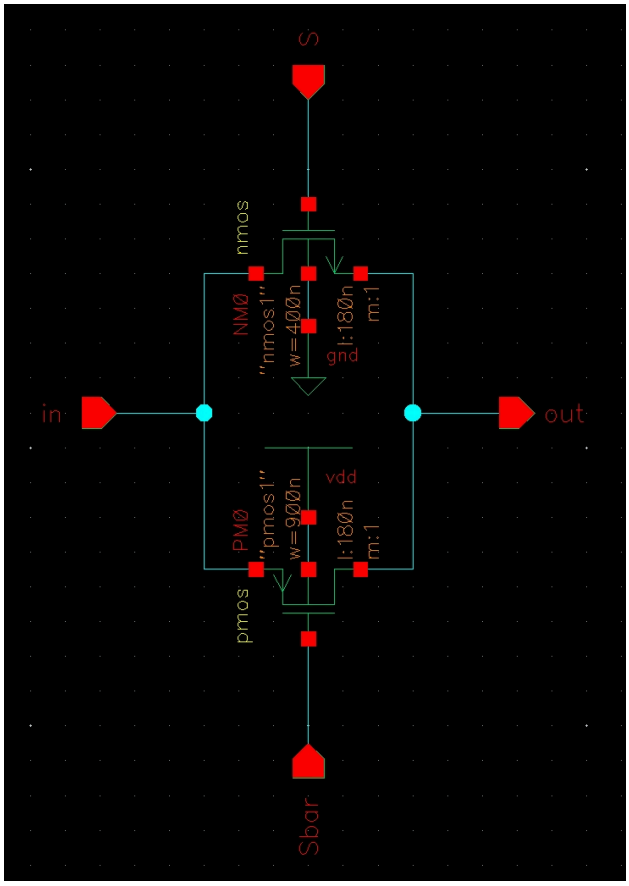

Navigator
Nets and Pins
nand
Search
Name
XL Status
nand
a
b
gnd!
net14
out
vdd!
Run: "nand"
Run: "nand" from
/Users/Student/amehrotr/lab/
Schematic and Layout Match.
You currently have an open run (project).
Do you want to close current project and view the results of new run?
Summary of LVS Issues
Extraction Information:
0 cells have 0 mal-formed device problems
0 cells have 0 label short problems
0 cells have 0 label open problems
Comparison Information:
0 cells have 0 Net mismatches
0 cells have 0 Device mismatches
0 cells have 0 Pin mismatches
0 cells have 0 Parameter mismatches
ELW Information:
Total DRC violations: 0
Yes
No
Help
Metal1
drw
Objects
Instances
Pins
Vias
vdd!
/B/vdd!
/B/gnd!
out
net14
gnd!

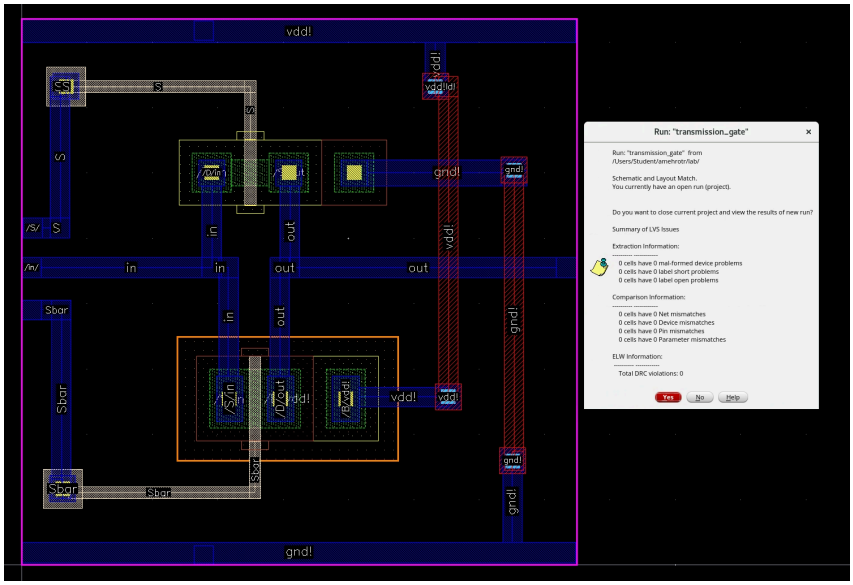
XOR Gate:



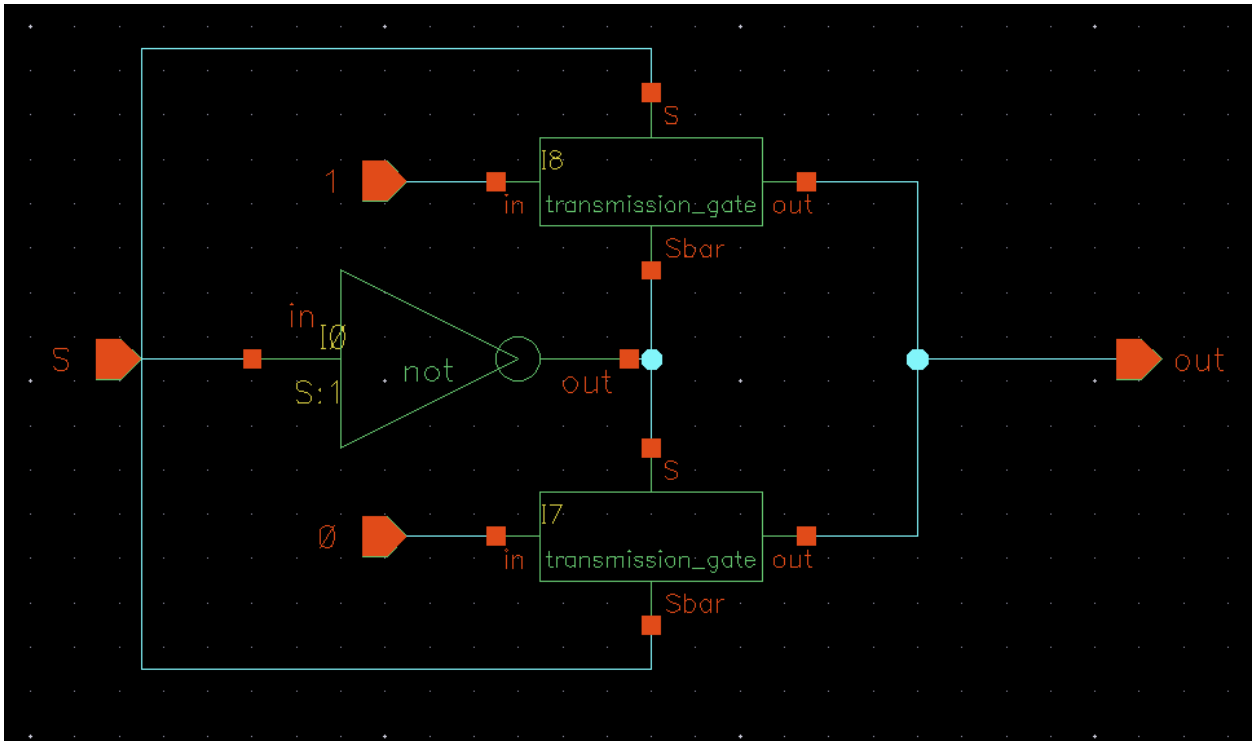

Run: "XOR"
Run: "XOR" from
/Users/Student/amehrotr/lab/
Schematic and Layout Match.
You currently have an open run (project).
Do you want to close current project and view the results of new run?
Summary of LVS Issues
Extraction Information:
0 cells have 0 mal-formed device problems
0 cells have 0 label short problems
0 cells have 0 label open problems
Comparison Information:
0 cells have 0 Net mismatches
0 cells have 0 Device mismatches
0 cells have 0 Pin mismatches
0 cells have 0 Parameter mismatches
ELW Information:
Total DRC violations: 0
Yes
No
Help
Out
Gnd
Vin
net6
net14
A
B

