

32 – Bit ALU

EECE 7353: VLSI Design

Professor Kim

4/26/2021

Team Members: Tiffany Chan, Erica Chen, Kyung Hwan 'David' Lee

Table of Contents

ALU Design Methodology	5
Scope.....	5
Design Requirements.....	5
Implementation	5
ALU Block Diagram.....	6
CMOS Gate Implementation.....	7
Scope.....	7
INV Gate (Kyung Hwan ‘David’ Lee).....	7
Schematic.....	7
Simulation	8
Layout.....	10
Post-Layout Simulation	11
AND Gate (Kyung Hwan ‘David’ Lee)	12
Schematic.....	12
Simulation	13
Layout.....	15
Post-Layout Simulation	16
OR Gate (Kyung Hwan ‘David’ Lee).....	17
Schematic.....	17
Simulation	18
Layout.....	20
Post-Layout Simulation	21
XOR Gate (Kyung Hwan ‘David’ Lee).....	22
Schematic.....	22
Simulation	23
Layout.....	25
Post-Layout Simulation	26
Lessons Learned.....	26
32-Bit Adder (Kyung Hwan ‘David’ Lee).....	27
Constraints & Functionality Definition.....	27
Design Methodologies	27
Ripple Carry Adder	27

Carry Look Ahead Adder	28
Chosen Design Methodology	29
Propagate and Generate Block	30
Schematic.....	30
Layout.....	31
Post-Layout Simulation	32
4 – Bit Propagate Generate Block.....	33
Scope.....	33
Schematic.....	33
Layout.....	34
Post – Layout Simulation.....	35
Lessons Learned.....	40
4 – Bit Sum and Carry Logic.....	41
Scope.....	41
Schematic.....	42
Layout.....	44
Post – Layout Simulation.....	45
Lessons Learned.....	52
4 – Bit Carry Look Ahead Adder	53
Scope.....	53
Schematic.....	53
Layout.....	54
Post Layout Simulation	55
32 – Bit Carry Look Ahead Adder	62
Scope.....	62
Schematic.....	62
Layout.....	65
Post – Layout Simulation.....	67
32 – Bit OR gate (Erica Chen)	72
Constraints and Functionality	72
Design Methodologies	72
Schematic.....	73
Layout.....	80

32 - Bit AND gate (Tiffany Chan)	82
Constraints and Functionality	82
Design Methodologies	82
Schematic.....	82
Simulation	84
Layout.....	85
Post-Layout Simulation	87
Lessons Learned	89
3 to 1 MUX (Tiffany Chan).....	90
Constraints and Functionality	90
Design Methodologies	90
1- Bit 3 to 1 MUX with Select Complements.....	91
Schematic.....	91
Simulation	92
Layout.....	94
1- Bit 3 to 1 MUX with Inverter and Select	95
Schematic.....	95
Simulation	95
Layout.....	96
32 - Bit 3 to 1 MUX.....	97
Schematic.....	97
Simulation	99
Layout.....	100
Post-Layout Simulation	102
Lessons Learned	104
Lesson's Learned	105
Bibliography	106
Appendix: Recorded Delays and Areas	107

ALU Design Methodology

Scope

A 32 Bit ALU Design was chosen for the topic of this project. This was chosen as the 32 – Bit ALU is a basic functional component of every modern CPU.

Design Requirements

ALU functionality requirements as defined by the initial group meeting are as follows.

1. 32 – bit Bitwise OR
2. 32 – bit Bitwise AND
3. 32 – bit Adder

Additional ALU functionalities were planned to be added if time allowed.

The inputs for the ALU are defined as the following.

1. 32 – bit A Input bus
2. 32 – bit B Input bus
3. 32 – bit S output bus
4. 2 – bit ALU OP Switch

Implementation

The implementation of the ALU was planned as the following.

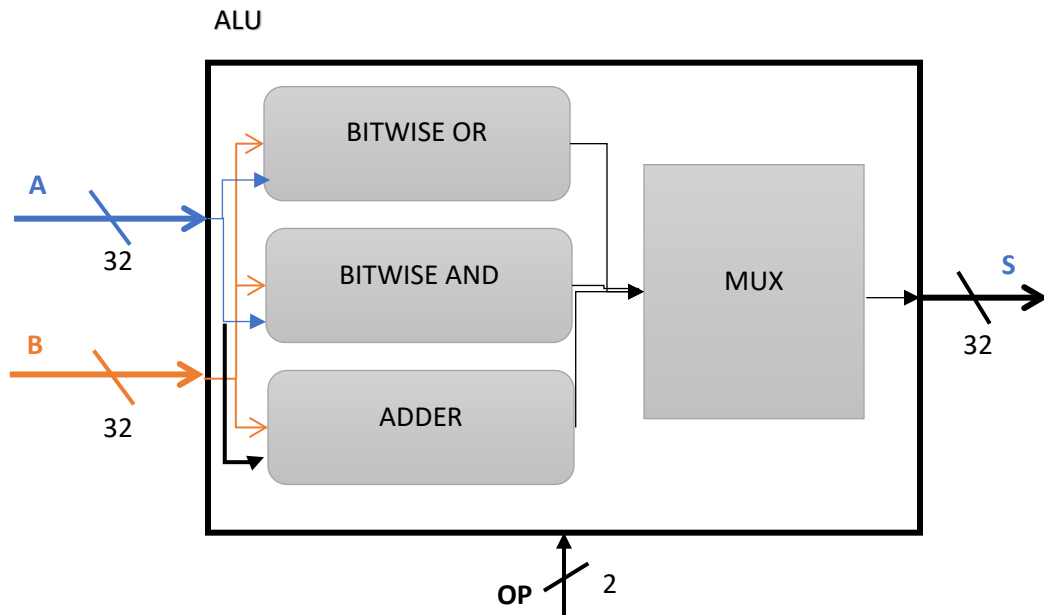
1. Individual functionalities are implemented (OR, AND, ADDER).
2. MUX created to enable ALU with multiple functions.
3. Block Integration (Integrate function blocks with MUX).

If time allowed, an additional function could be added to fill in the additional ‘switch’ available on a 2-bit selector MUX.

The functional block diagram of the ALU is described in the following section.

The ALU is implemented using the *GPDK180* design package in *Cadence Virtuoso*.

ALU Block Diagram



CMOS Gate Implementation

Scope

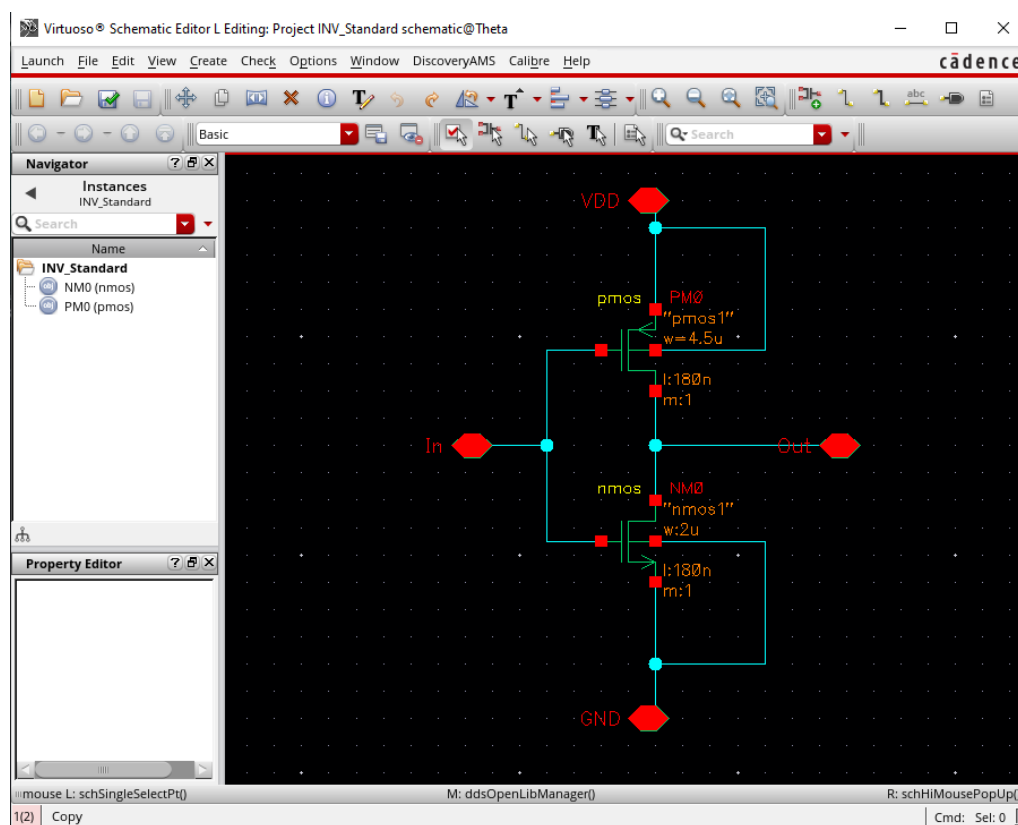
In this section, the implementation of the following gates will be described.

1. INV Gate
2. AND Gate
3. OR Gate
4. XOR Gate

These gates are specifically described in because they will be used as the building blocks to create all other functional blocks of the ALU.

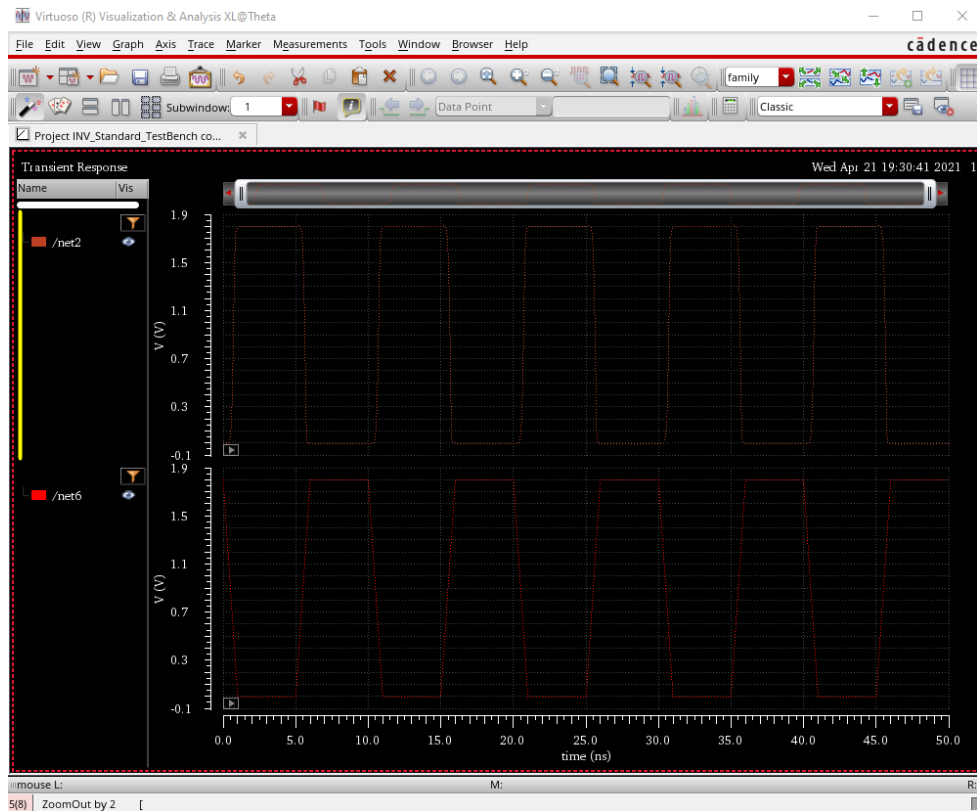
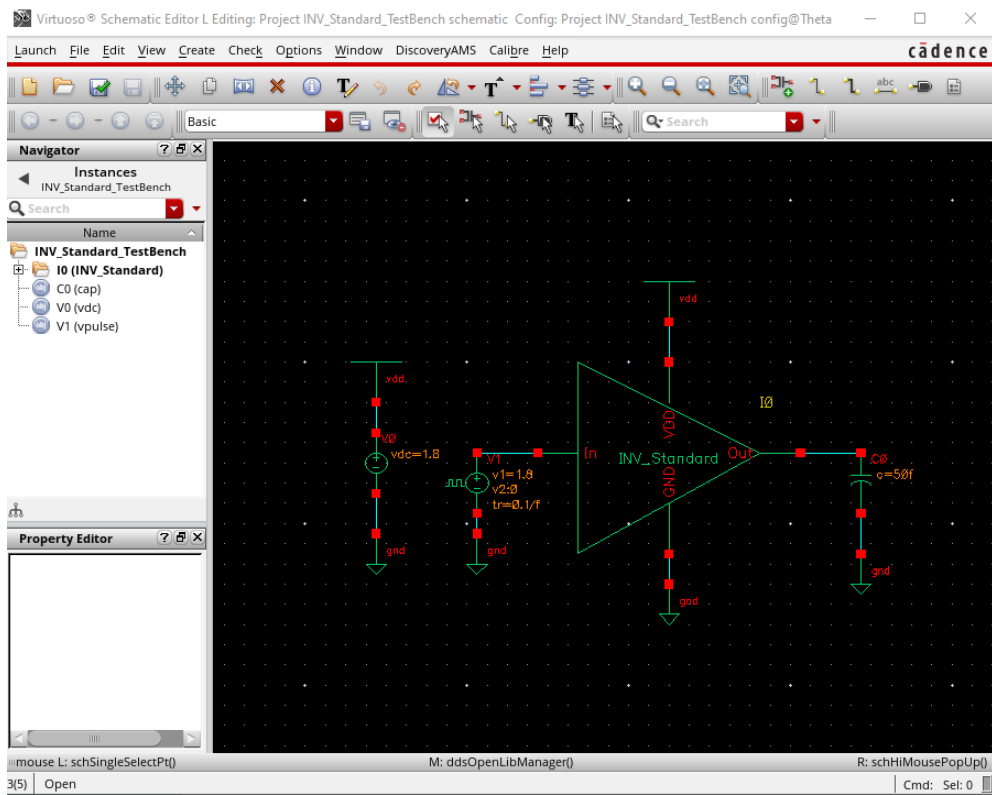
INV Gate (Kyung Hwan 'David' Lee)

Schematic



The minimum size of a NMOS or PMOS transistor in the project is assumed to be $W/L = 2\mu/180\text{n}$. All sizing is calculated based on this assumption. The PMOS transistor is sized to be 2.5x larger than the NMOS minimum size to ensure that the VTC curve of the inverter is balanced. The following simulation is captured before layout.

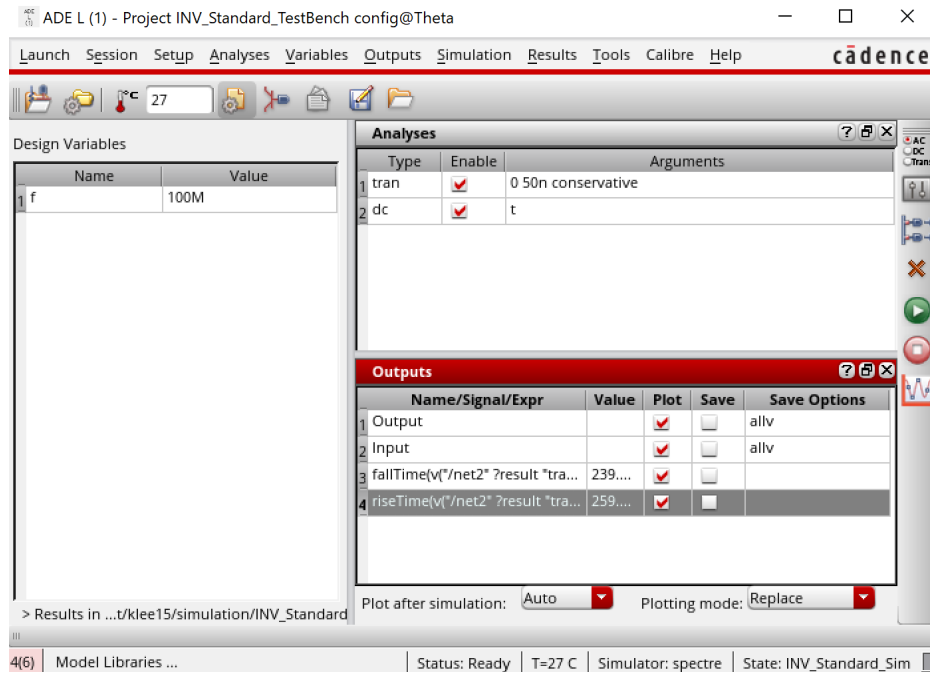
Simulation



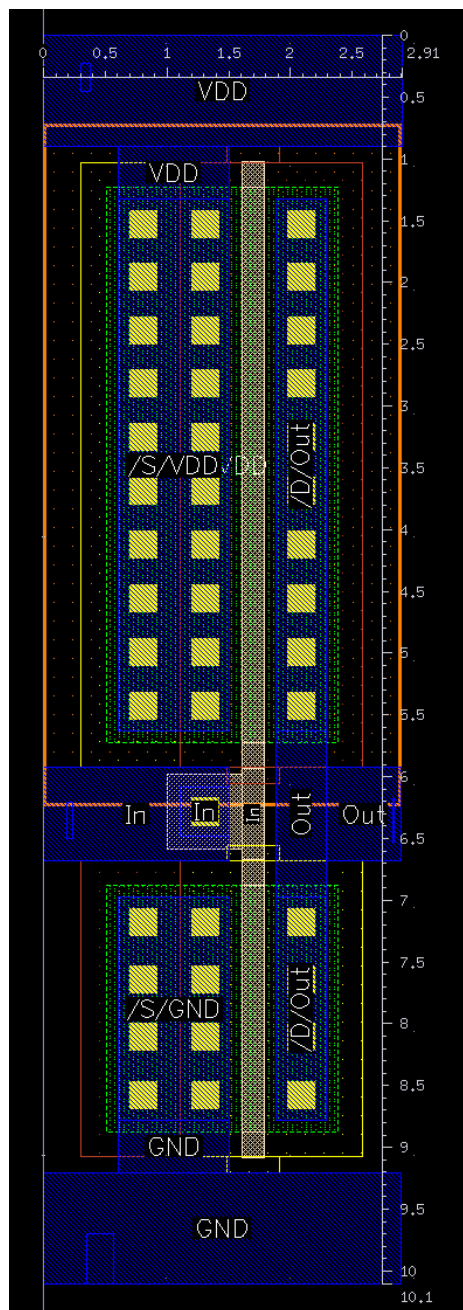
The calculated rise and fall time are as follows:

Rise Time = 259.505 ps

Fall Time = 239.539 ps



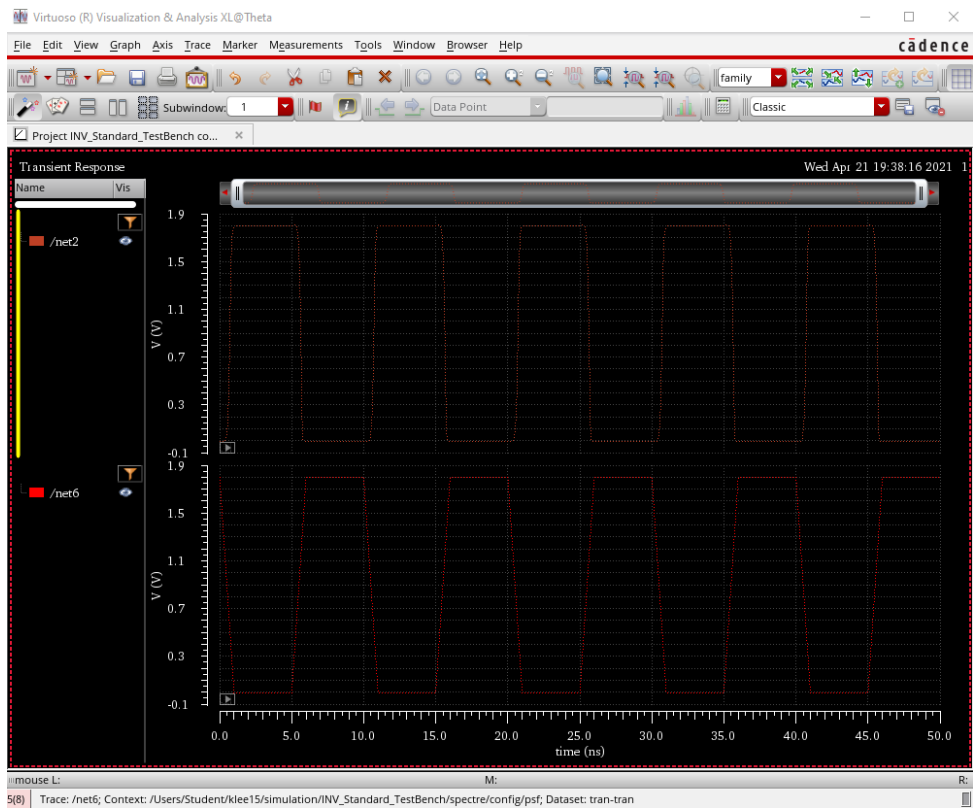
Layout



The calculated area of the INV gate is $10.1\mu\text{m} \times 2.91\mu\text{m}$ or $29.391 \mu\text{m}^2$.

Post-Layout Simulation

The following was simulated post layout using SPICE.



The figure shows the ADE L (1) configuration window in Virtuoso. It displays the simulation analysis and output results. The analysis table shows two analyses: 'tran' (transient) and 'dc' (DC). The output table shows four outputs: 'Output', 'Input', 'fallTime', and 'riseTime'. The calculated rise time is 263.563 ps and the fall time is 241.239 ps.

Type	Enable	Arguments
1 tran	<input checked="" type="checkbox"/>	0 50n conservative
2 dc	<input checked="" type="checkbox"/>	t

Name/Signal/Expr	Value	Plot	Save	Save Options
1 Output		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
2 Input		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
3 fallTime(v("/net2" ?result "tran") 1.8 nil 0 nil 10 90 nil ...)	241.239p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
4 riseTime(v("/net2" ?result "tran") 0 nil 1.8 nil 10 90 nil...)	263.563p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Plot after simulation: **Auto** Plotting mode: **Replace**

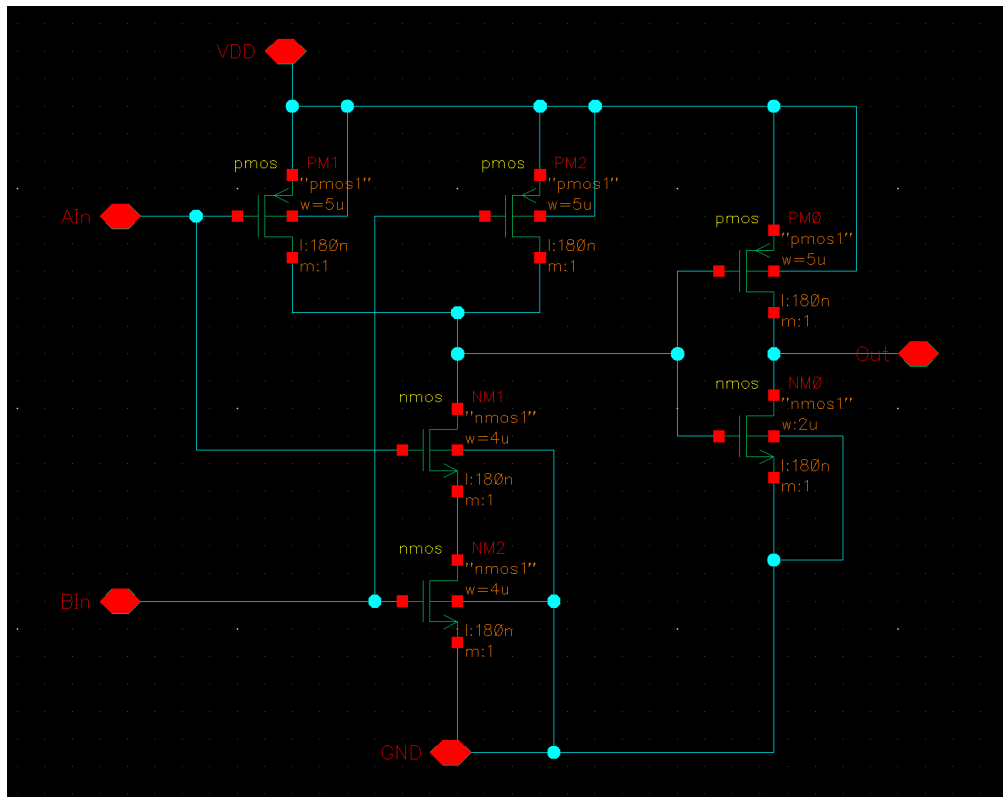
The calculated rise time and fall time are as follows.

Rise Time = 263.563 ps

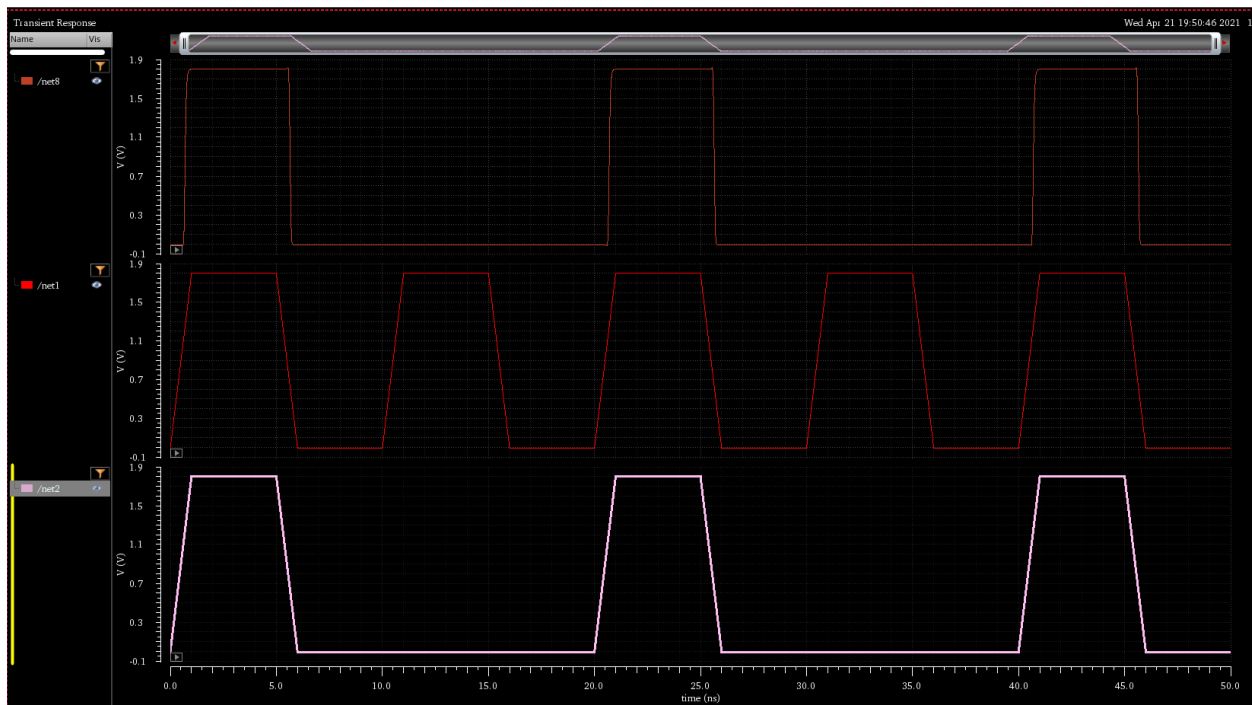
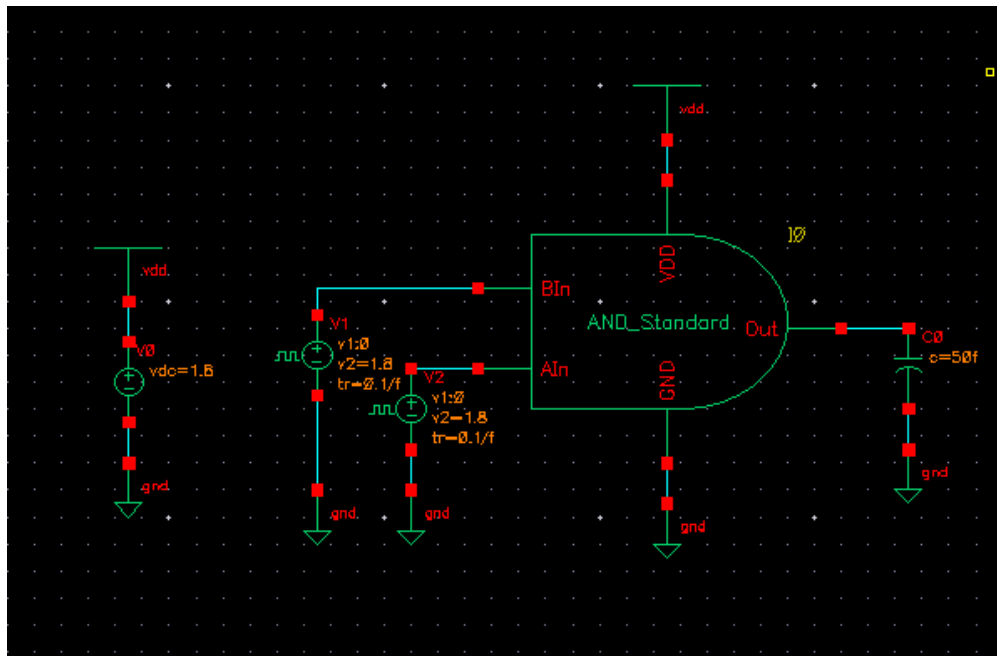
Fall Time = 241.239 ps

AND Gate (Kyung Hwan 'David' Lee)

Schematic



Simulation



Outputs					
	Name/Signal/Expr	Value	Plot	Save	Save Options
1	net8		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
2	net1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
3	net2		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
4	I0/VDD		<input type="checkbox"/>	<input type="checkbox"/>	no
5	I0/GND		<input type="checkbox"/>	<input type="checkbox"/>	no
6	riseTime(v("/net8" ?result "tran") 0 nil 1.8 nil 10 ...	123.93p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
7	fallTime(v("/net8" ?result "tran") 1.8 nil 0 nil 10 9...	100.816p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

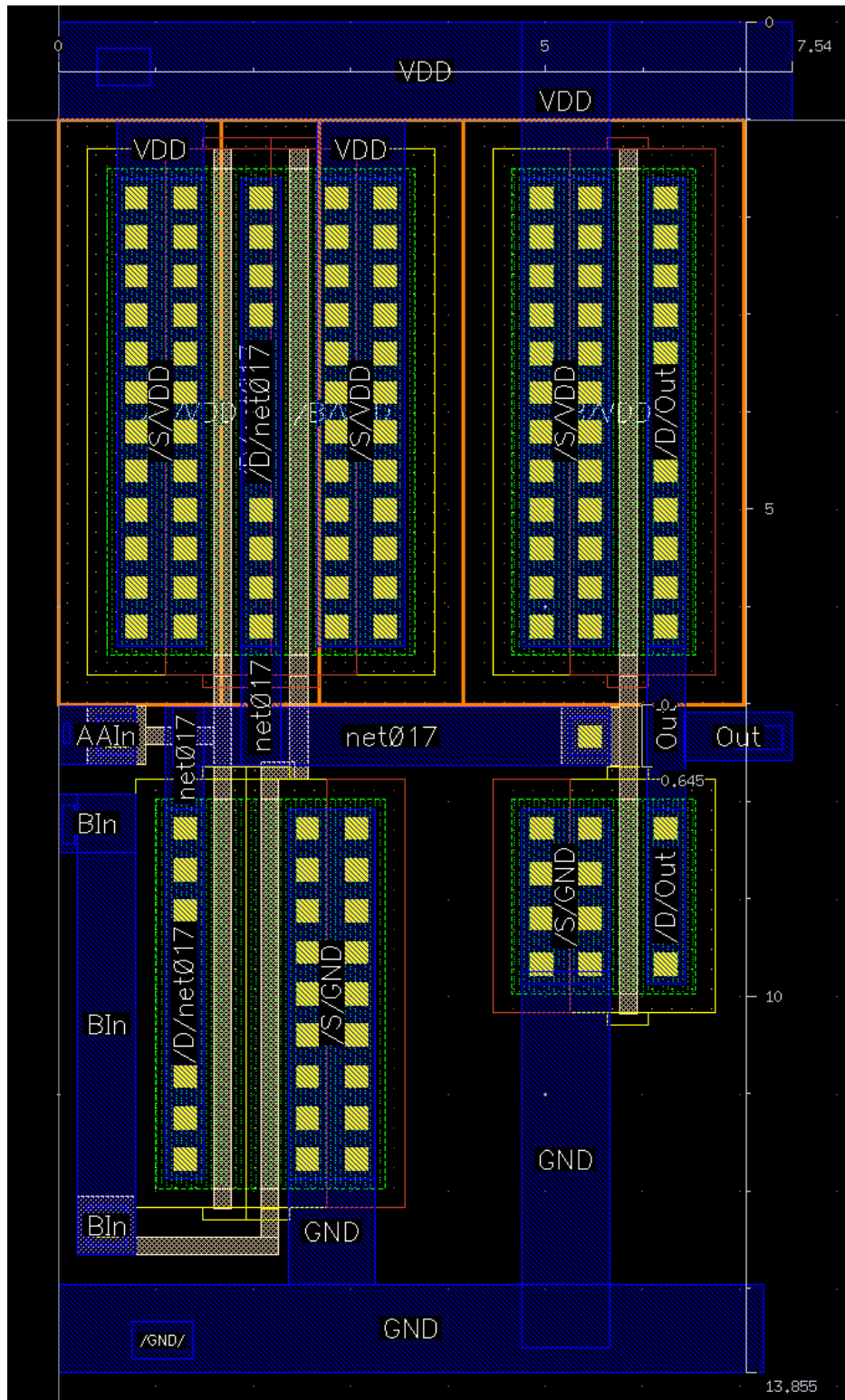
The calculated Rise Time and Fall Time of the AND Gate are as follows.

Rise Time = 123.93 ps

Fall Time = 100.816 ps

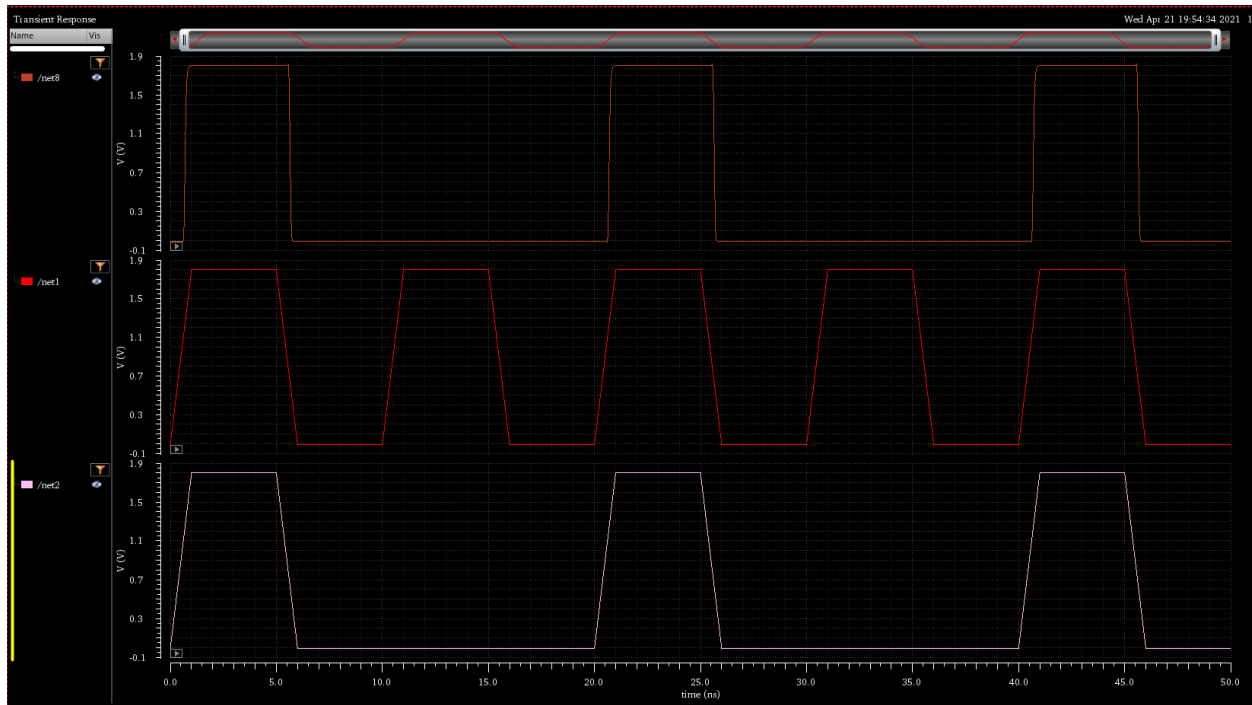
The propagation delay is calculated to be 200.81 ps.

Layout



The calculated area of the AND gate is 13.855 μm x 7.54 μm or 104.4667 μm^2 .

Post-Layout Simulation



	Name/Signal/Expr	Value	Plot	Save	Save Options
1	net8		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
2	net1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
3	net2		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
4	I0/VDD		<input type="checkbox"/>	<input type="checkbox"/>	no
5	I0/GND		<input type="checkbox"/>	<input type="checkbox"/>	no
6	riseTime(v("/net8" ?result "tran") 0 nil 1.8 nil 10 ...	126.434p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
7	fallTime(v("/net8" ?result "tran") 1.8 nil 0 nil 10 9...	102.133p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

The calculated Rise Time and Fall Time are as follows.

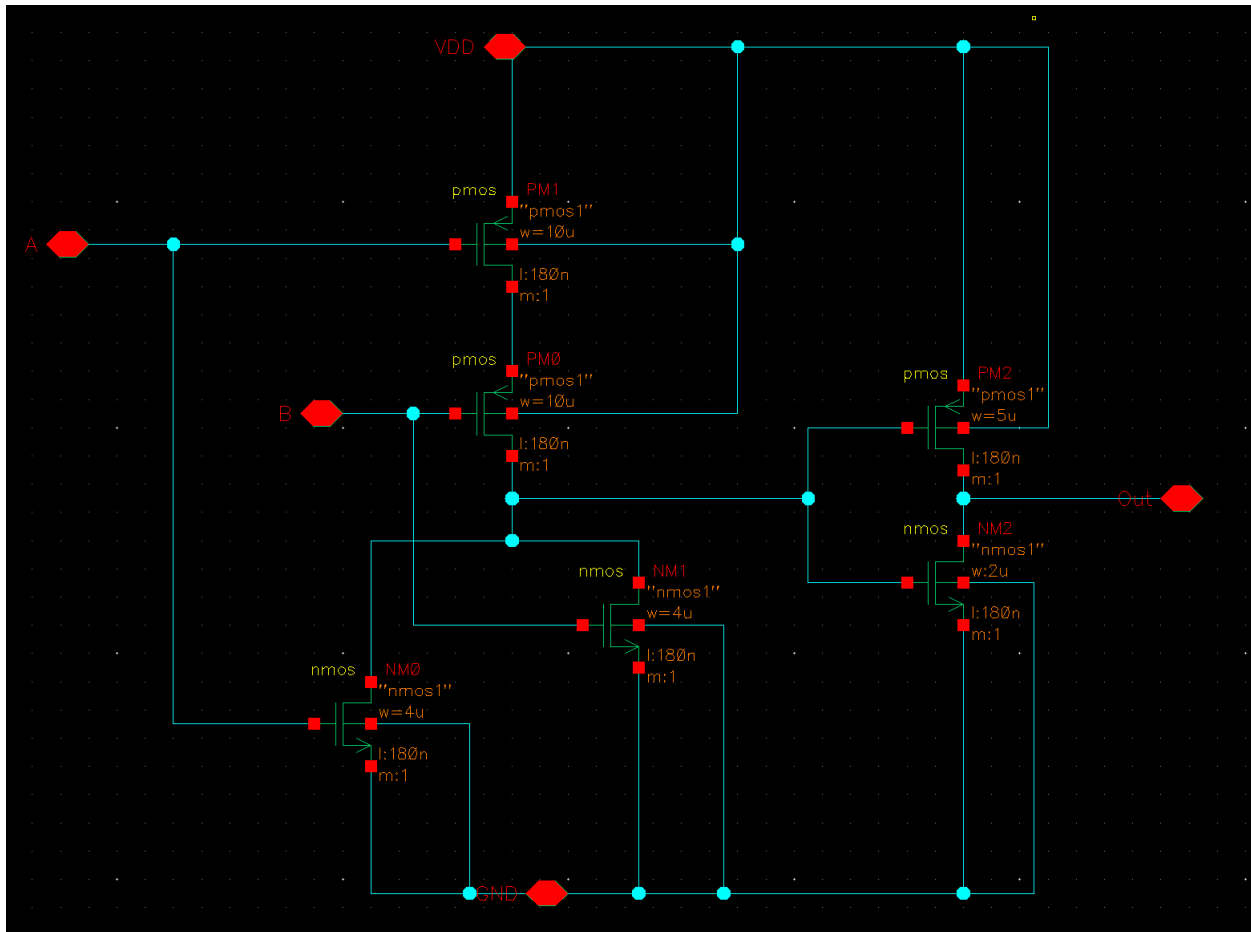
Rise Time = 126.434 ps

Fall Time = 102.133 ps

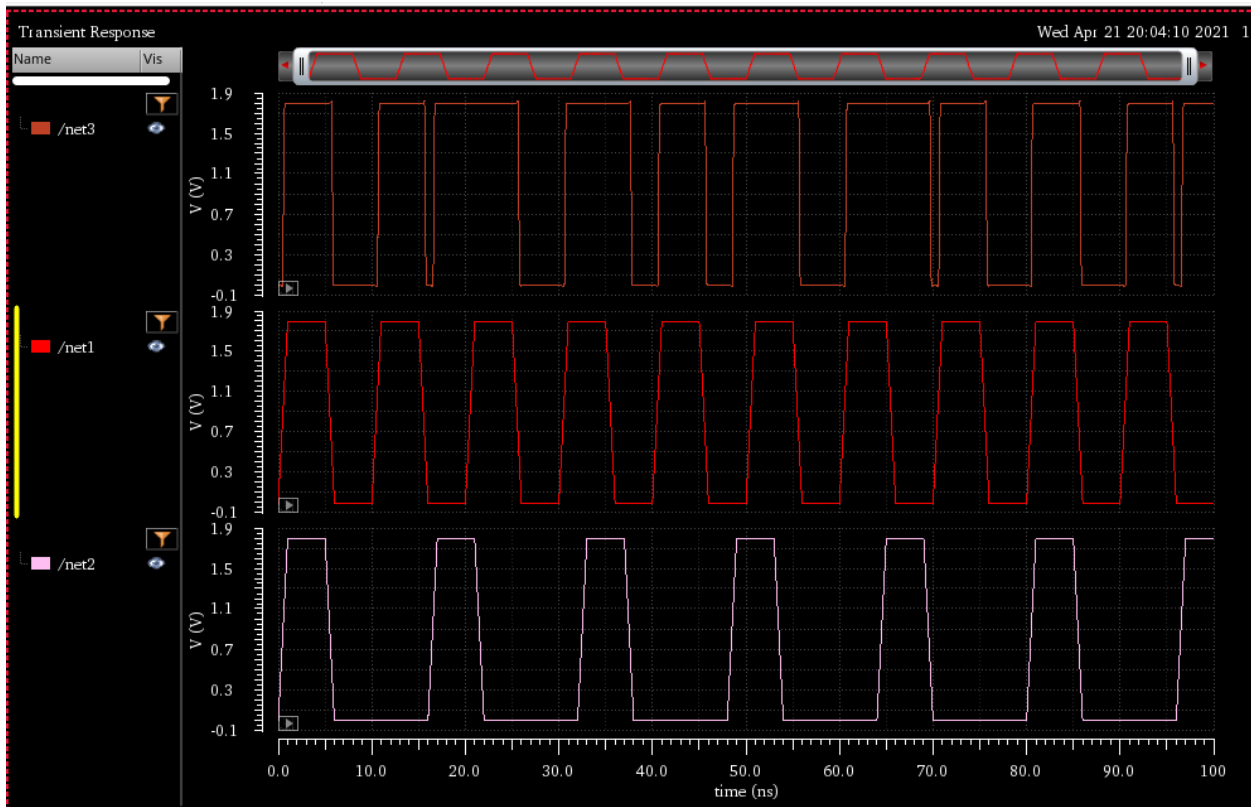
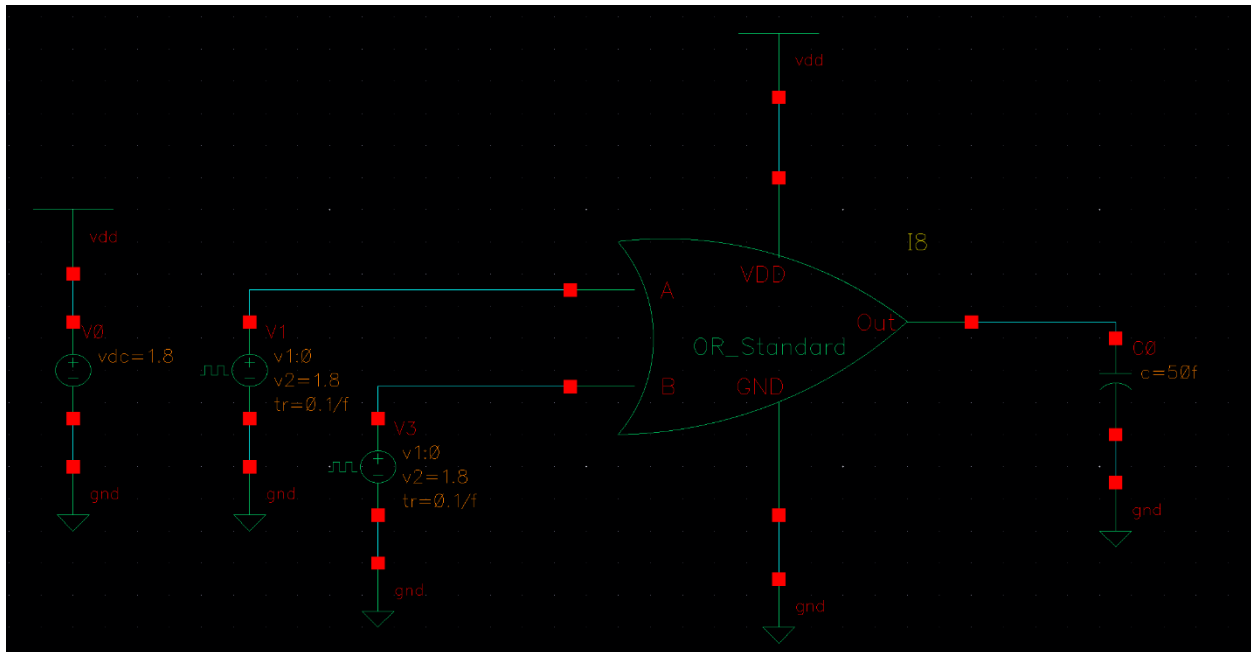
The calculated Propagation Delay is 202.33 ps.