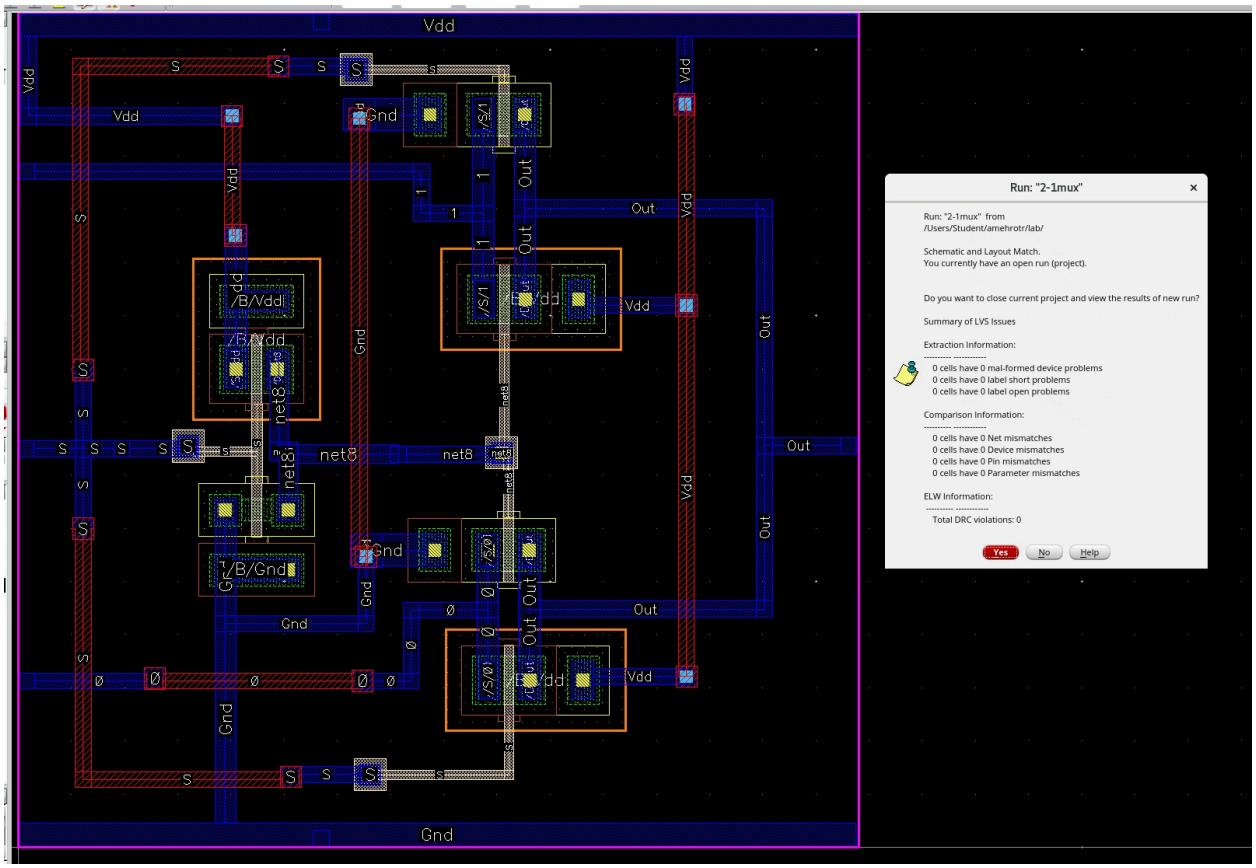
Transmission Gate:



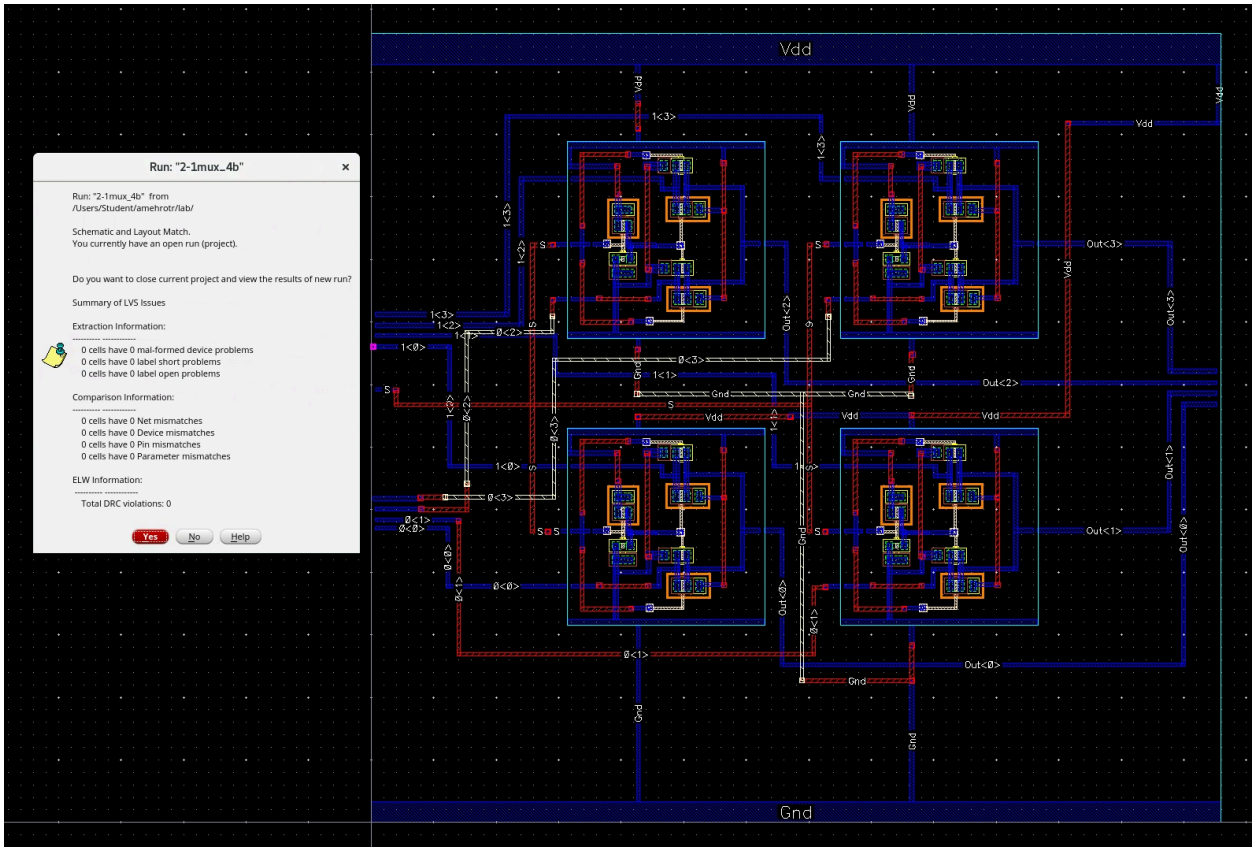


2-1 MUX:

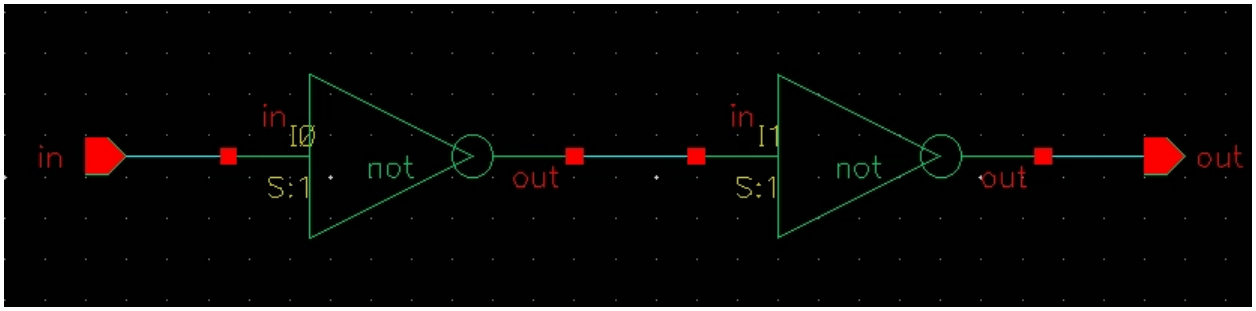

S
1
I8
in
transmission_gate
out
Sbar
in
I0
S
not
S:1
out
out
S
Ø
I7
in
transmission_gate
out
Sbar

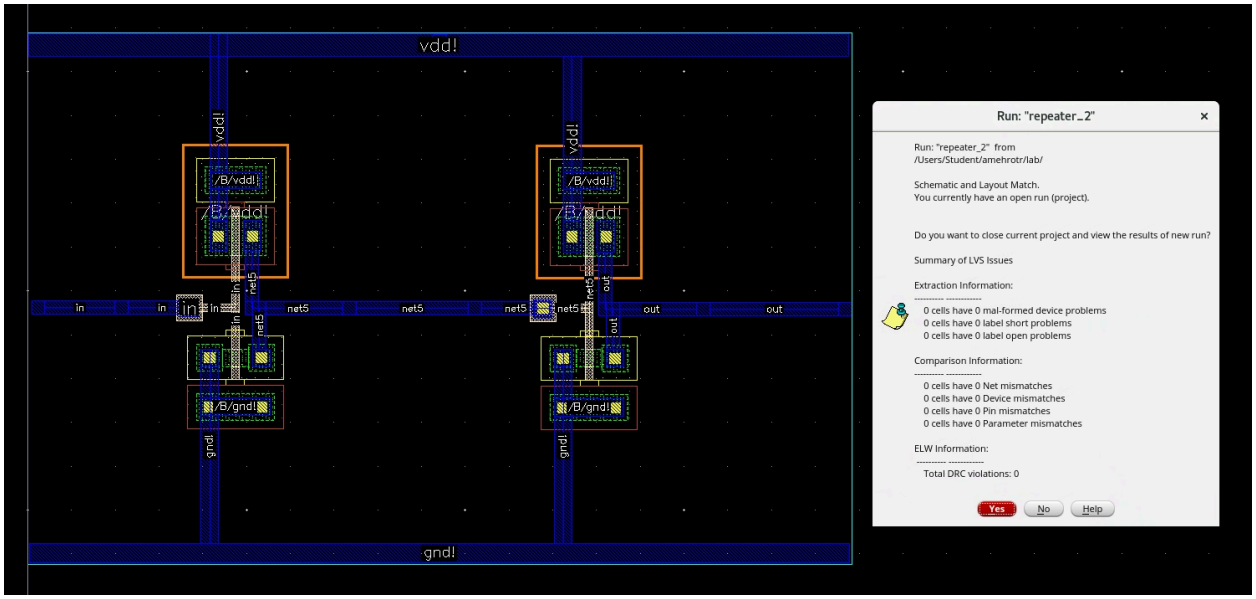

Vdd
Gnd
Out
net8
/B/Vdd
/B/Gnd
Run: "2-1mux"
Run: "2-1mux" from
/Users/Student/amehrotr/lab/
Schematic and Layout Match.
You currently have an open run (project).
Do you want to close current project and view the results of new run?
Summary of LVS Issues
Extraction Information:
0 cells have 0 mal-formed device problems
0 cells have 0 label short problems
0 cells have 0 label open problems
Comparison Information:
0 cells have 0 Net mismatches
0 cells have 0 Device mismatches
0 cells have 0 Pin mismatches
0 cells have 0 Parameter mismatches
ELW Information:
Total DRC violations: 0
Yes
No
Help

4-Bit 2-to-1 MUX Layout:

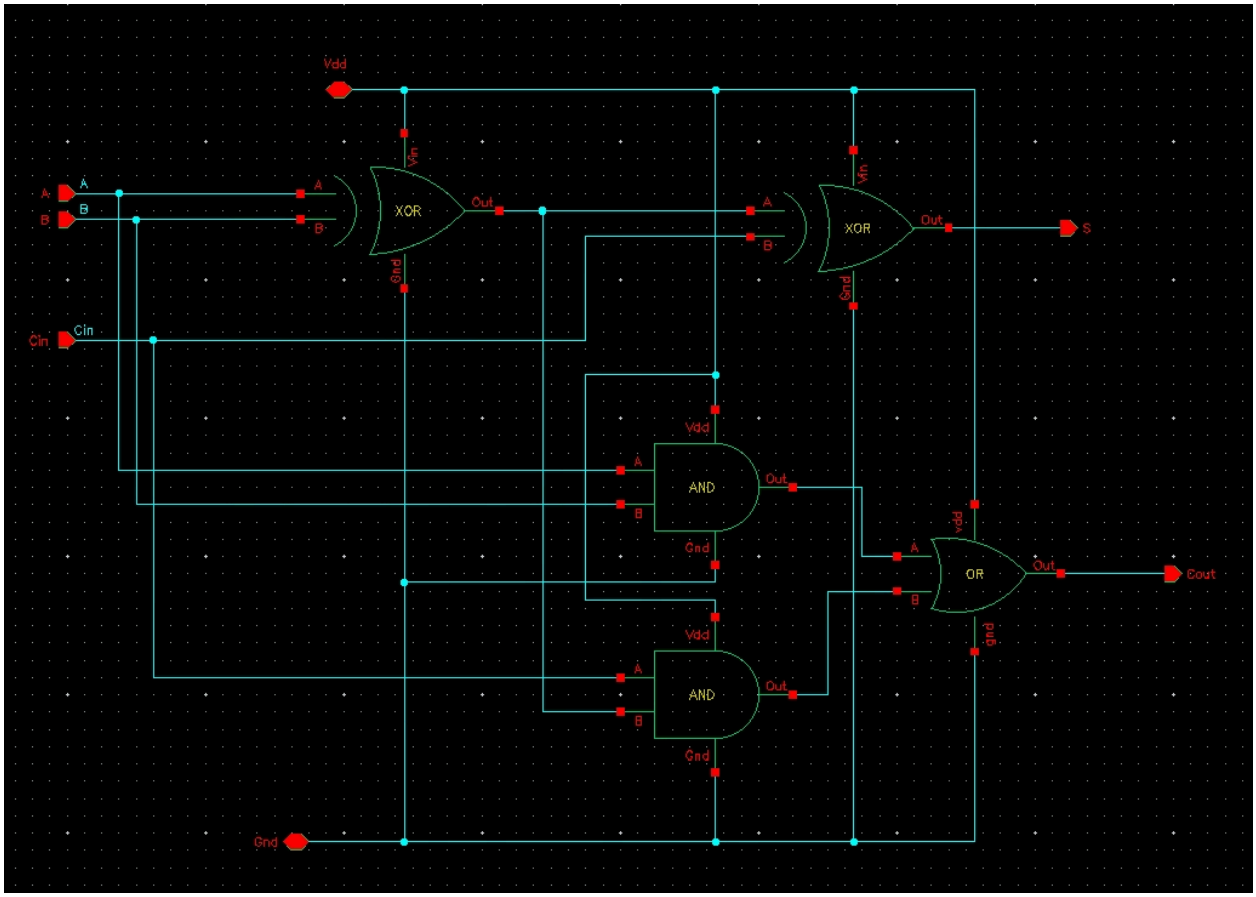


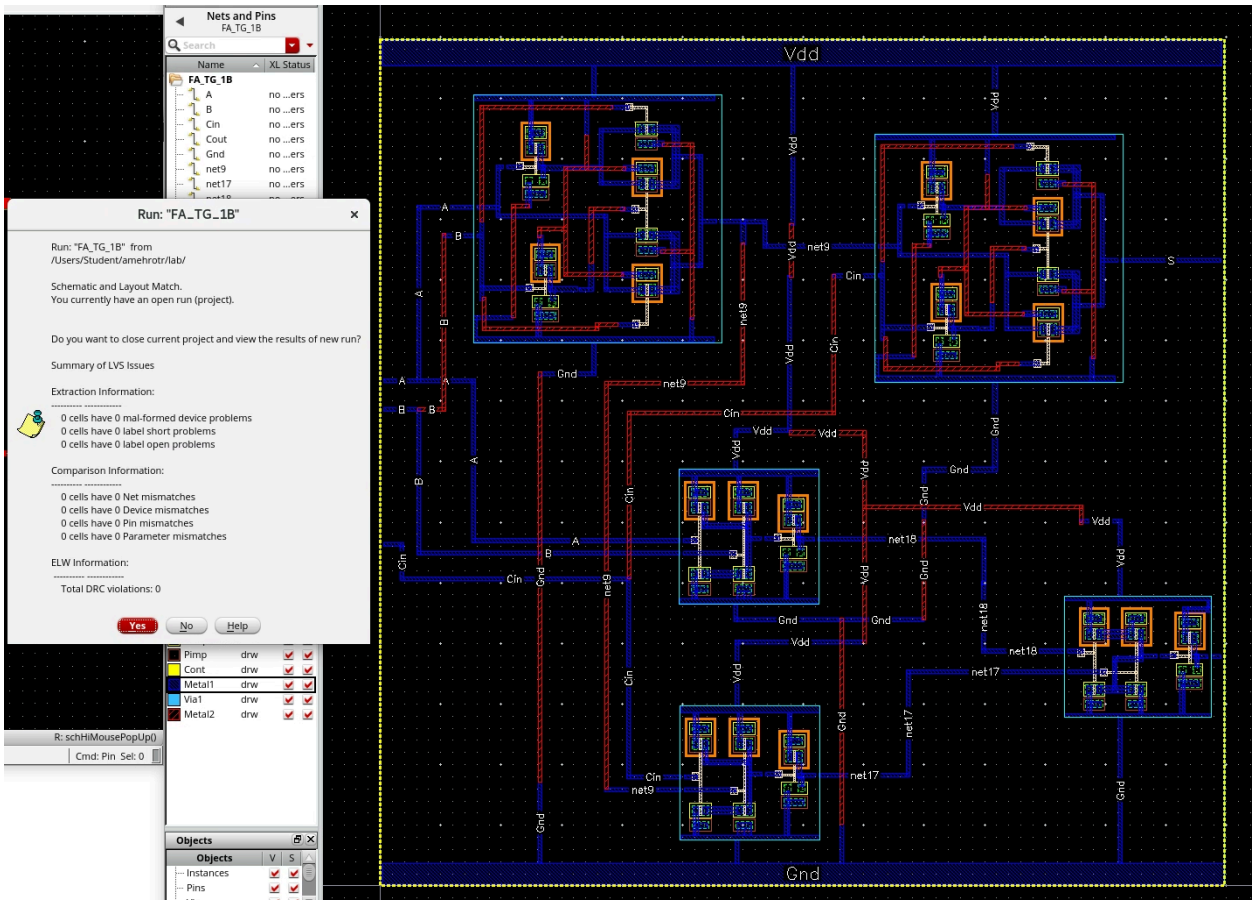
2 Stage Repeater:



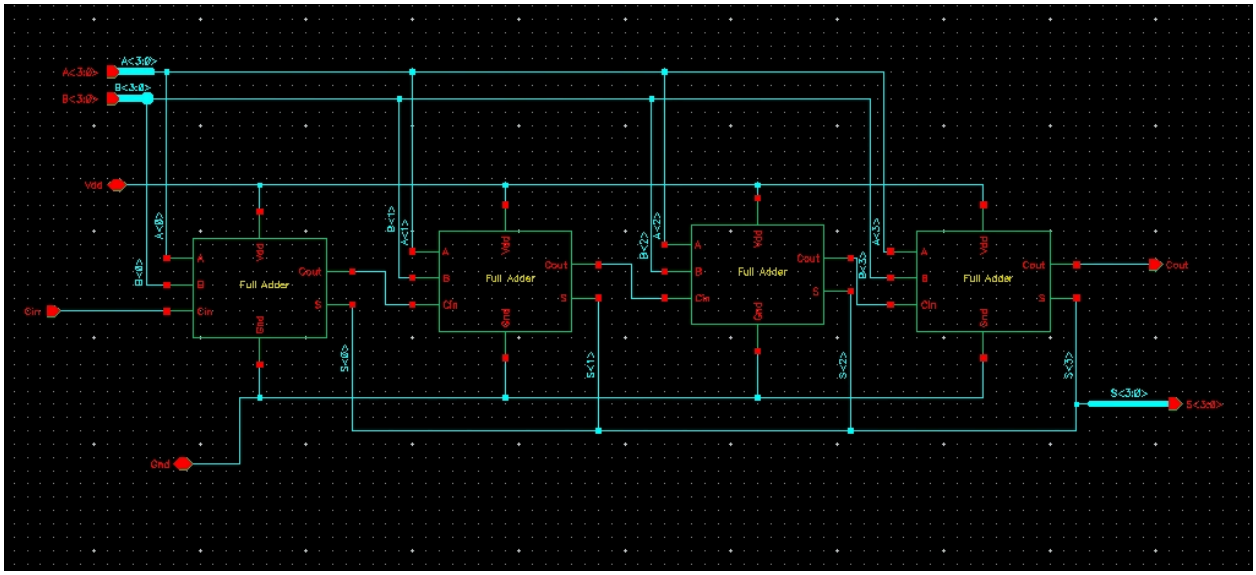


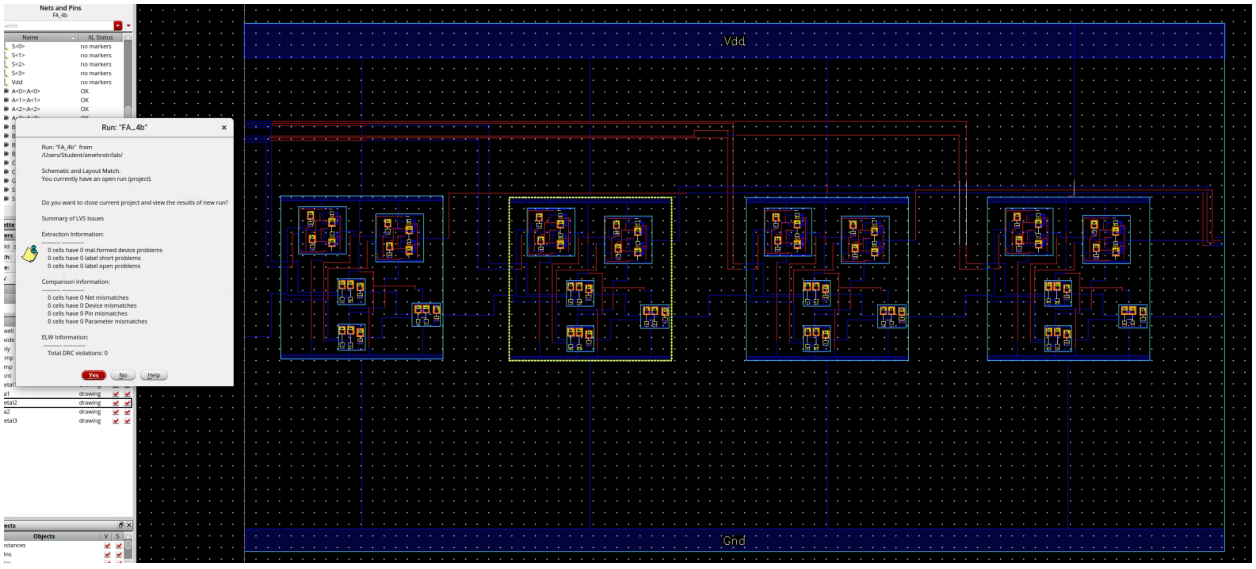
1-Bit Full Adder:



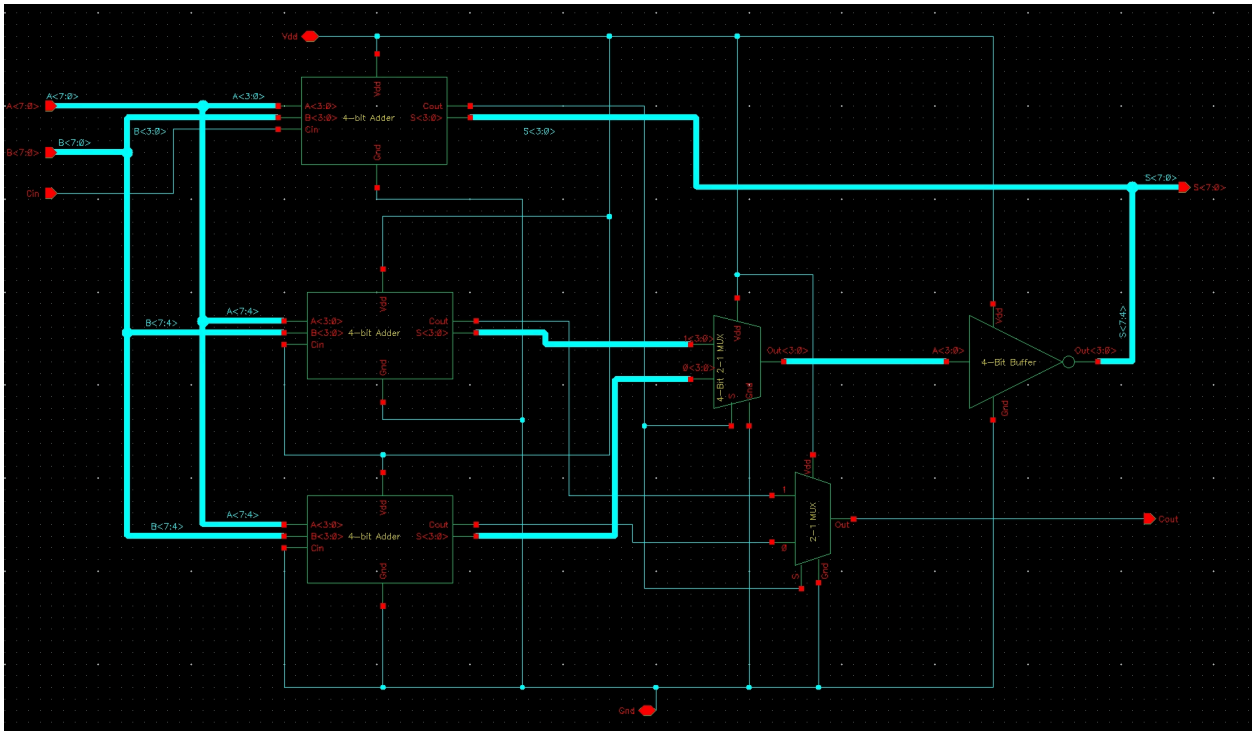


4-Bit Ripple Carry Adder:

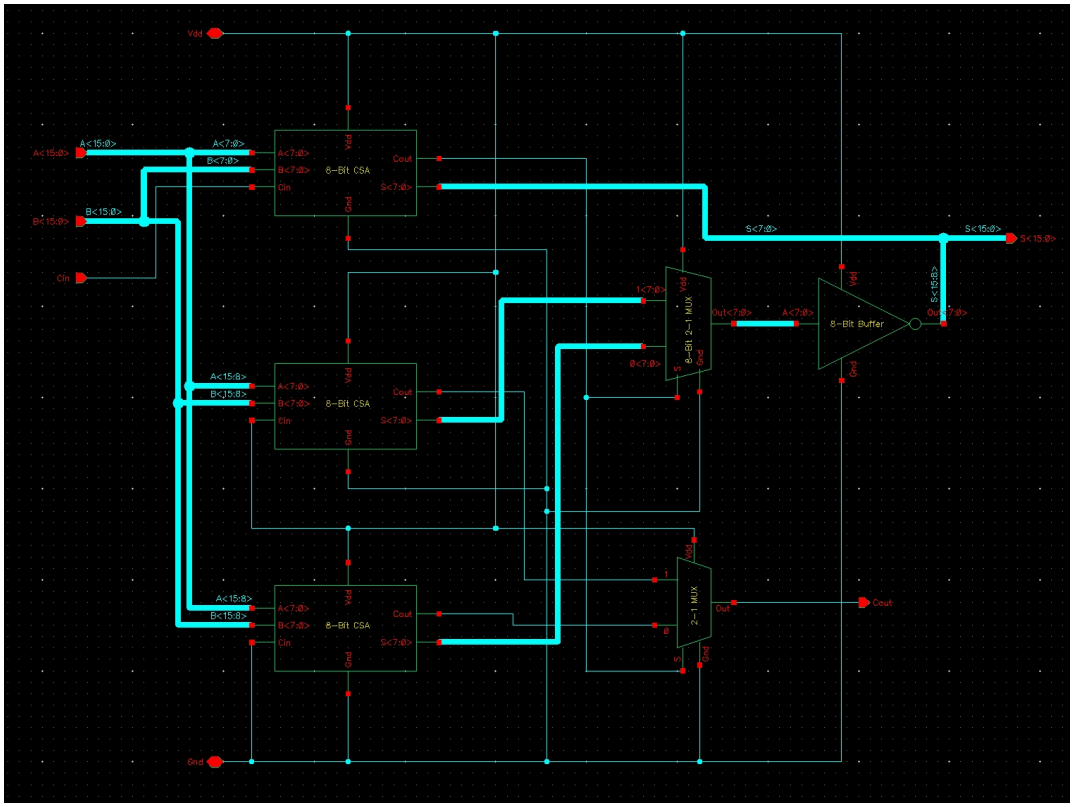




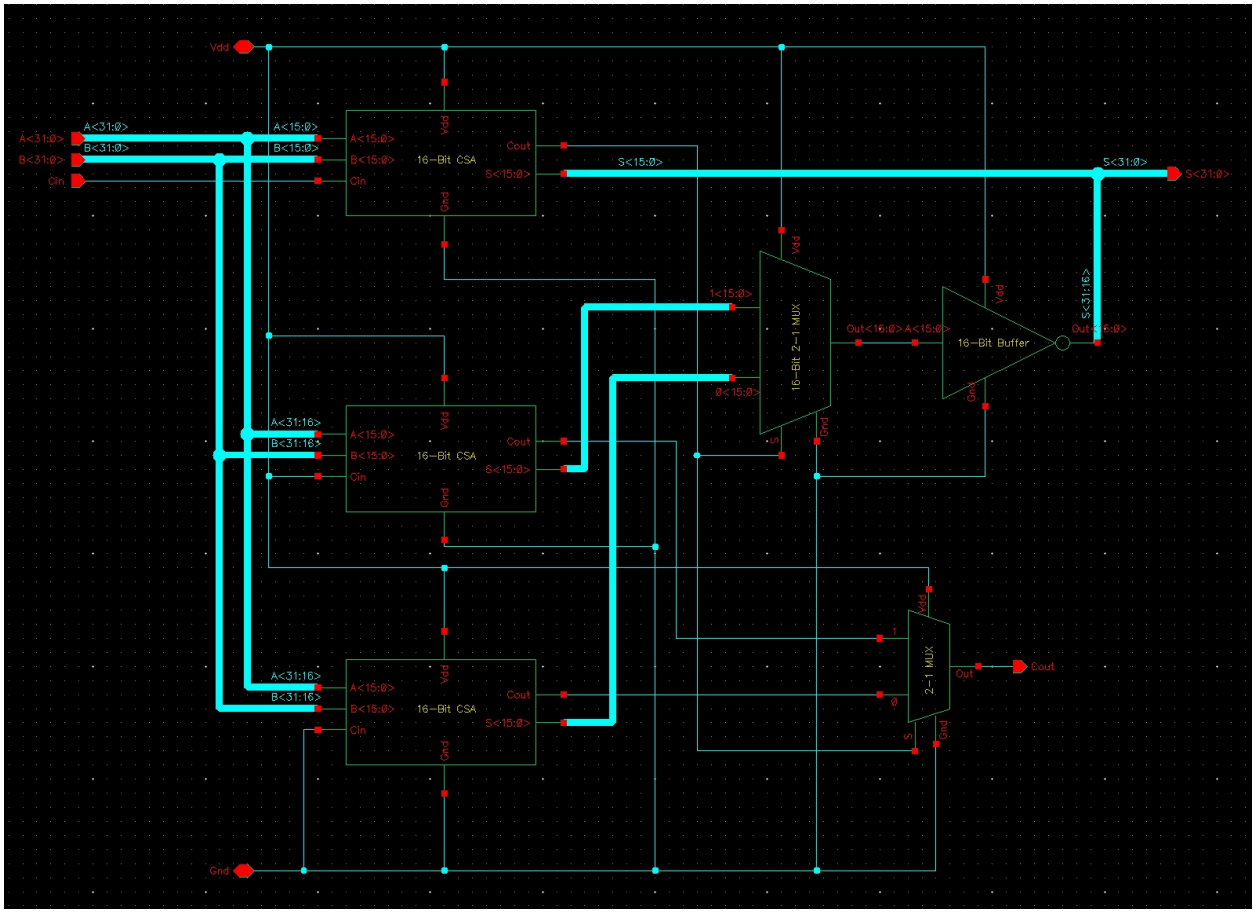
8-Bit Carry Select Adder:



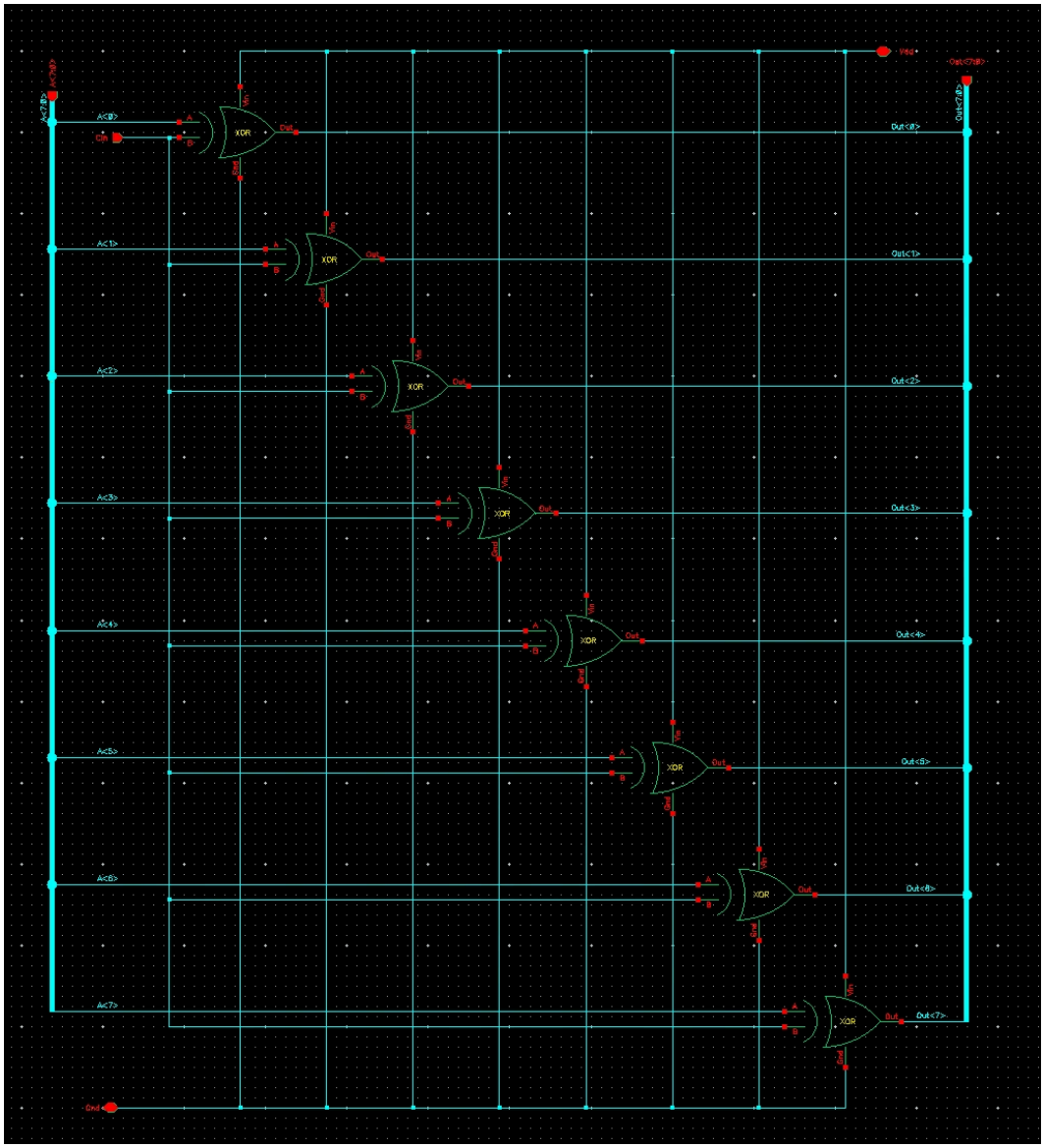
16-Bit Carry Select Adder:



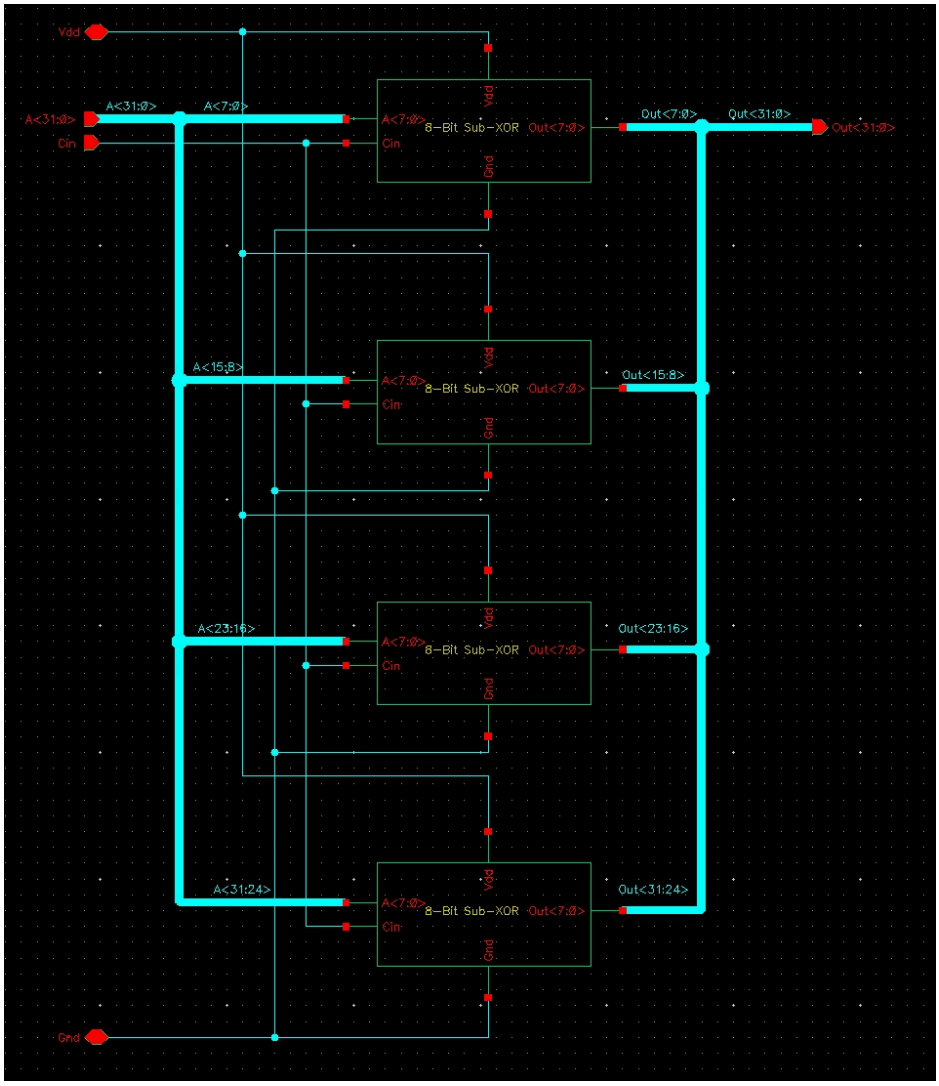
32-Bit Carry Select Adder:



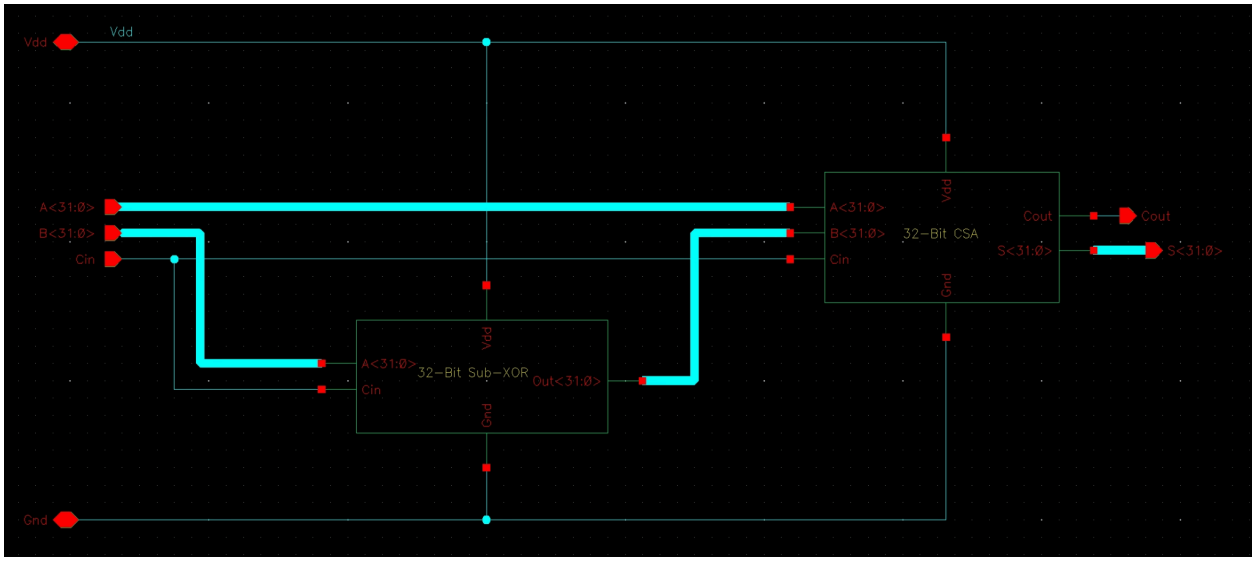
8-Bit Subtractor XOR Unit:



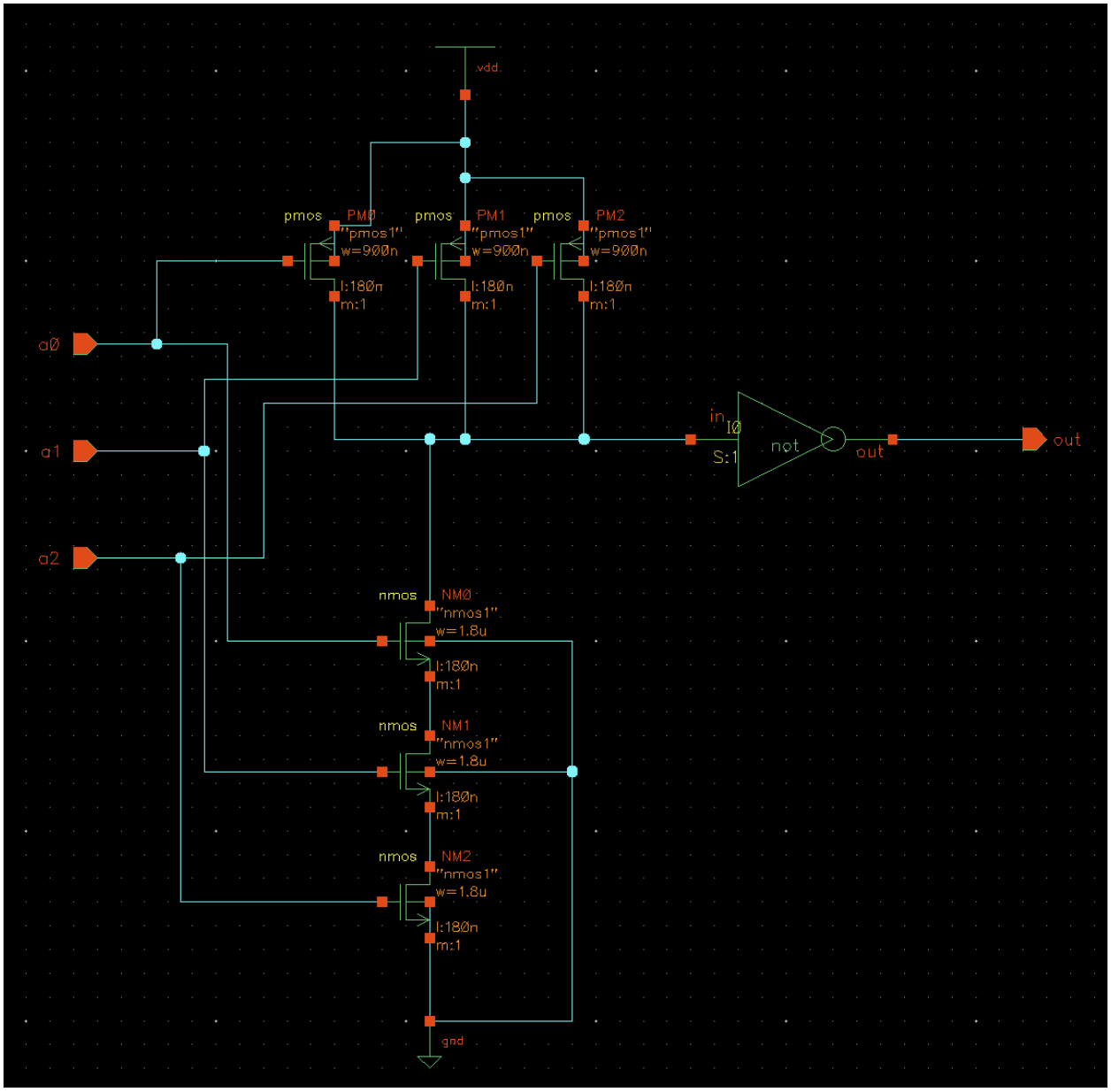
32-Bit Subtractor XOR Unit:



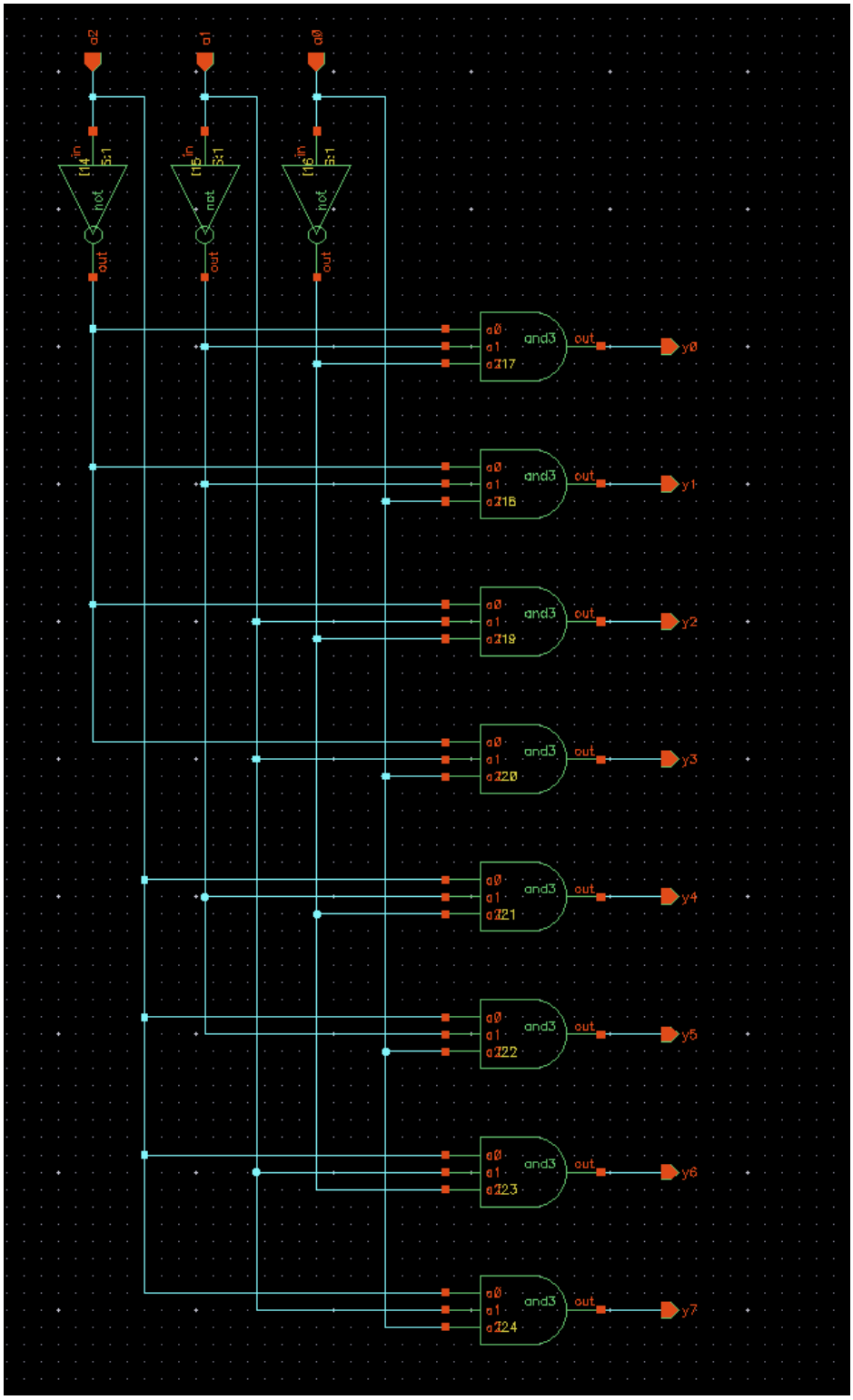
32-Bit Adder/Subtractor:



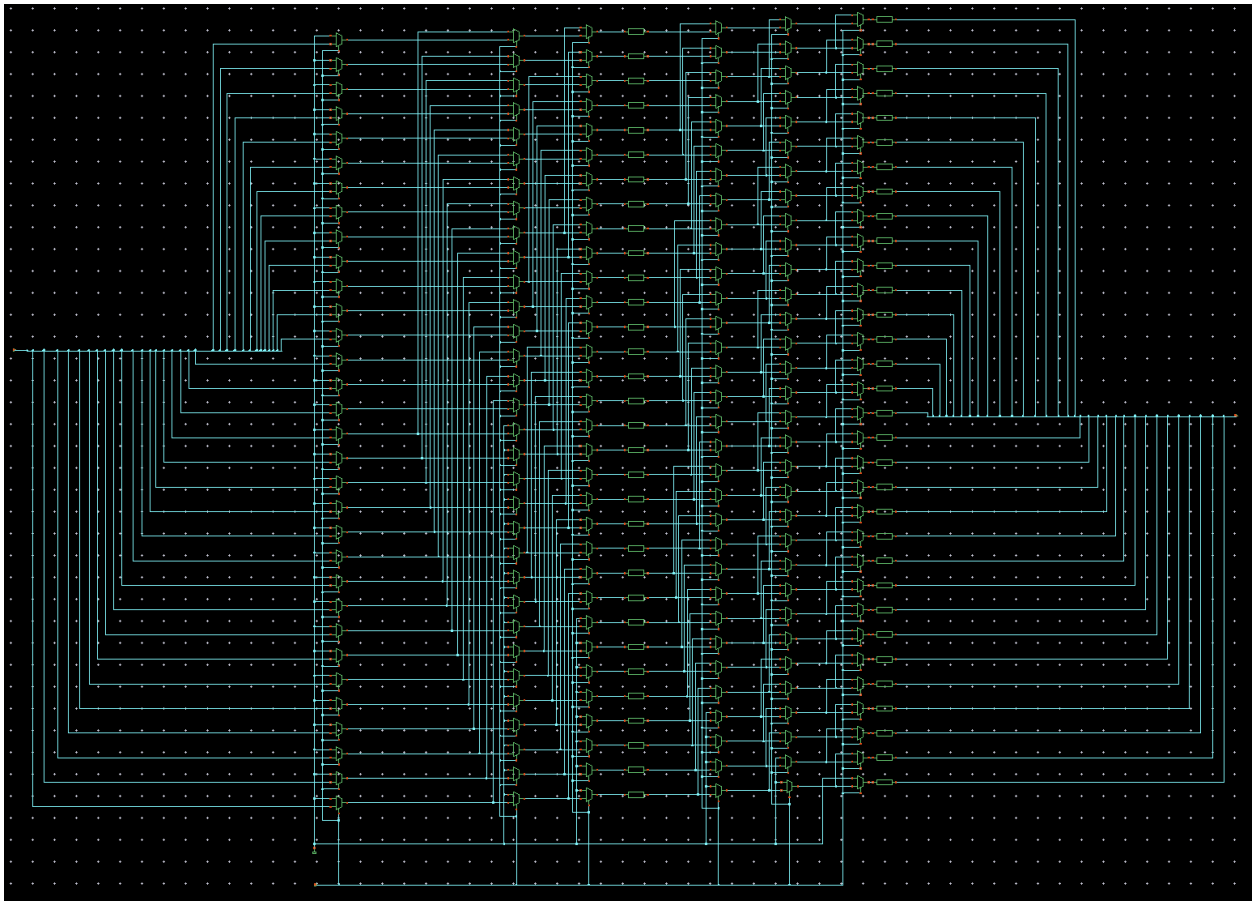
3 Input AND:


vdd
pmos
PM0
"pmos1"
w=900n
l:180n
m:1
pmos
PM1
"pmos1"
w=900n
l:180n
m:1
pmos
PM2
"pmos1"
w=900n
l:180n
m:1
a0
a1
a2
in
I0
S:1
not
out
out
nmos
NM0
"nmos1"
w=1.8u
l:180n
m:1
nmos
NM1
"nmos1"
w=1.8u
l:180n
m:1
nmos
NM2
"nmos1"
w=1.8u
l:180n
m:1
gnd