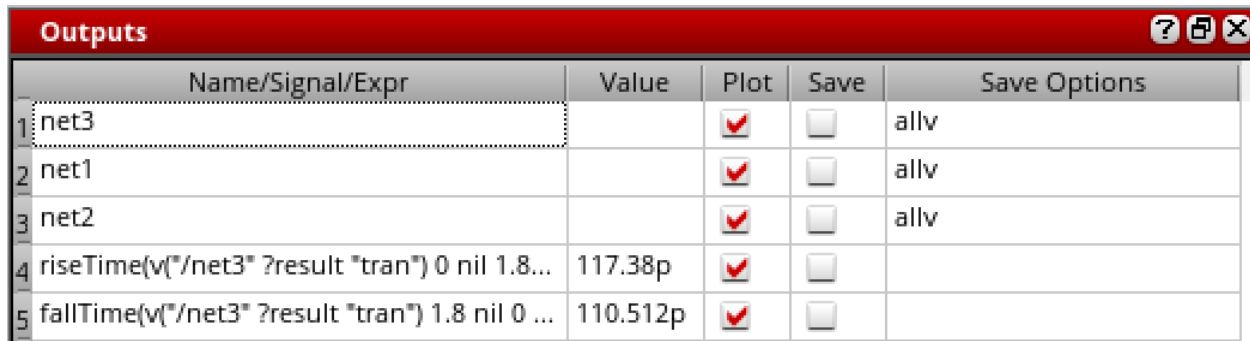
OR Gate (Kyung Hwan 'David' Lee)

Schematic



Simulation





	Name/Signal/Expr	Value	Plot	Save	Save Options
1	net3		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
2	net1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
3	net2		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
4	riseTime(v("/net3" ?result "tran") 0 nil 1.8...	117.38p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5	fallTime(v("/net3" ?result "tran") 1.8 nil 0 ...	110.512p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

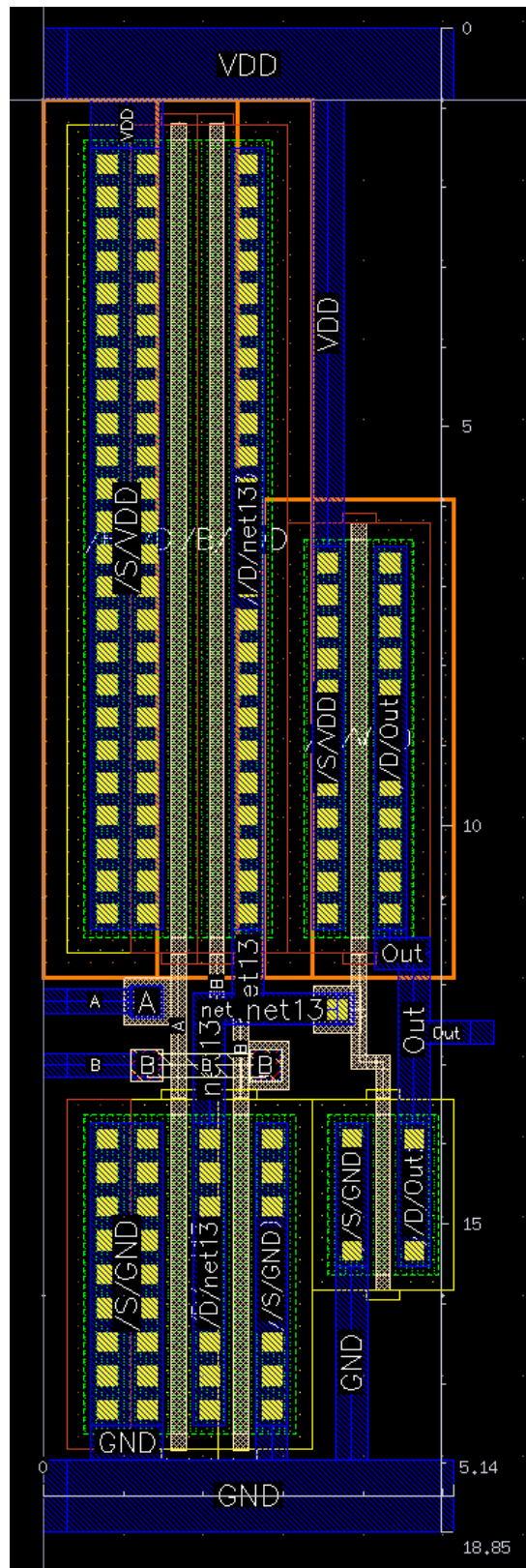
The calculated Rise Time, Fall Time, and Propagation Delay are as follows.

Rise Time = 117.38 ps

Fall Time = 110.512 ps

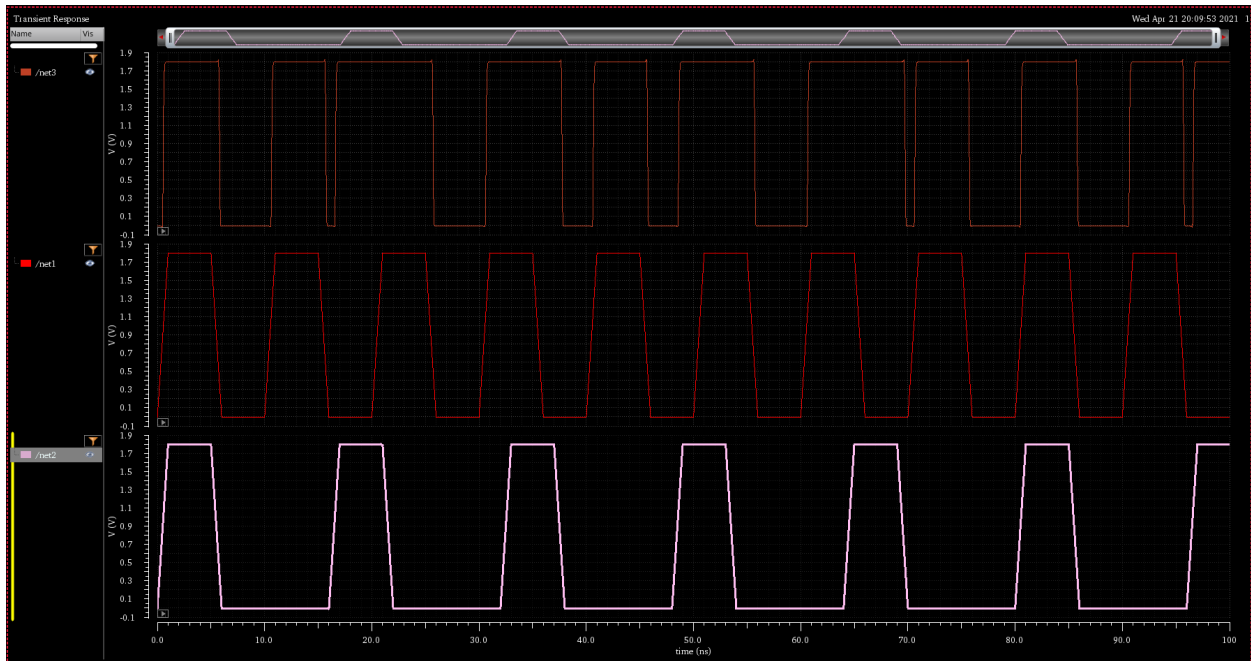
Propagation Delay = 49.3438 ps

Layout



The calculated area of the OR gate is 18.85 μm x 5.14 μm or 96.889 μm^2 .

Post-Layout Simulation



Outputs					
	Name/Signal/Expr	Value	Plot	Save	Save Options
1	net3		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
2	net1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
3	net2		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
4	riseTime(v("/net3" ?result "tran") 0 nil 1.8...	120.198p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5	fallTime(v("/net3" ?result "tran") 1.8 nil 0 ...	112.327p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

The calculated Rise Time, Fall Time, and Propagation Delay is as follows.

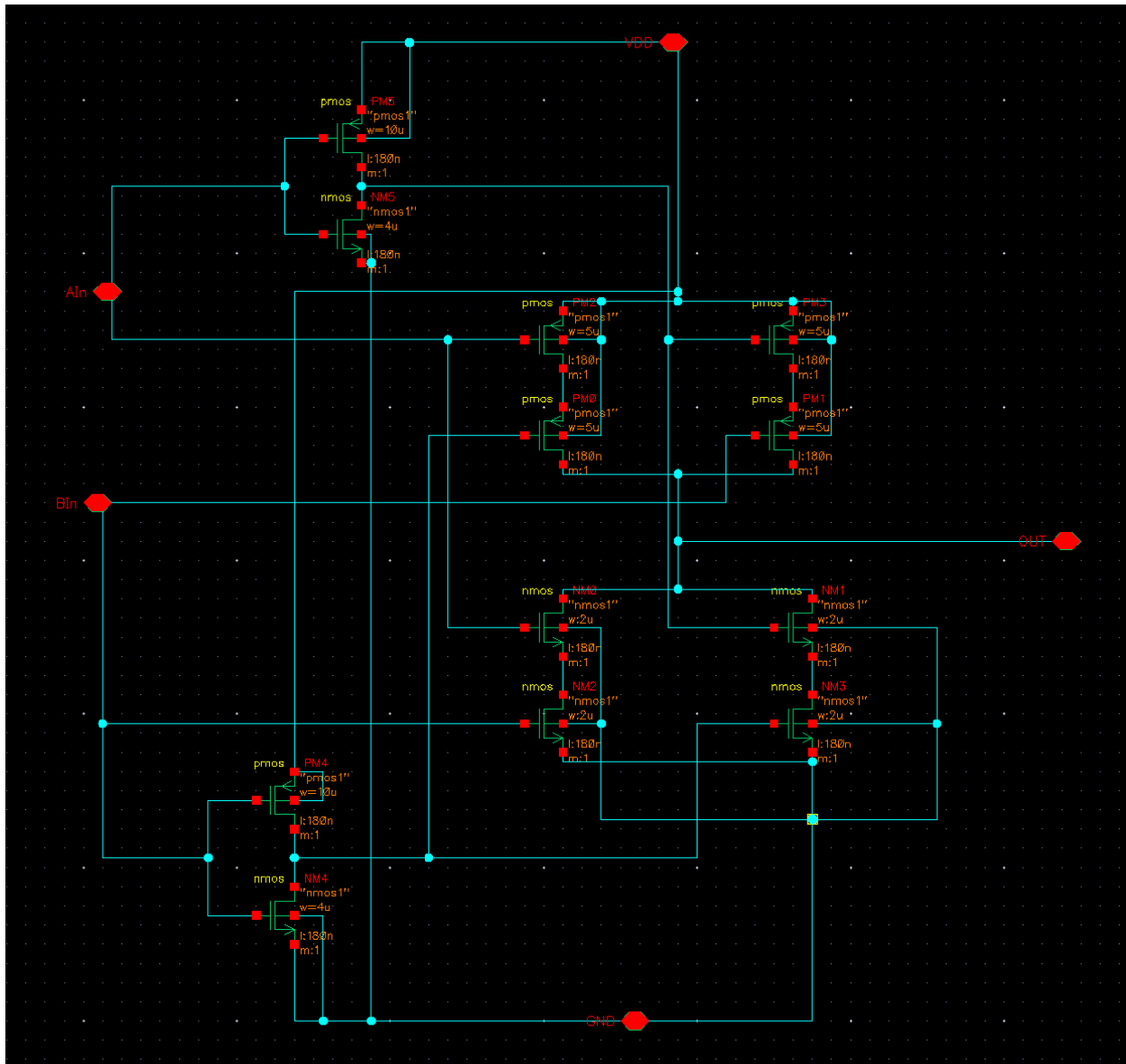
Rise Time = 120.198 ps

Fall Time = 112.327 ps

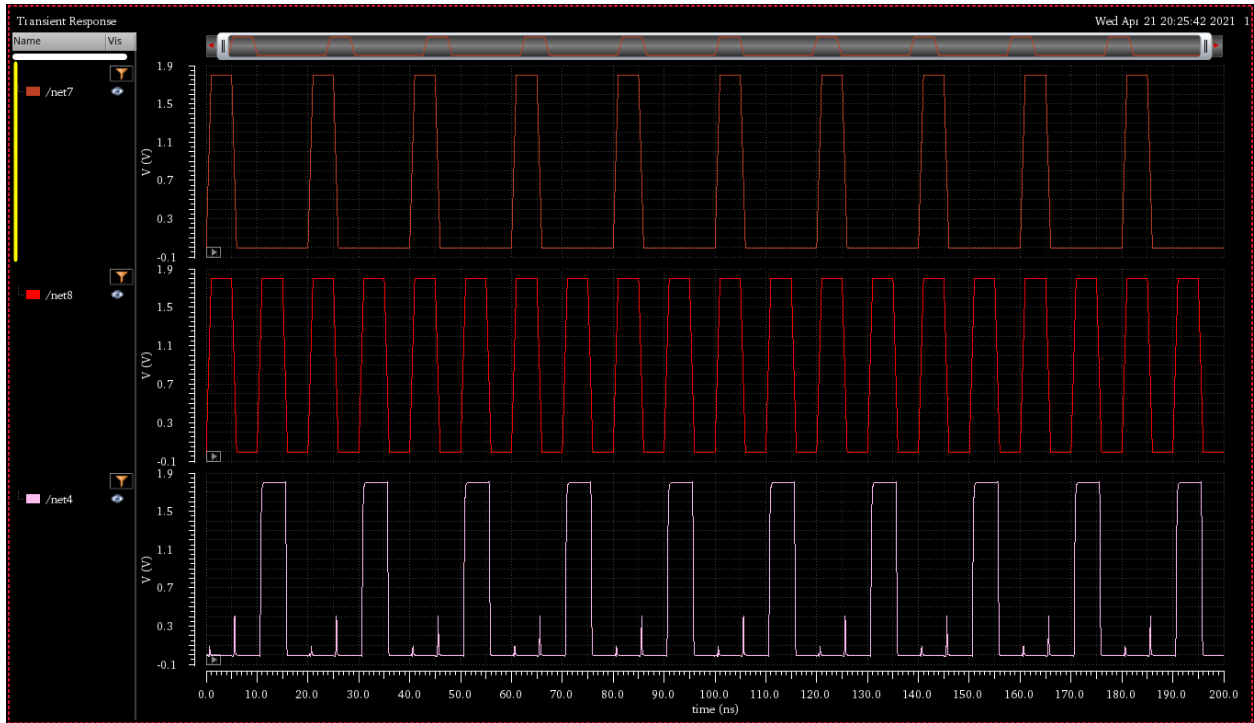
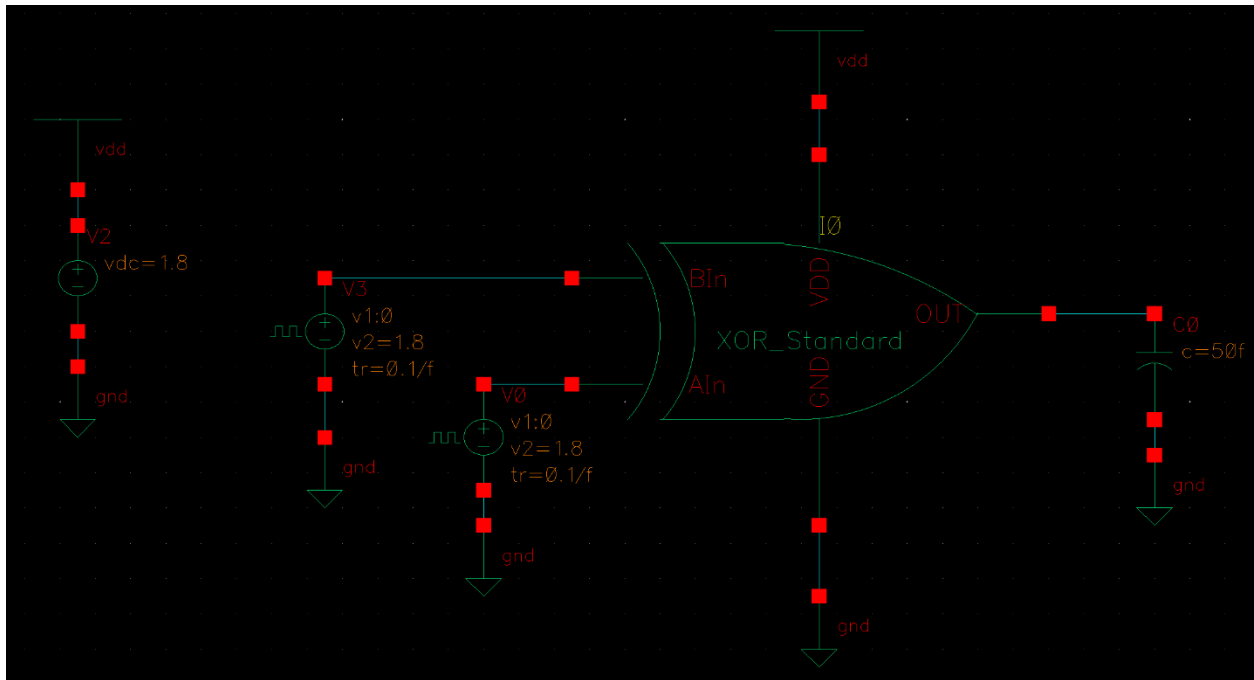
Propagation Delay = 54.538 ps

XOR Gate (Kyung Hwan 'David' Lee)

Schematic



Simulation



Outputs					
	Name/Signal/Expr	Value	Plot	Save	Save Options
1	net7		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
2	net8		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
3	net4		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
4	fallTime(v("/net4" ?result "tran") 1.8 nil 0 nil 10 90...	194.92p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5	riseTime(v("/net4" ?result "tran") 0 nil 1.8 nil 10 90...	270.592p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

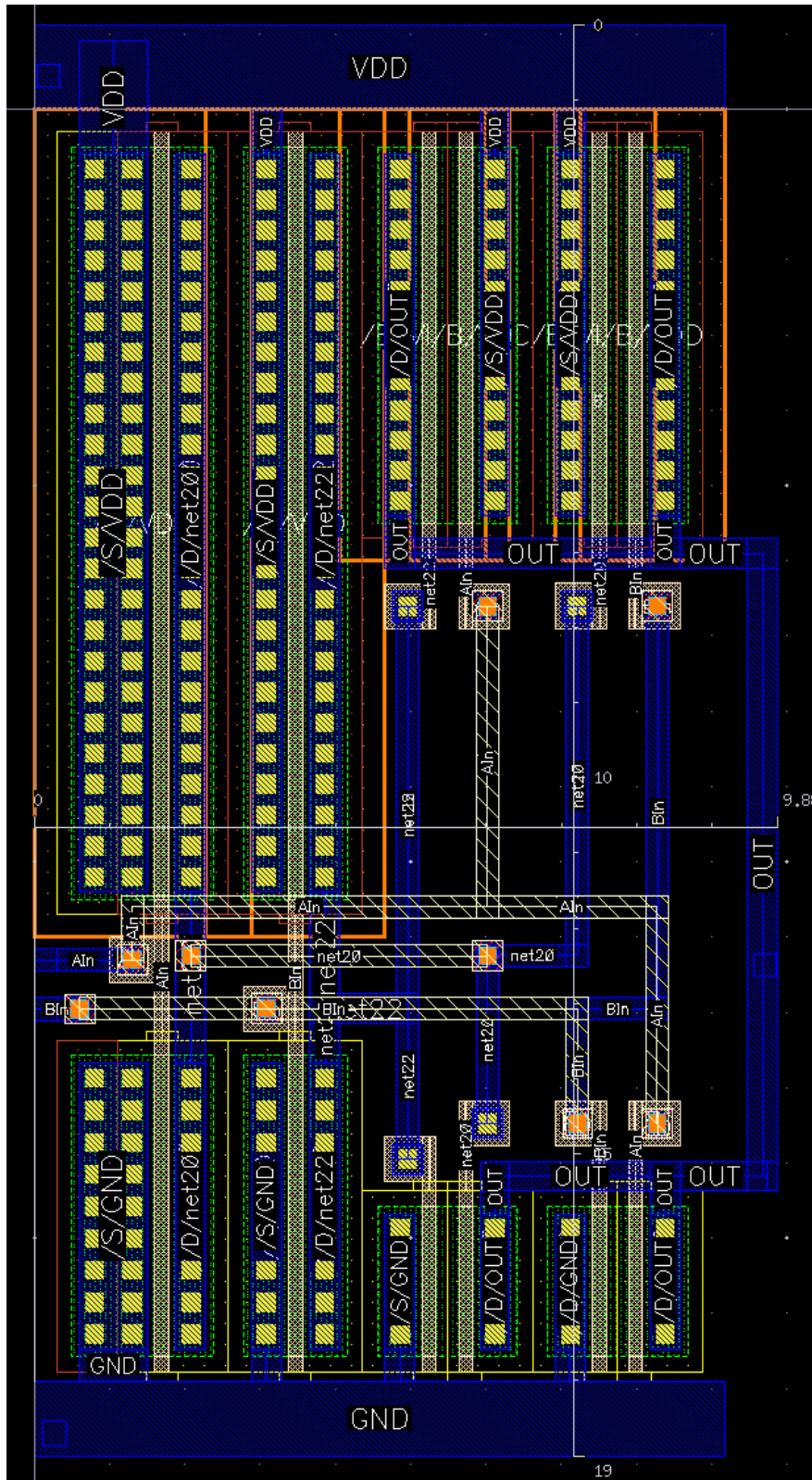
The Rise Time, Fall Time, and Propagation Delay are calculated to be the following.

Rise Time = 270.592 ps

Fall Time = 194.92 ps

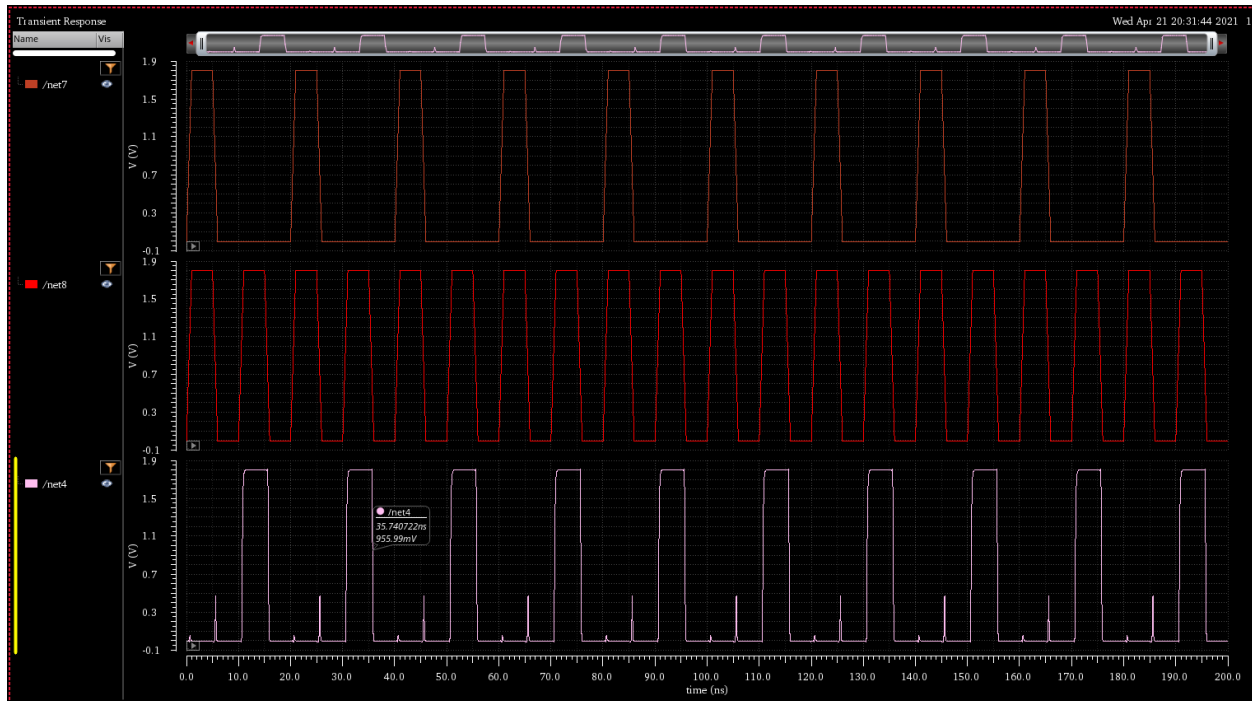
Propagation Delay = 213.368 ps

Layout



The calculated area of the XOR Gate is 19 μm x 9.88 μm or 187.72 μm^2 .

Post-Layout Simulation



Outputs					
	Name/Signal/Expr	Value	Plot	Save	Save Options
1	net7		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
2	net8		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
3	net4		<input checked="" type="checkbox"/>	<input type="checkbox"/>	allv
4	fallTime(v["/net4" ?result "tran"] 1.8 nil 0 nil 10 90 nil ...	197.425p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5	riseTime(v["/net4" ?result "tran"] 0 nil 1.8 nil 10 90 nil ...	278.319p	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

The calculated Rise Time, Fall Time, and Propagation Delay are as follows.

Rise Time = 278.319 ps

Fall Time = 197.425 ps

Propagation Delay = 224.763 ps

Lessons Learned

In the simulations, it is observed that the XOR gate outputs small spikes at the output with changing inputs. This is due to the delay caused by the inverters to generate the A' and B' signals used. In future designs, we can mitigate this spike by generating simultaneous A and B complementary signals. We can reduce this spike by sizing up the inverters. This will cause the complementary signals to rise and fall faster, reducing the time where the outputs may spike. The sizing of the inverters was chosen via multiple simulations to reduce the observed output spikes. The output spikes are observed to be less than 25% of VDD, eliminating any downstream 'false' signals from propagating. The regenerating properties of Static CMOS design should eliminate these spikes from propagating forward.

32-Bit Adder (Kyung Hwan 'David' Lee)

Constraints & Functionality Definition

The 32-Bit adder design and implementation was decided upon based on the following constraints.

1. Design Complexity
2. Estimated Time of Completion

The Design Complexity constraint is loosely defined as:

Design Complexity – The perceived and researched complexity of design to be implemented.

These design constraints were chosen when selecting the implementation due to the inexperience of the designer and the limited time available to research the design, learn *Cadence Virtuoso*, and implementation.

In general, a more complex design implementation will result in faster implementation of the 32-Bit adder. The implementation will have valid outputs faster than other implementations. However, the area, transistor count, and power consumption will generally be higher. This is because of the higher transistor count due to the higher design complexity.

In addition, only static CMOS based designs were considered for the 32-Bit Adder implementation.

The functionality of the 32-Bit adder can be described in the following way.

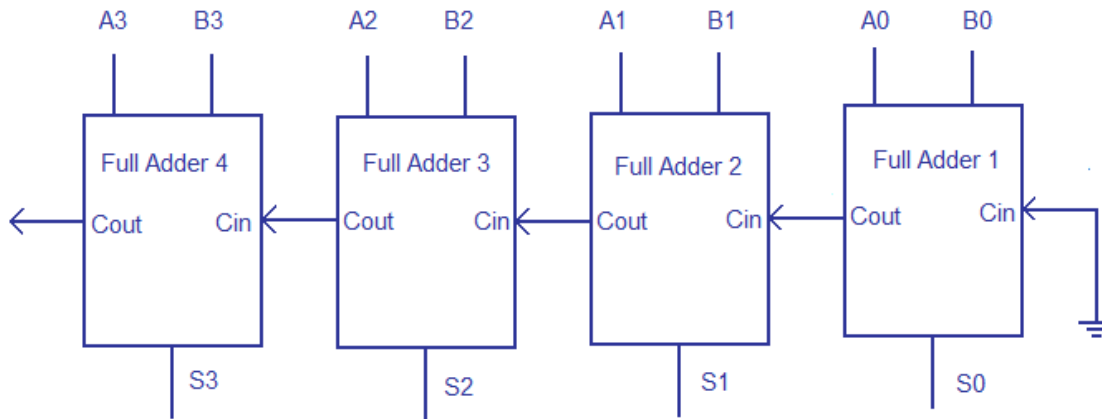
$$A_{32} + B_{32} = S_{32}$$

Design Methodologies

There are two major families of adders that can be considered for the 32-Bit Adder design: Ripple Carry Adder and Carry Look Ahead Adder.

Ripple Carry Adder

The ripple-carry adder is comprised of full adder circuits which are cascaded to N times to fulfill the addition of N bits. The ripple-carry adder has a low design complexity but is a slower implementation of the 32-bit adder because the carry bit must be generated sequentially at each N stage. Because of this characteristic, if 1 stage of an N stage ripple carry adder took x seconds to generate the output S and Carry, the ripple-carry adder will take at least $x * N$ time.



4 bit ripple carry adder

www.circuitstoday.com

[1]

A Full Adder stage can be made by cascading 2 half-adder circuits together.

Carry Look Ahead Adder

A carry look ahead adder, or CLA, is an adder that improves upon the ripple carry adder in speed by eliminating the need to ‘wait’ for each stage’s *Carry* bit generation. The *Carry* bit of all the stages can be generated simultaneously and used simultaneously to compute the summation of the input bits.

The basic operation of the CLA adder is described as the following [2]. We can observe the truth table of a Full Adder and determine modes of operation, in terms of the *Carry* bit’s generation and propagation.

A	B	C_i	C_o	Type of Carry
0	0	0	0	None
0	0	1	0	None
0	1	0	0	None
0	1	1	1	Propagate
1	0	0	0	None
1	0	1	1	Propagate
1	1	0	1	Generate
1	1	1	1	Generate/Propagate

[2]

The types of *Carry* bit is described as Propagate, Generate, and Generate/Propagate. The carry bit will be generated only if A and B are ‘On’, ie. The carry output bit will only be generated if the A and B term will result in a carry over.

The Propagated Carry bit is conceptual a carry bit that should be considered for the following stages. ie. For each stage or bit on the bus, should the generated carry bit be considered during the calculation.

The following equations summarize the general function of a CLA [2].

$$P_i = A_i \oplus B_i$$

$$G_i = A_i \cdot B_i$$

The valid outputs of the CLA are summarized in the following equations [2].

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i \cdot C_i$$

In this way, all the propagate and generate signals can be computed simultaneously for multiple stages of the CLA. With these generated signals, the Sum and Carry Out can be computed simultaneously for multiple stages.

This basic concept of generating multiple *Carry* bits from multiple stages simultaneously can be implemented in a variety of ways. The *Kogge-Stone Adder*, *Brent-Kung Adder*, and *Ling Adder* are all examples of CLA implementations. These adders improve upon the CLA implementation by parallelizing the CLA operation.

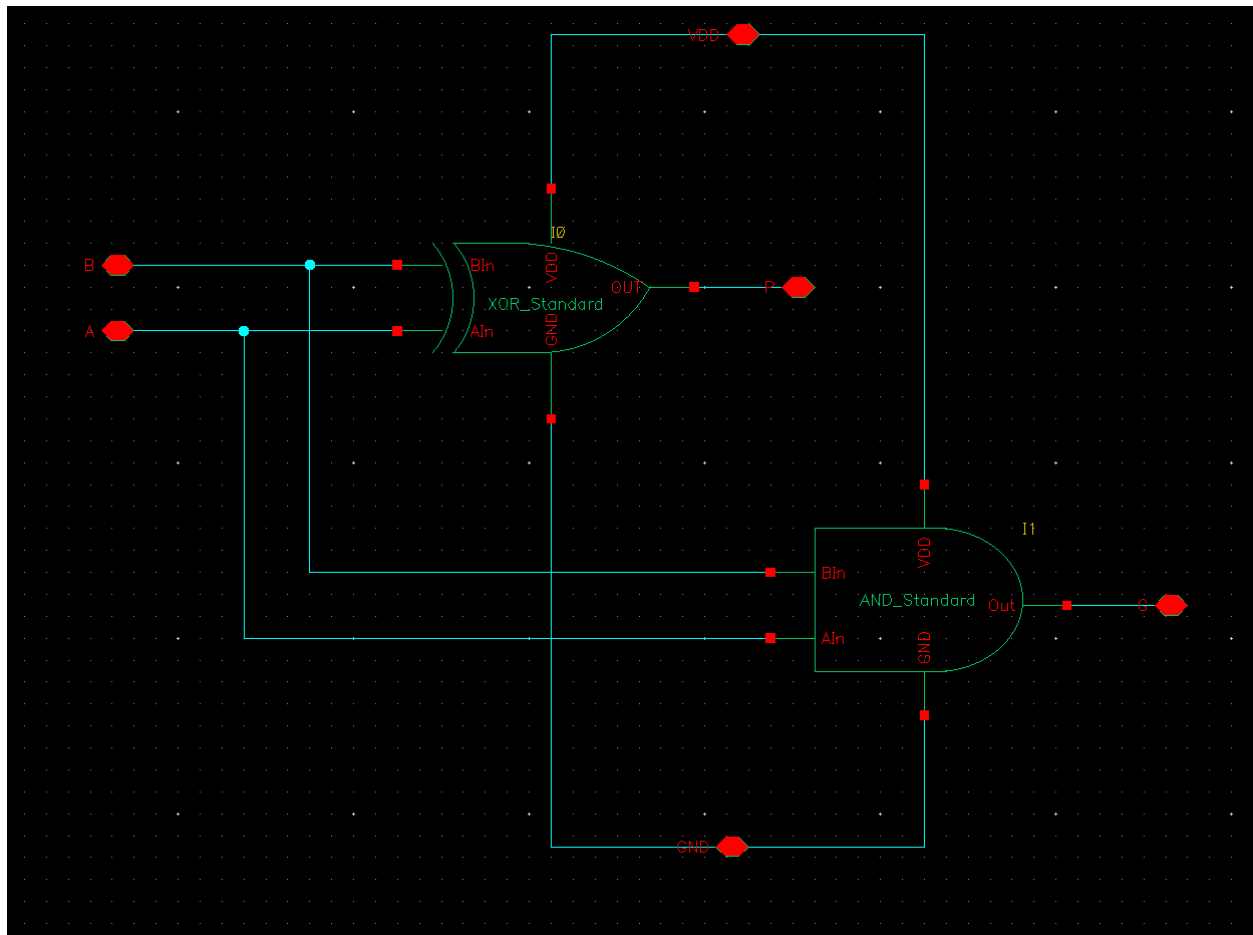
We can explore how the CLA adder can be further optimized by looking at the *Ling Adder* as described in *High Speed Binary Parallel Adder* [3]. The *Ling Adder* optimizes the CLA by grouping stages of the CLA together. By grouping these stages together, another intermediary term of *Group Generate* and *Group Propagate* can be generated. These terms can be used to simultaneously calculate a larger amount of CLA stages without exceeding gate input limitations.

Chosen Design Methodology

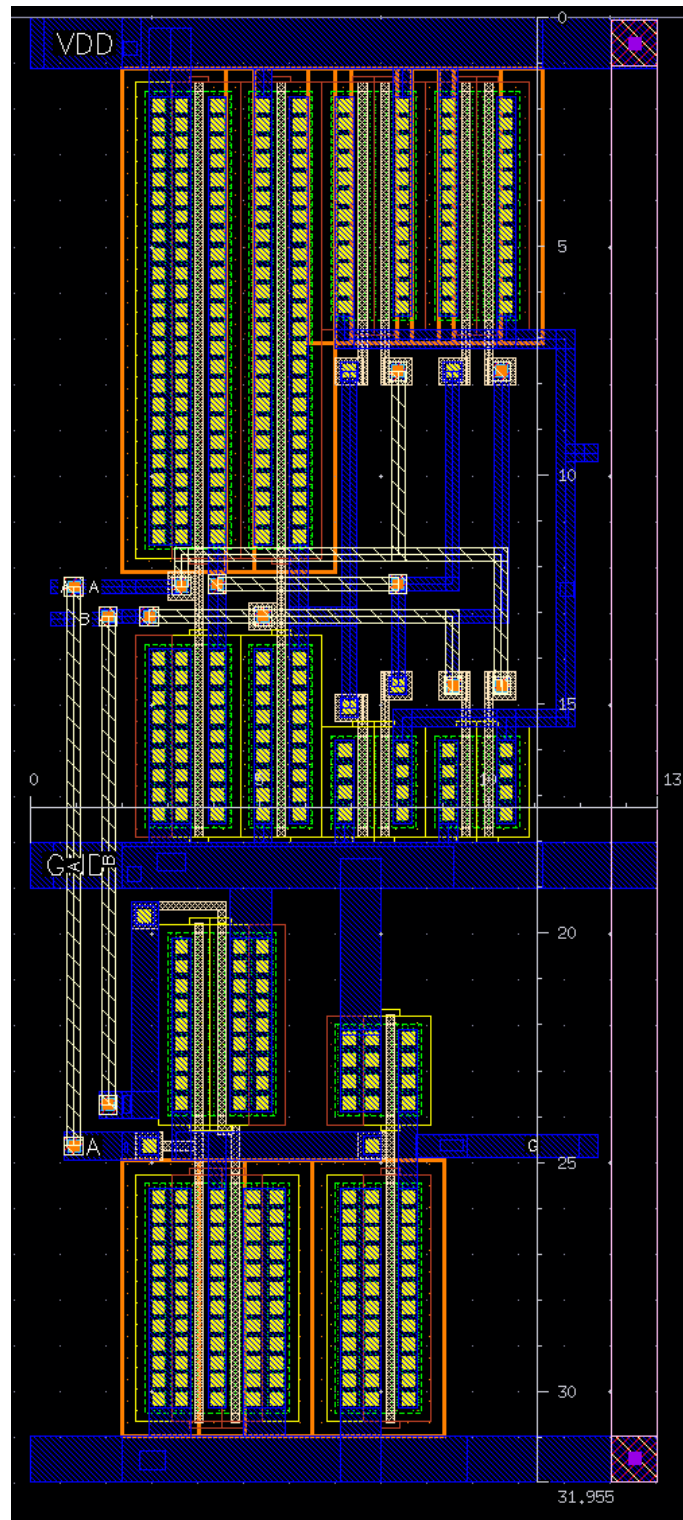
In this project, the adder will take themes from both the Ripple Carry Adder and the Carry Look Ahead Adder. In the implemented 32-bit Adder design, a 4-bit Carry Look Ahead adder will be cascaded 8 times. Theoretically, this should result in a less complex design, which is easier to implement, but is still faster than a traditional 32-bit Ripple Carry Adder.

Conceptually, 4 input bits and the *Carry* bit for this block will be generated simultaneously. The first carry bit will be used as the following carry input bit. Instead of computing sums sequentially 32 times, the implemented design will sequentially compute 4 bits only 8 times.

Propagate and Generate Block
Schematic

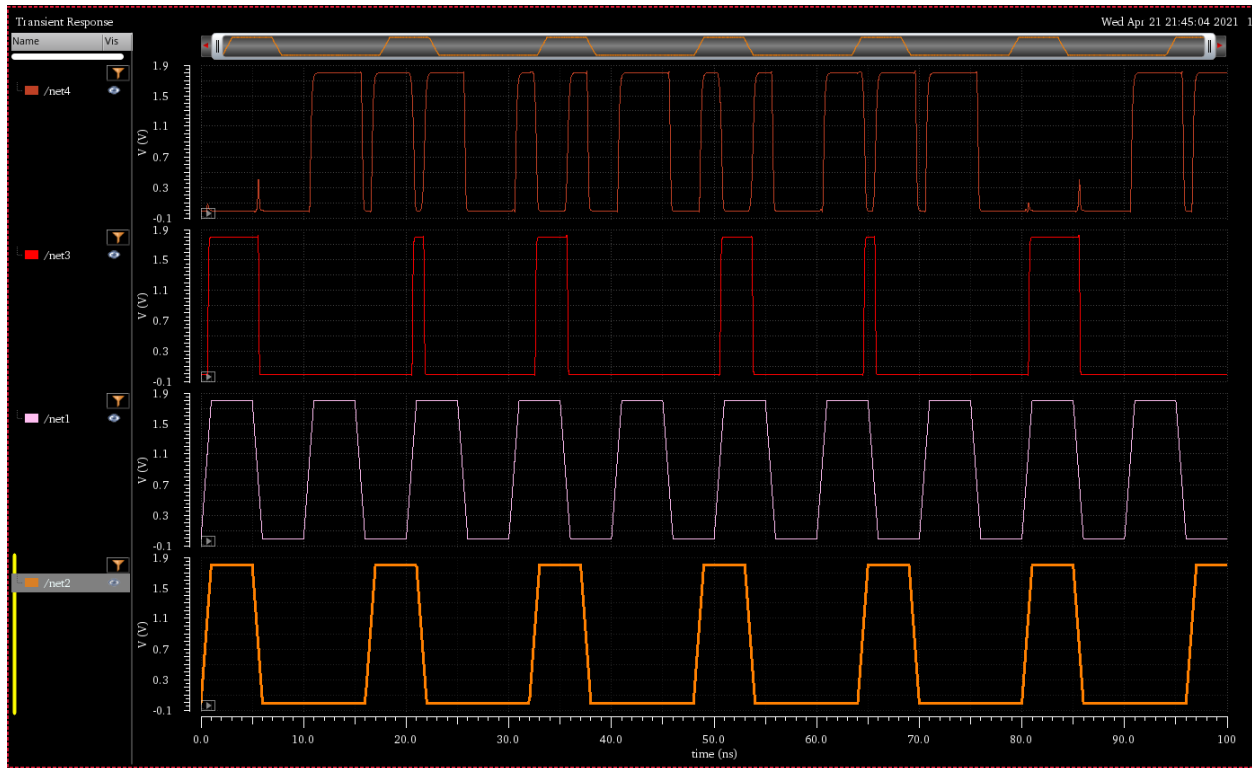


Layout



The calculated area of the Propagate & Generate Block is 31.995 μm x 13.68 μm or 437.6916 μm^2 .

Post-Layout Simulation



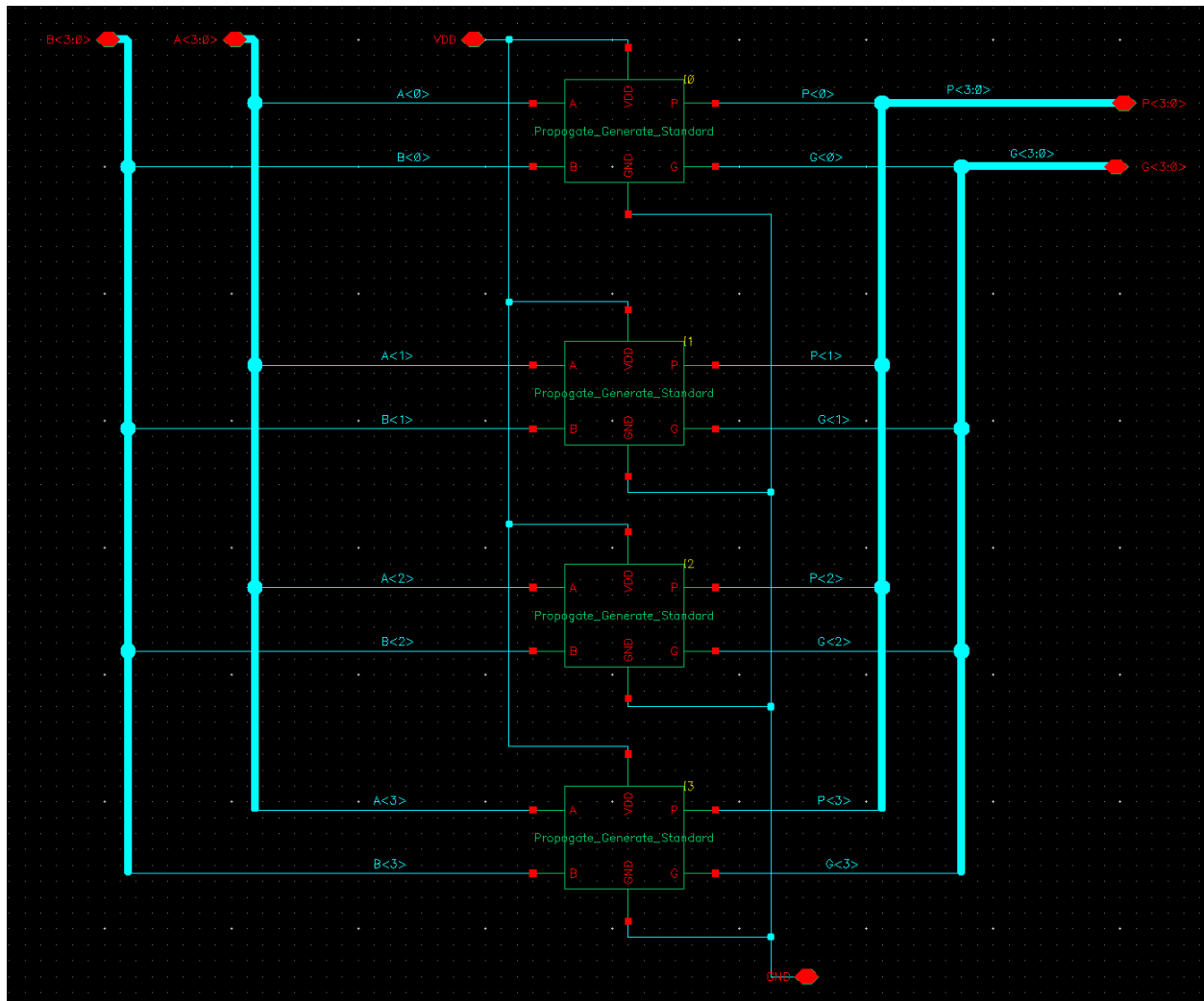
The propagate and generate signals can be validated using the above simulation.

4 – Bit Propagate Generate Block

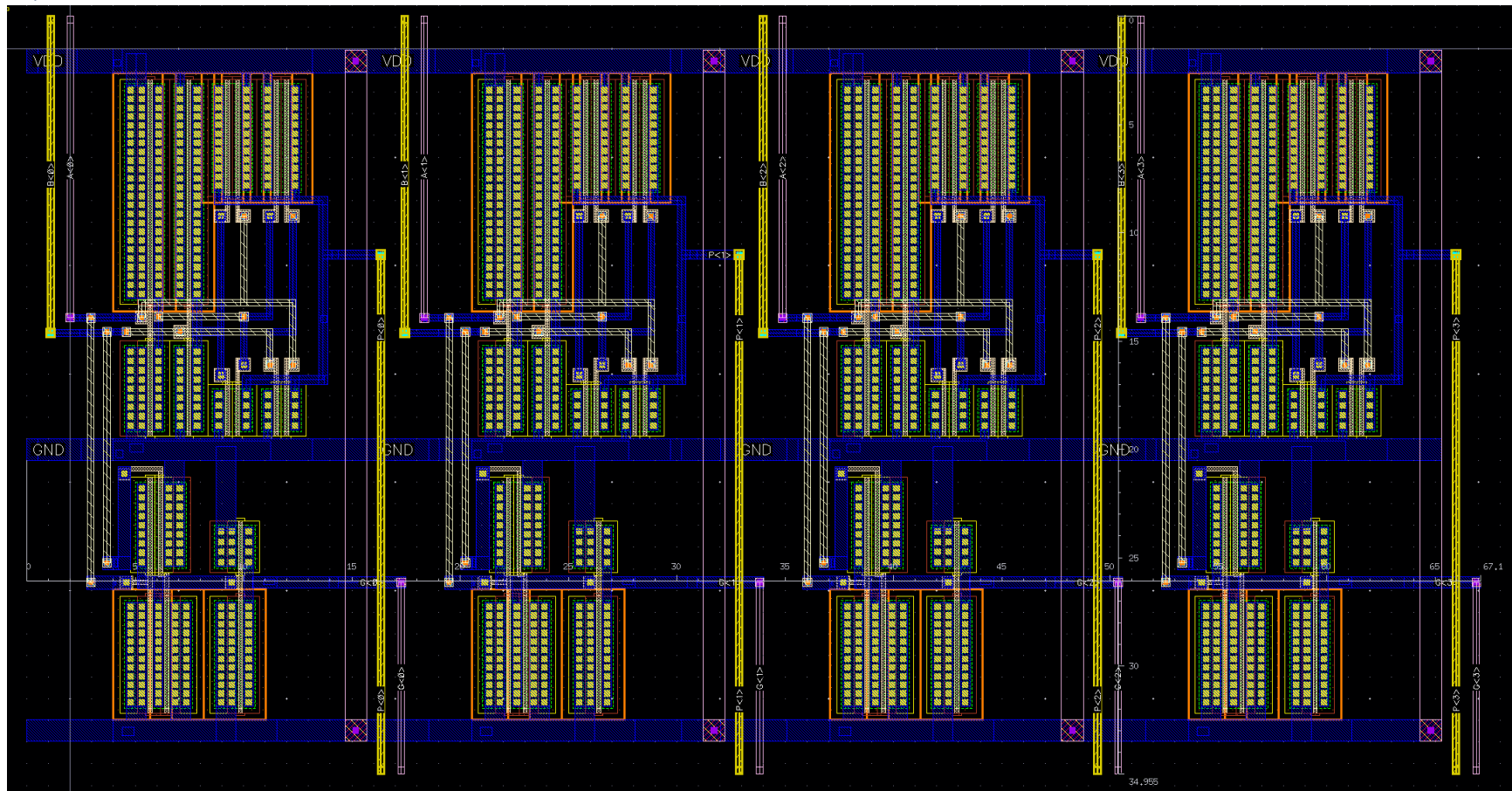
Scope

To help aid in the design flow of the 32 – Bit CLA adder, a 4 bit Propagate Generate Block was created. This would simplify the future designs and help in the implementation of the of the 4 bit CLA adders cascaded 8 times to achieve the 32 – Bit adder.

Schematic



Layout



The calculated area of the 4 – Bit Propagate and Generate Block is 34.955 um x 67.1 um or 2345.481 um².

Post – Layout Simulation

The following test vectors were used to simulate the 4-Bit Propagate Generate Block to verify its functionality.

$$A_1 = \langle 1100 \rangle$$

$$B_1 = \langle 0011 \rangle$$

$$A_2 = \langle 0011 \rangle$$

$$B_2 = \langle 0101 \rangle$$

As described above, the Propagate and Generate Signals are expected to be as follows.

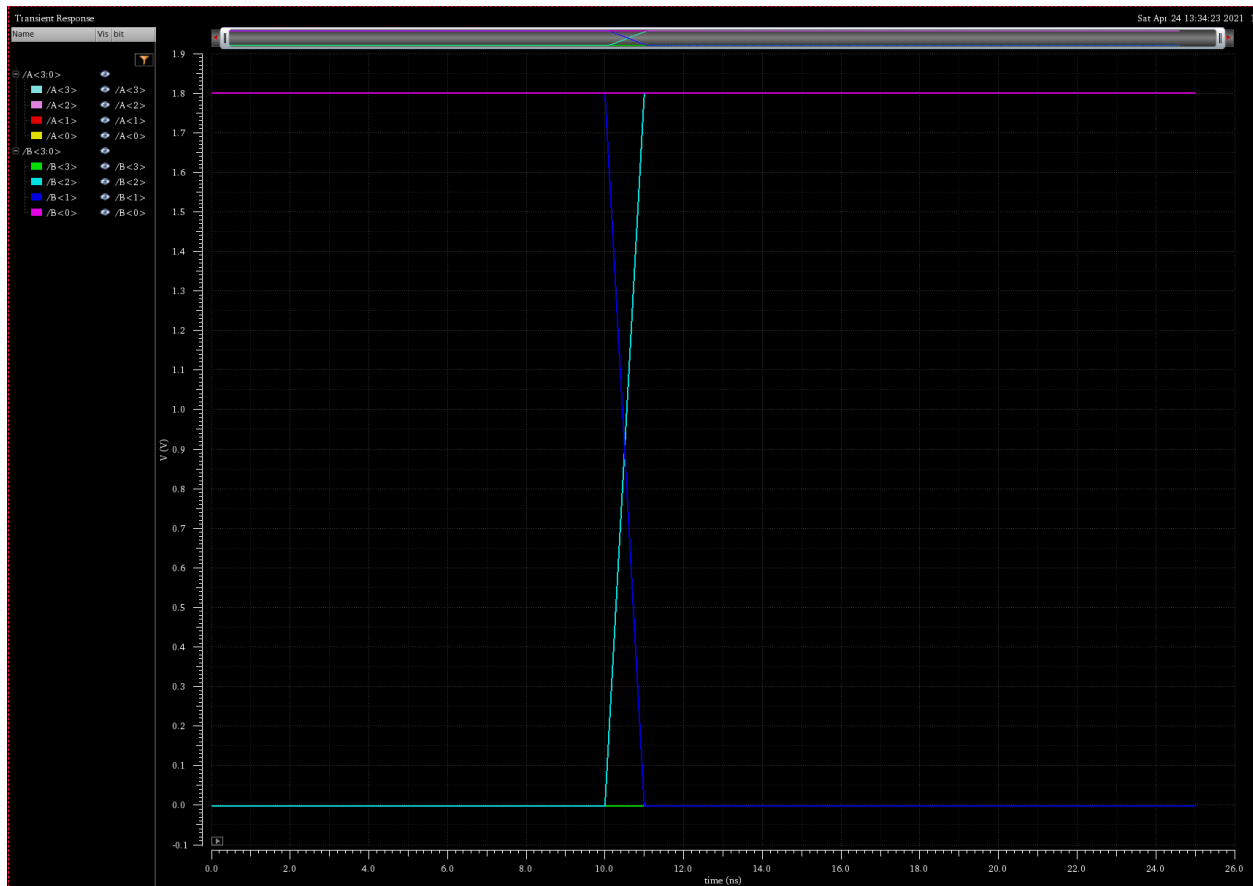
$$A_1 \oplus B_1 = P_1 = \langle 1111 \rangle$$

$$A_1 \cdot B_1 = G_1 = \langle 0000 \rangle$$

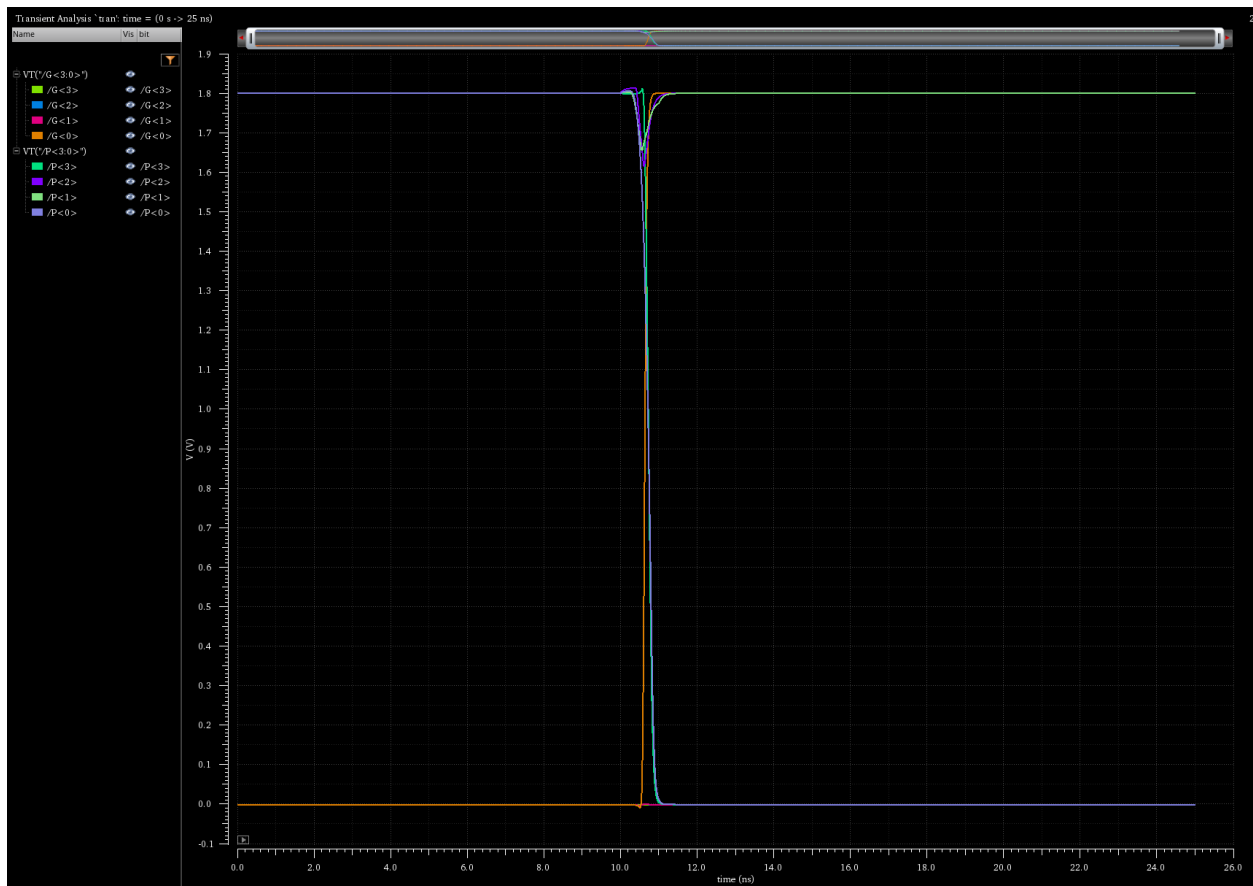
$$A_1 \oplus B_1 = P_2 = \langle 0110 \rangle$$

$$A_1 \cdot B_1 = G_2 = \langle 0001 \rangle$$

The test vectors can be seen below.

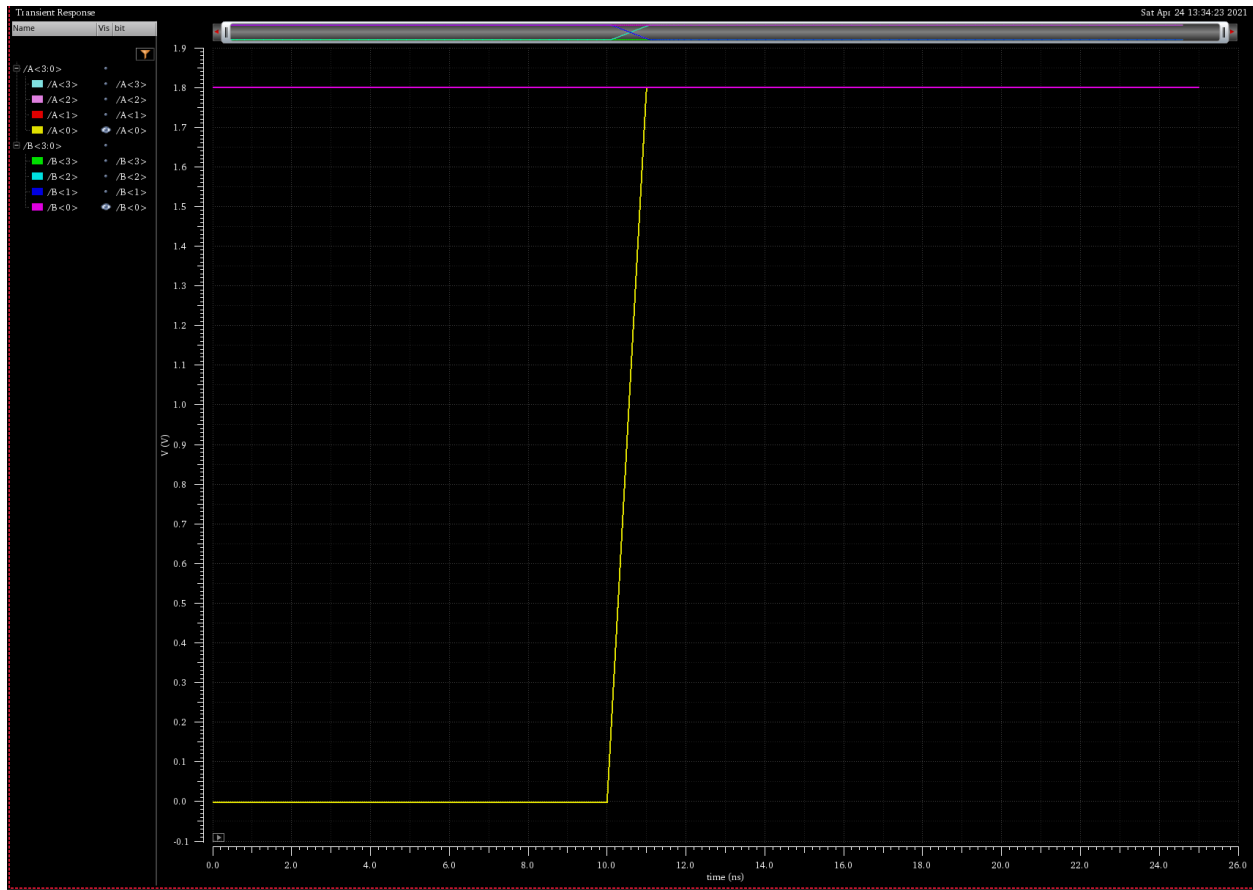


The post layout simulation outputs can be seen below.

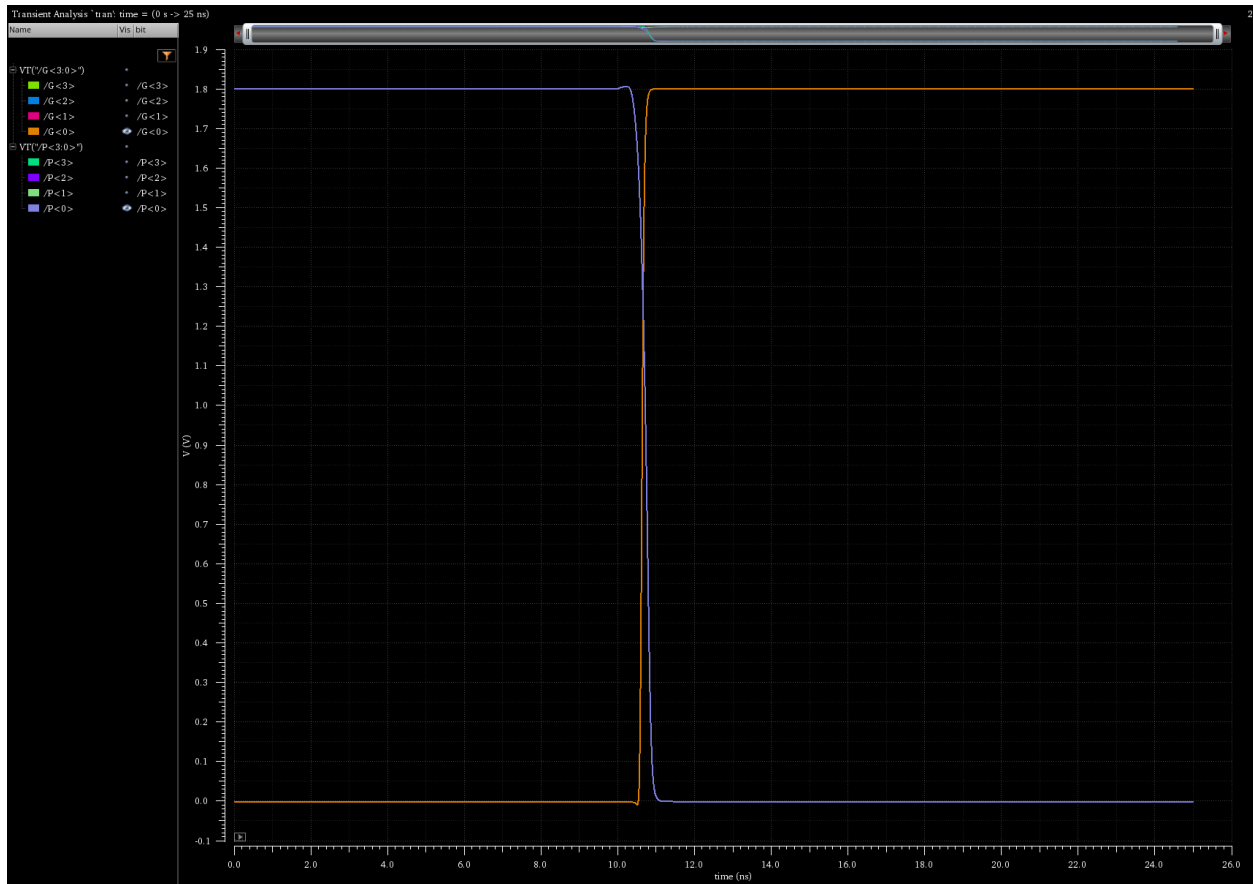


We can confirm the correct functionality of the 4 - Bit Propagate and Generate block by observing each bus line.

A<0> and B<0> are shown below.



Our expectation is that P<0> should output 1 then 0 while G<0> should output 0 then 1. This is confirmed in the below image.



The propagation delay for Propagation signal is calculated to be 235.4 us. The propagation delay for Generate signal is calculated to be 135.9 us.

Using this method, we can verify that all bus lines are working correctly. Below is a list of what the Propagation and Generate signals are expected to be per bus line. Simulation results (post-layout) are given below to verify the correct functionality of the 4-Bit Propagate and Generate Signals.

$$P_1 = < 11 >$$

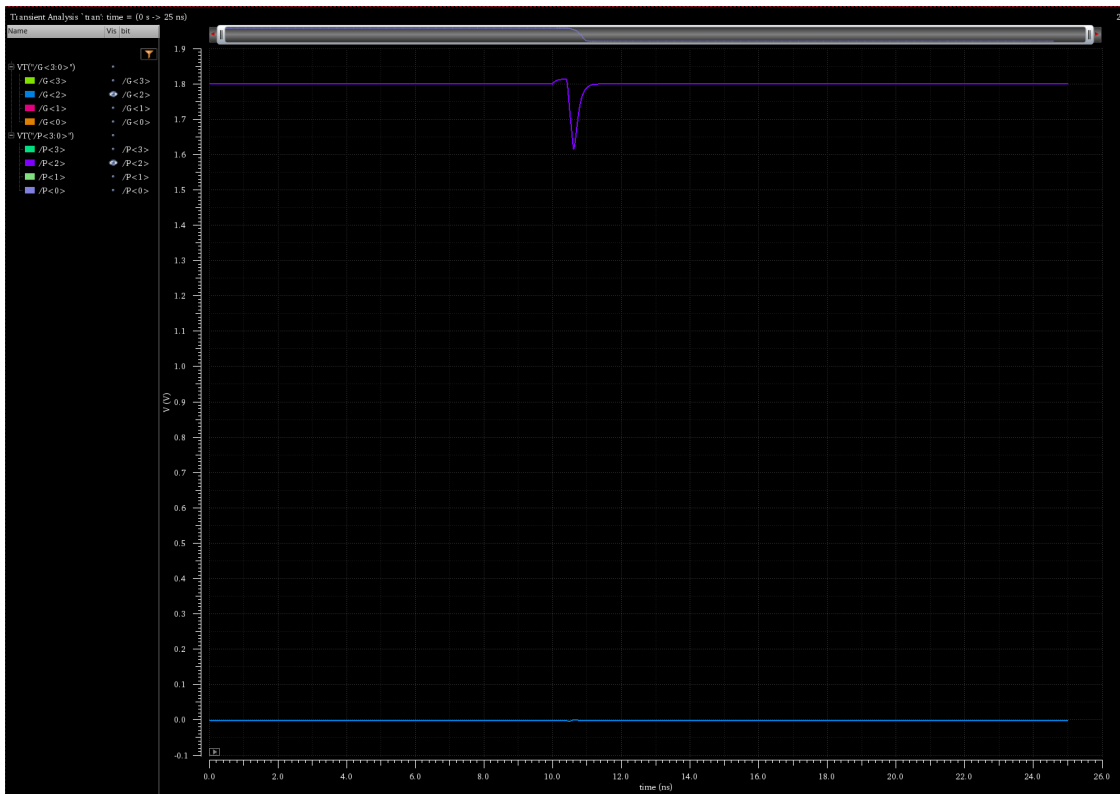
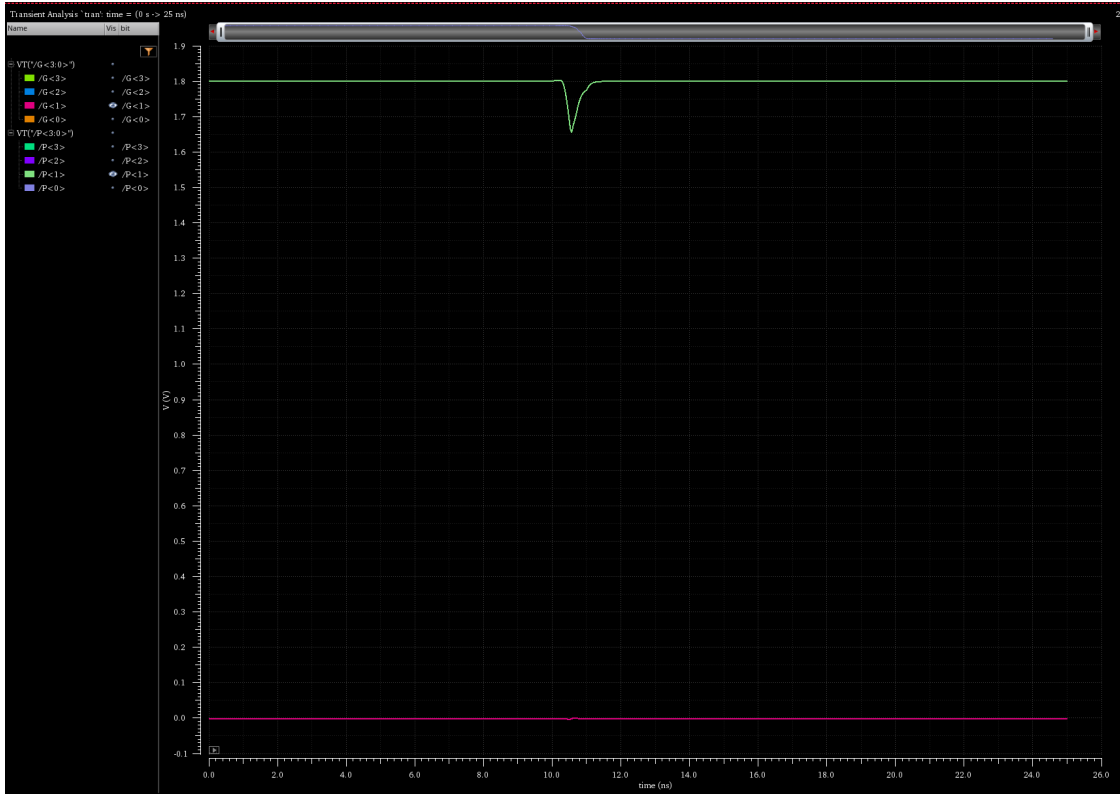
$$G_1 = < 00 >$$

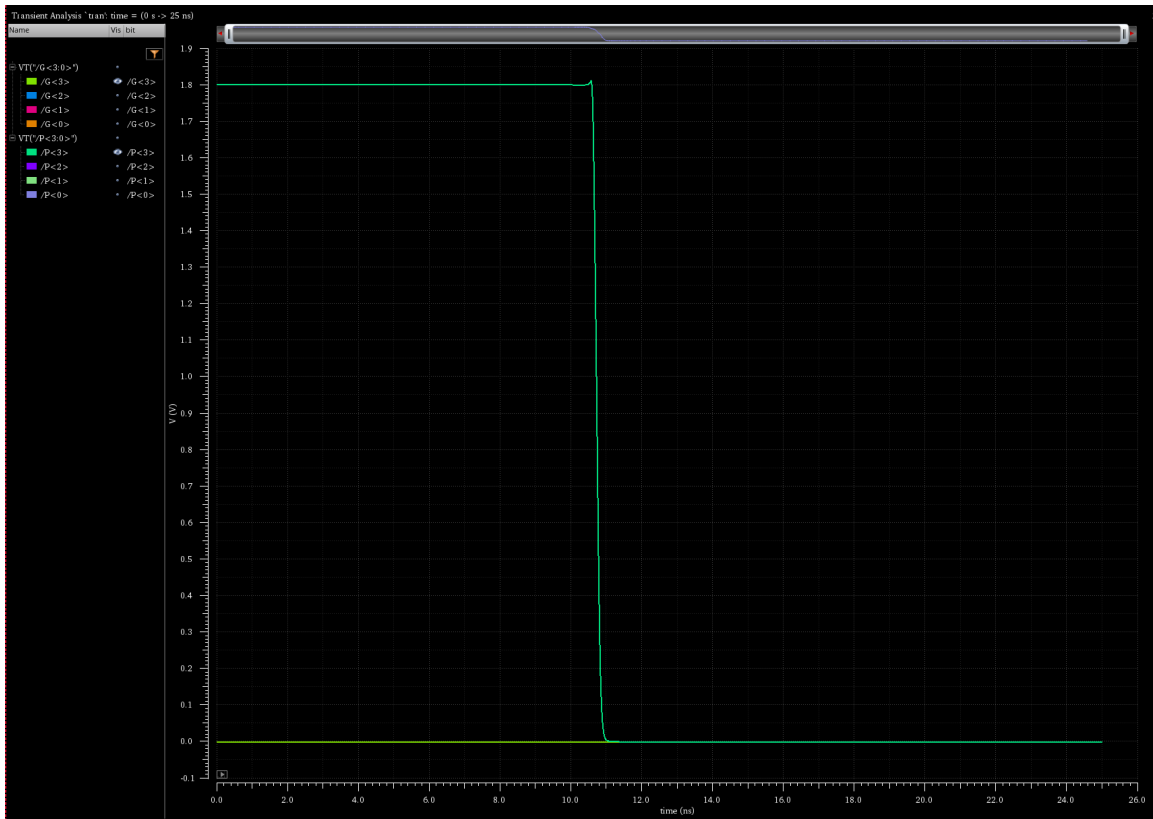
$$P_2 = < 11 >$$

$$G_2 = < 00 >$$

$$P_3 = < 10 >$$

$$G_3 = < 00 >$$





Lessons Learned

It is observed in the P<2> and P<1> signals, there is a slight dip in output voltage. The lowest voltage dip is measured at 1.6 V. This is substantially above the $V_{dd}/2$ transition point. For this project, this was deemed acceptable. This voltage delay is assumed to be caused by the XOR gate elaborated upon previously in this report. The XOR gate inputs create a complementary signal that cause a delay within the XOR gate.

4 – Bit Sum and Carry Logic

Scope

The 4 – Bit Sum and Carry Logic is needed to generate the final Sum and Carry out signals needed in the 32 – Bit adder. The Sum and Carry Logic was designed to work with a 4 – bit block because of our chosen design methodology. The Sum and Carry Logic will compute the Propagate and Generate signals described previously.

In general, the Sum and Carry Logic is as follows.

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i \cdot C_i$$

The Carry bit logic can be expanded upon.

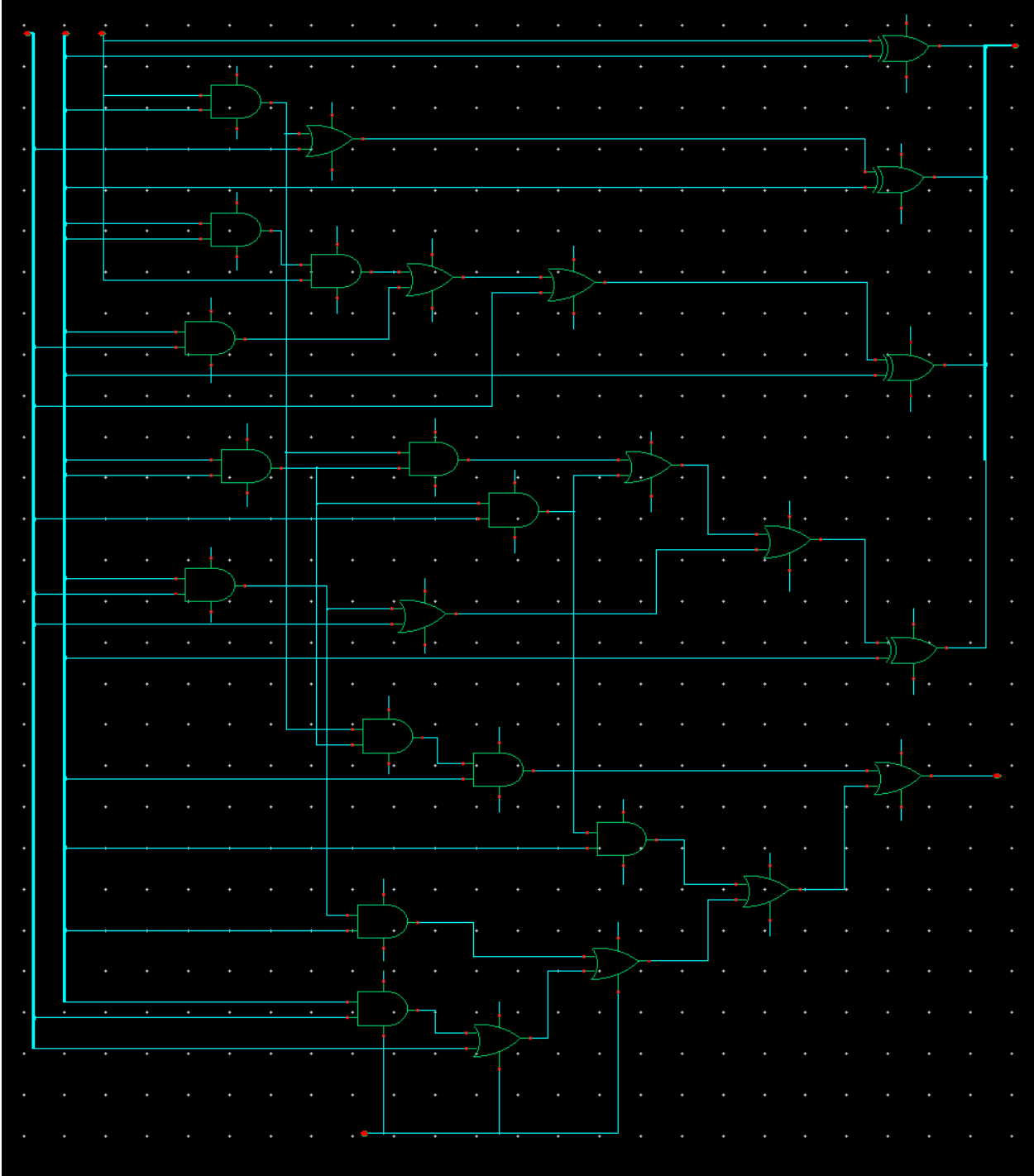
$$C_1 = G_0 + P_0 \cdot C_{in}$$

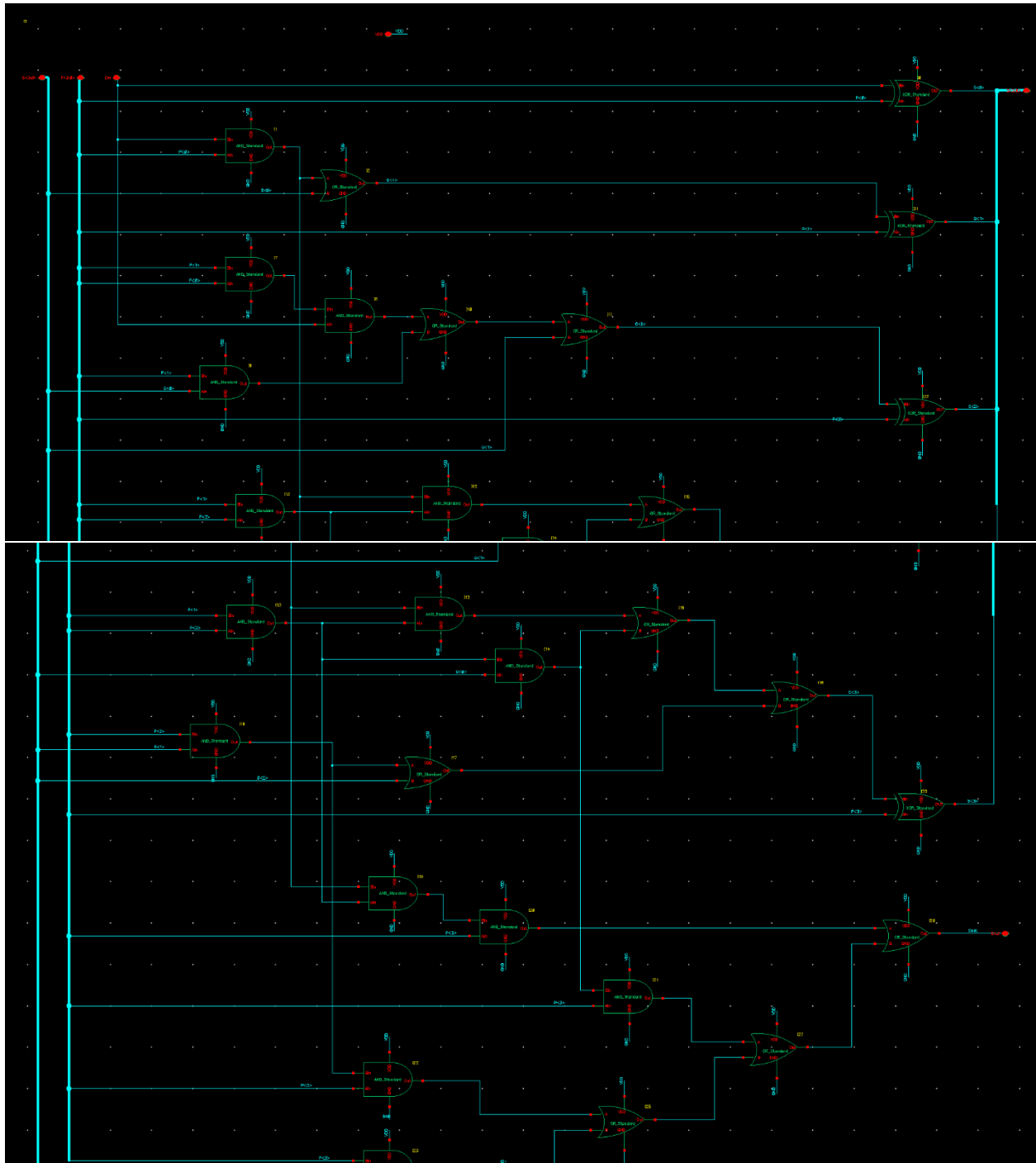
$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$

$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$

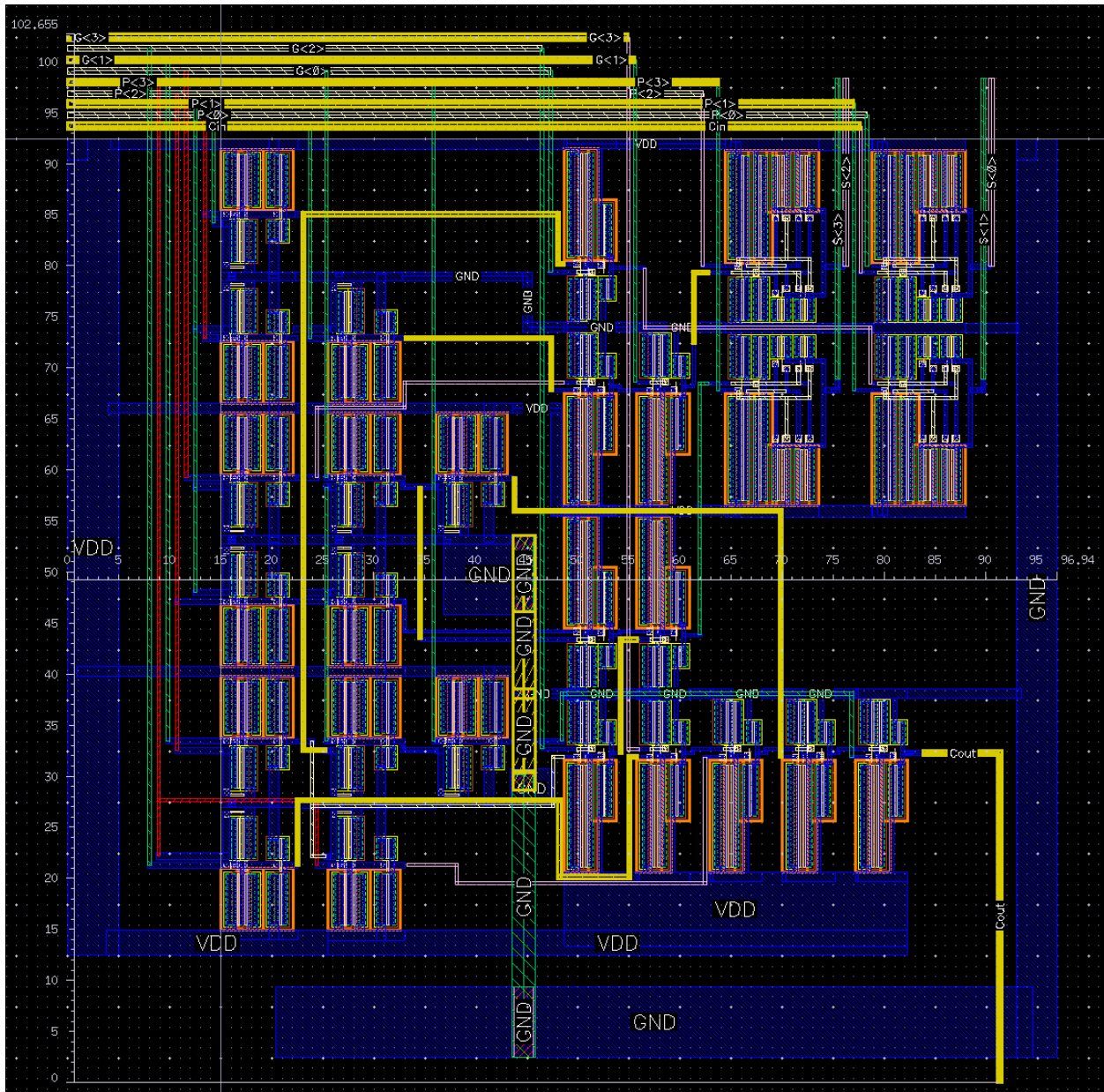
$$C_{out} = G_3 + P_3 \cdot C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

Schematic





Layout



The calculated area of the 4 – Bit Sum and Carry Logic is 102.655 μm x 96.94 μm or 9951.376 μm^2 .

Post – Layout Simulation

We will assume that the A and B inputs for the propagate and generate block is kept the same.

$$A_1 = \langle 1100 \rangle$$

$$B_1 = \langle 0011 \rangle$$

$$A_2 = \langle 0011 \rangle$$

$$B_2 = \langle 0101 \rangle$$

This will result in a Propagate and Generate signals of the following, which has been theoretically inferred and verified through simulation.

$$A_1 \oplus B_1 = P_1 = \langle 1111 \rangle$$

$$A_1 \cdot B_1 = G_1 = \langle 0000 \rangle$$

$$A_1 \oplus B_1 = P_2 = \langle 0110 \rangle$$

$$A_1 \cdot B_1 = G_2 = \langle 0001 \rangle$$

We will use these Propagation and Generate Signals to simulate the 4 – Bit Sum and Carry Logic. We will also have a C_{in} vector of the following.

$$C_{in,1} = \langle 0 \rangle$$

$$C_{in,2} = \langle 1 \rangle$$

The expected Sum and Carry out signals are as follows.

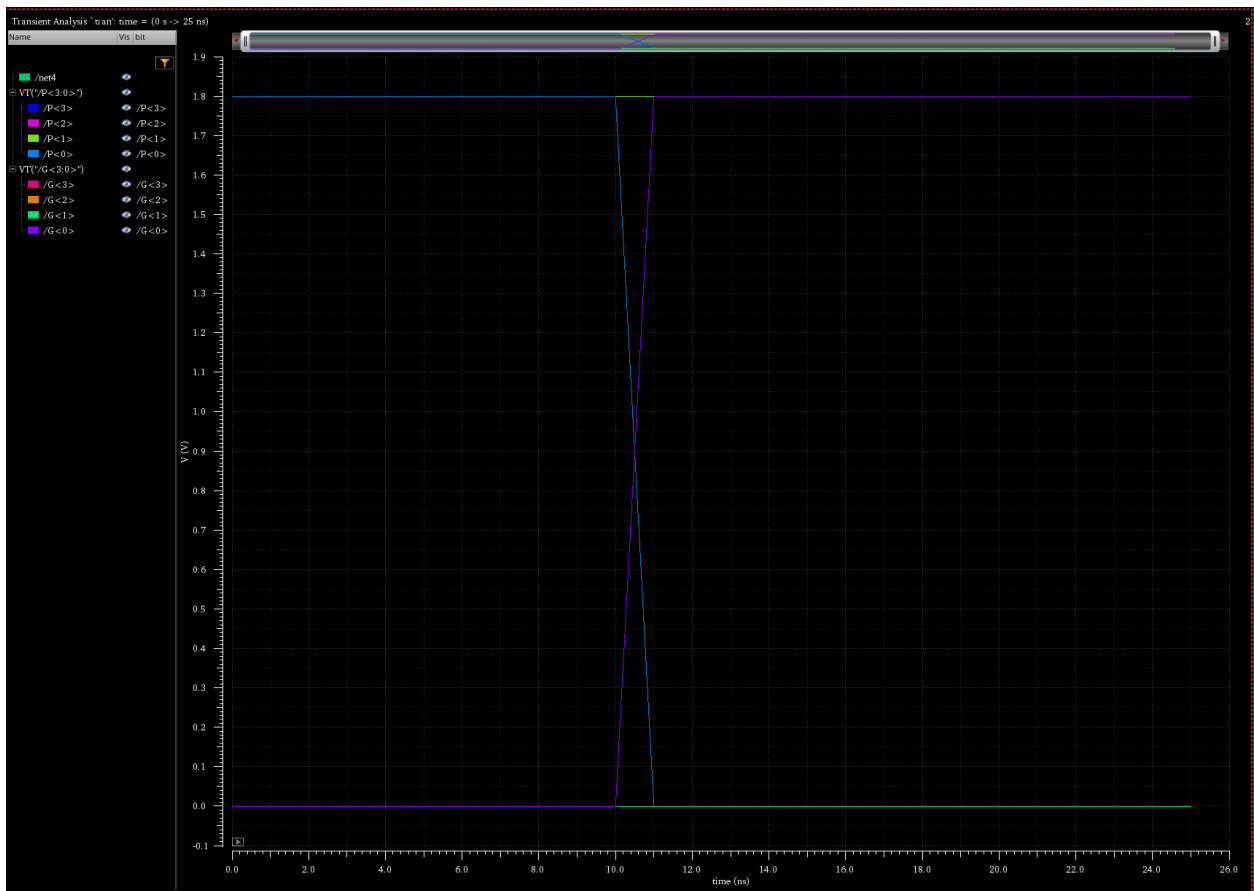
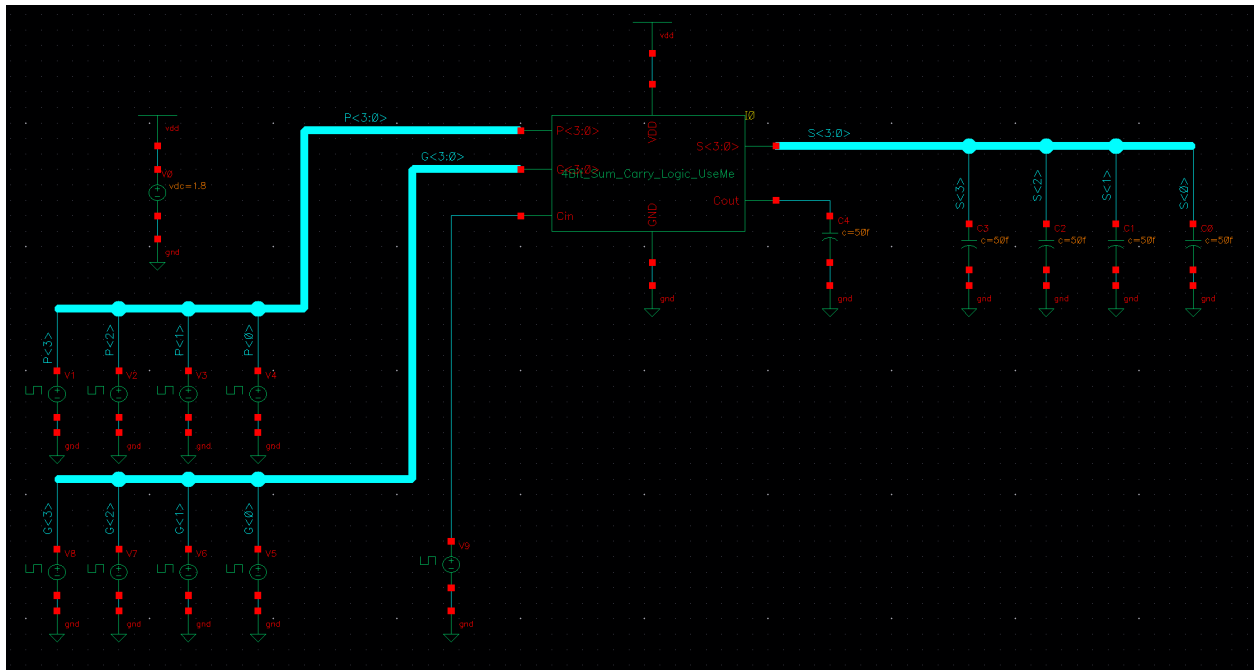
$$S_1 = \langle 1111 \rangle$$

$$C_{out,1} = \langle 0 \rangle$$

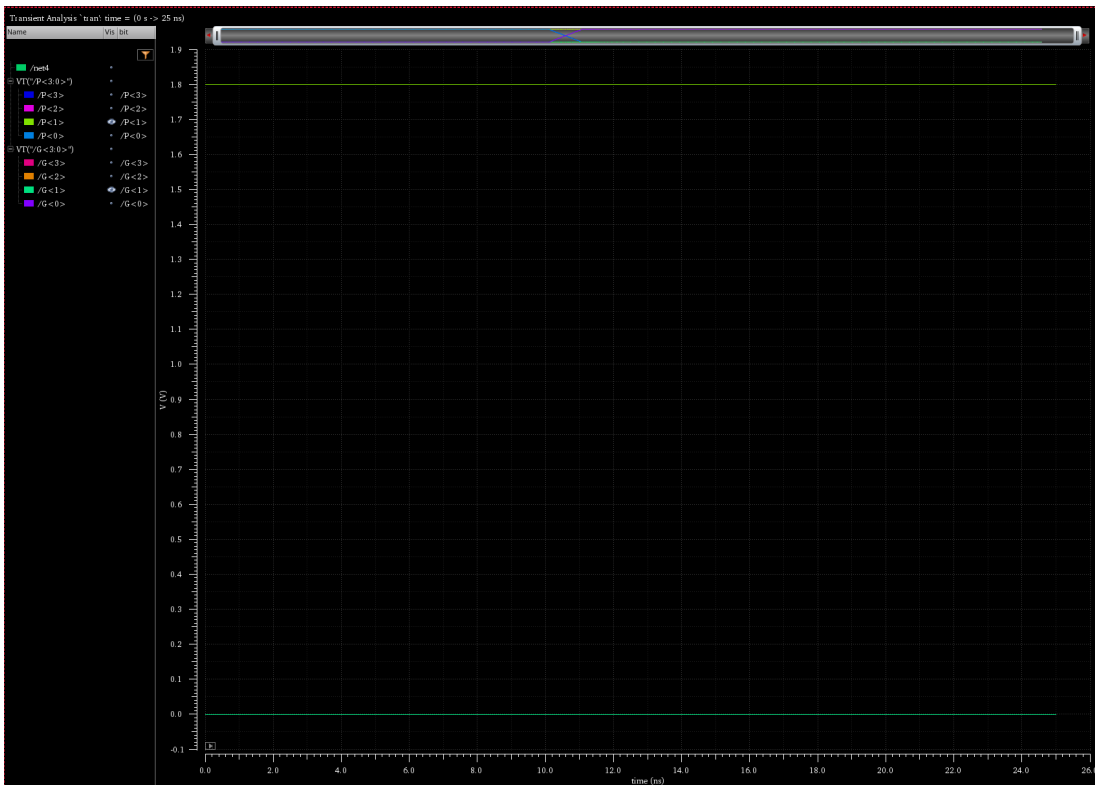
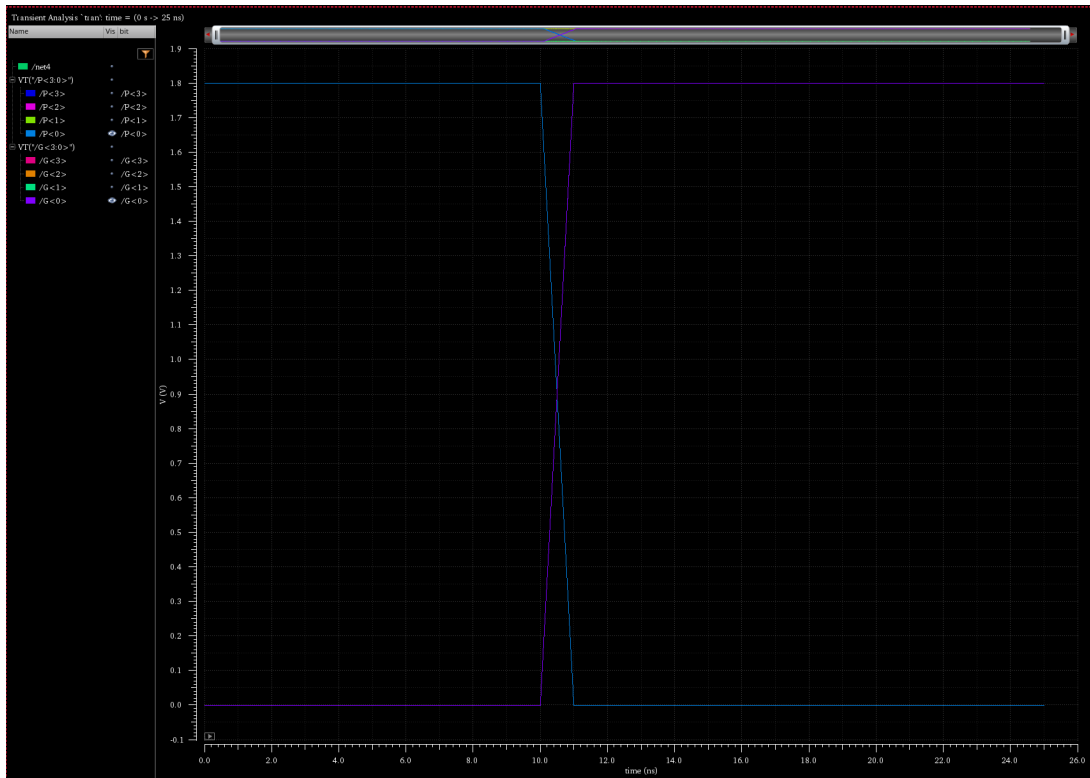
$$S_2 = \langle 1001 \rangle$$

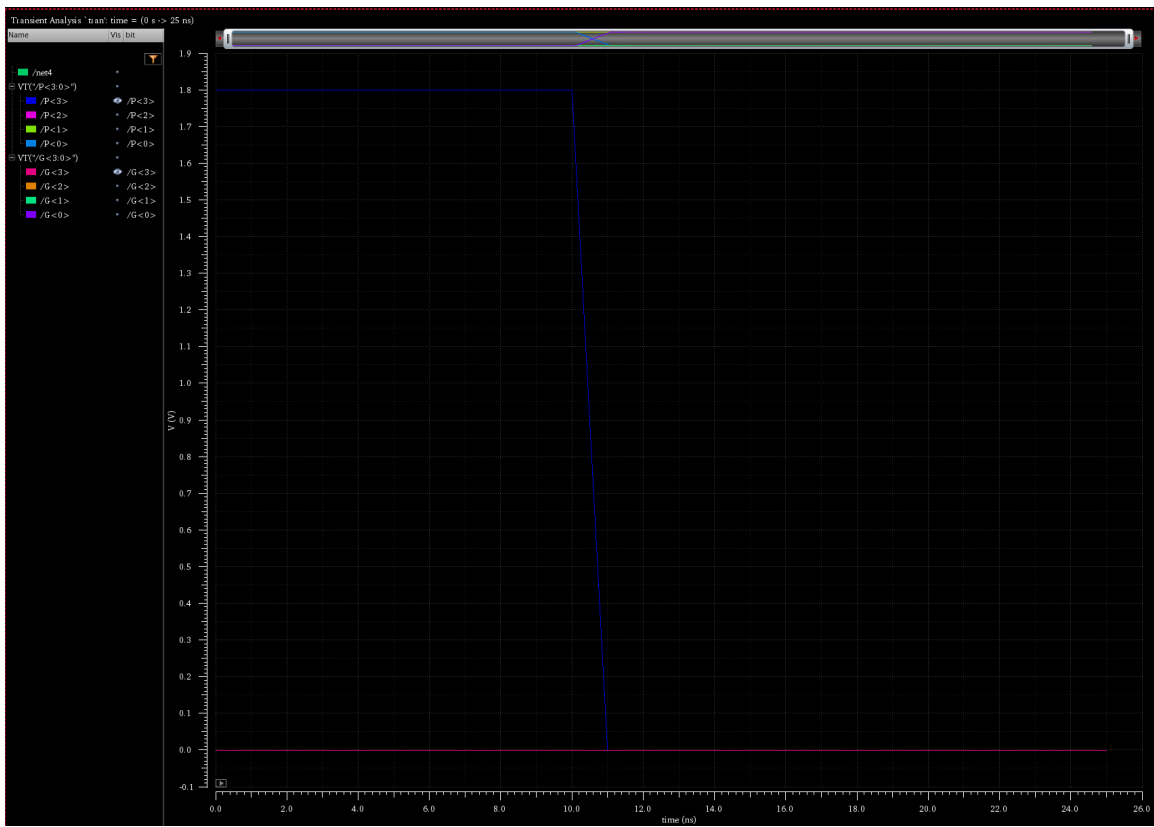
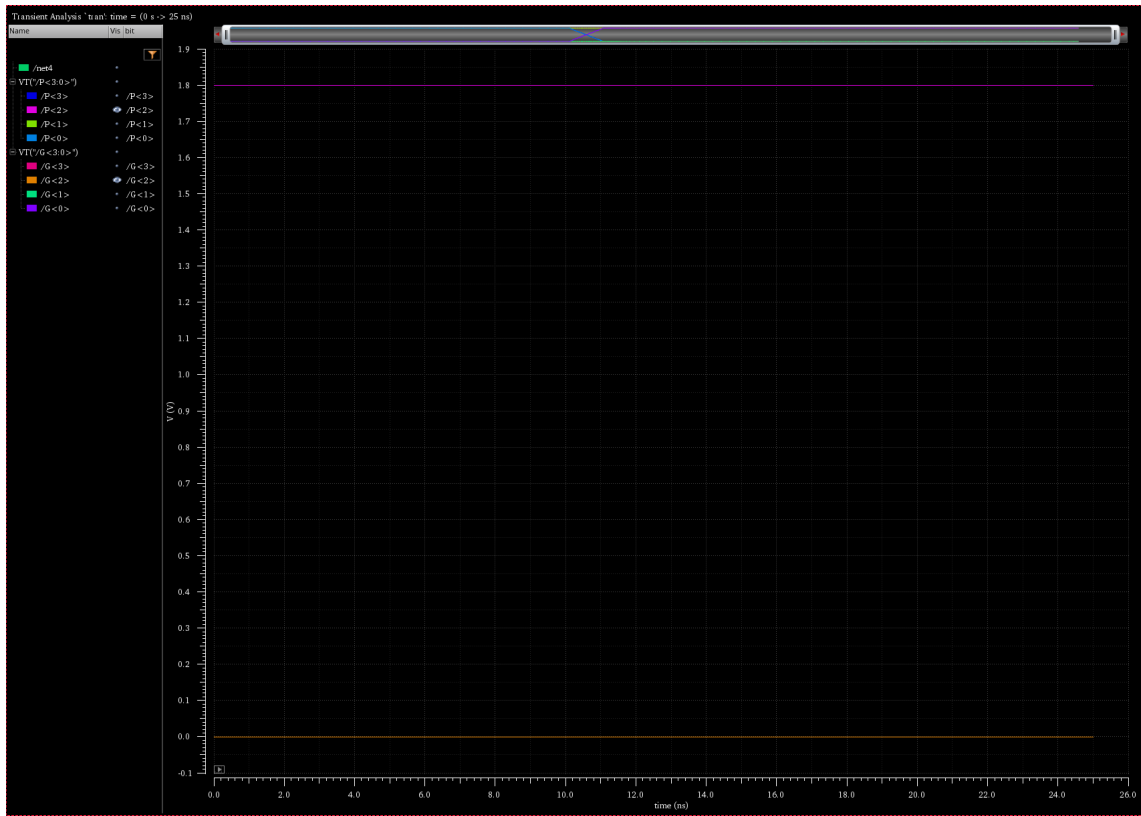
$$C_{out,2} = \langle 0 \rangle$$

The test vectors and testbench are shown below.

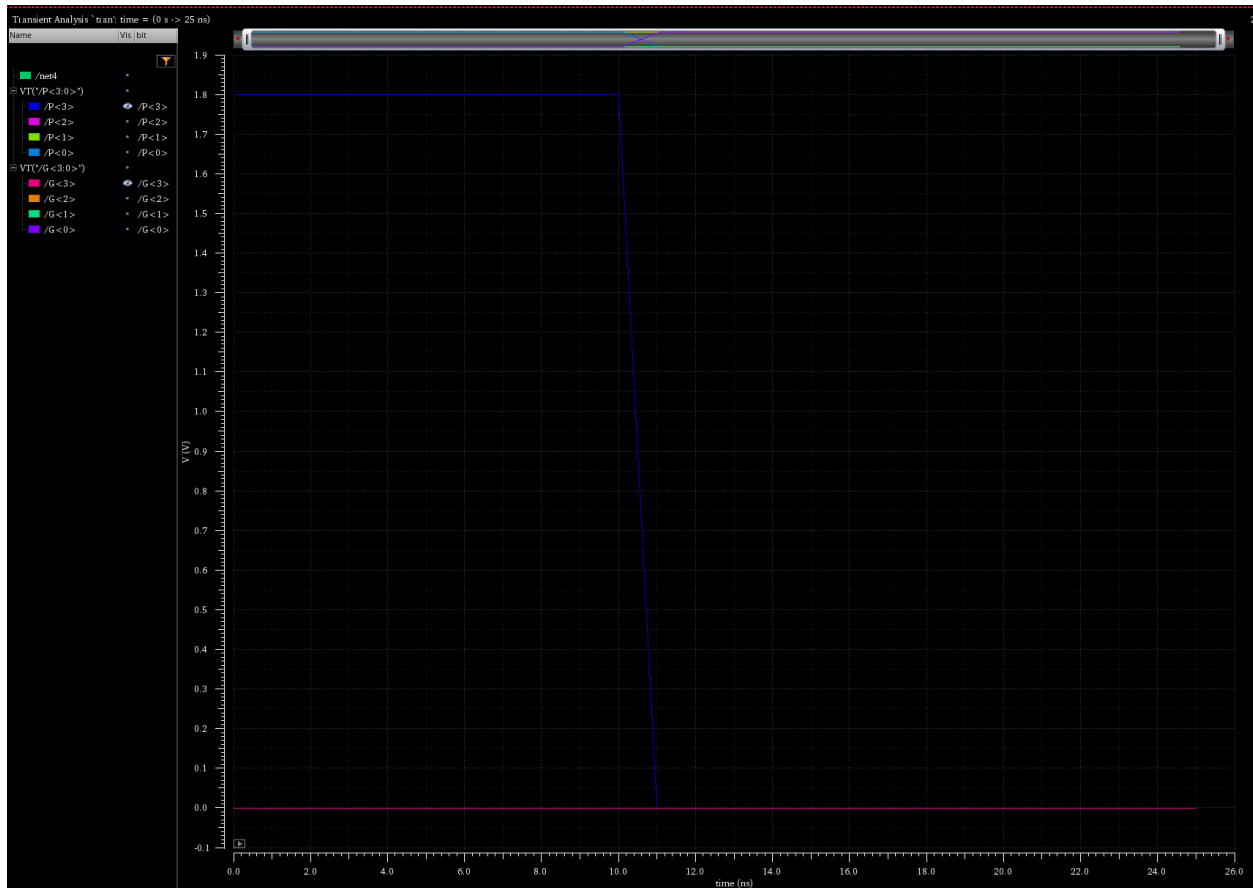


Each Propagate and Generate Simulated input signal can be verified correct below.





C input is verified below.



The expected output bit's behavior is described below.

$$S_0 = \langle 11 \rangle$$

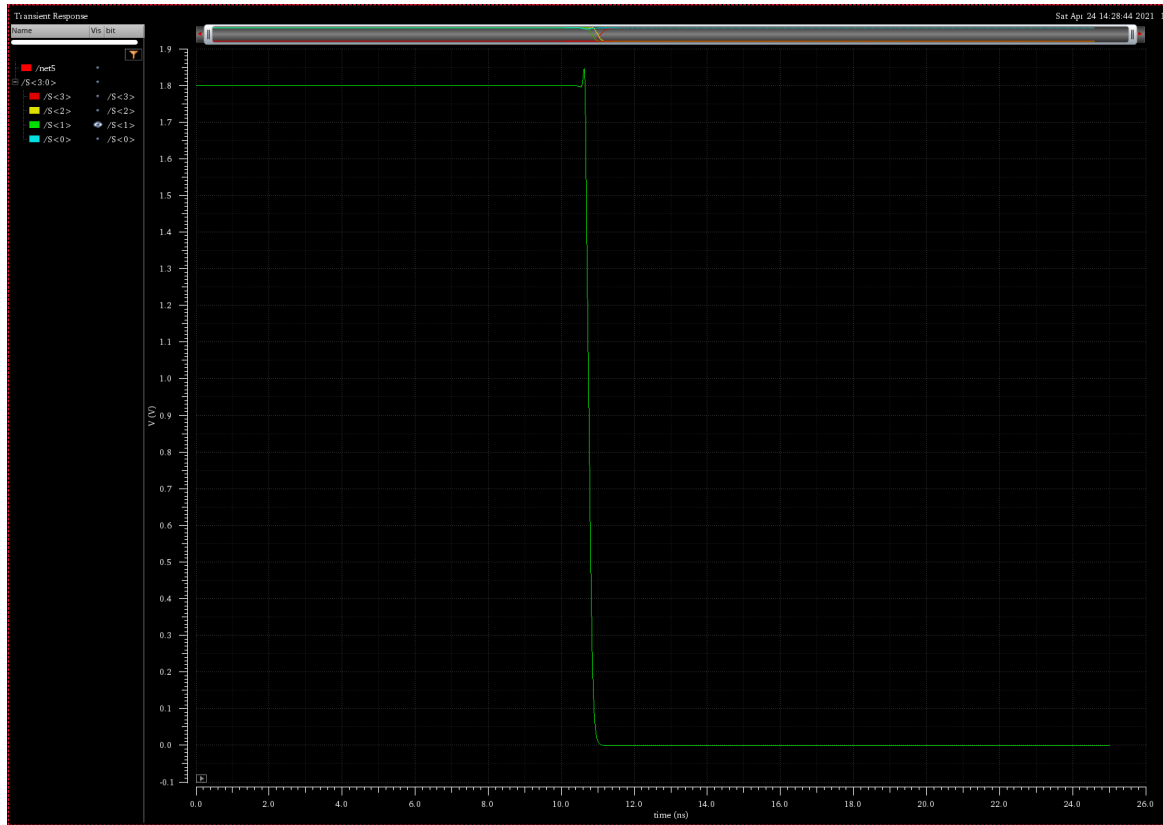
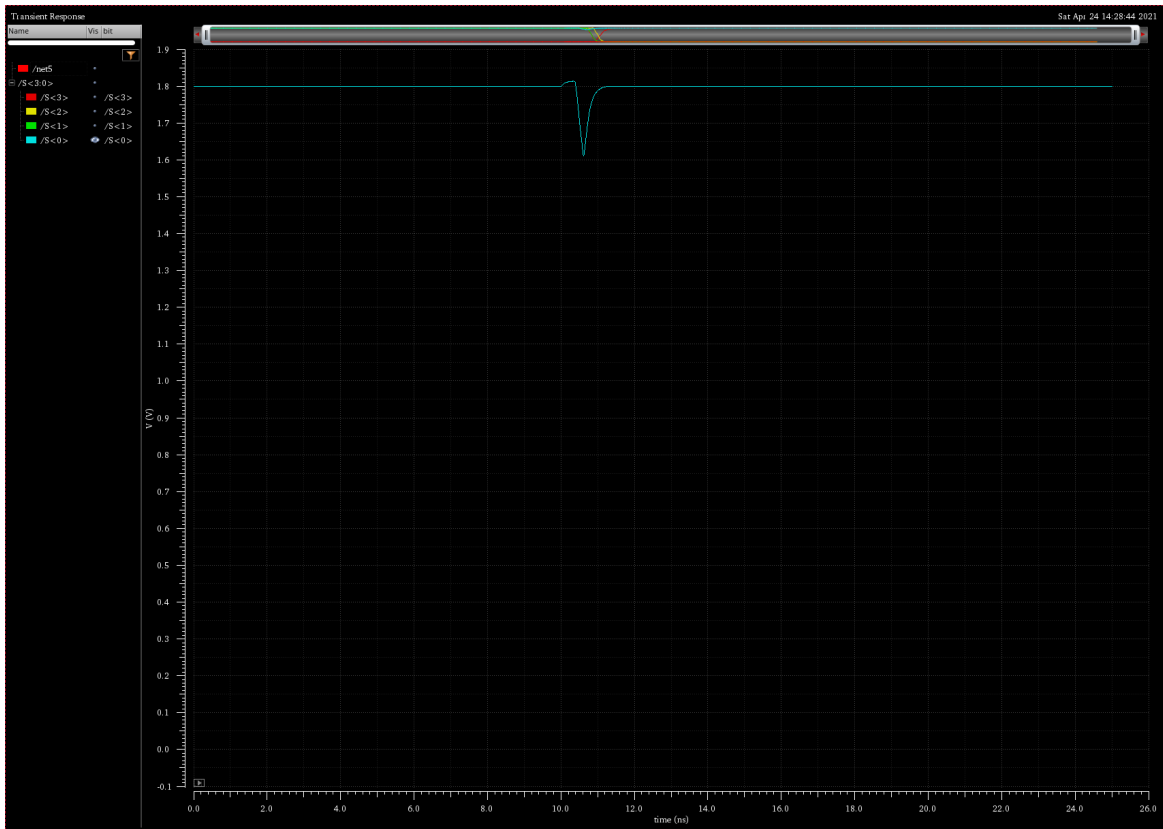
$$S_1 = \langle 10 \rangle$$

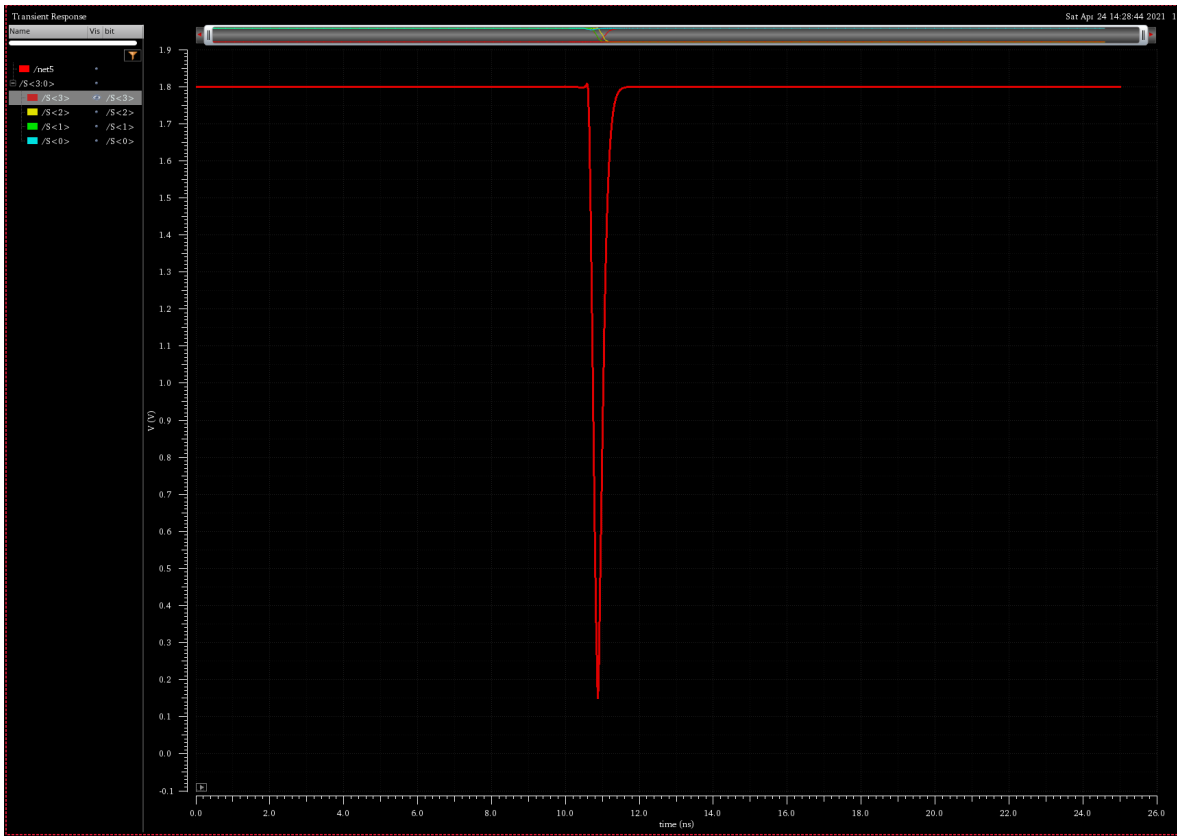
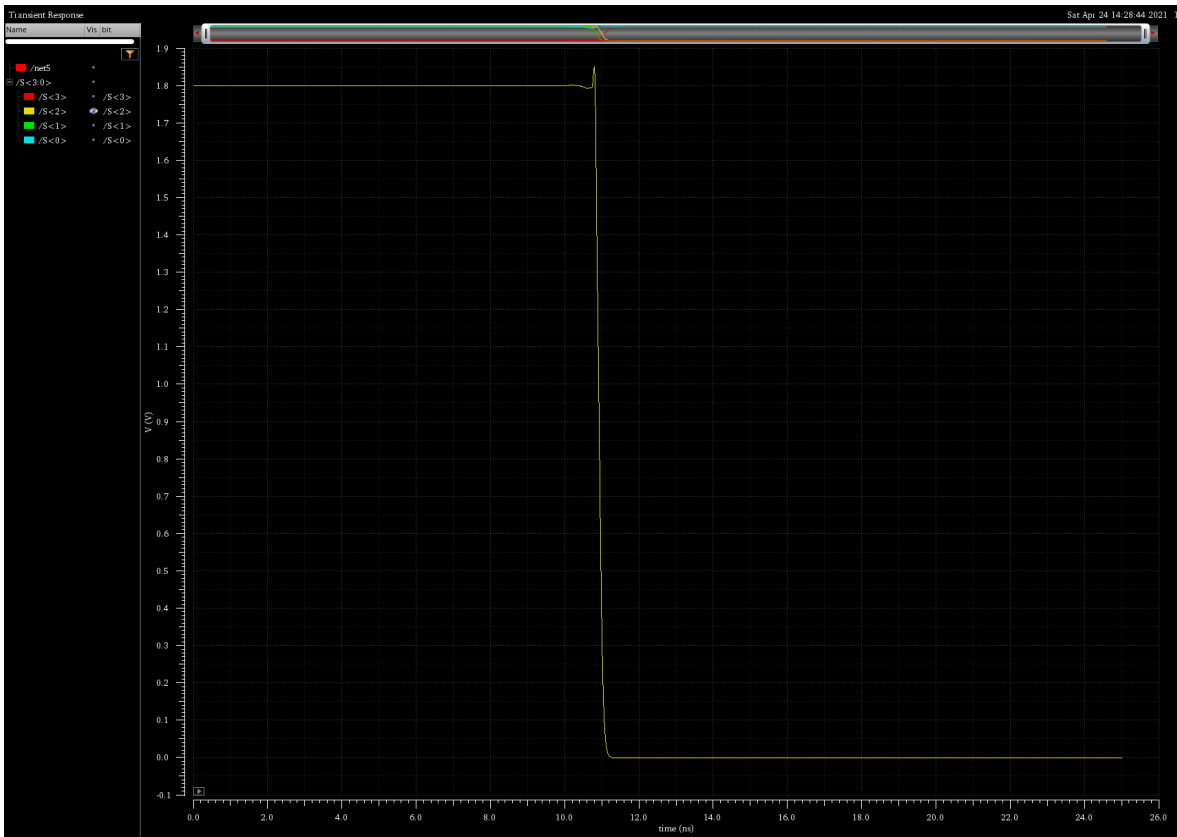
$$S_2 = \langle 10 \rangle$$

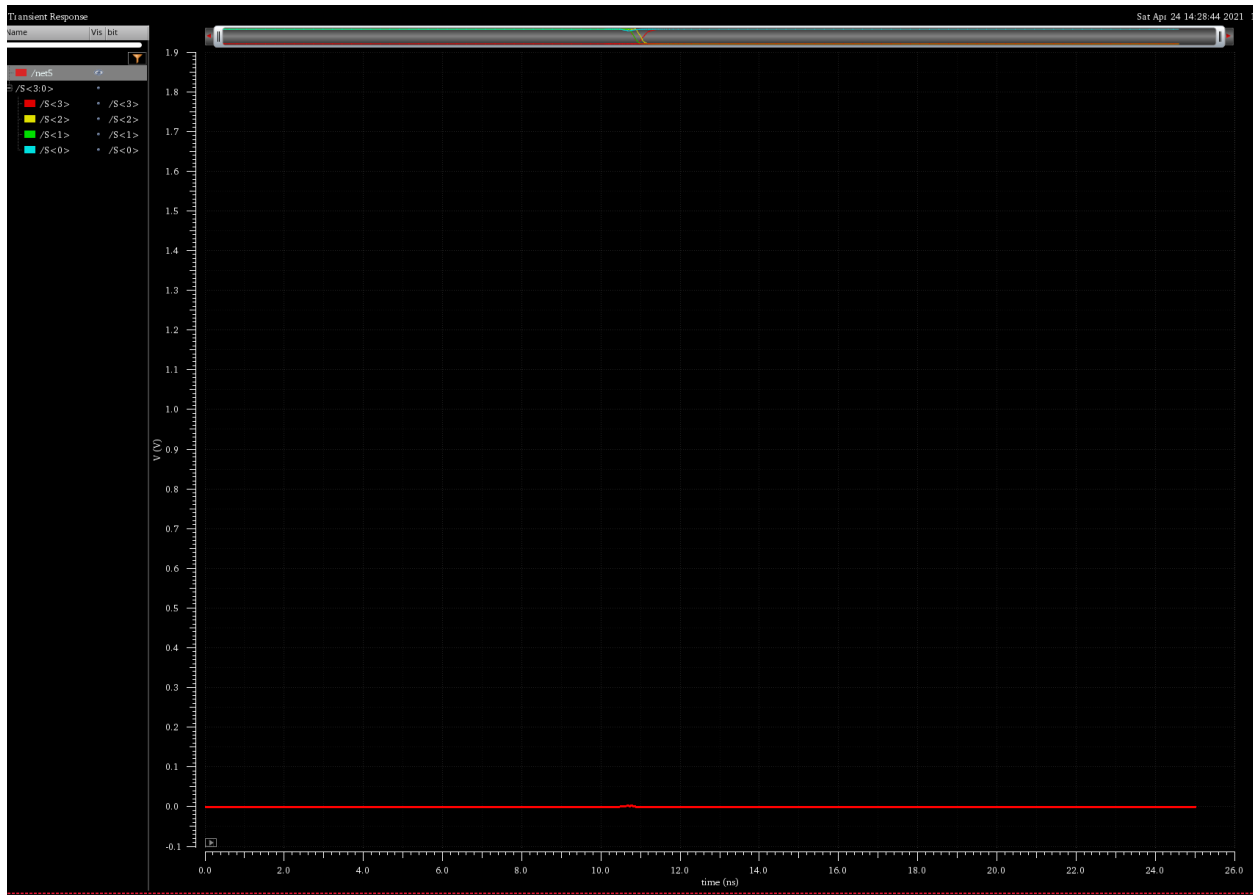
$$S_3 = \langle 11 \rangle$$

$$C_{out} = \langle 00 \rangle$$

Each Output bit's simulated behavior is verified below.







Lessons Learned

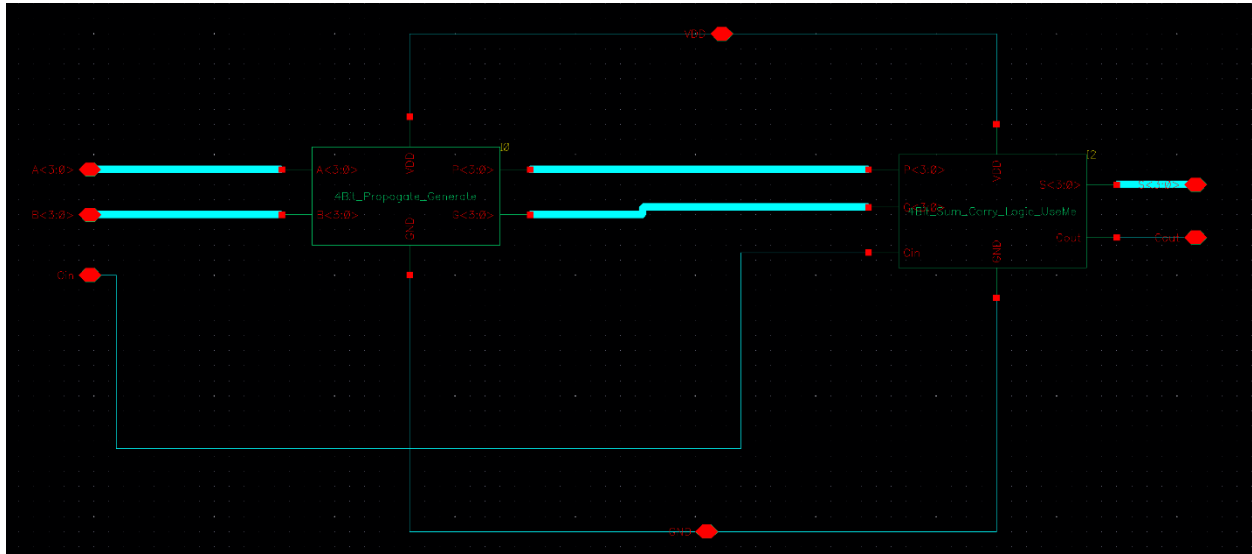
It is observed in S<3> that there is a major dip in the output voltage. This is due to the internal gate delays in the Sum and Carry Logic. The cascading delays within this logic block will induce momentary voltage dips between input transitions.

4 – Bit Carry Look Ahead Adder

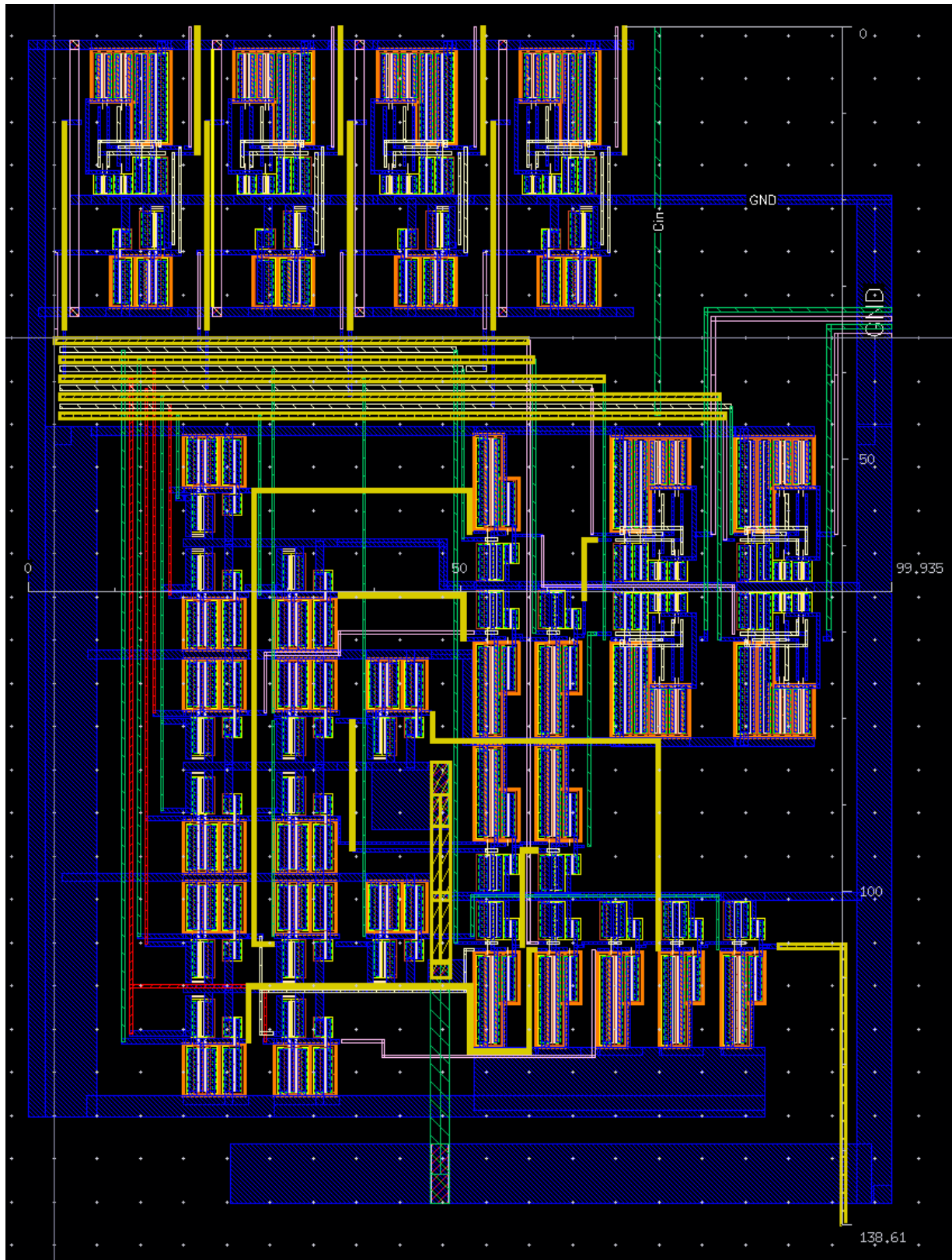
Scope

The 4 – Bit CLA Adder is created. This is due to the chosen design methodology described previously in this report. The 4 – Bit CLA Adder will be cascaded 8 times to create a 32 – Bit Adder.

Schematic

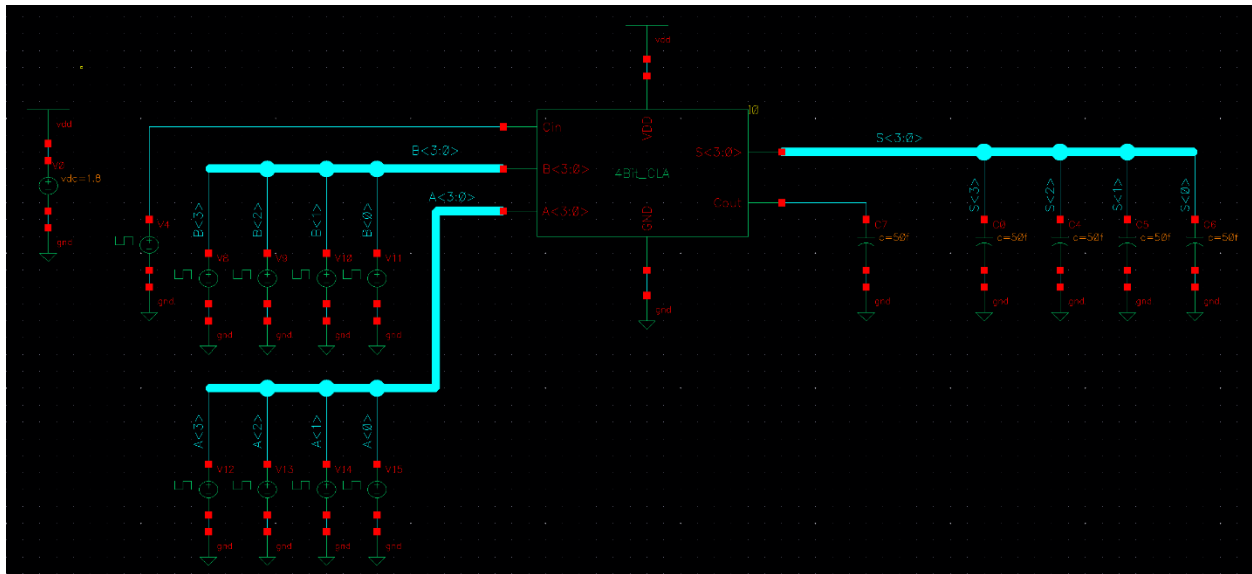


Layout



The calculated area of the 4 – Bit CLA Adder is 138.61 μm x 99.935 μm or 13851.99 μm^2 .

Post Layout Simulation



The following test vectors are chosen in this simulation.

$$A_1 = \langle 1000 \rangle$$

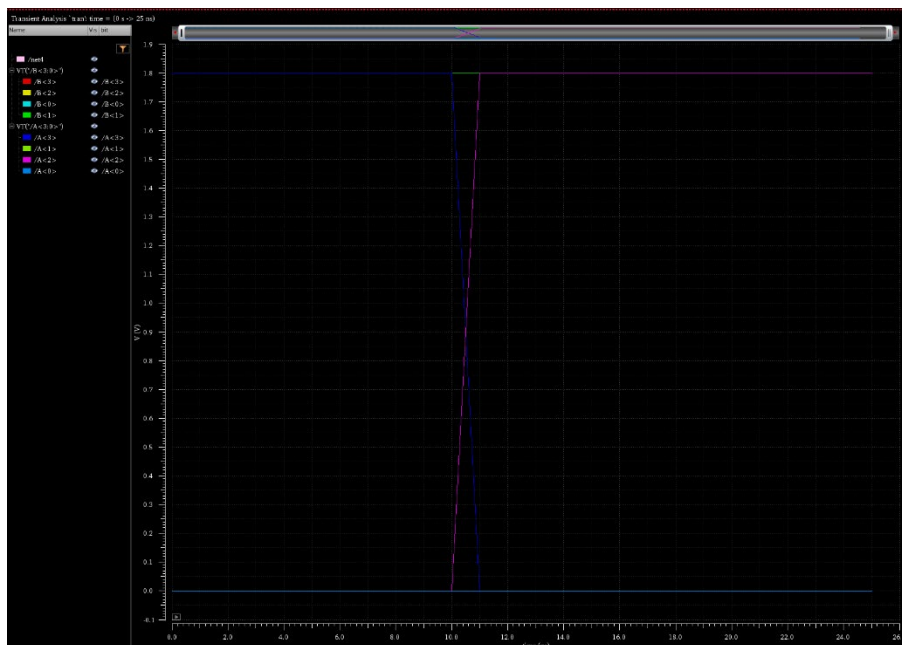
$$B_1 = \langle 1111 \rangle$$

$$C_{in,1} = \langle 1 \rangle$$

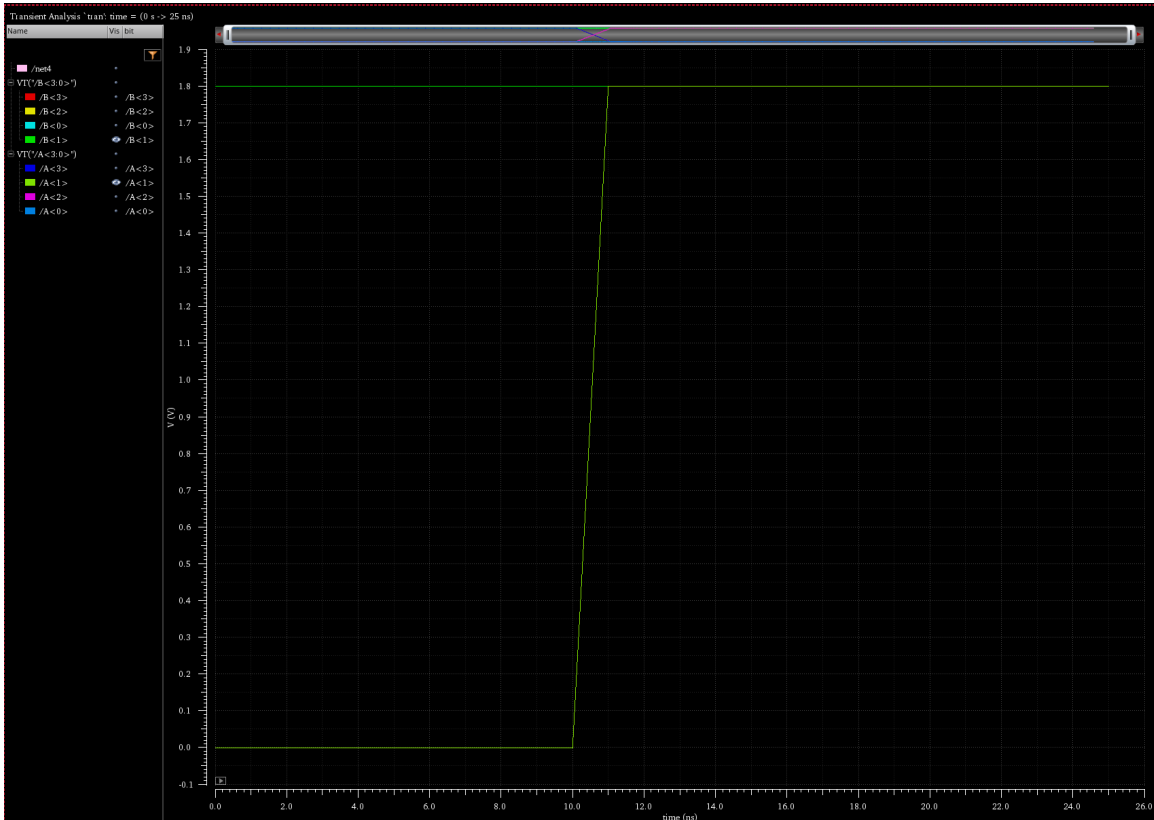
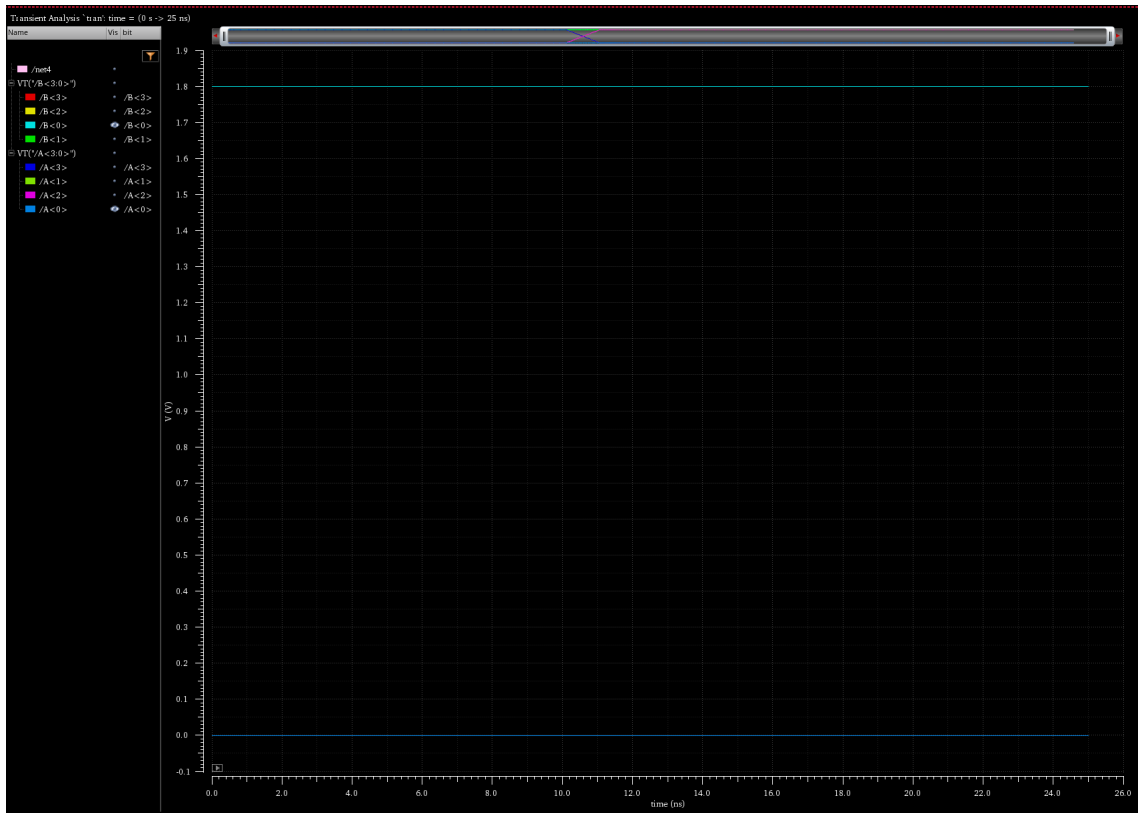
$$A_2 = \langle 0110 \rangle$$

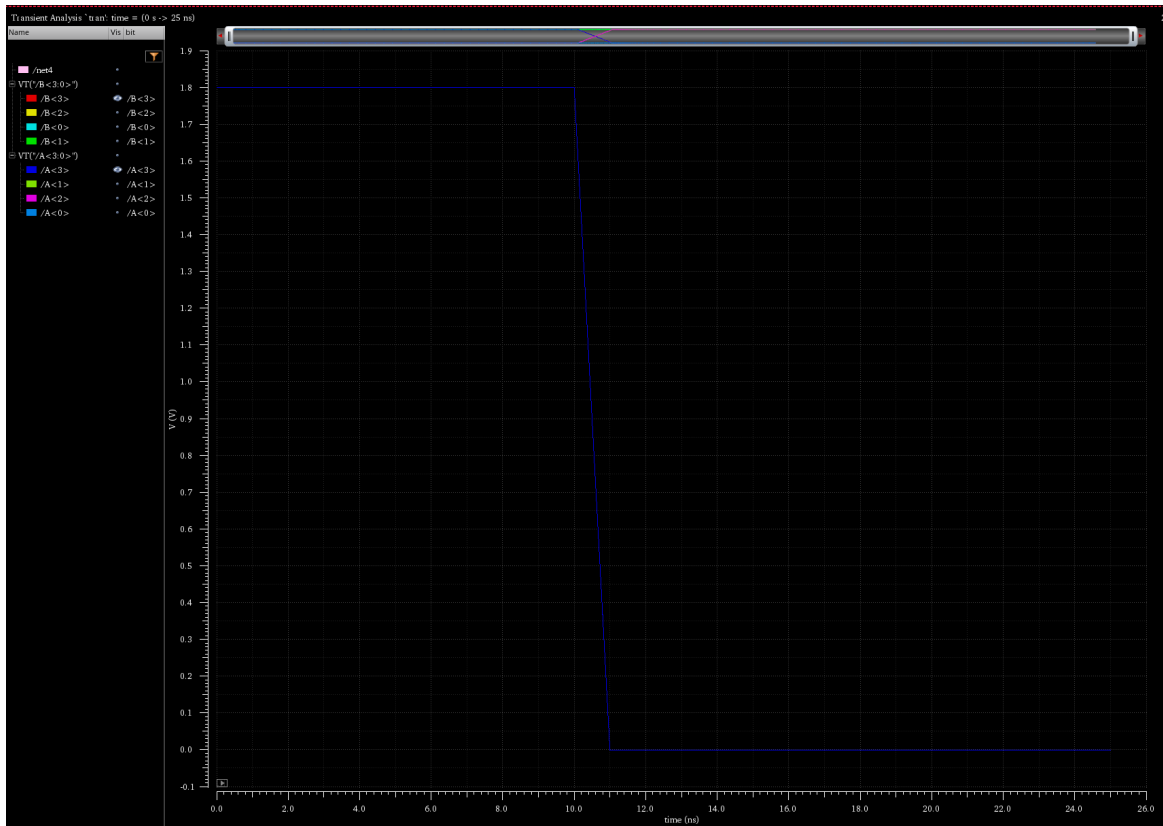
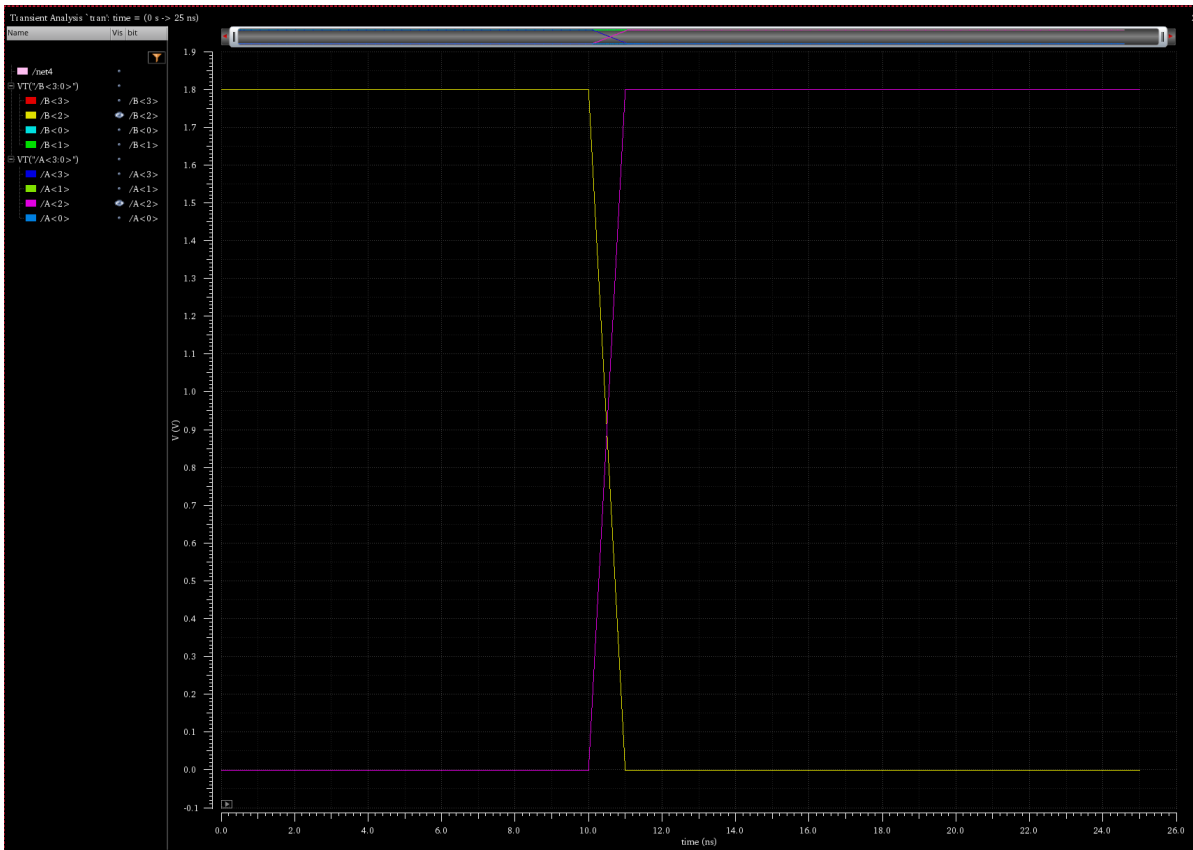
$$B_2 = \langle 0011 \rangle$$

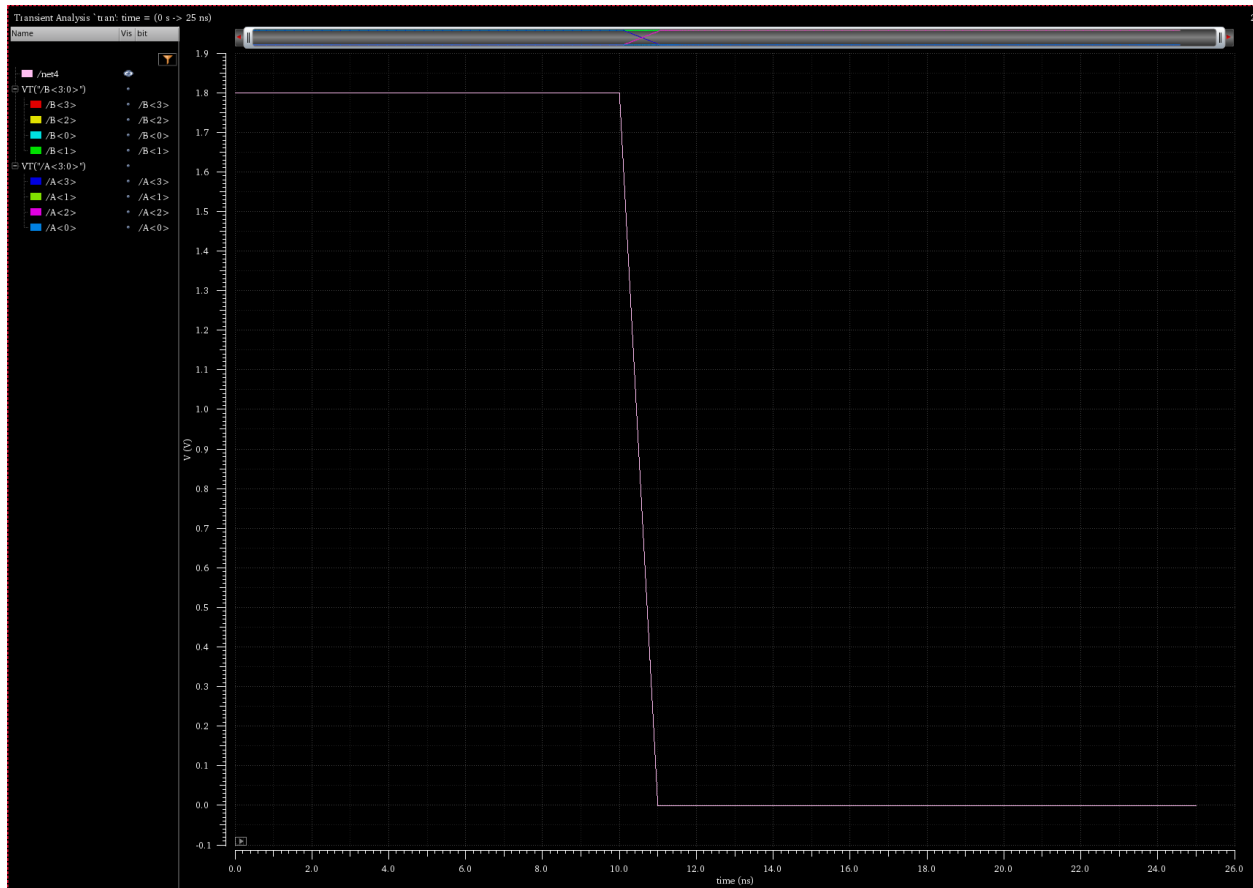
$$C_{in,2} = \langle 0 \rangle$$



The specific behavior of each simulated input is verified below.







The expected Sum and Carry Outputs are as follows.

The simulation inputs are verified in the following figure.

$$S_0 = \langle 1000 \rangle$$

$$C_{out,0} = \langle 1 \rangle$$

$$S_1 = \langle 1001 \rangle$$

$$C_{out,1} = \langle 0 \rangle$$

$$S \langle 0 \rangle = \langle 01 \rangle$$

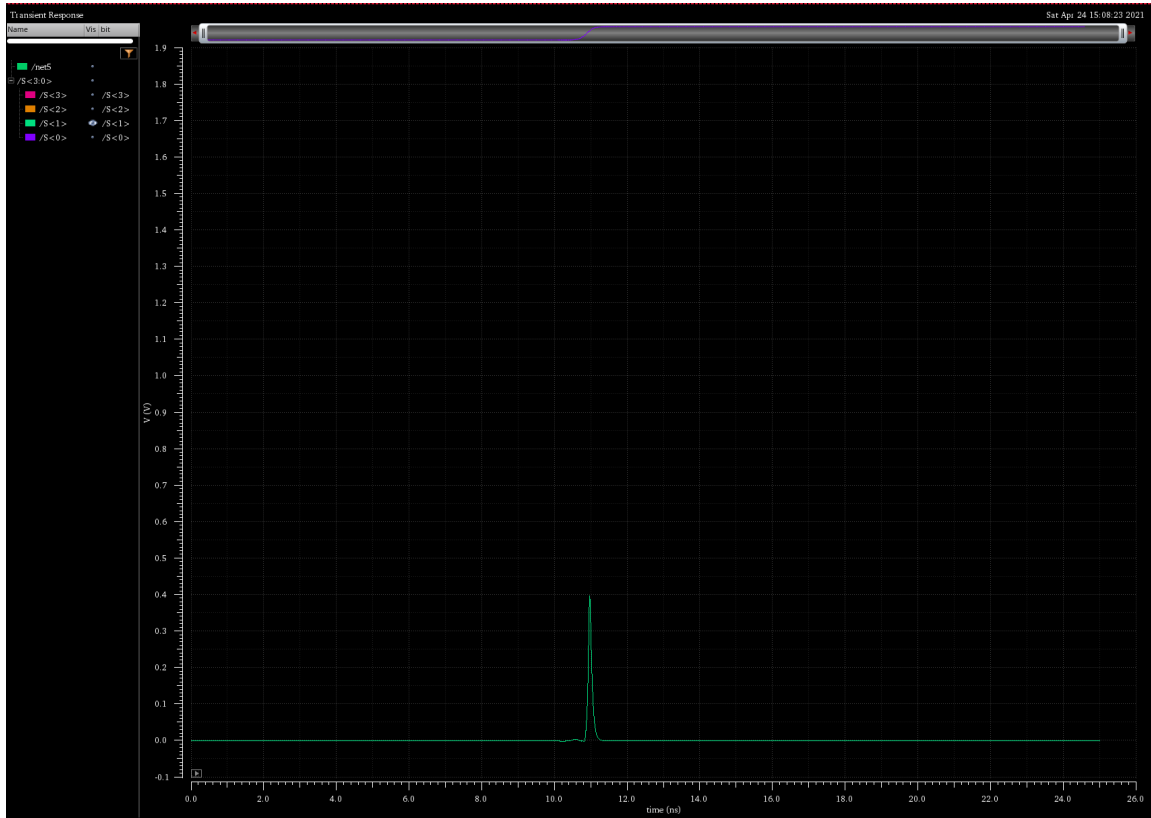
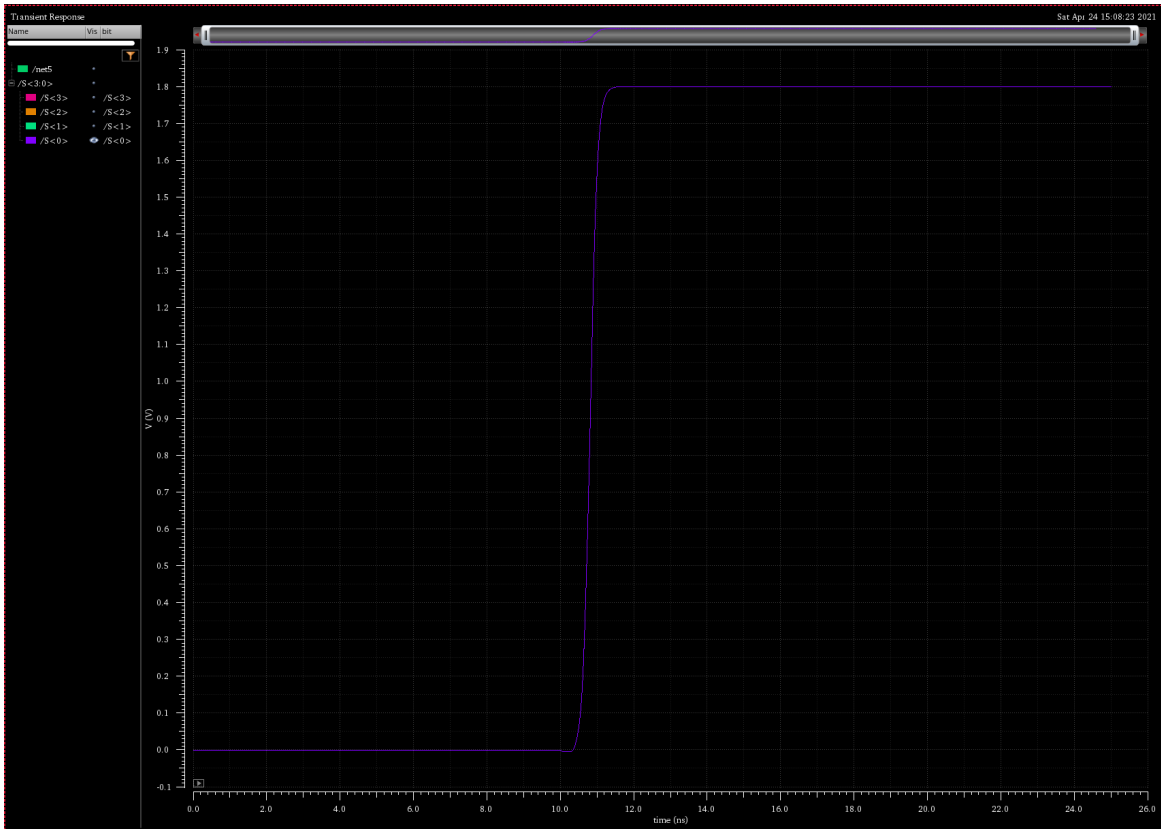
$$S \langle 1 \rangle = \langle 00 \rangle$$

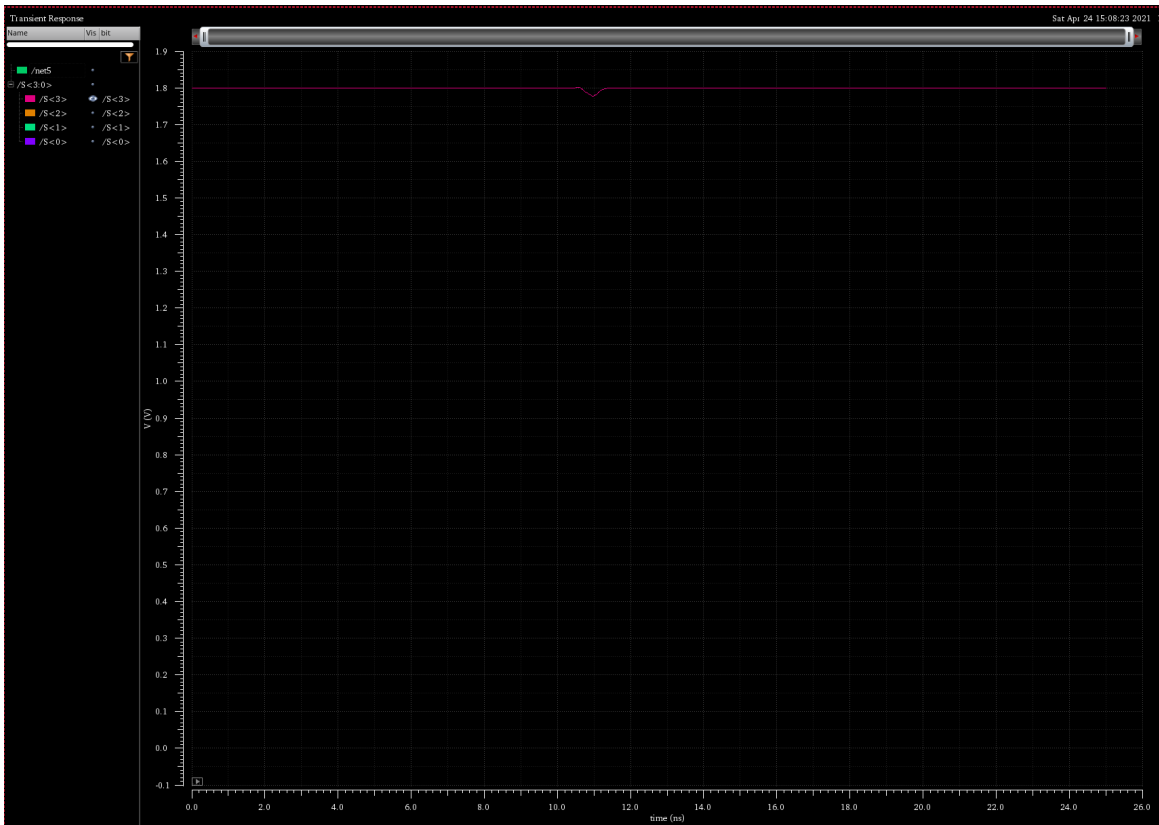
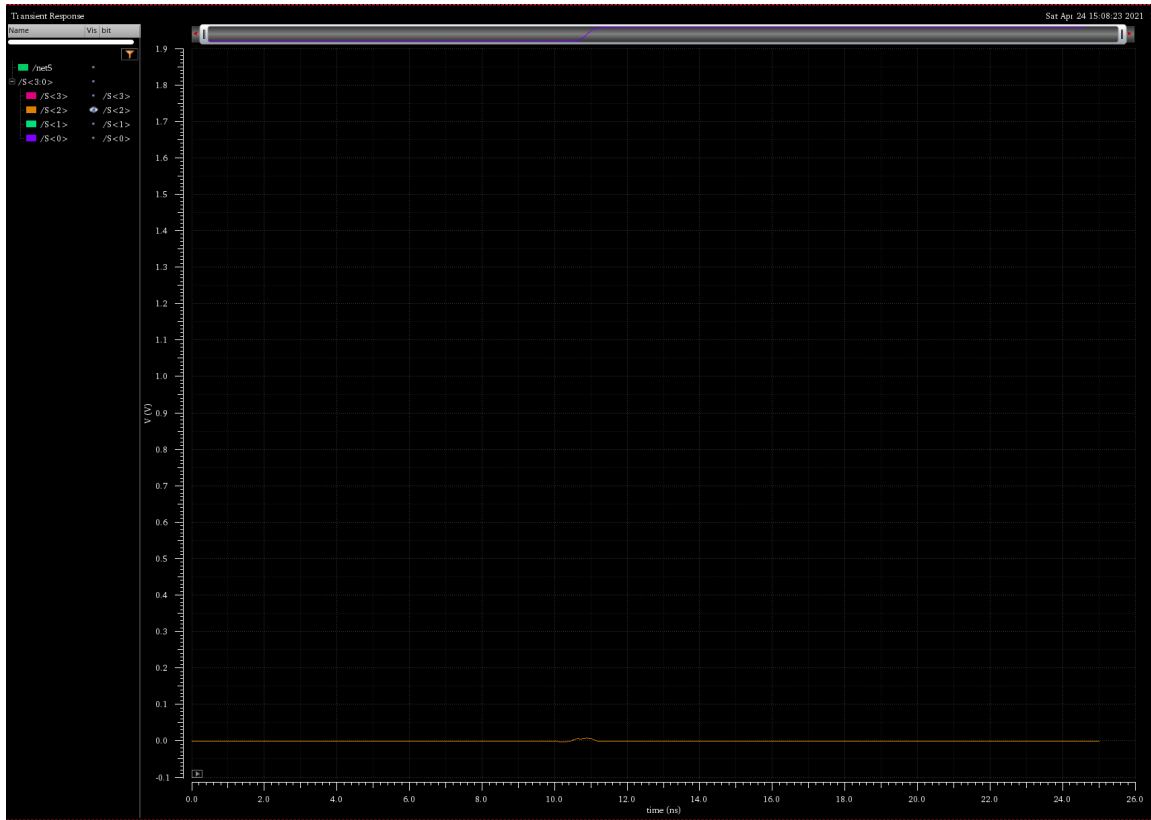
$$S \langle 2 \rangle = \langle 00 \rangle$$

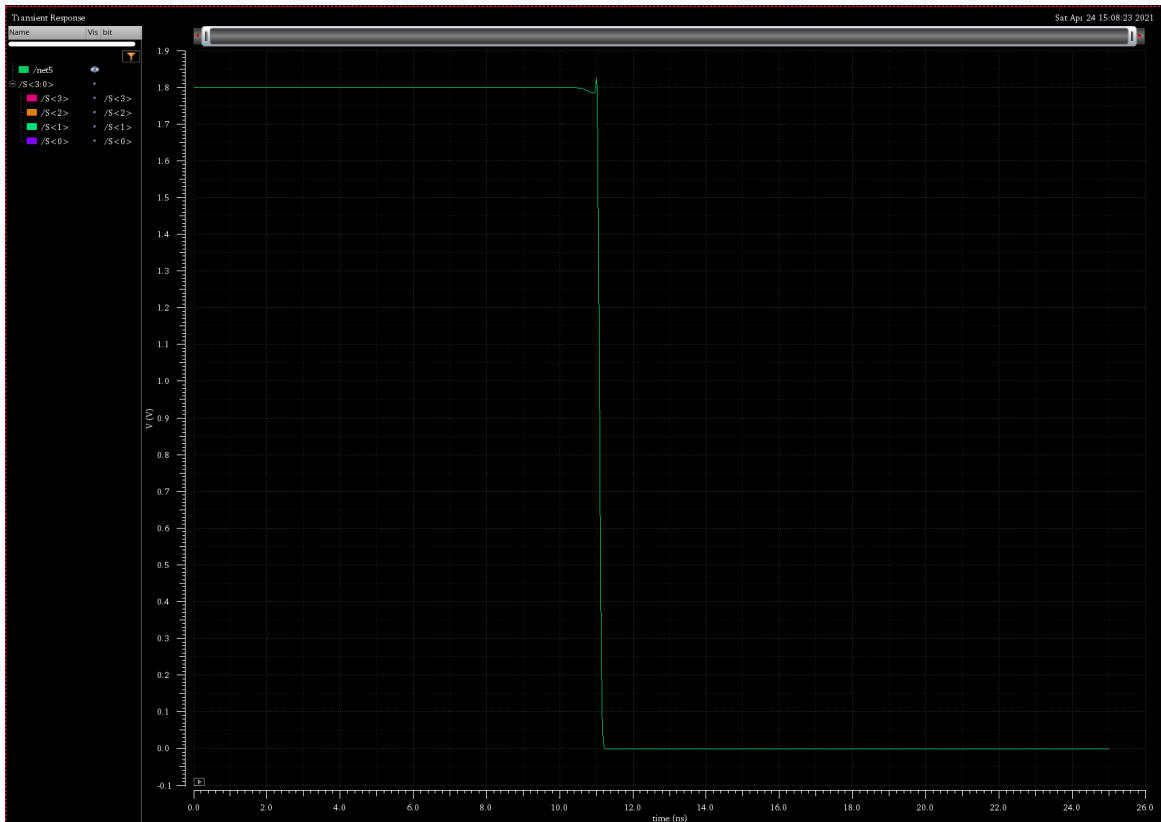
$$S \langle 3 \rangle = \langle 11 \rangle$$

$$C_{out} = \langle 10 \rangle$$

This behavior is verified below.







The above figures verify the correct functionality of the 4 – Bit Carry Look Ahead Adder.

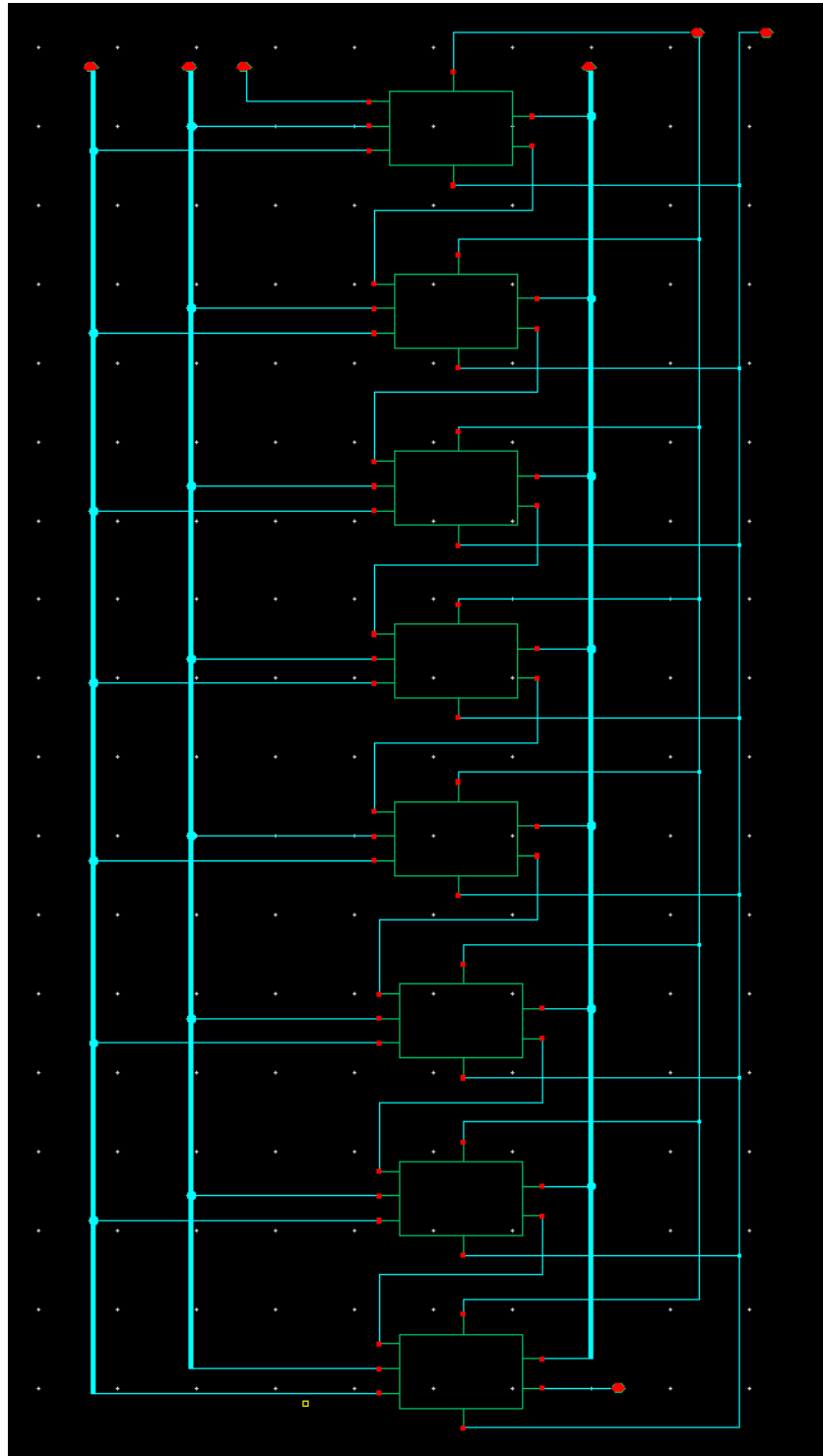
The calculated propagation delay is 591.3 us.

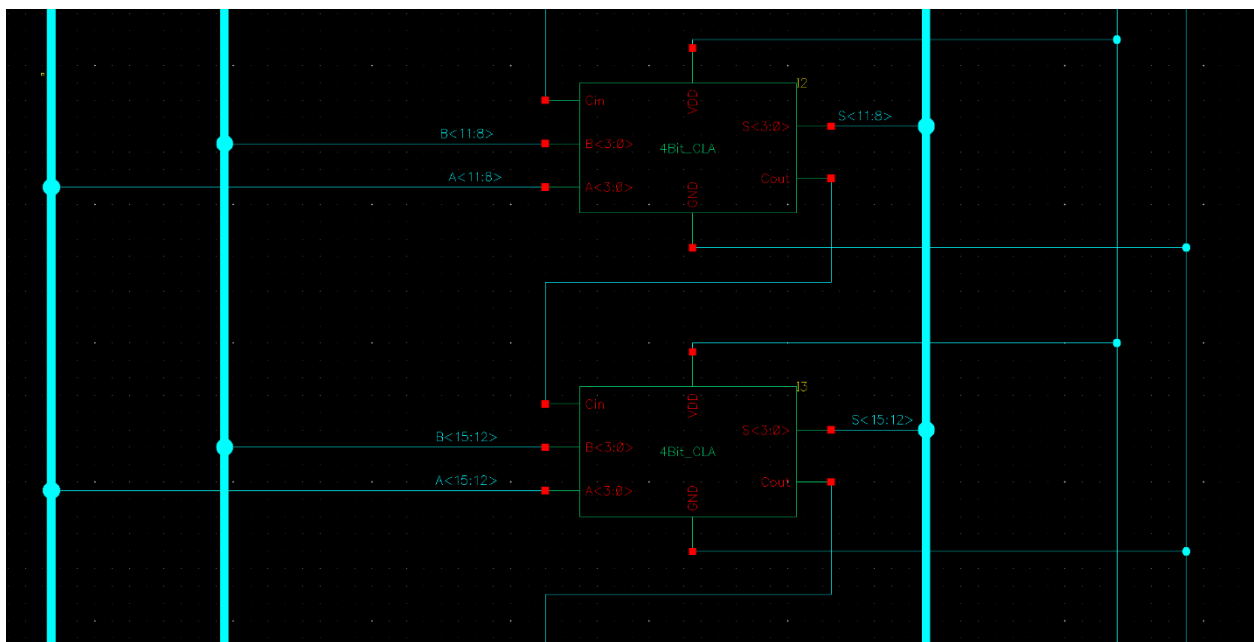
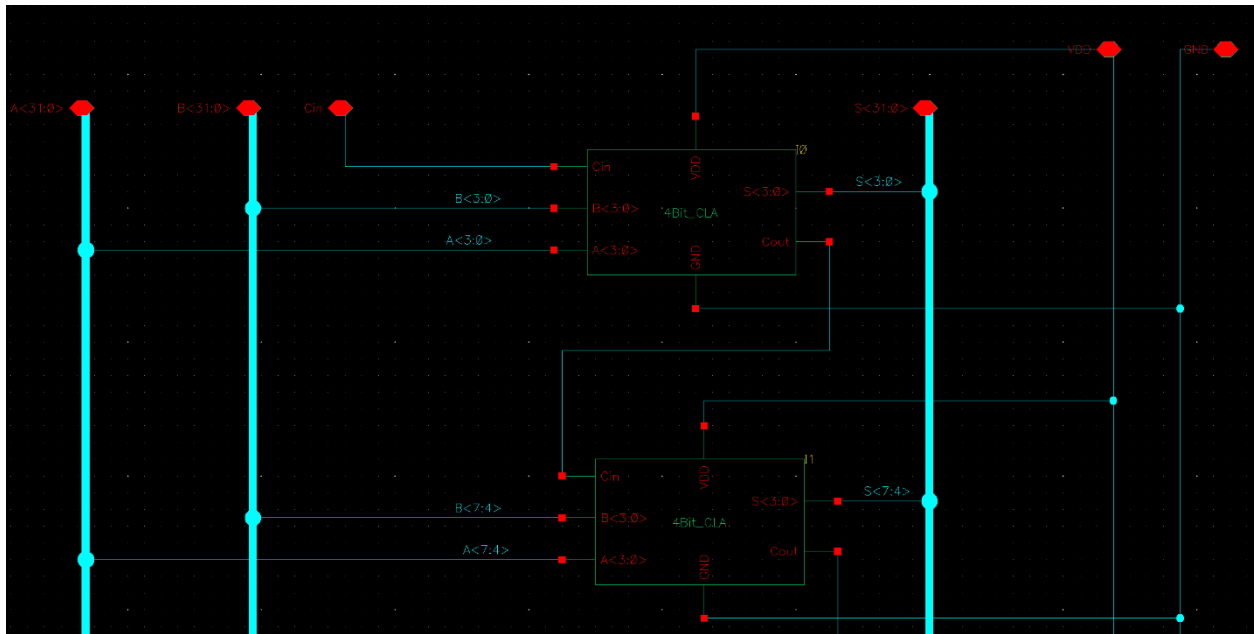
32 – Bit Carry Look Ahead Adder

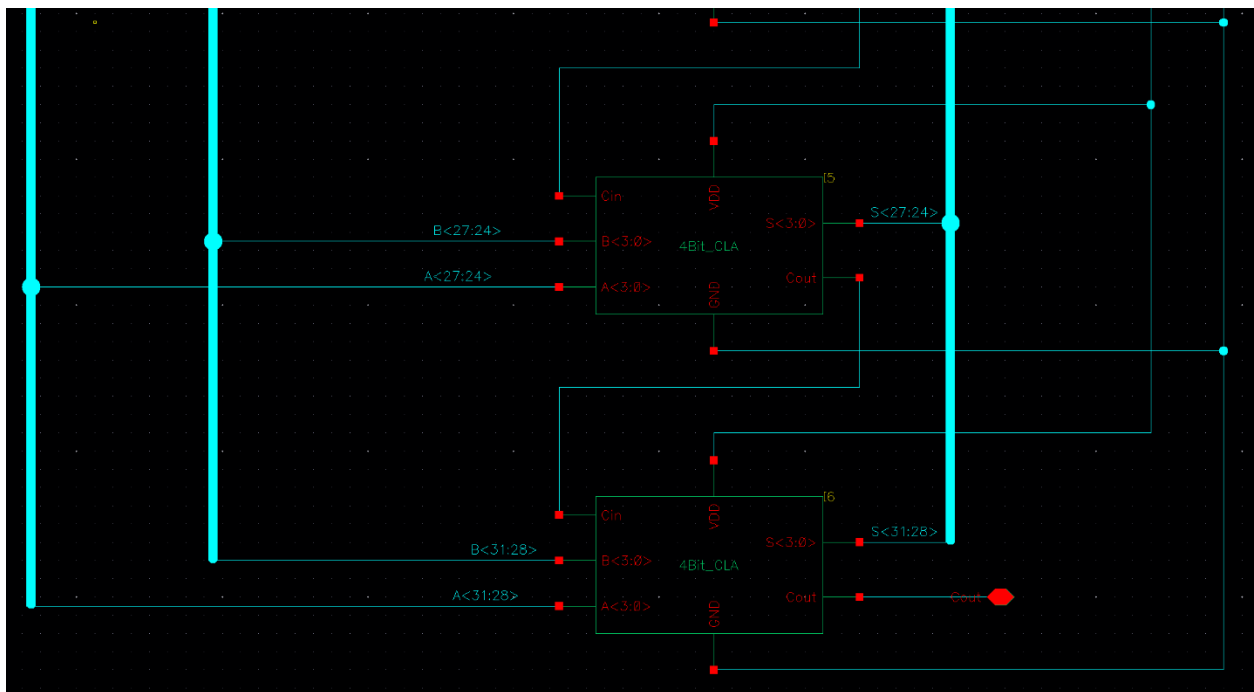
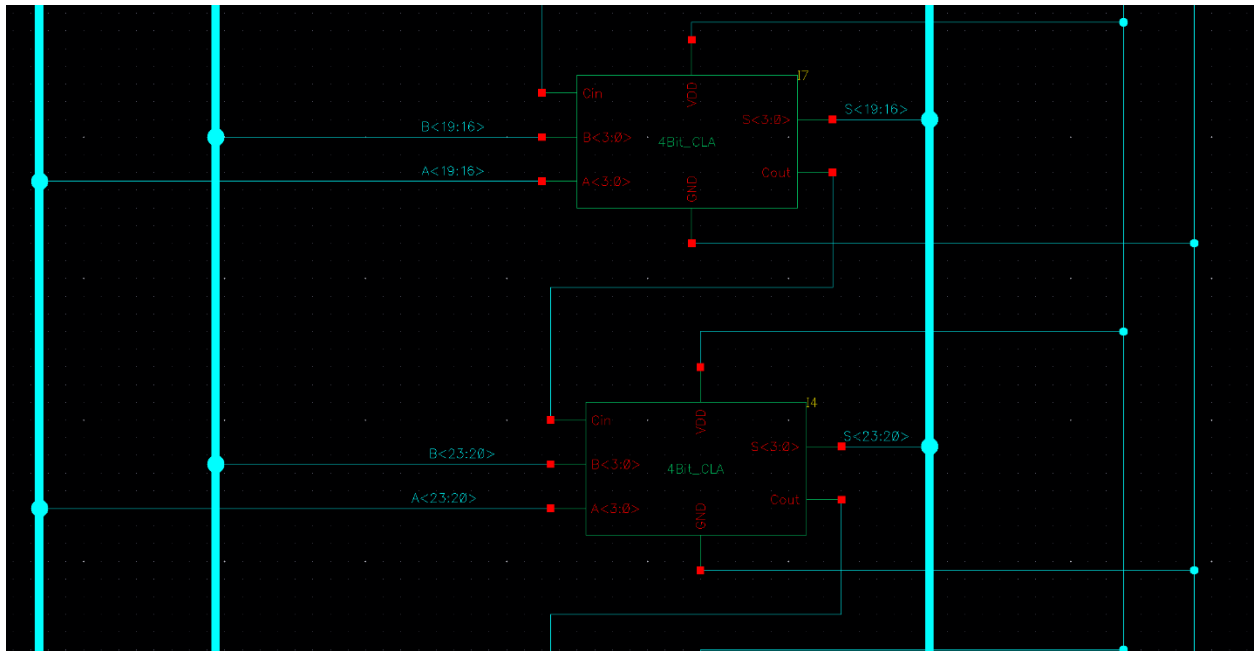
Scope

As described in the *Chosen Design Methodology* section of this report, the 4 – Bit CLA Adder will be cascaded 8 times to achieve the 32 – bit adder.

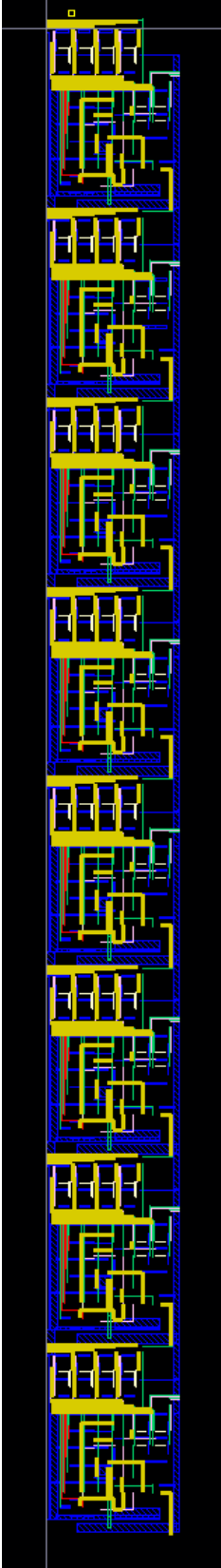
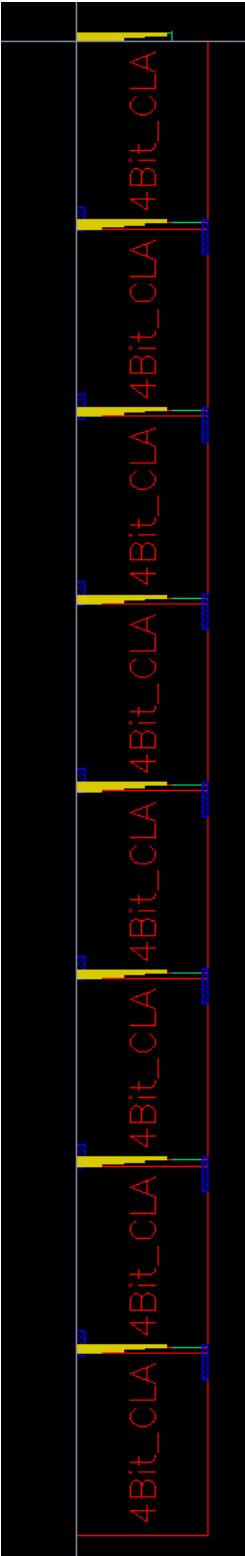
Schematic

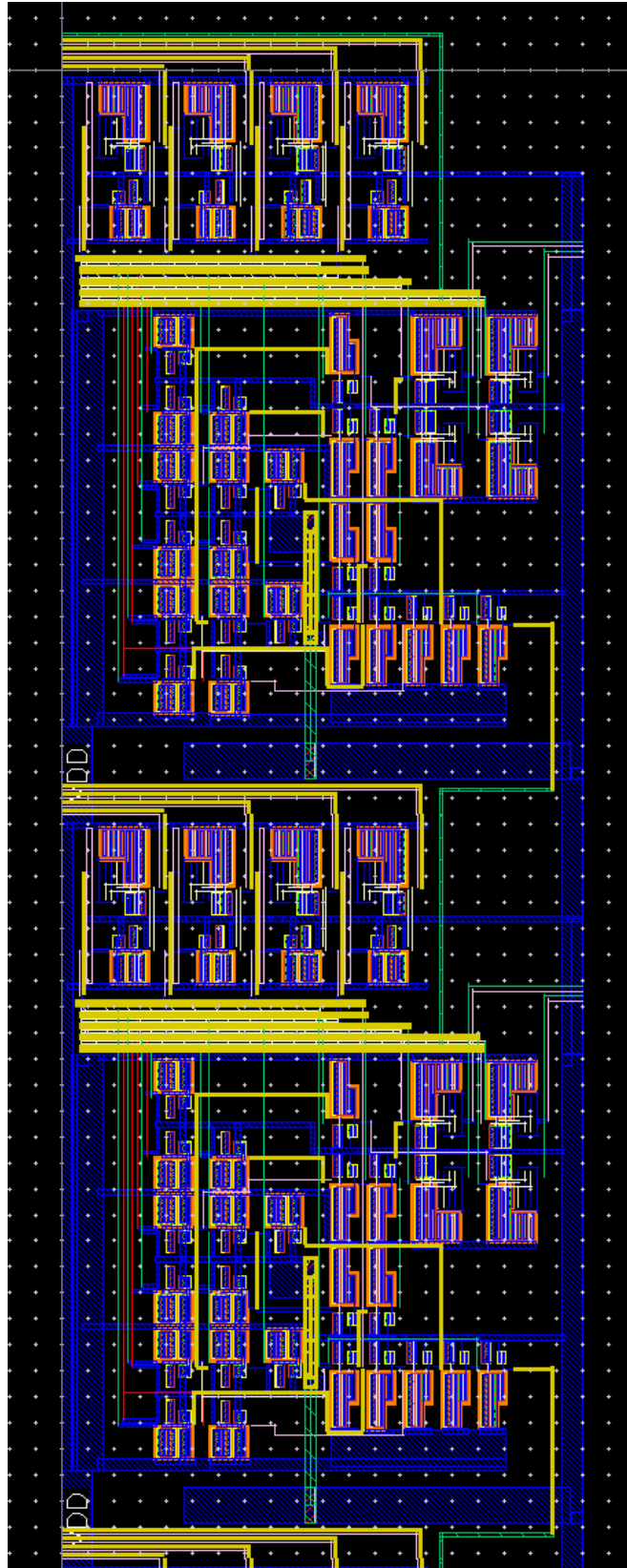






Layout

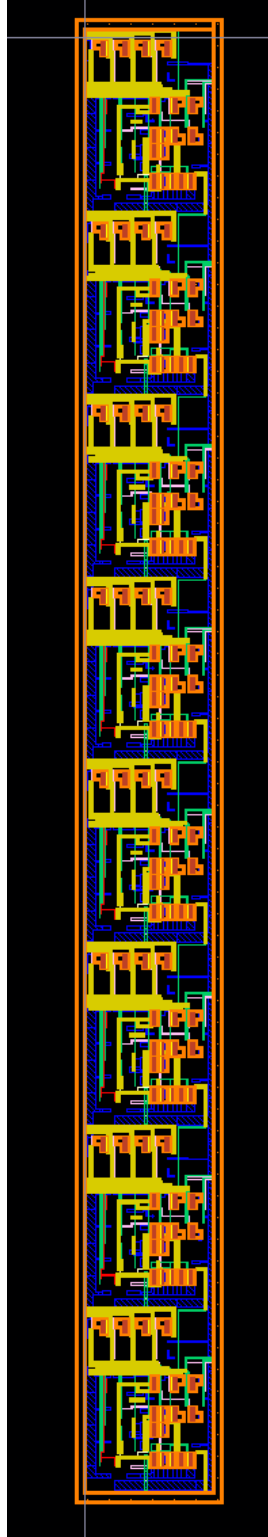


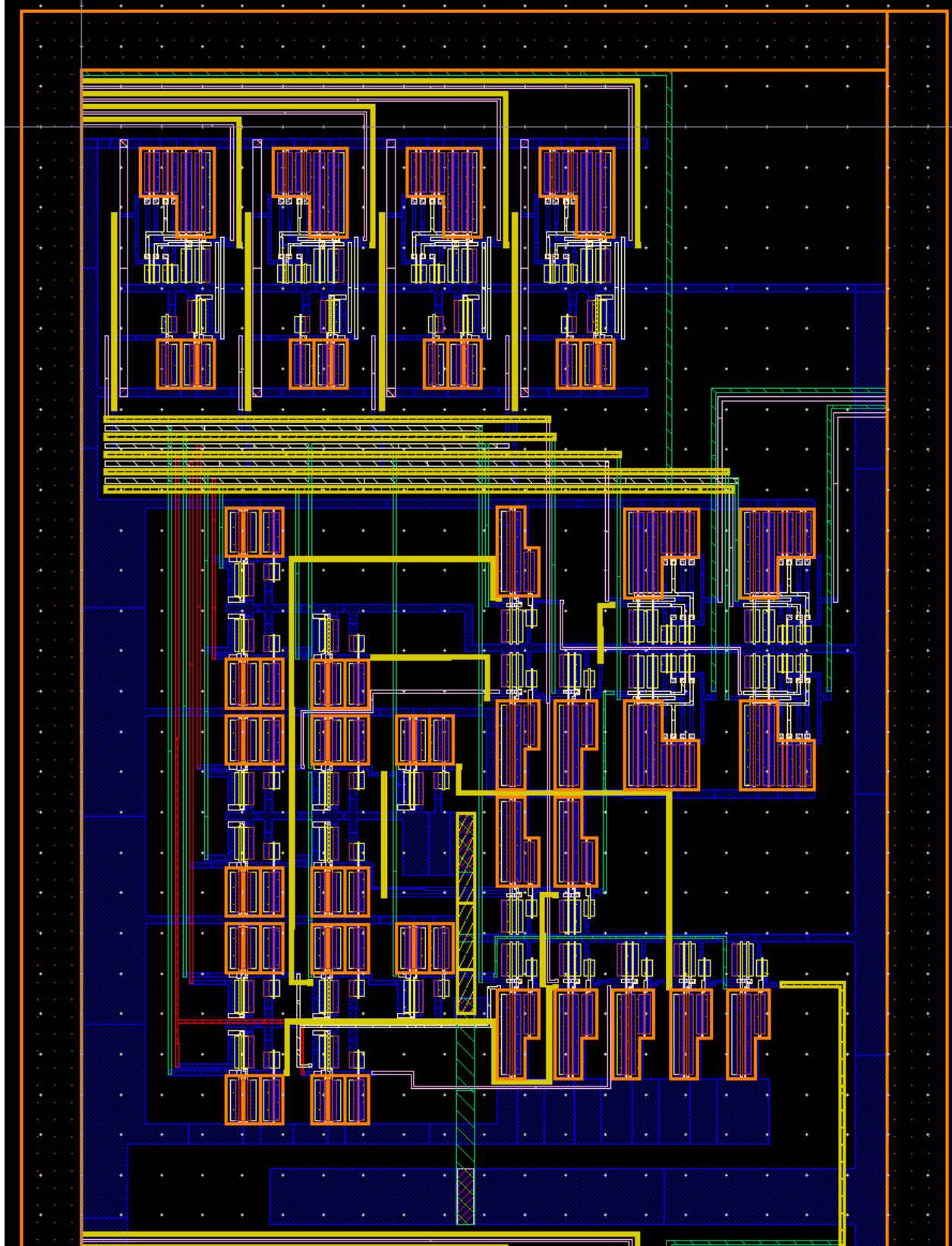


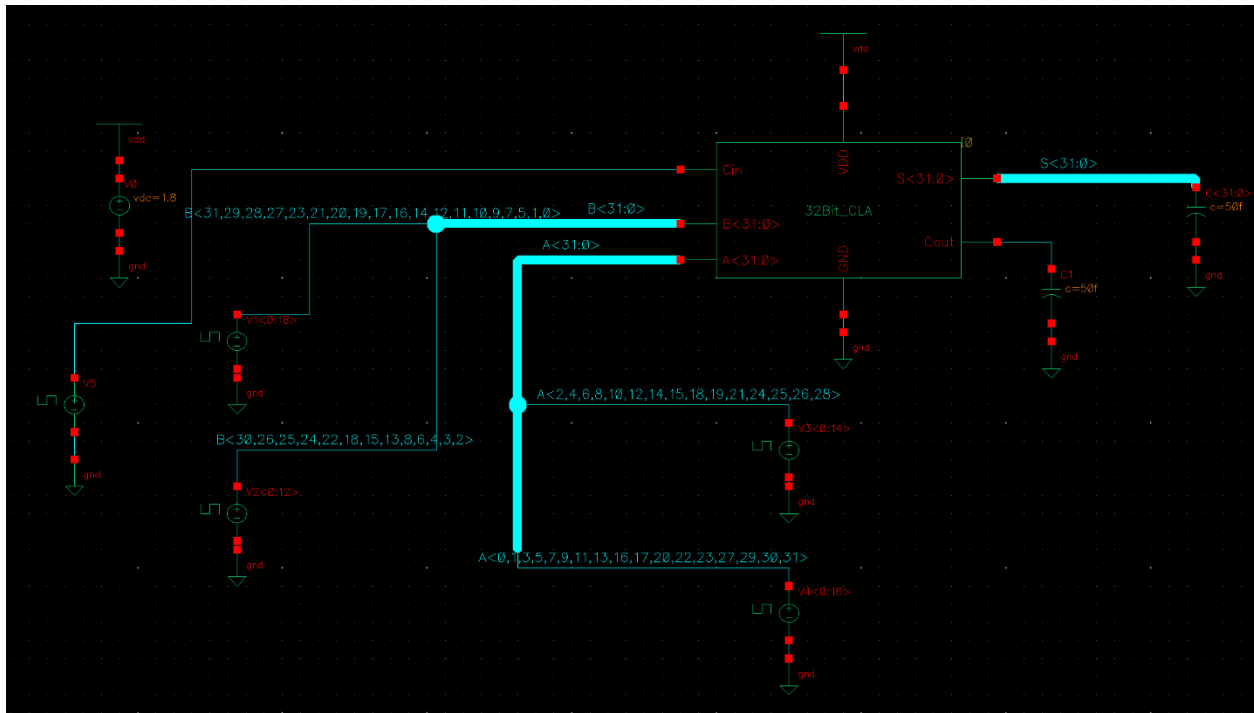
The calculated area of the 32 – Bit CLA area is $1144.78 \text{ um} \times 99.935 \text{ um}$ or 114403.6 um^2 .

Post – Layout Simulation

The *AV_Extracted* file for the 32 – Bit CLA adder is included below to confirm that all components used in this 32 – Bit adder has passed DRC, LVS, and Quantas.







The simulated input vectors and expected output vectors are shown below.

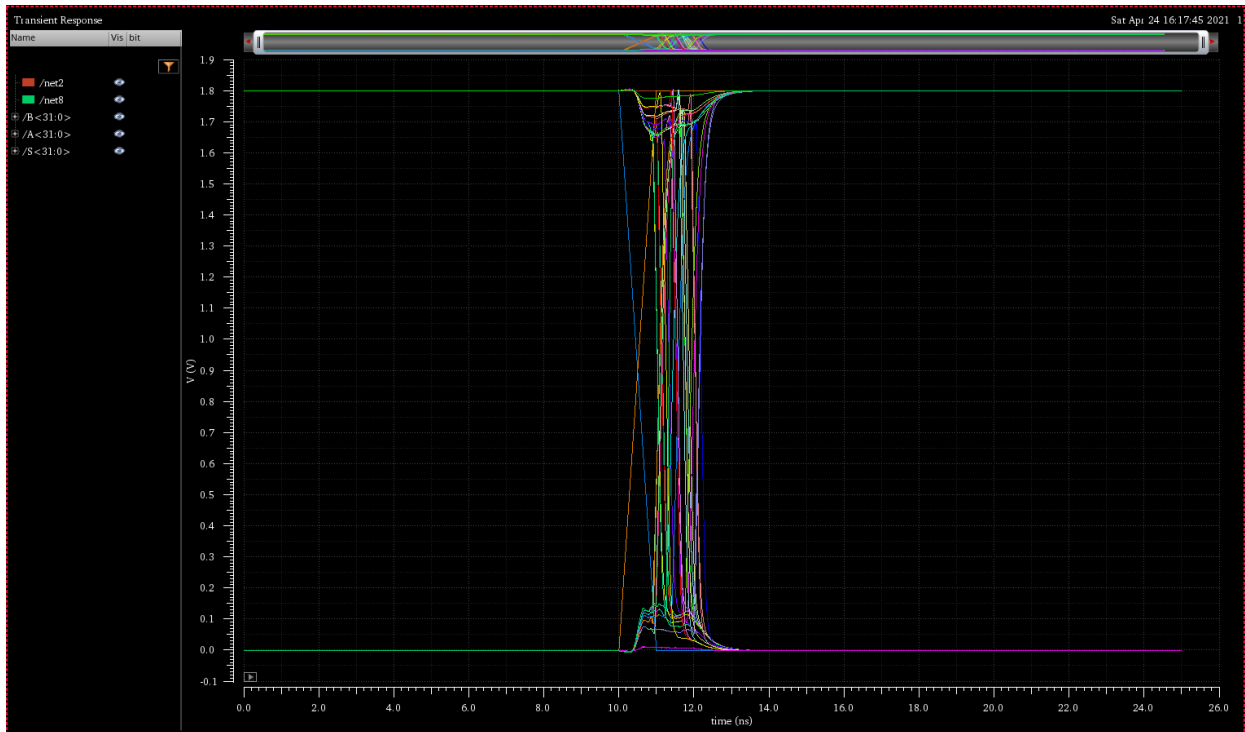
$$A = 388814164$$

$$B = 3099287203$$

$$Cin = 1$$

$$S = 3488101368$$

$$Cout = 0$$



The calculated propagation delay is 1027.3 us. This is calculated by observing the 32nd bit change as different inputs were given to the ALU. This is the worst case propagation delay as the 32nd bit of the Sum will become valid only once the initial input signal is propagated through the 8 cascaded 4 – bit CLA adders.

32 – Bit OR gate (Erica Chen)

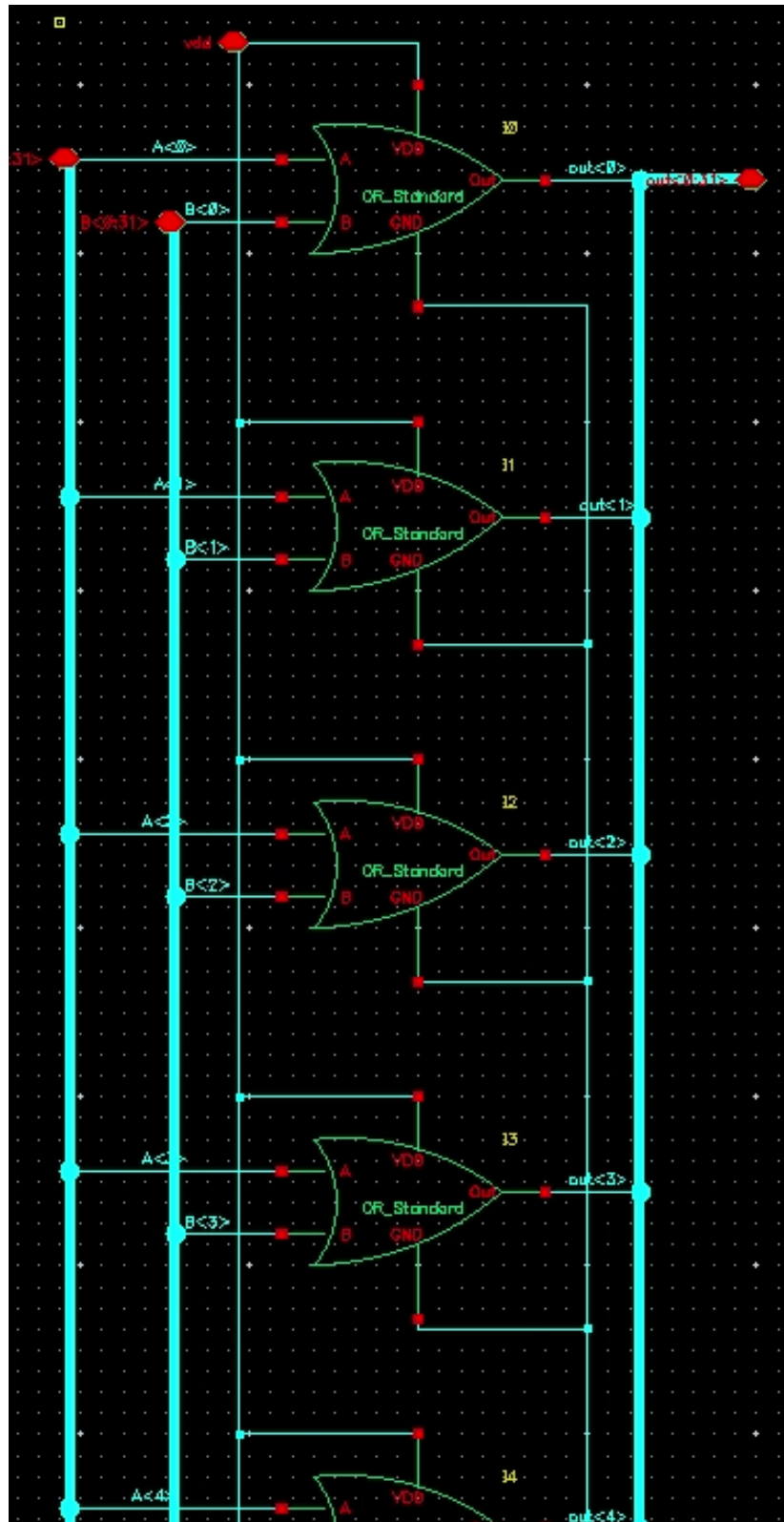
Constraints and Functionality

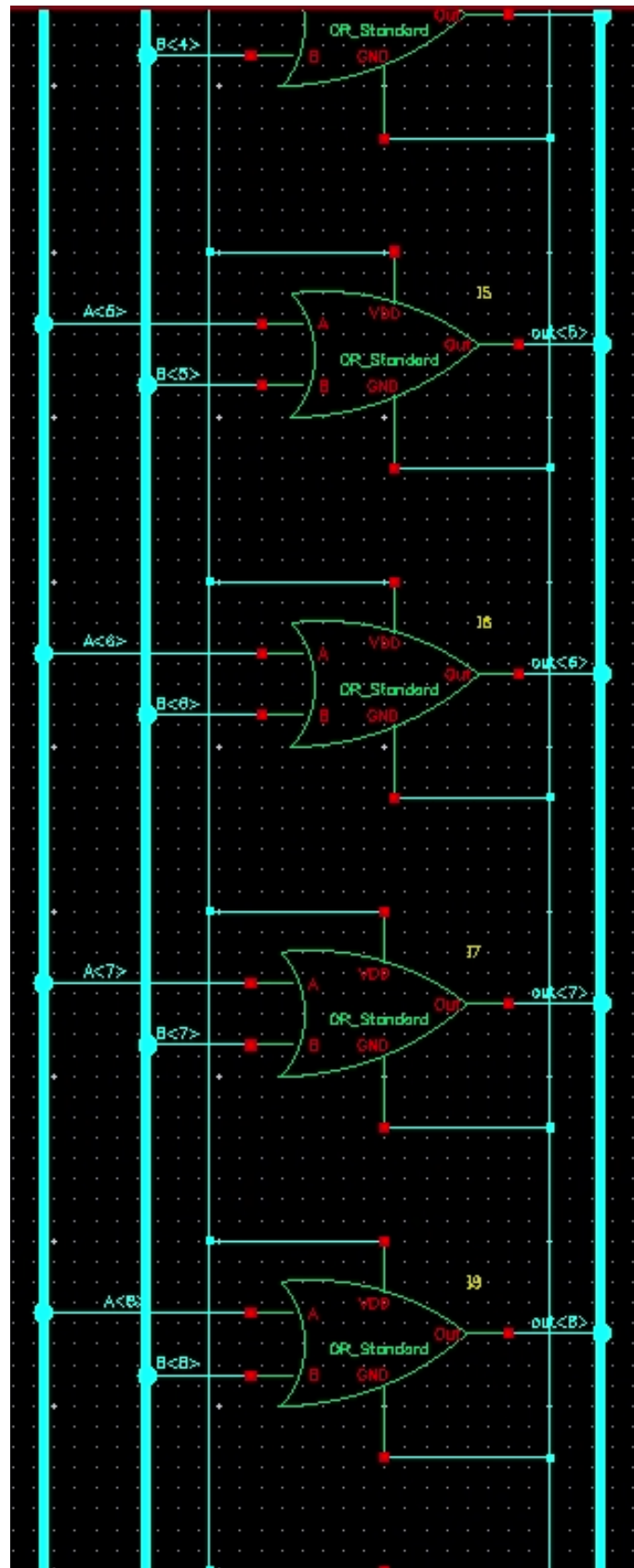
This is a bitwise 32-bit OR gate following the function $A + B = \text{OUT}$

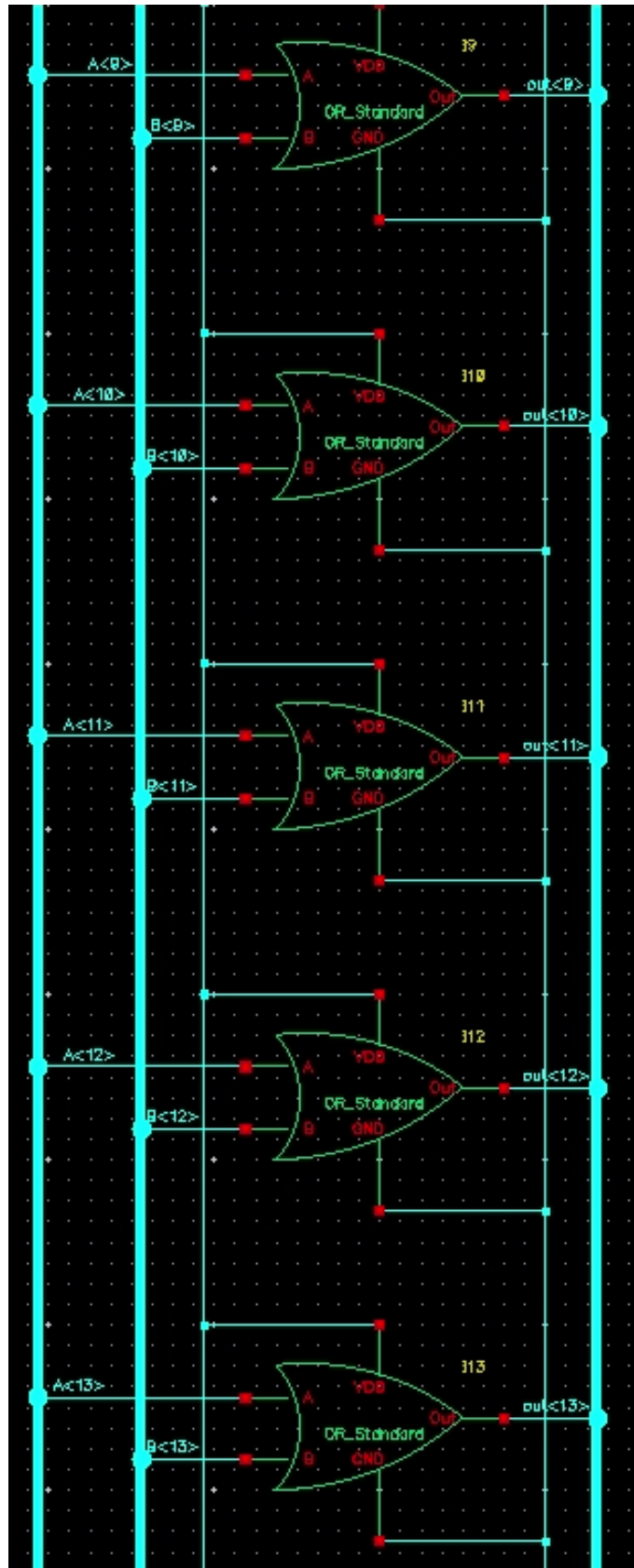
Design Methodologies

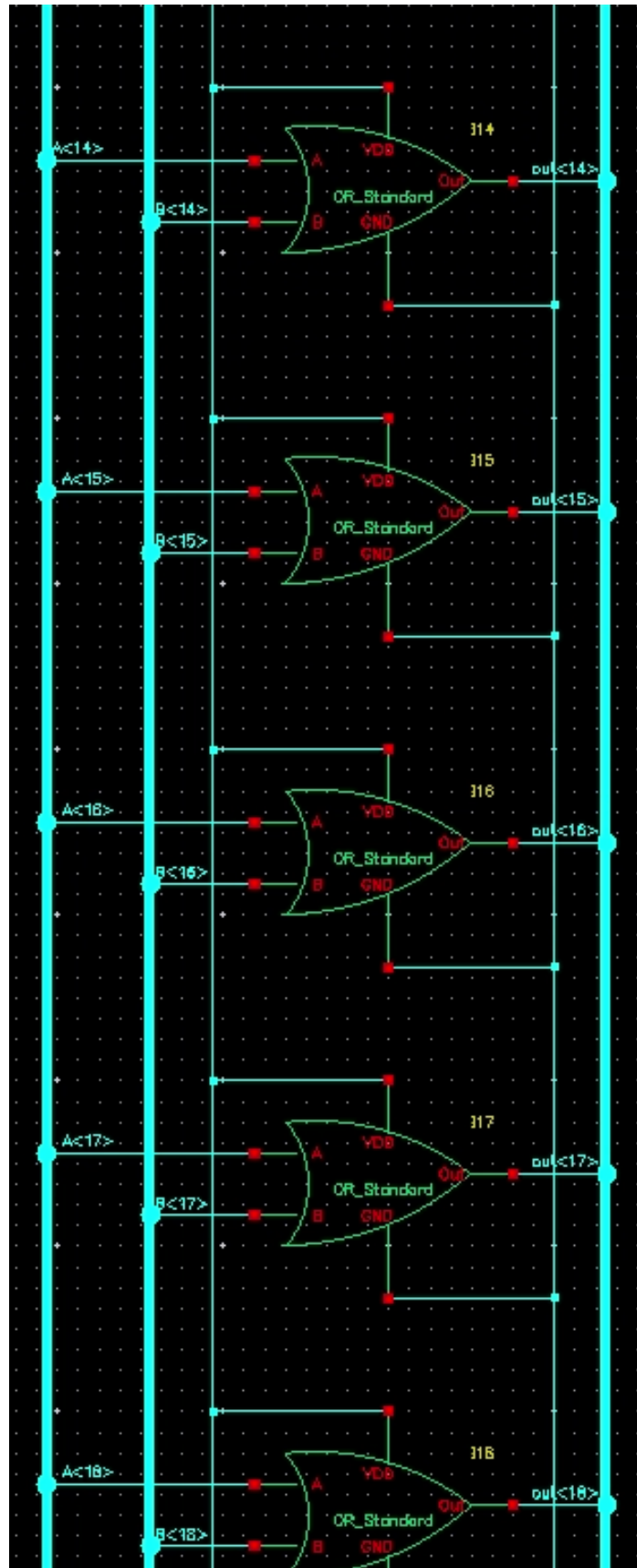
The 1-bit OR gate, previously made by David, was cascaded 32 times. A 32-bit bus was assigned to A, B, and OUT. All design rules were made following the class tutorials' steps.

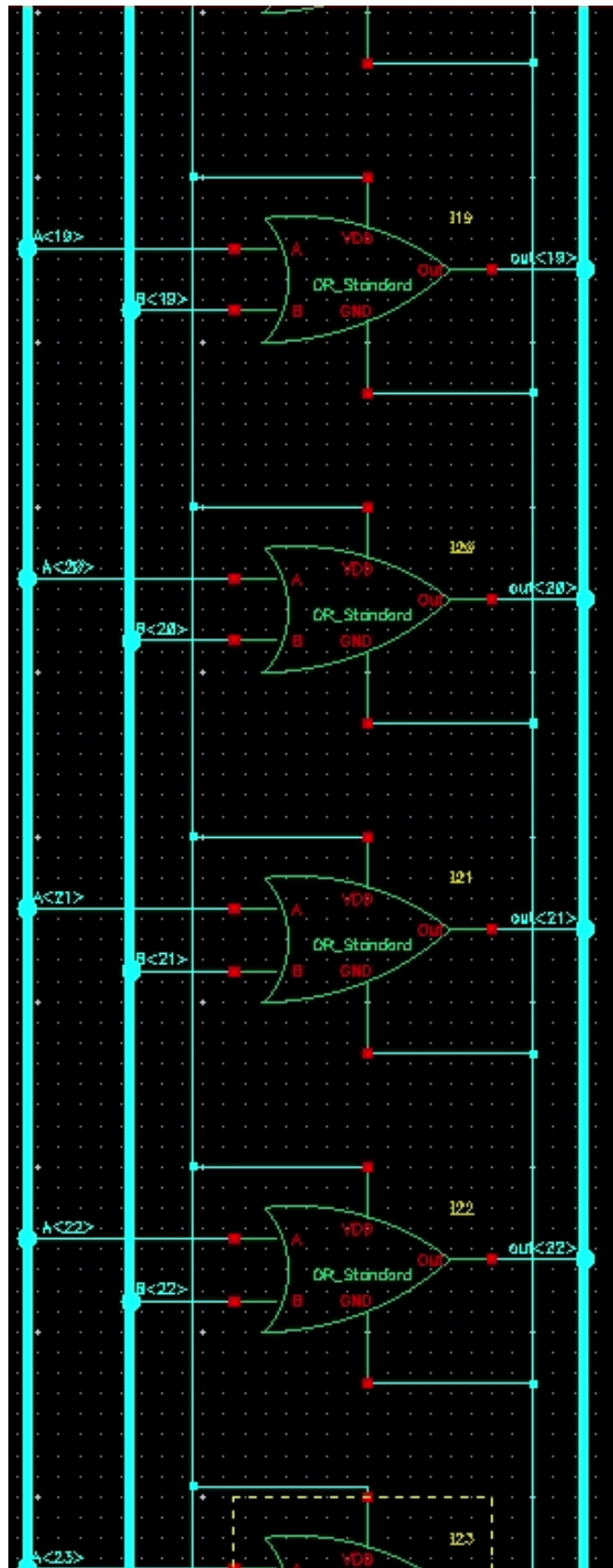
Schematic

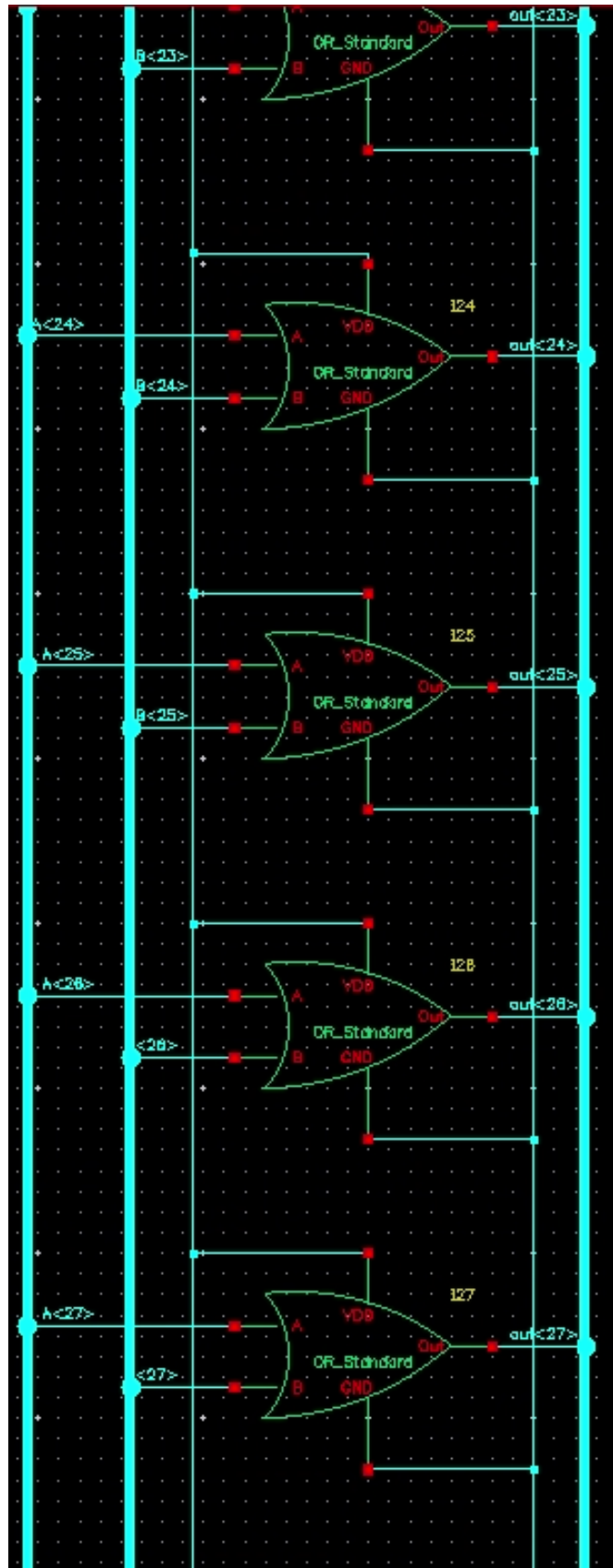


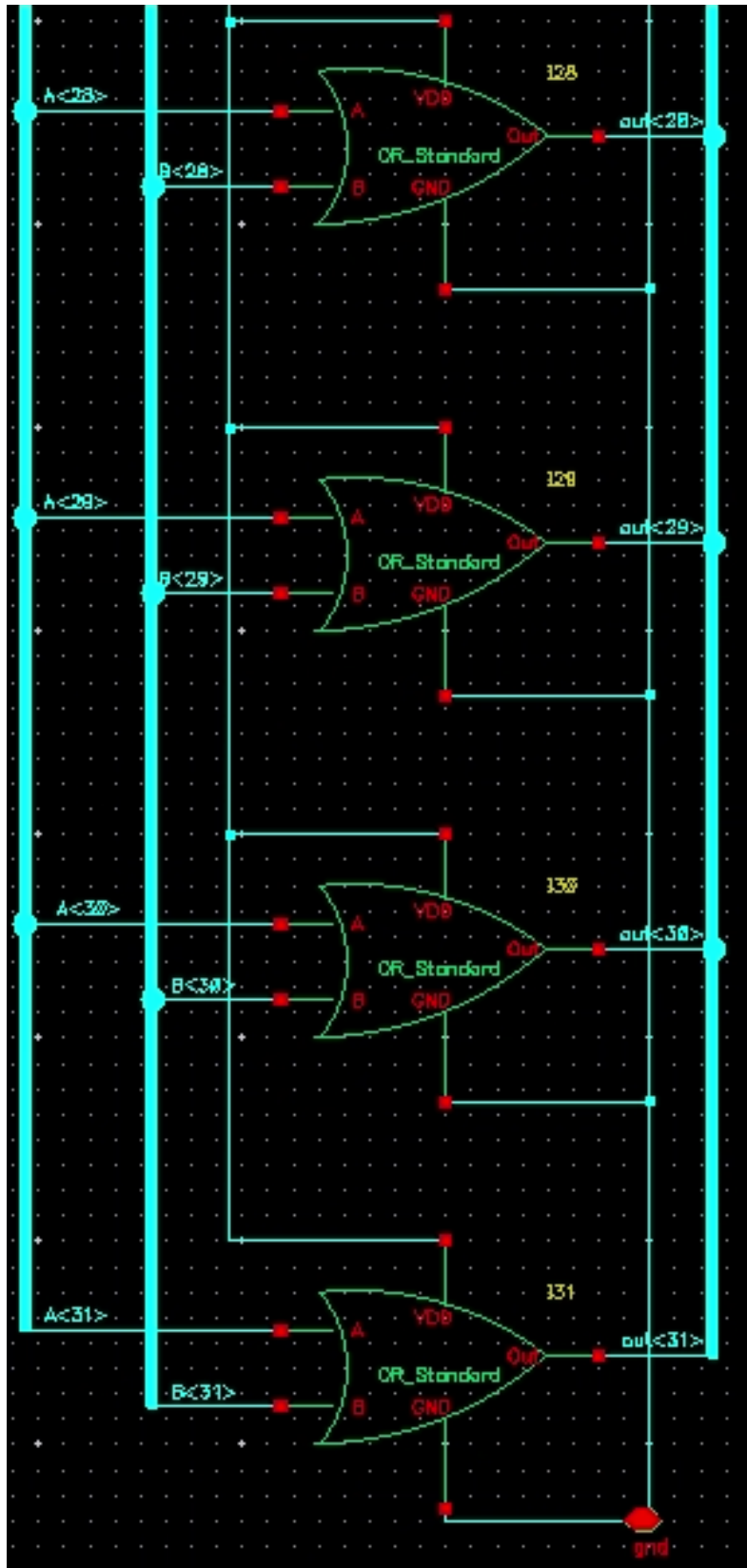




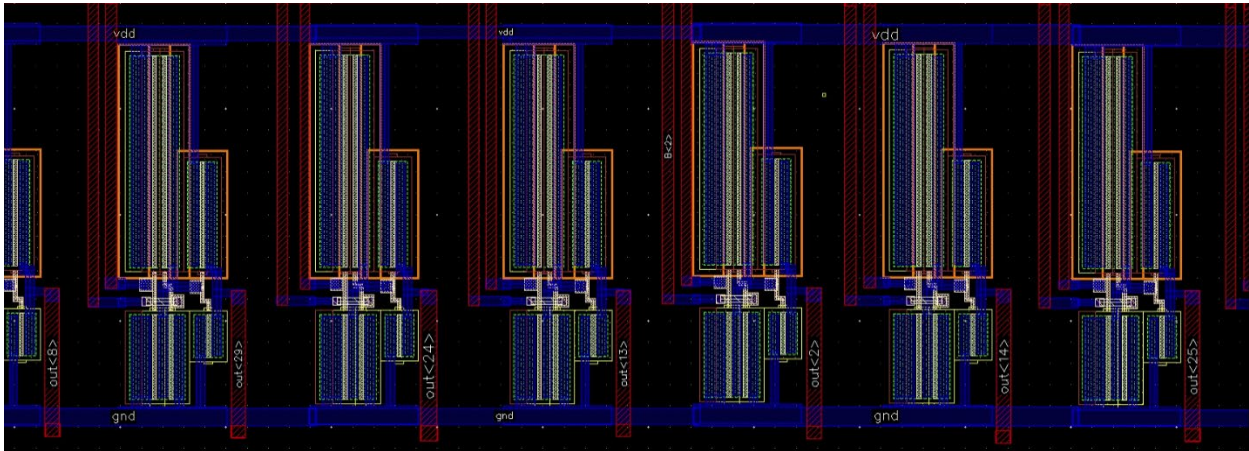
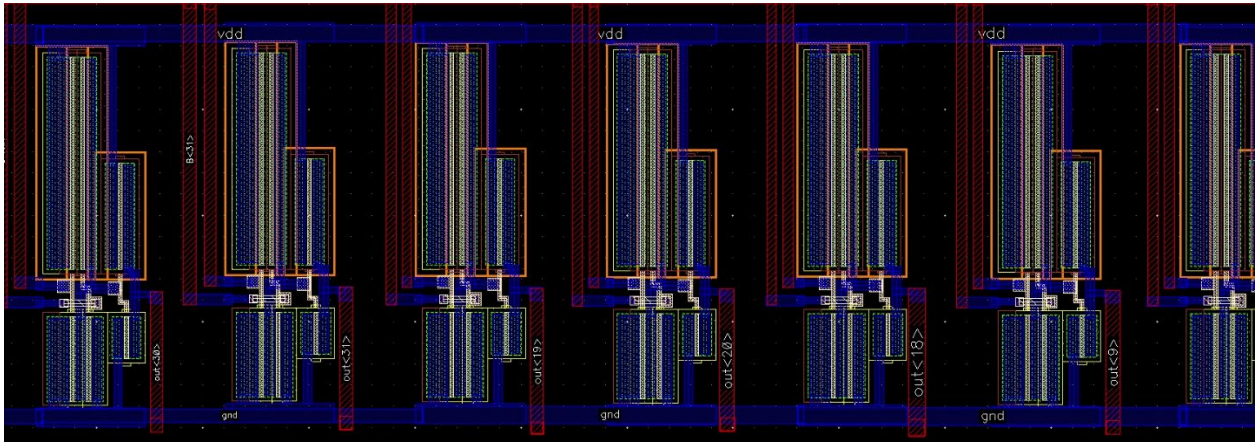
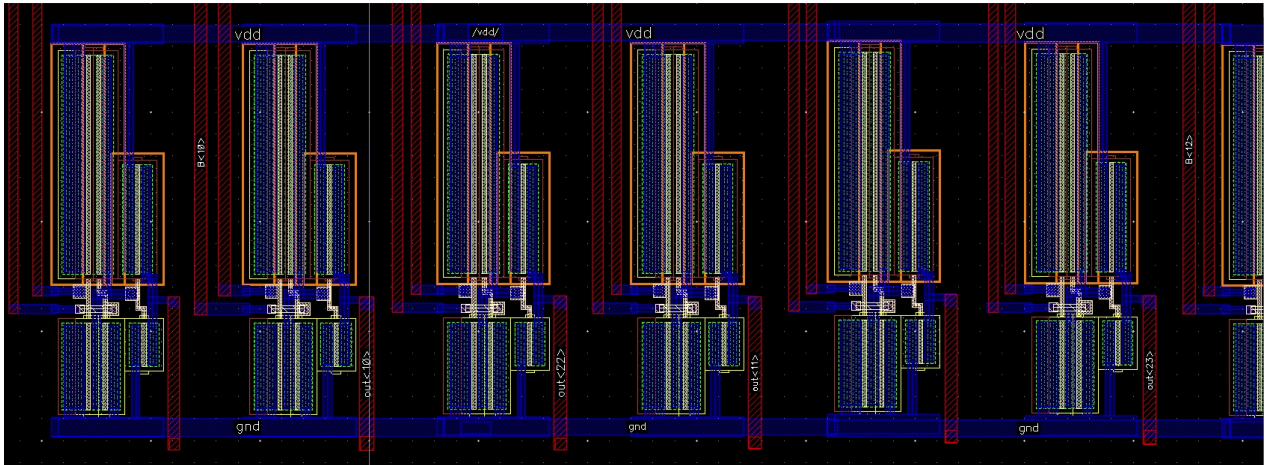


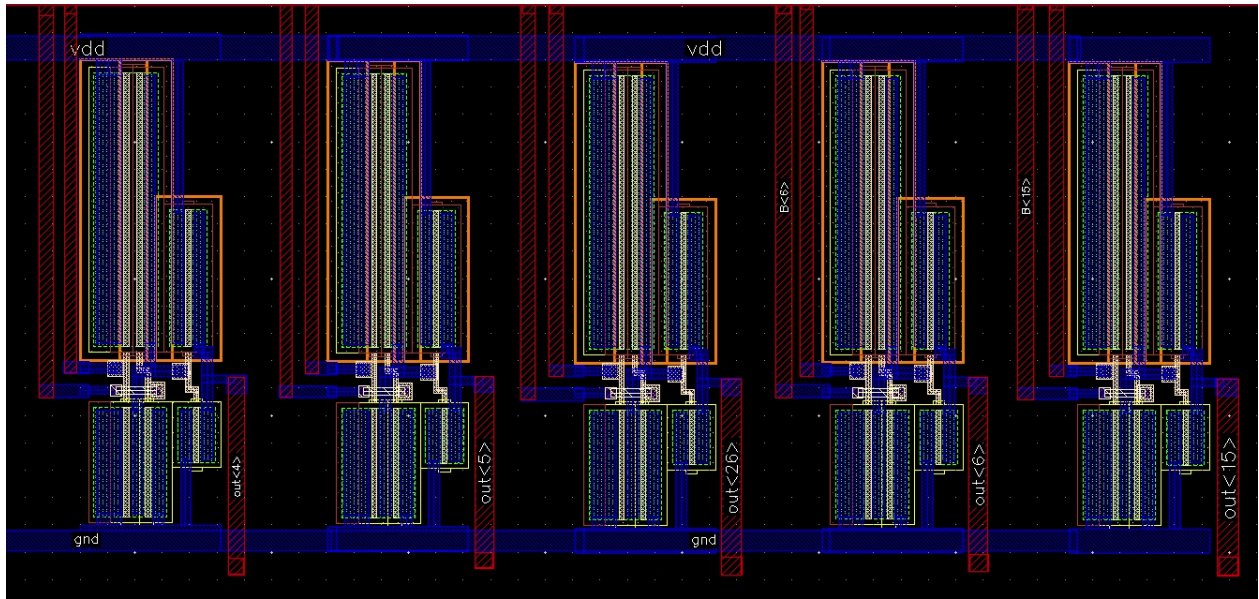
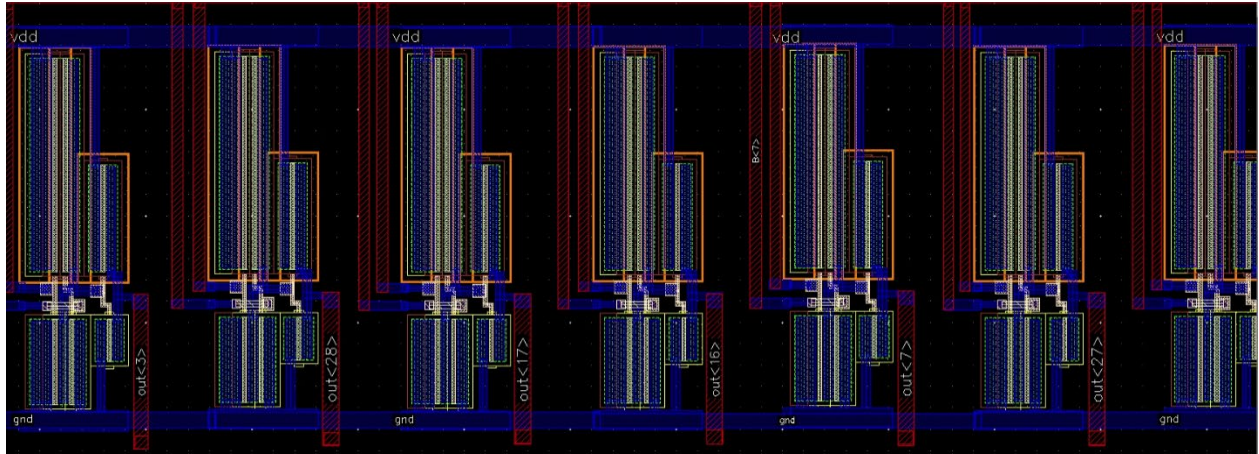






Layout





32 - Bit AND gate (Tiffany Chan)

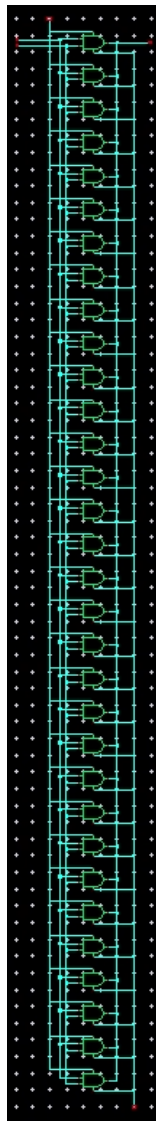
Constraints and Functionality

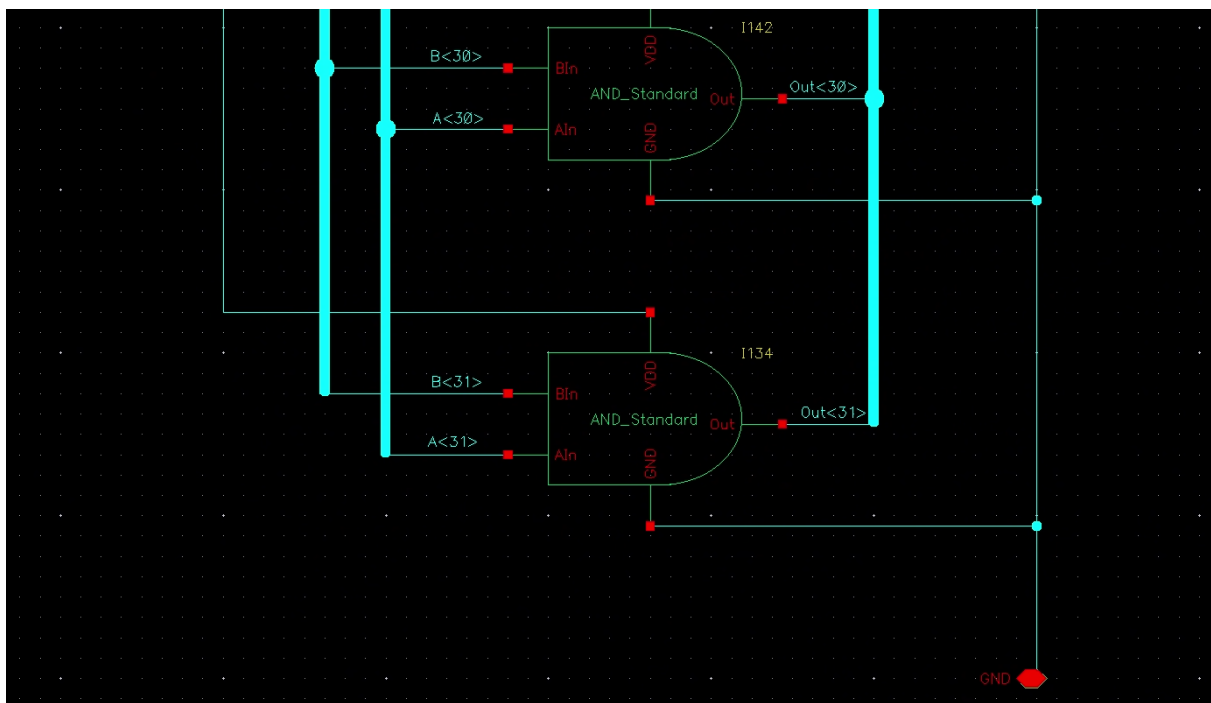
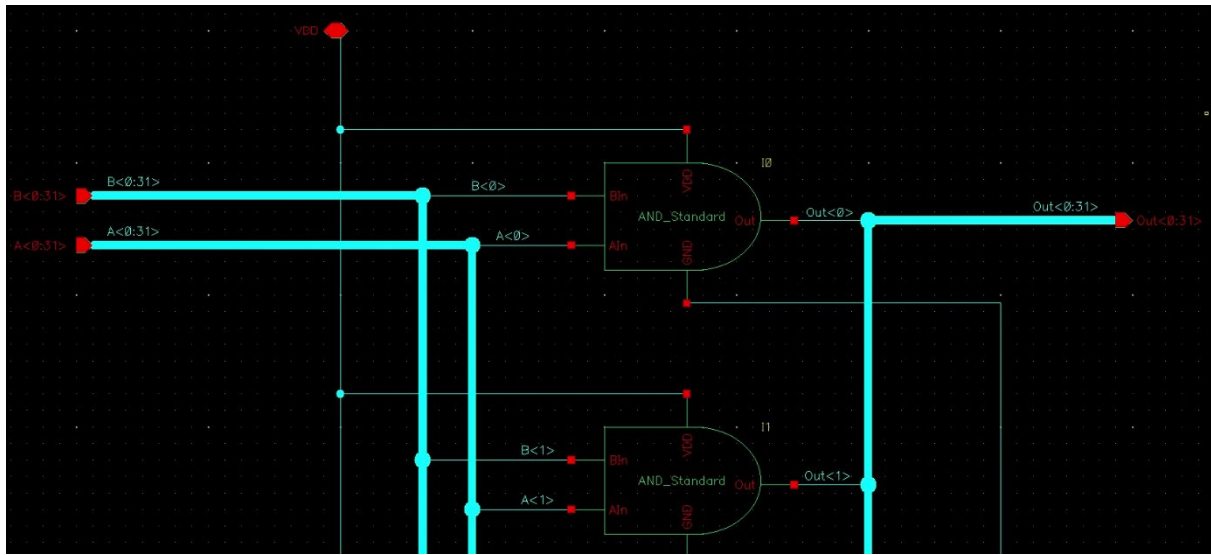
This is a bitwise 32-bit AND gate following the function $A \bullet B = \text{OUT}$

Design Methodologies

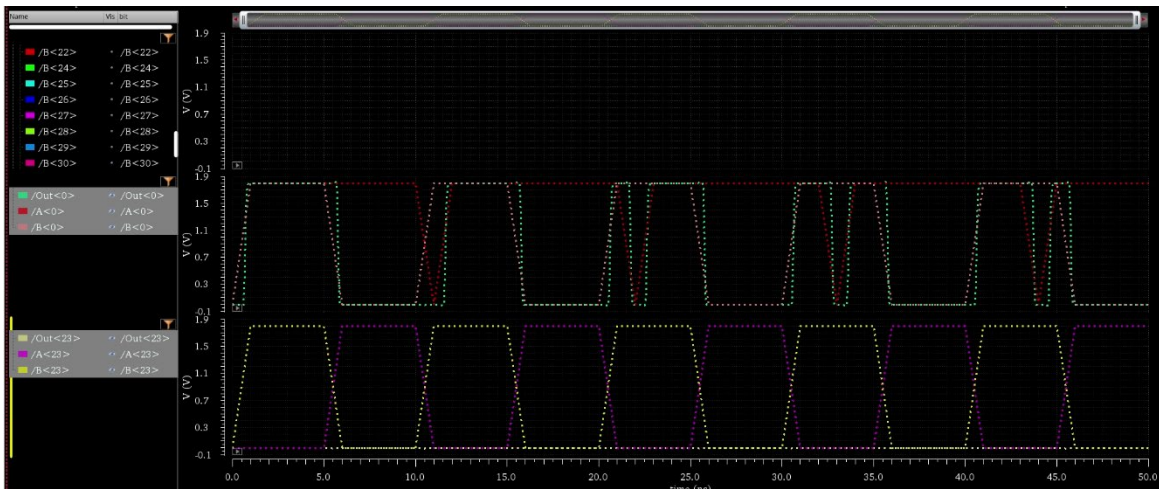
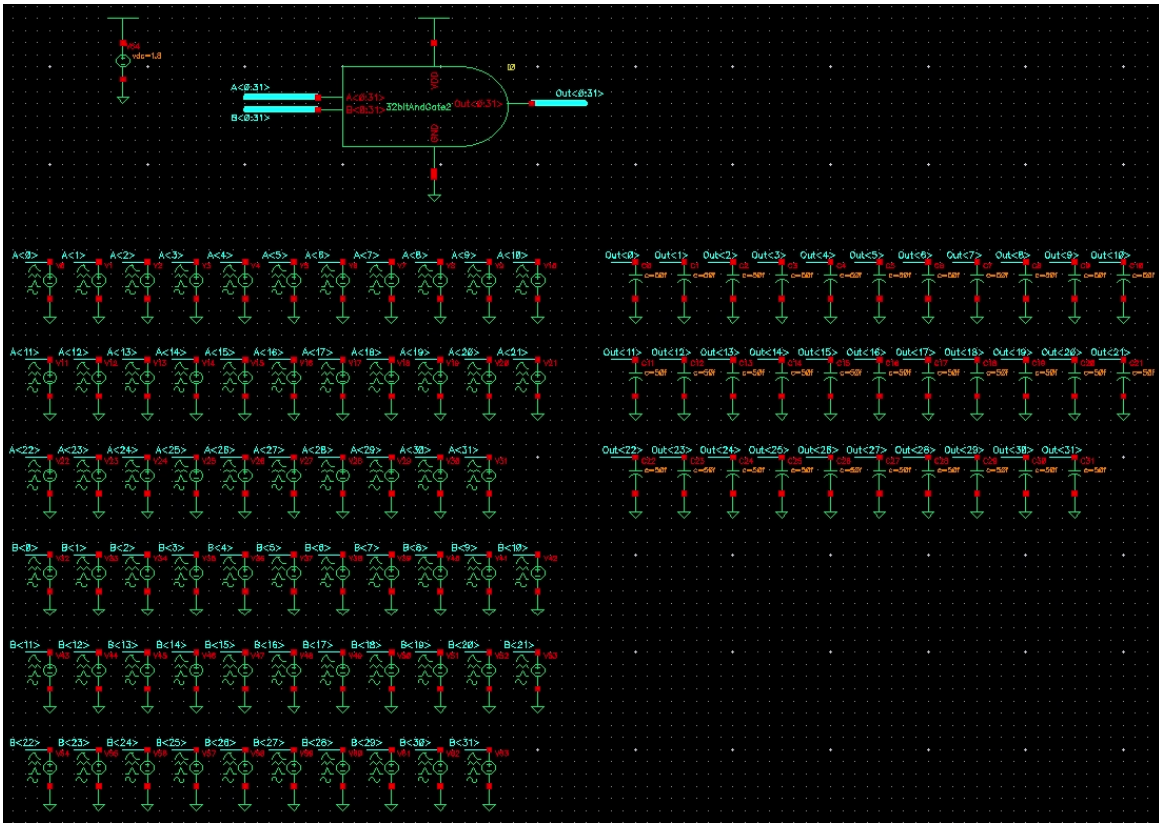
The 1-bit AND gate, previously made by David, was cascaded 32 times. A 32-bit bus was assigned to A, B, and OUT. All design rules were made following the class tutorials' steps.

Schematic



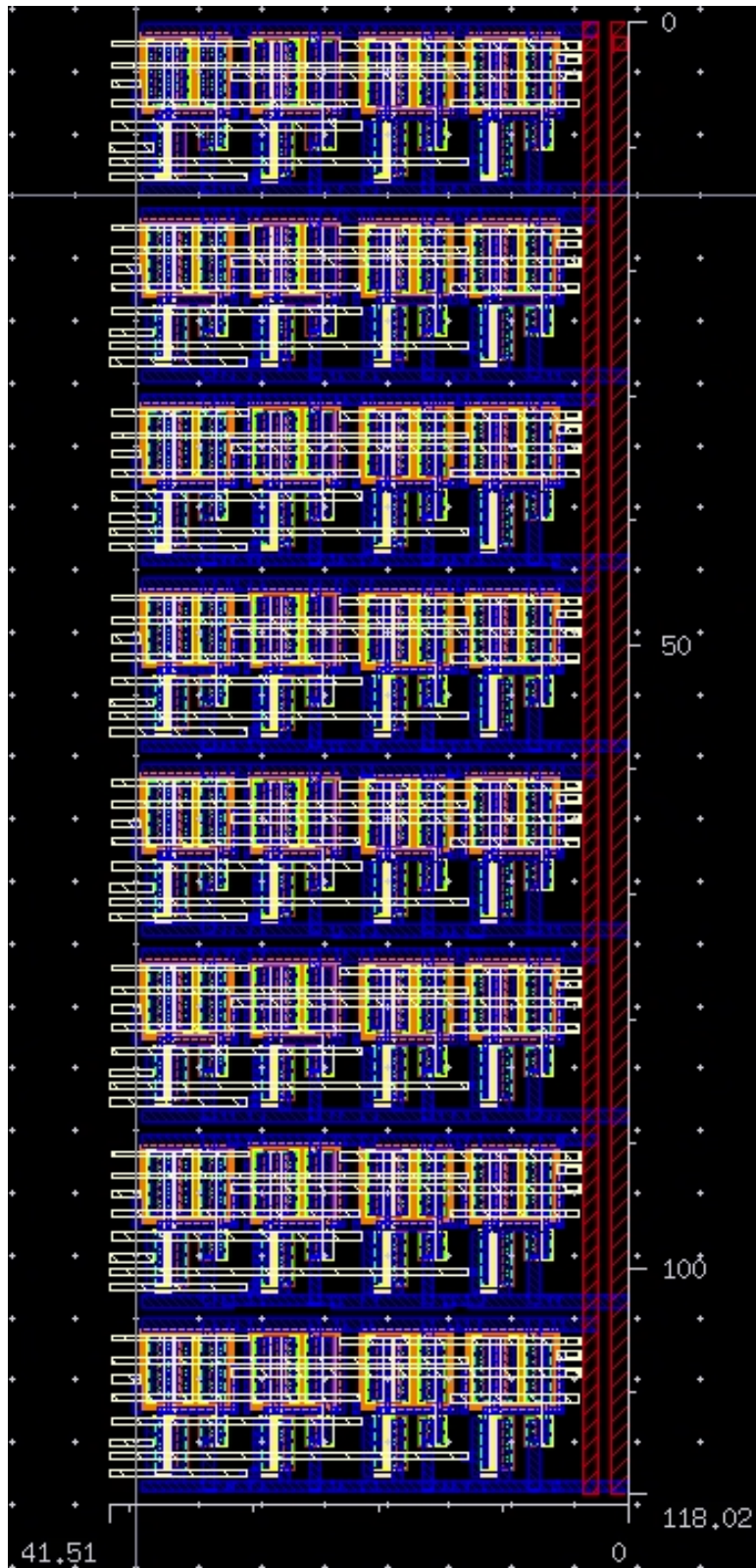


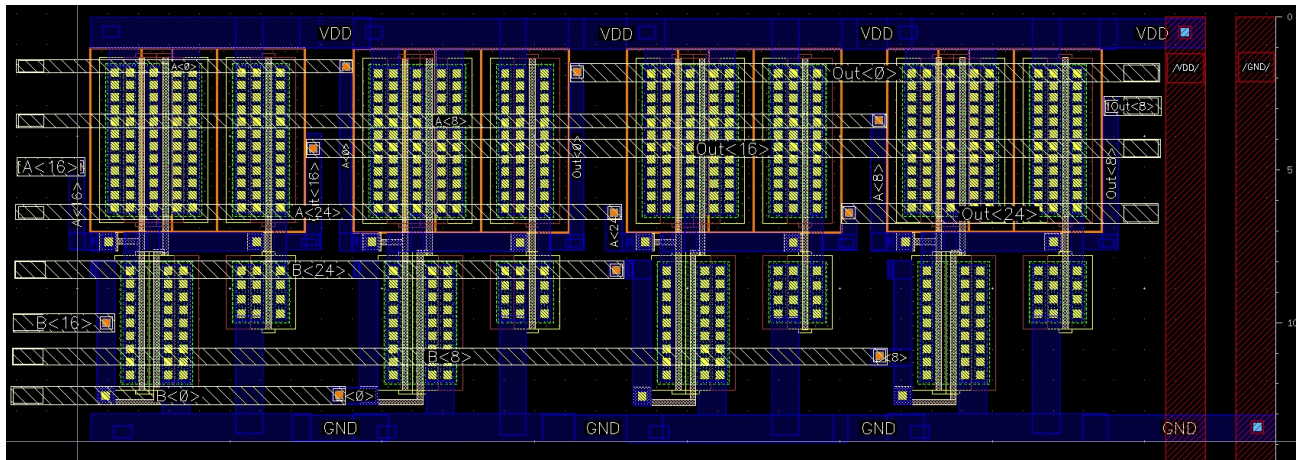
Simulation



Corresponding example bits are shown for A, B, and OUT. A<0> is a waveform with a pulse width that is 2 times longer than B<0>. Thus, the OUT of $A \cdot B$ is the pictured green waveform. Meanwhile, A<23> and B<23> have pulses of opposite polarity, so the OUT of $A \cdot B$ is 0V.

Layout



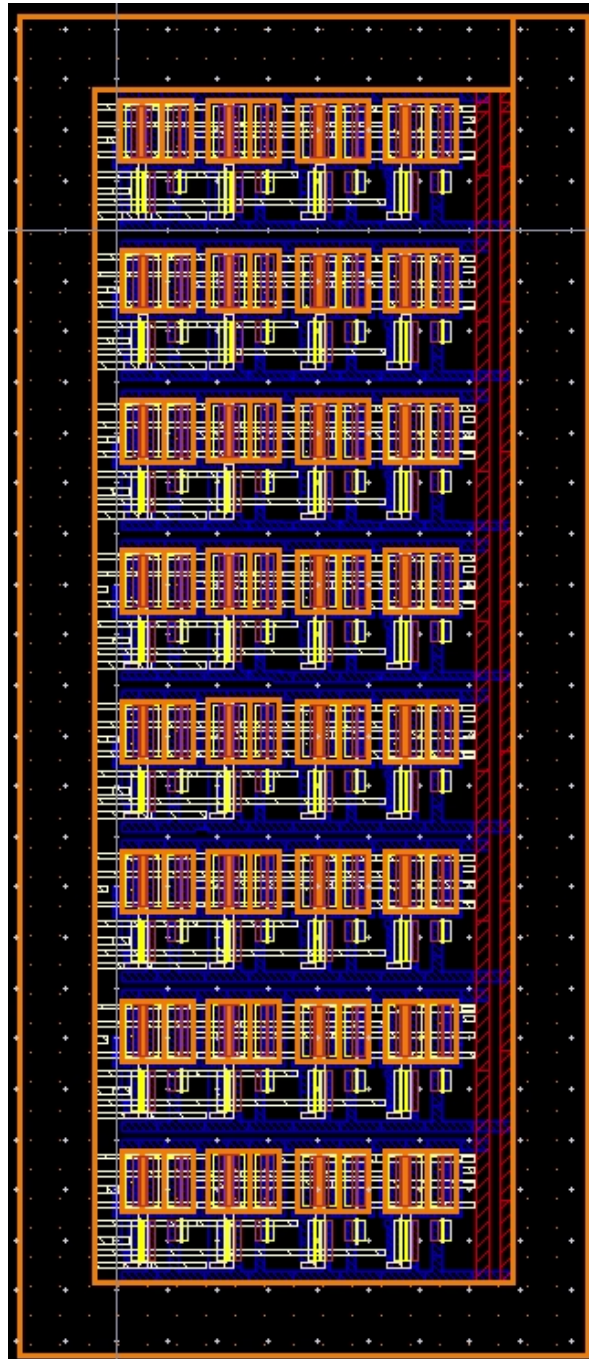


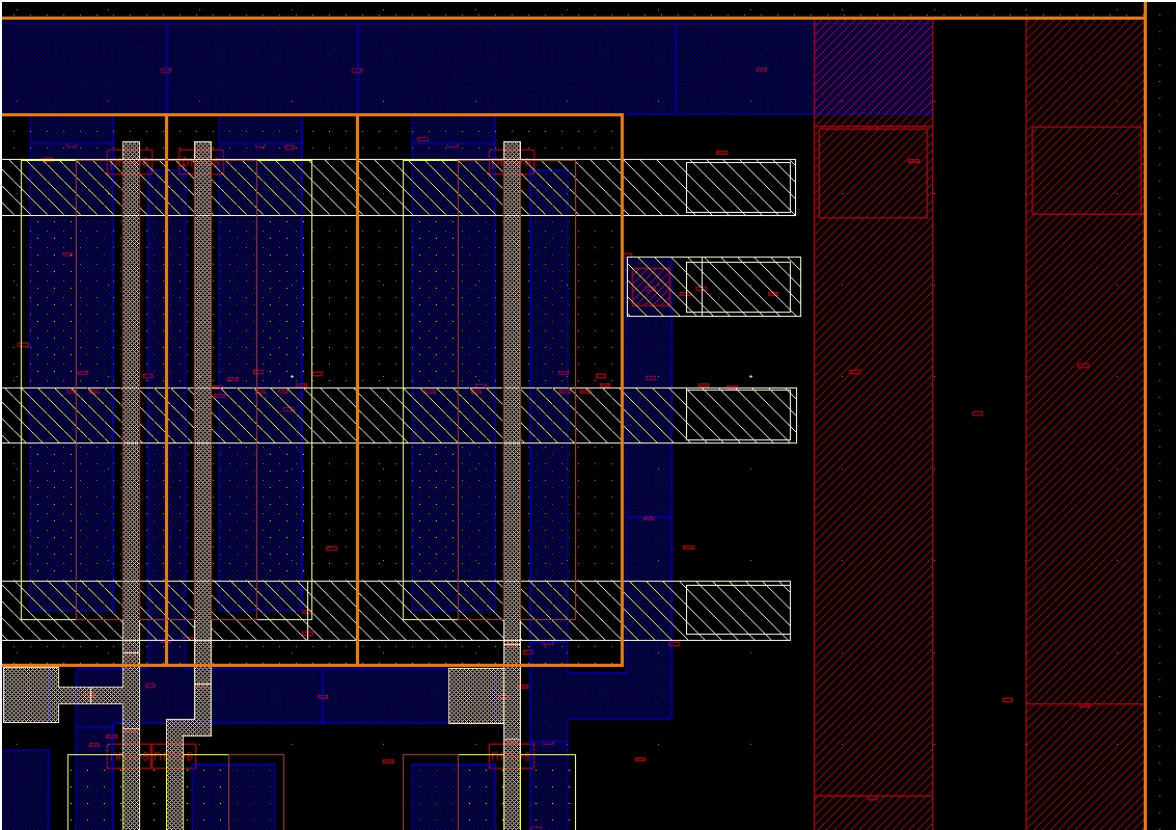
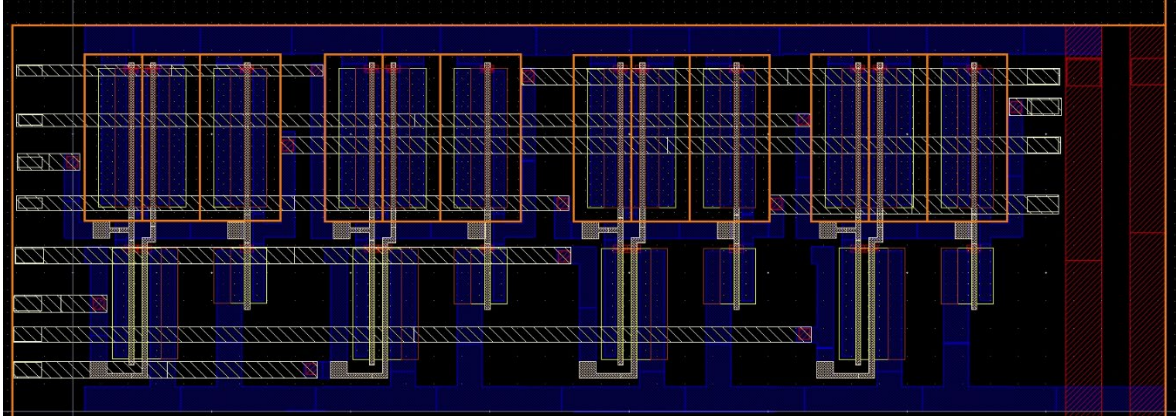
The layout is a 4x8 array of 1-bit AND gates. The area is 4897 μm^2 .

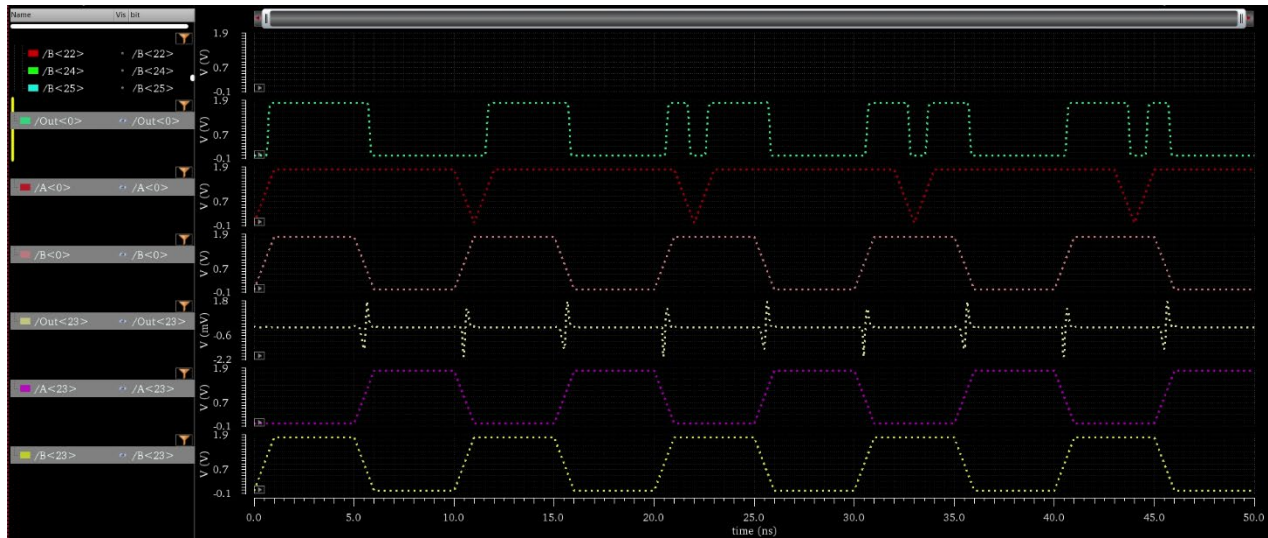
There is a VDD rail at the top of every row and a GND rail at the bottom of every row, and VDD and GND busses are on the right at the same side as OUT. Inputs A and B are on the right.

Post-Layout Simulation

The *AV_Extracted* file for the 32 – Bit AND gate is included below to confirm that all components used in this 32 – Bit AND gate has passed DRC, LVS, and Quantas.







The same bits as the pre-layout simulation are shown, but it is interesting to note that there is a ~1mV dip/spike for OUT<23> during the switching between 1.8V and 0V for A<23> and B<23>. Otherwise, the simulations don't look that different, probably because conservative design rules were followed.

Lessons Learned

Since this was the first part of my contribution to the project that I did, a lot of time was spent learning how to more efficiently use Cadence. Additionally, I encountered off grid errors constantly, and I learned that the best way to mitigate those was to do DRC very often (after I moved or drew a small group of things). And after much troubleshooting, if I did encounter them in my DRC, the next best thing to do was undo whatever I had done before the DRC and restart Cadence.

Additionally, something I wish I had done better was organize the gates in the layout so that when I drew the input and output pins, they would be in numerical order (ie: A<0> on the bottom and A<31> on the very top). This would make integration with other blocks easier. Since we didn't end up getting to integration in the scope of this project, this wasn't a big issue.

3 to 1 MUX (Tiffany Chan)

Constraints and Functionality

The MUX was made using a transmission gate configuration.

The pass gate configuration was considered but advised against during class. And the transmission gate configuration (10 transistors) uses significantly less transistors than the AND-gate configuration (32 transistors) [4].

The functional truth table for the MUX is as follows:

S2	S1	OUT<n>
0	0	A<n>
0	1	B<n>
1	0	C<n>

Design Methodologies

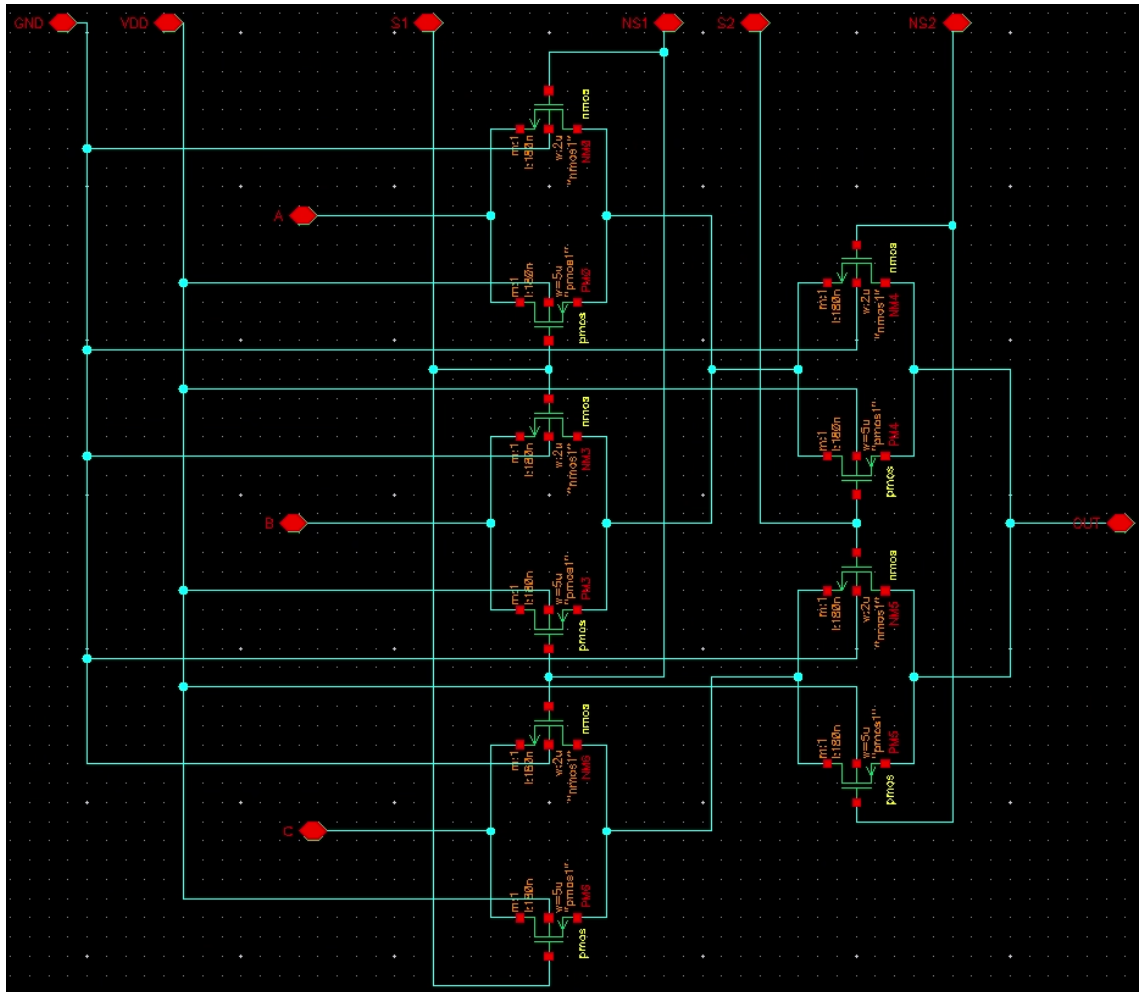
First, a 1 – bit 3 to 1 MUX was made with both select (S2, S1) and not-select (NS2, NS1) signals available. (Originally, a 1-bit transmission gate was going to be the first building block, but confirming functionality in the test bench simulation was going to be confusing and difficult.)

Then two inverters were added to provide NS2 and NS1 signals from S2 and S1 for each 1 – bit 3 to 1 MUX. (Originally a single inverter was combined with a 4 – bit configuration of the 3 to 1 MUX, and tested in the test bench simulation for inverter sizing. It was determined that the size of the transistors in the inverter didn't significantly affect the switching of the output. Because a great deal of errors were encountered in LVS for Layout for this configuration and the errors were difficult to solve because of how many different individual parts were in the layout, it was scrapped and the 1 – bit 3 to 1 MUX with 2 pre-made inverters was used instead.

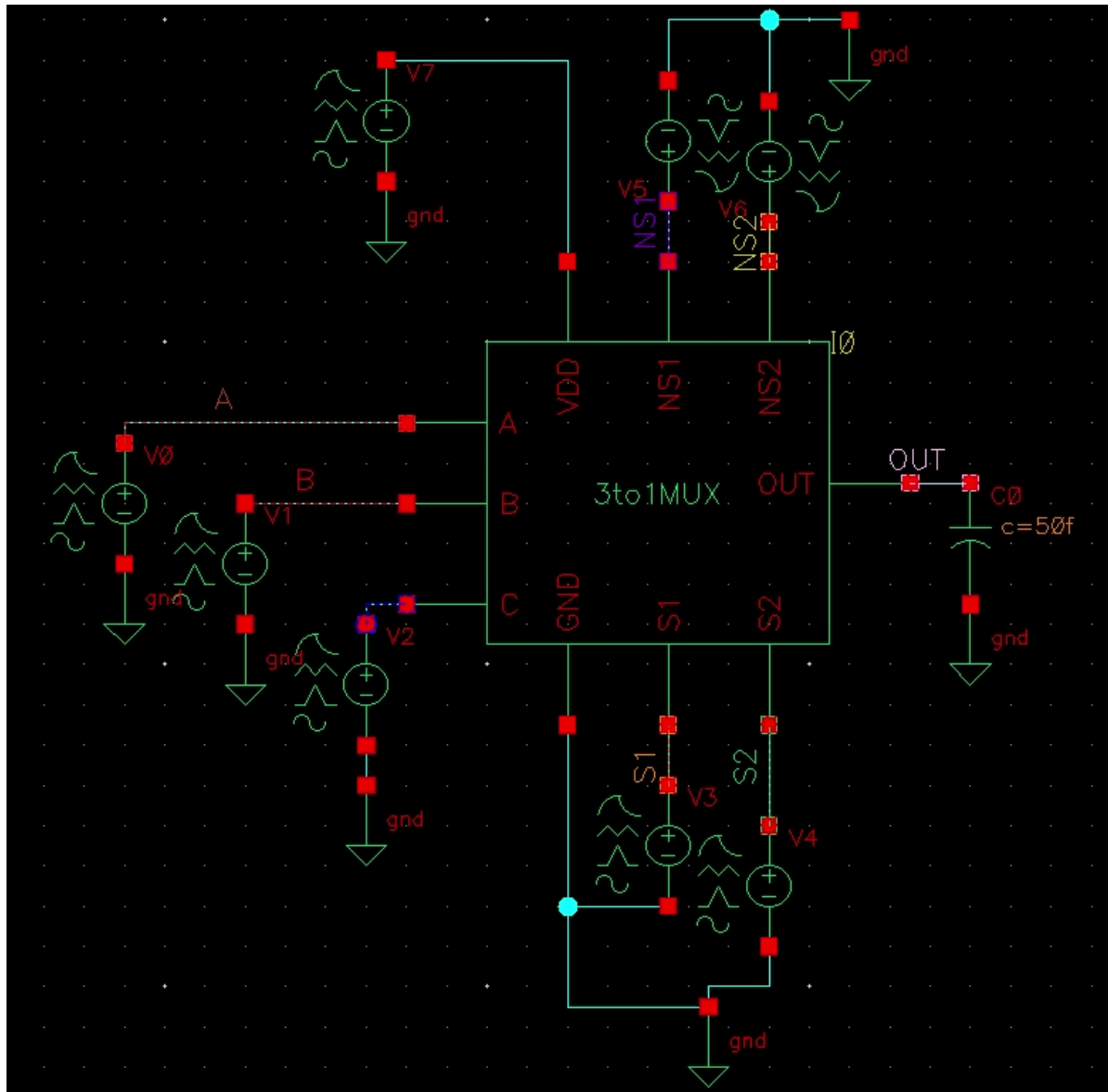
Finally, the 1 – bit 3 to 1 MUX with S2, S1 inputs was cascaded 32 times for the final 32 – bit 3 to 1 MUX.

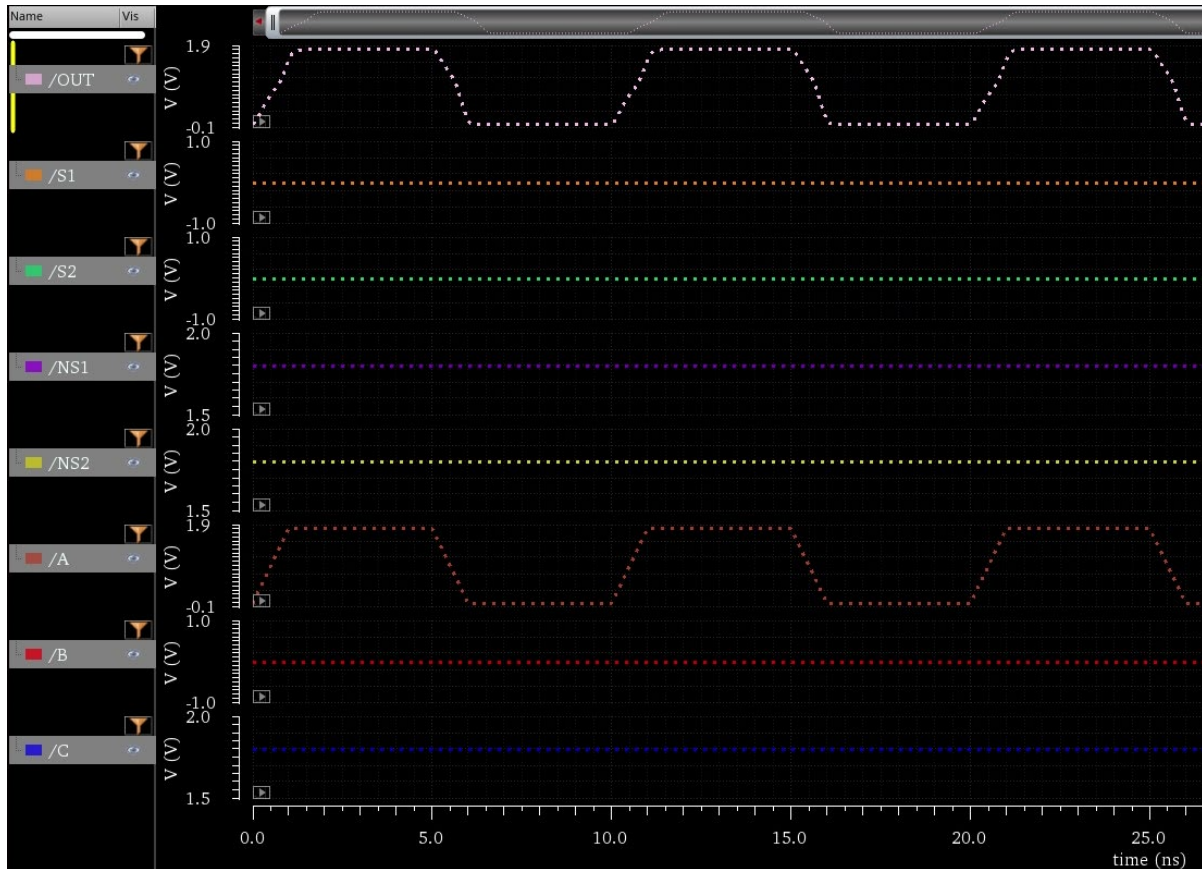
1- Bit 3 to 1 MUX with Select Complements

Schematic



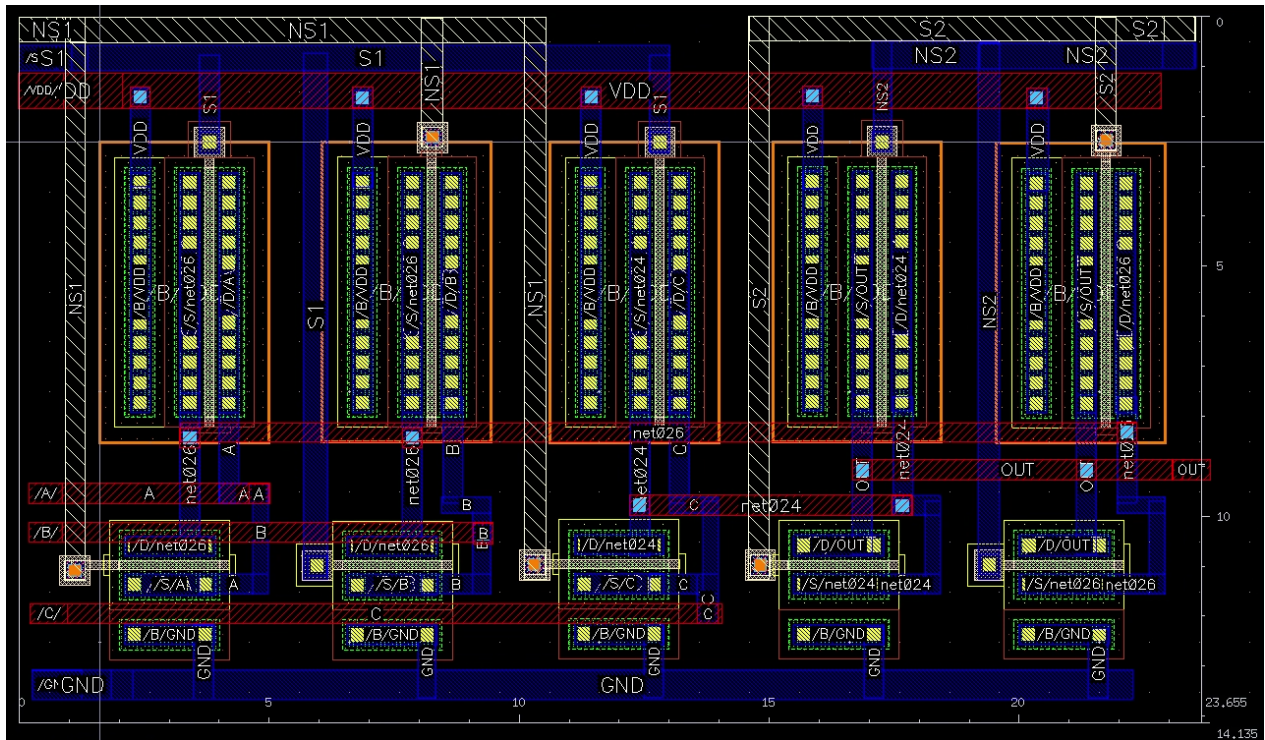
Simulation





For this simulation, A was assigned as a pulse, B was assigned as 0 V and C was assigned as 1.8 V. Then the selector pins, S2 and S1, were changed from 00 for A 01 for B and 10 for C, as explained in the constraints and functionality section.

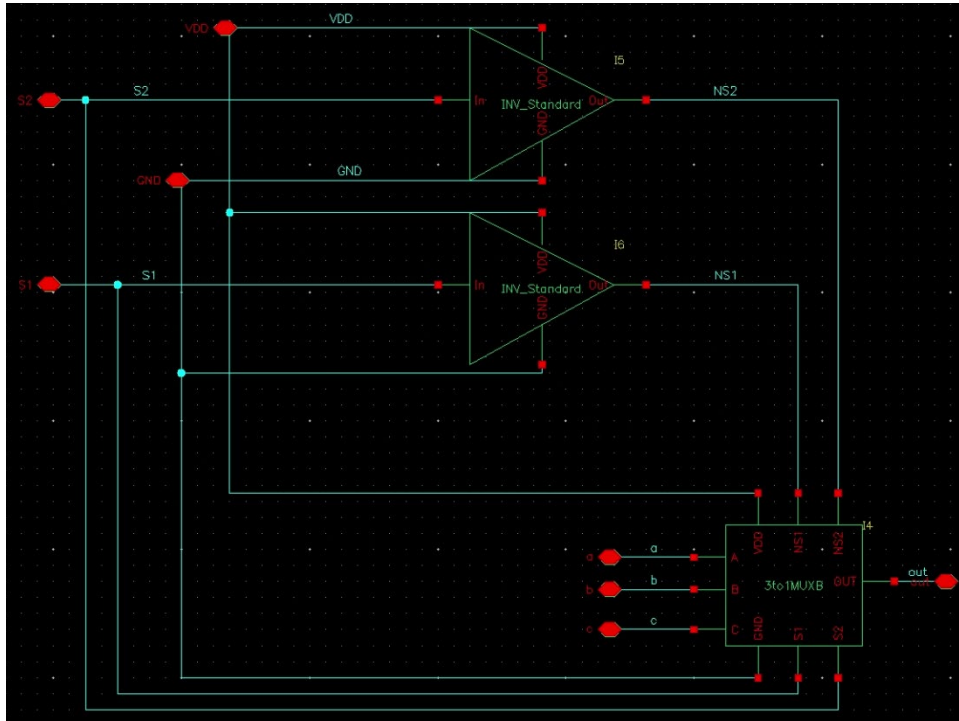
Layout



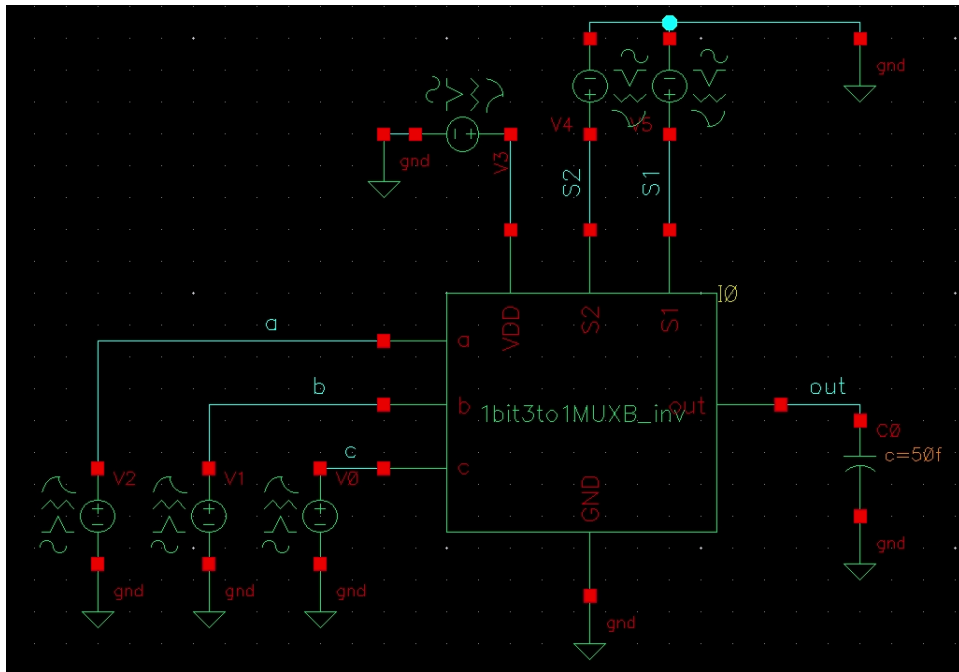
The select pins are placed on the top as well as the VDD rail, the GND rail is placed on the bottom, the inputs A, B, and C are on the left side, and the output OUT is on the right side.

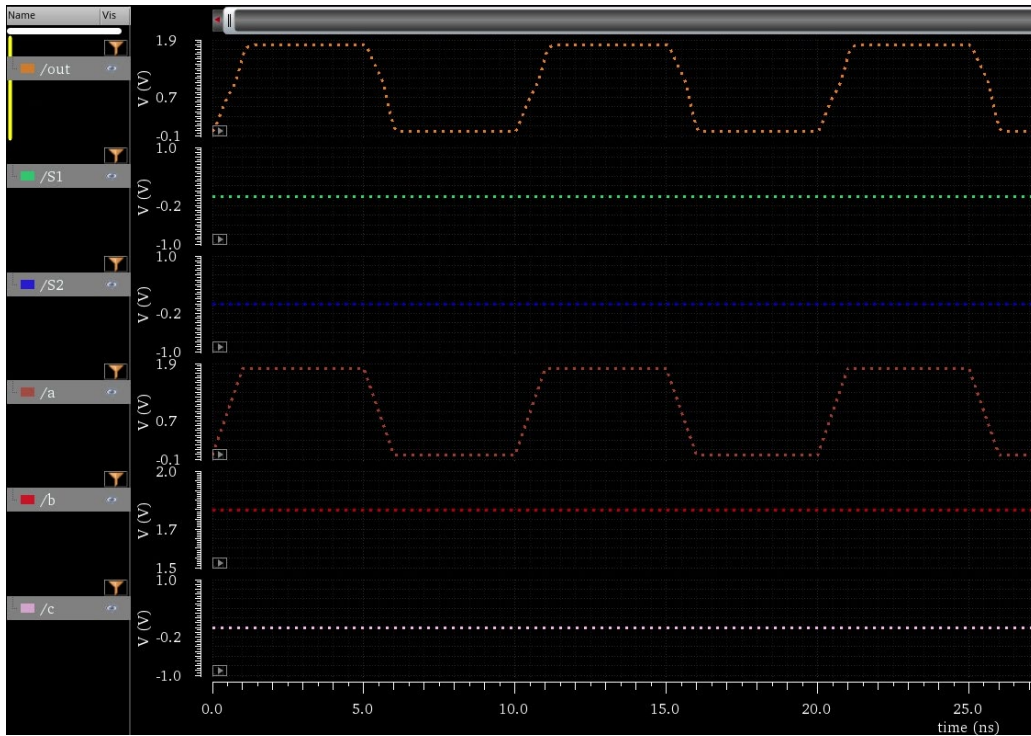
The area is 335 μm^2 .

1- Bit 3 to 1 MUX with Inverter and Select Schematic



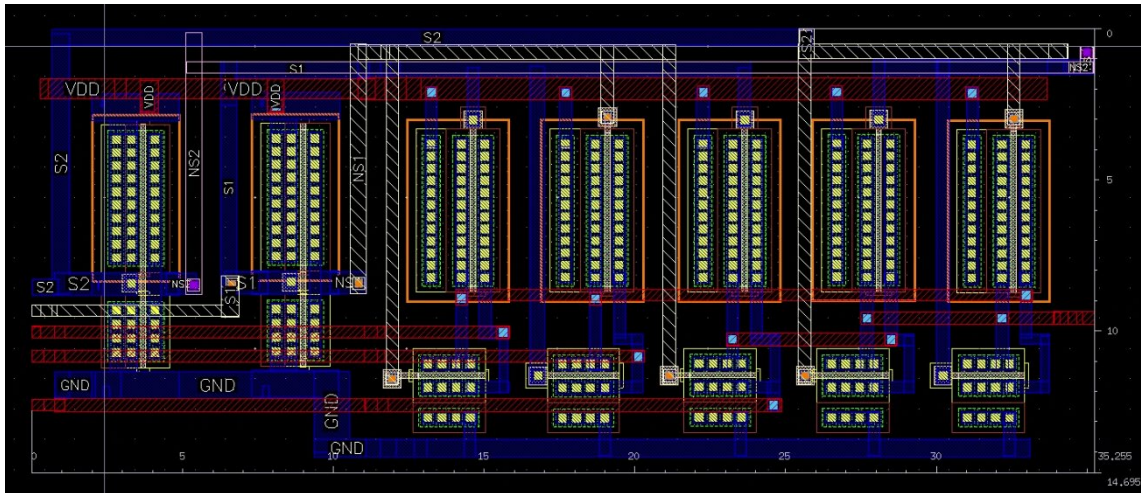
Simulation





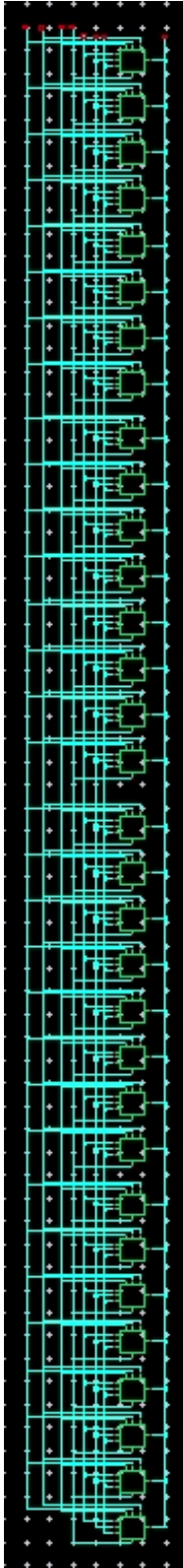
The same simulation as the previous 1 – bit 3 to 1 MUX with NS2 and NS1 signals was performed to confirm that the inverters were in fact translating S2 and S1 to NS2 and NS1 correctly.

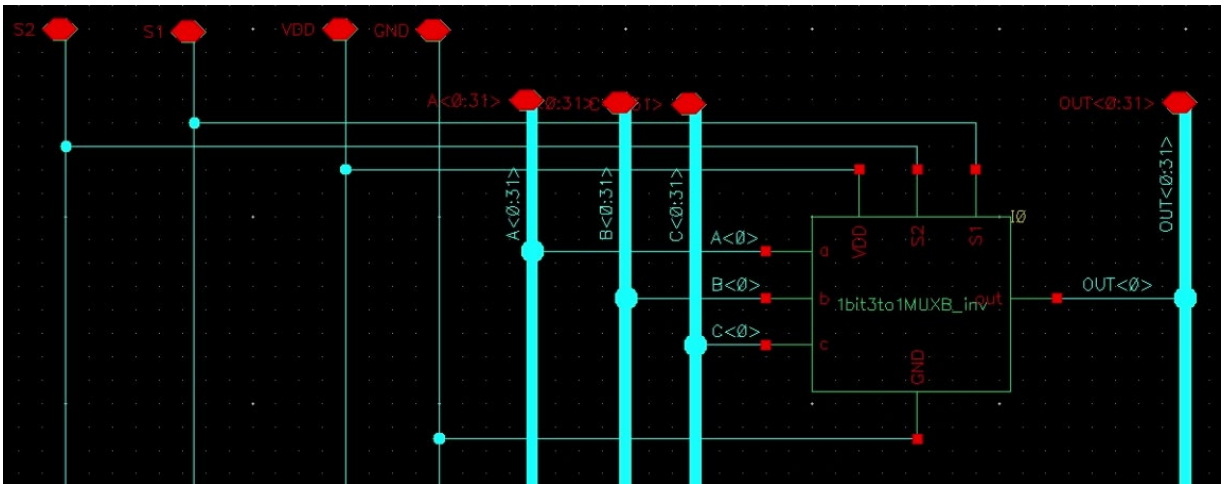
Layout



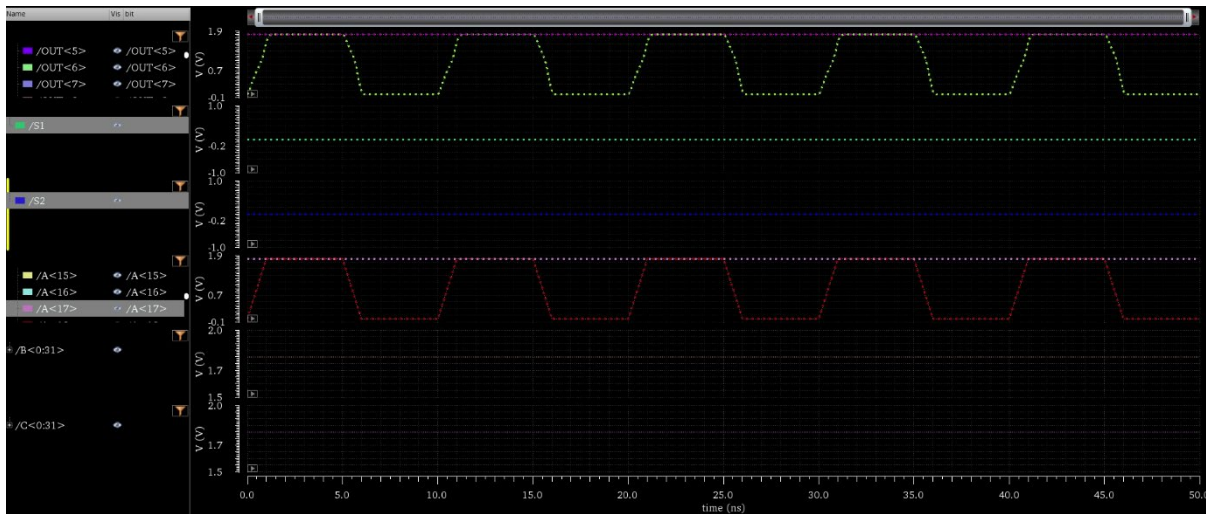
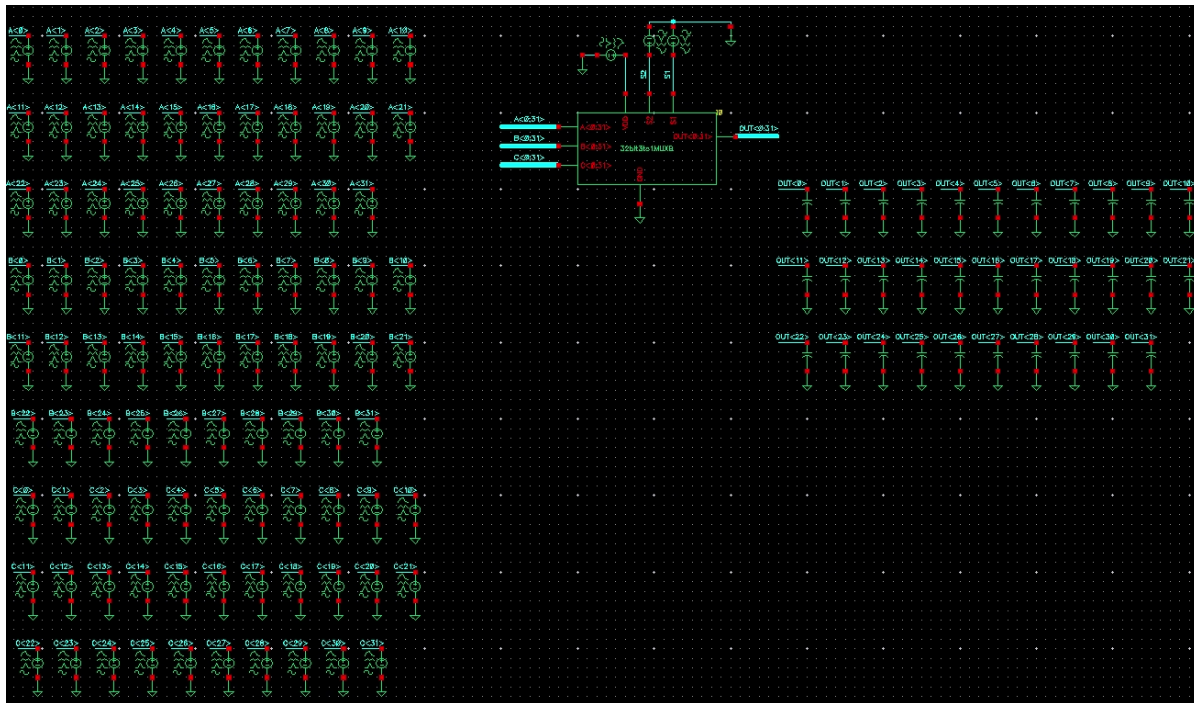
Just like the previous 1 – bit 3 to 1 MUX with NS2 and NS1 signals, the select pins are placed on the top as well as the VDD rail, the GND rail is placed on the bottom, the inputs A, B, and C are on the left side, and the output OUT is on the right side. The area is 518 μm^2 .

32 - Bit 3 to 1 MUX
Schematic



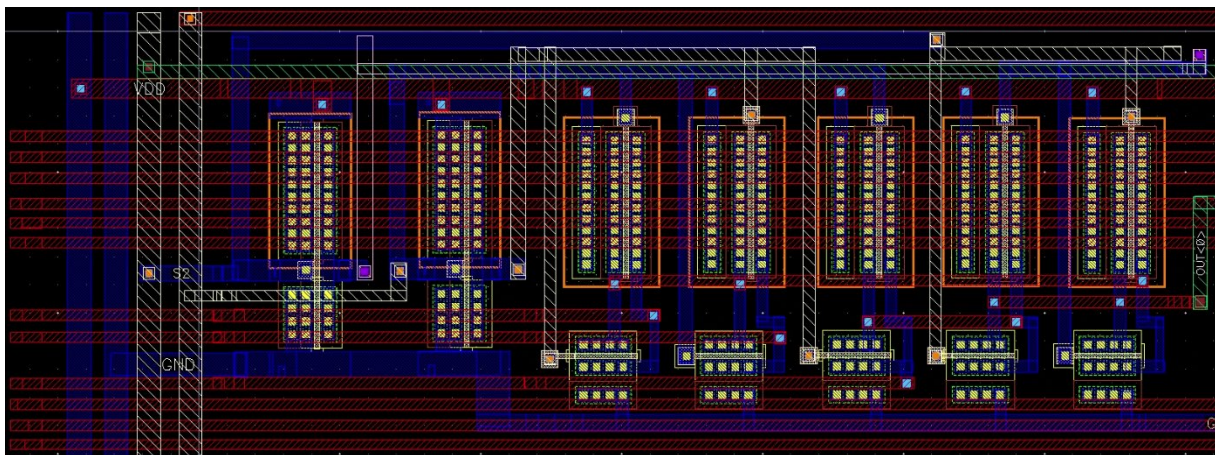
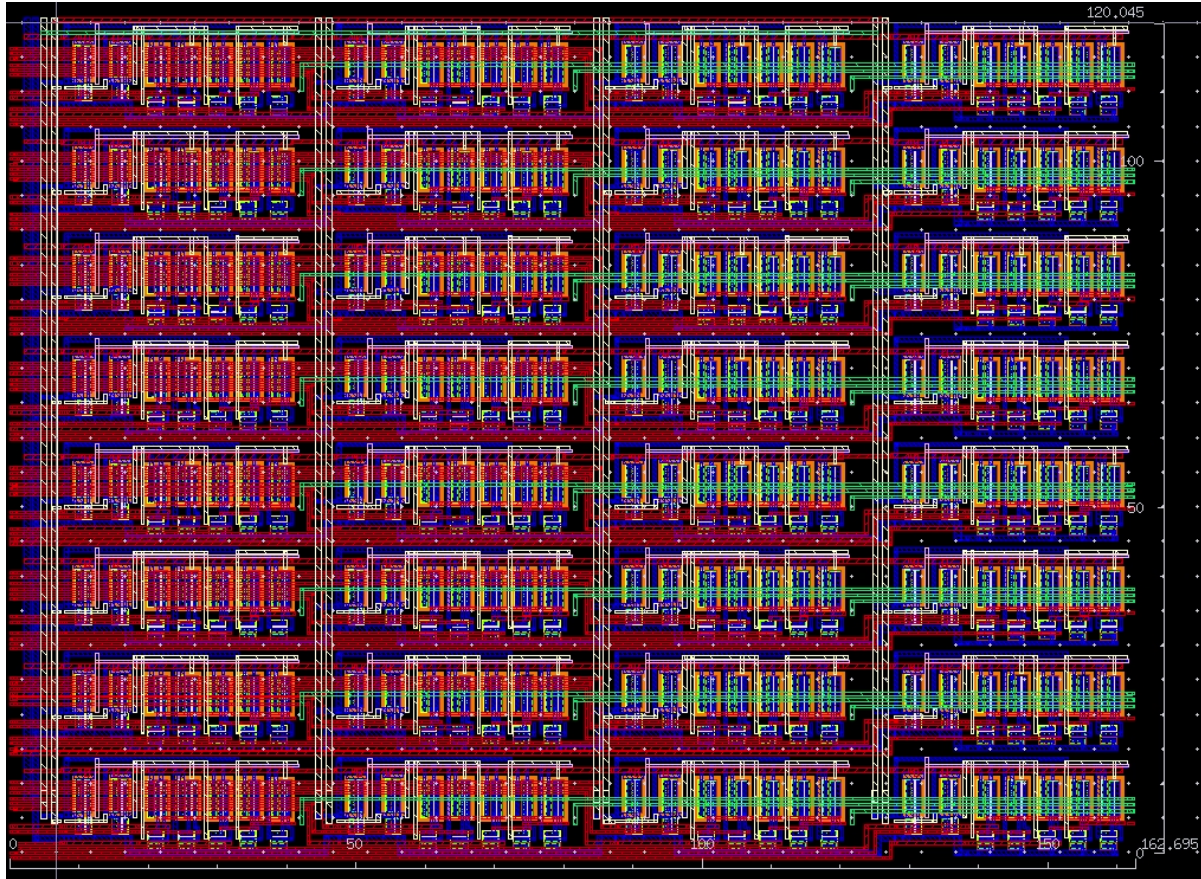


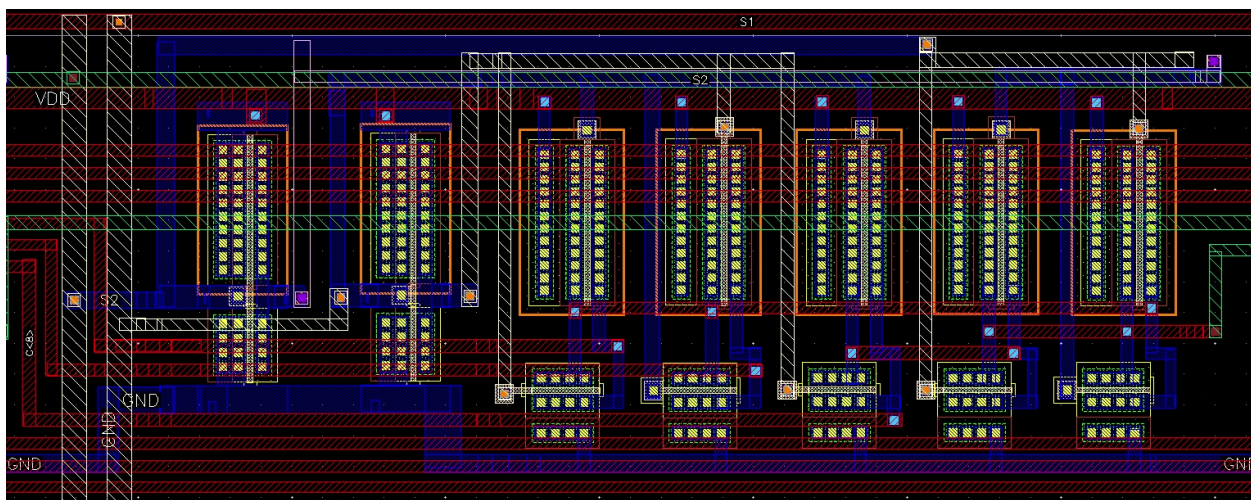
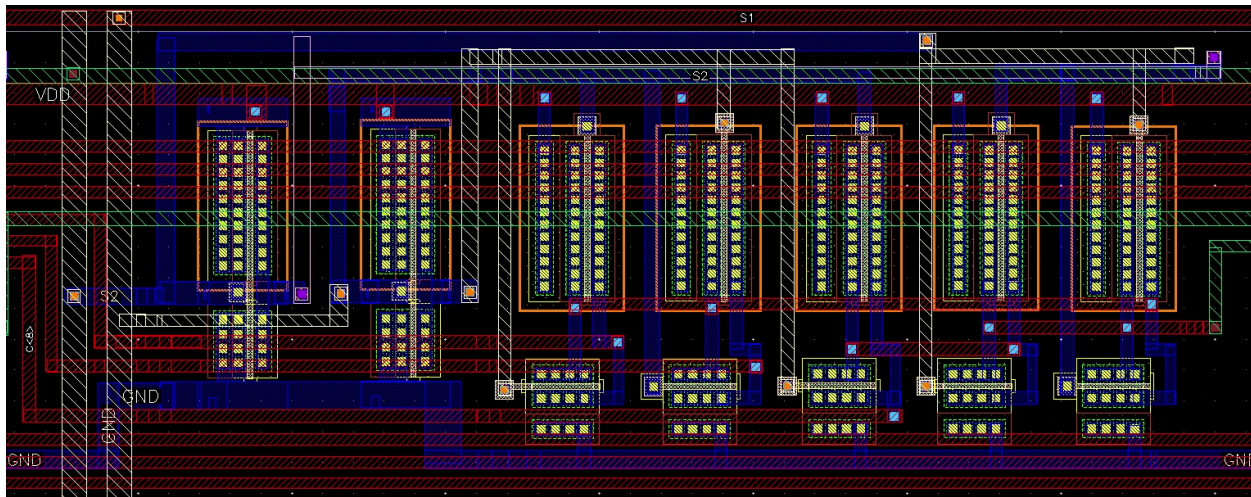
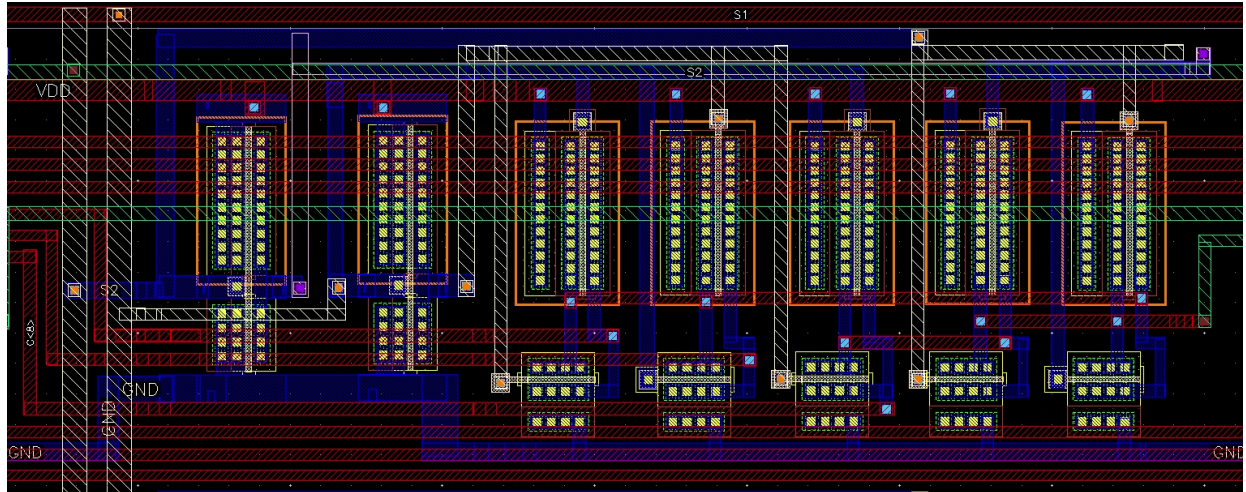
Simulation



Spot tests on OUT<0>, OUT<17>, A<0>, and A<17> are highlighted to confirm that the 32 bit 3 – to 1 MUX has been cascaded correctly.

Layout

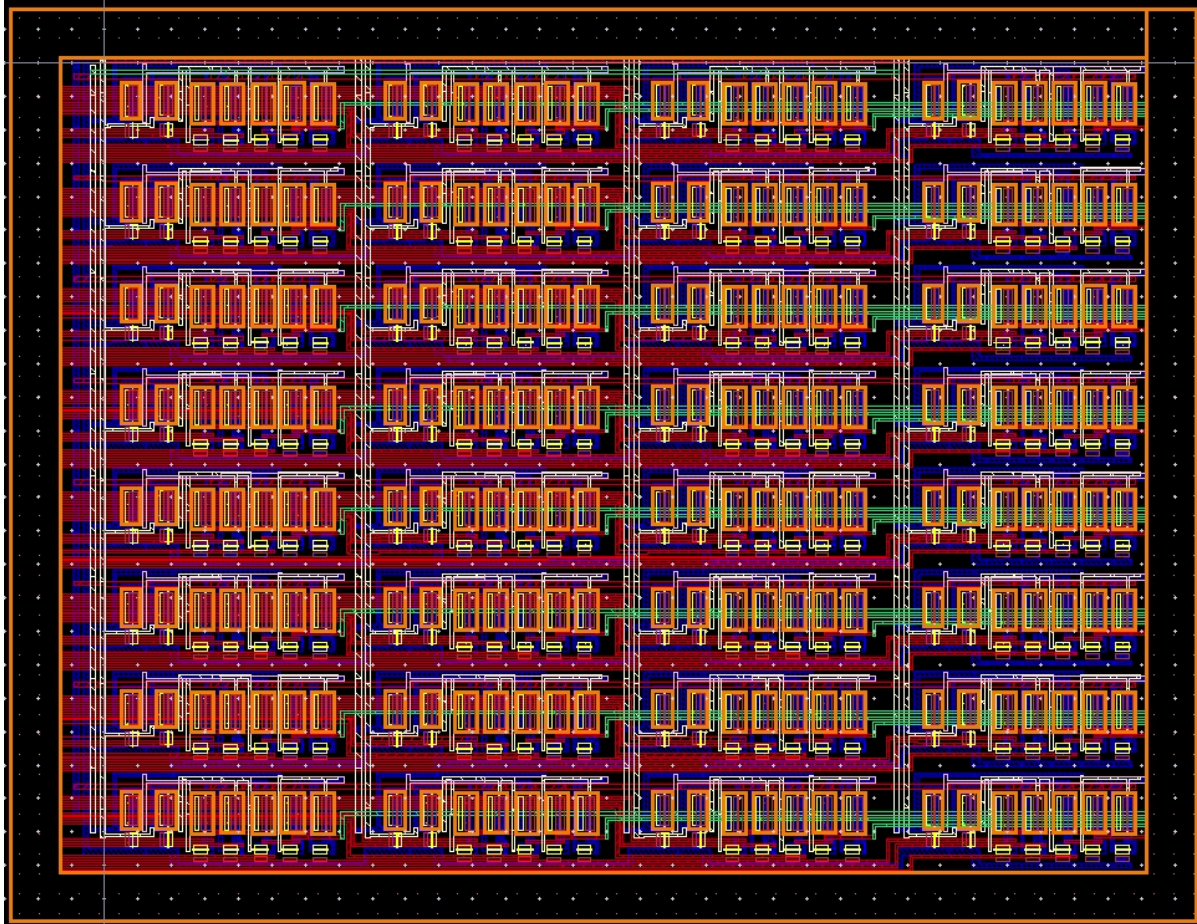


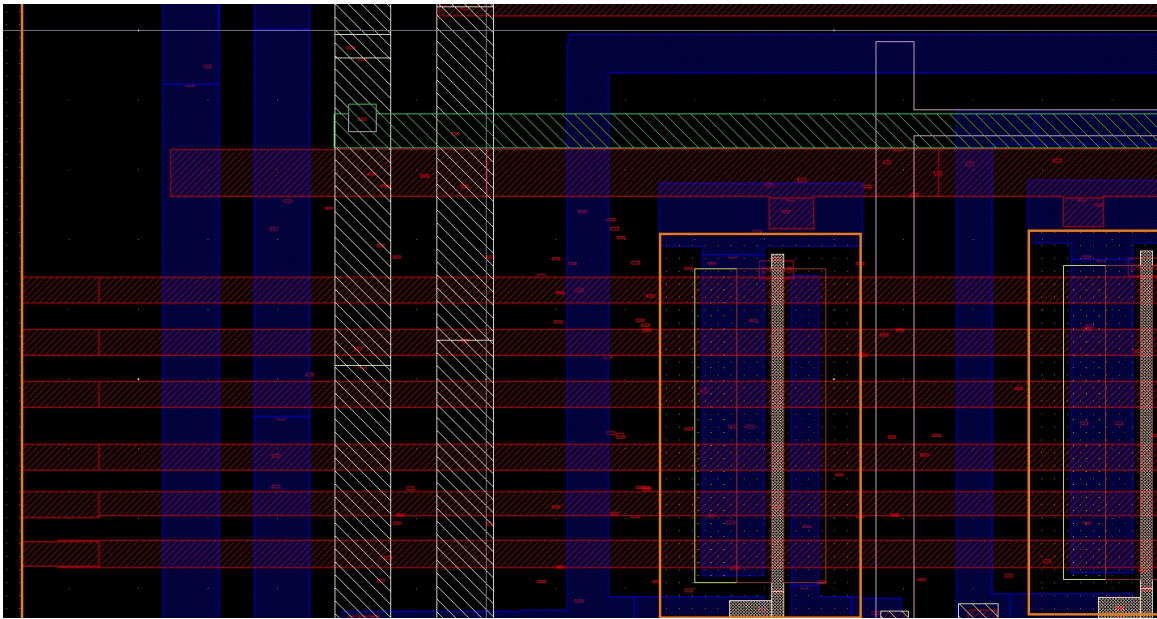
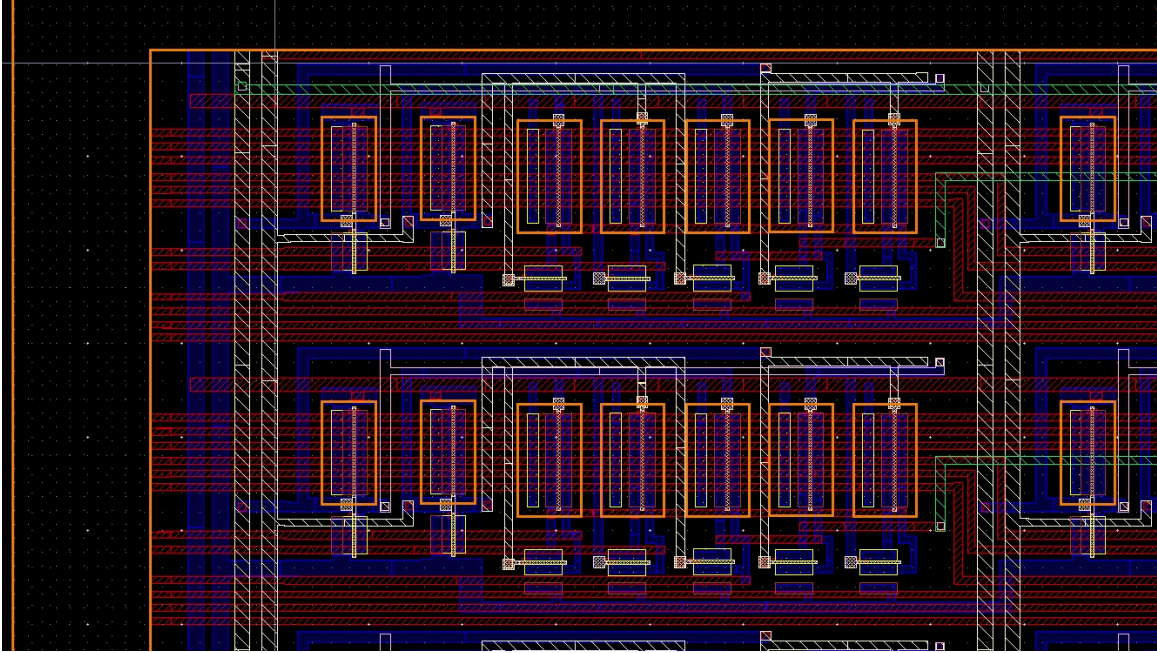