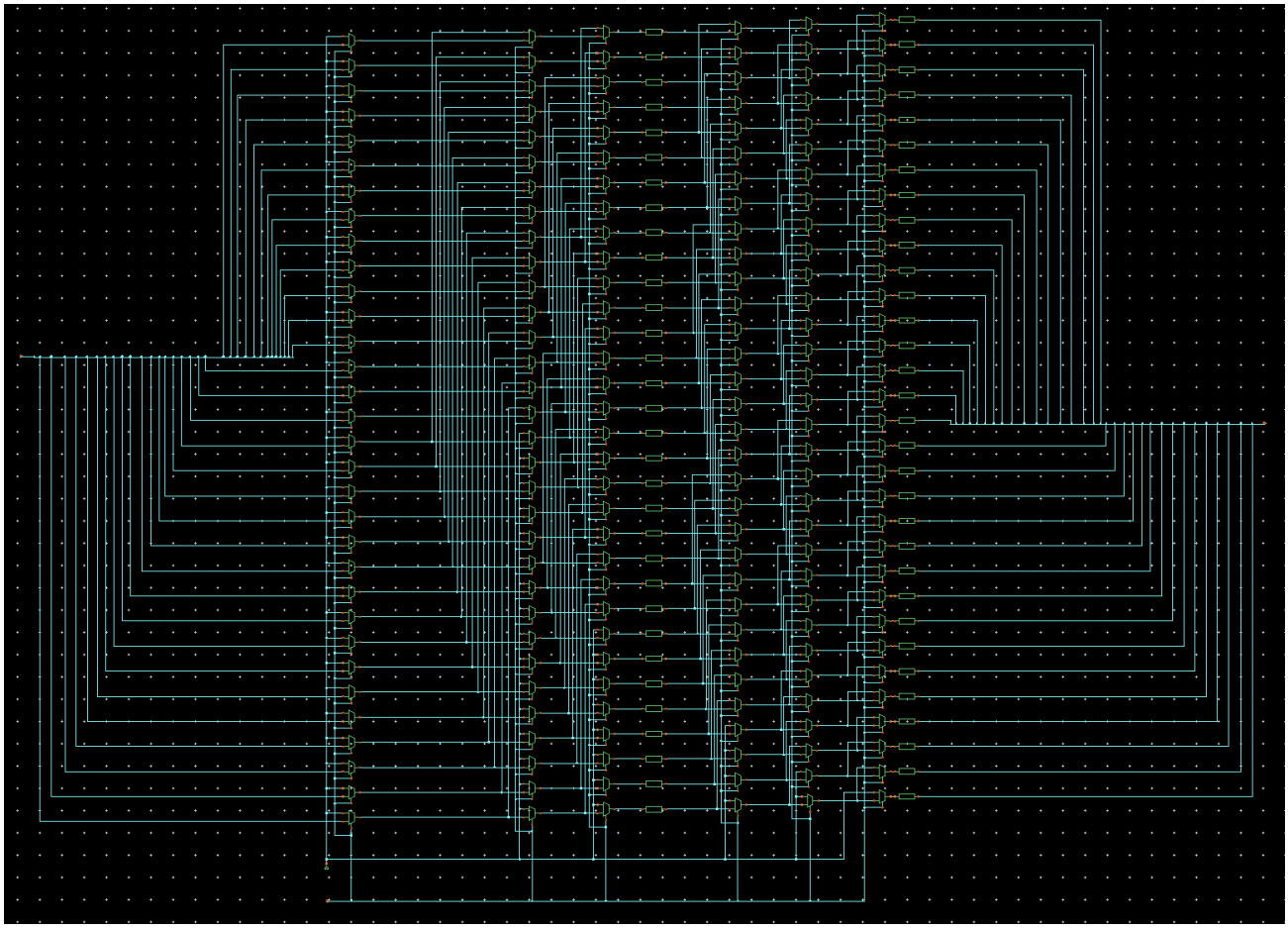
3 Bit Decoder:



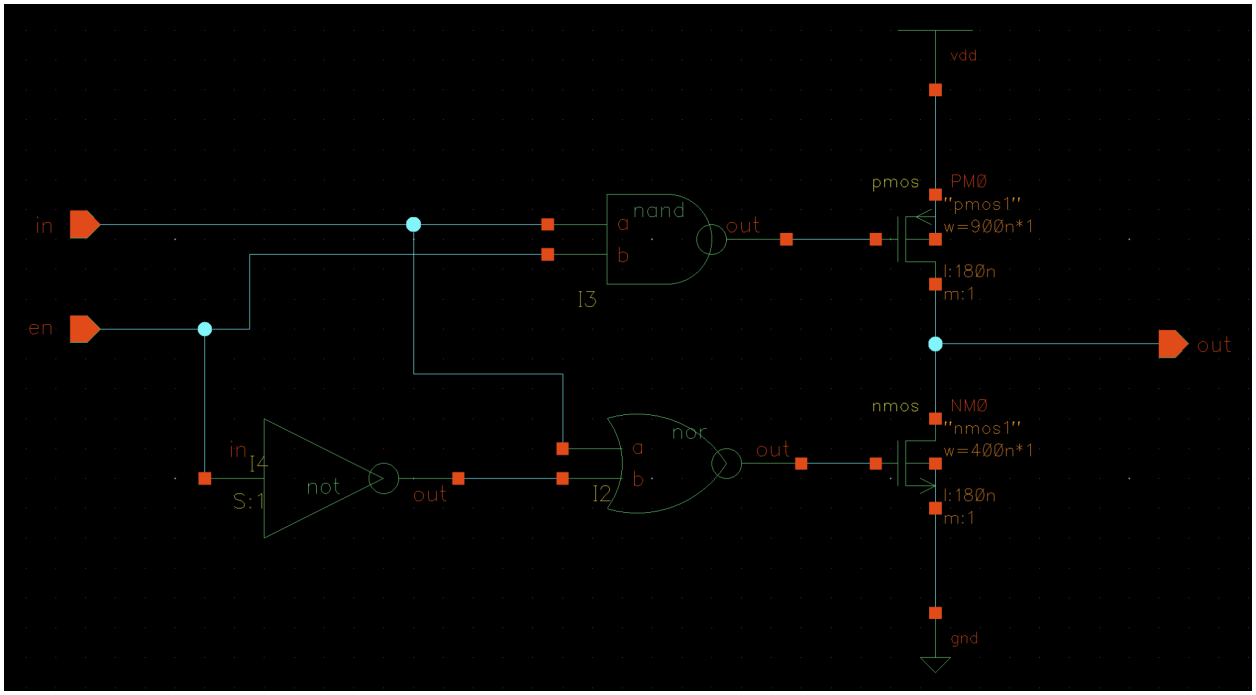
32 Bit Shift Left:



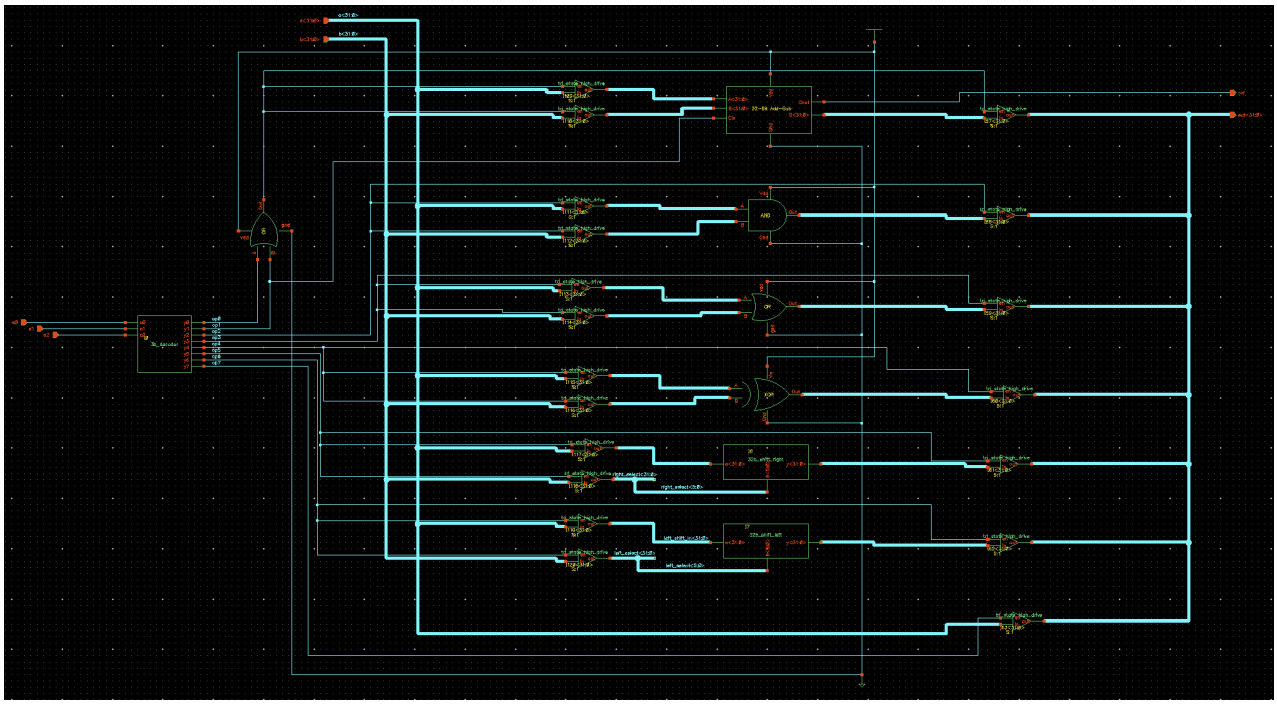
32 Bit Shift Right:



Tri-State Buffer:



Top Level (ALU):



ALU Testbench: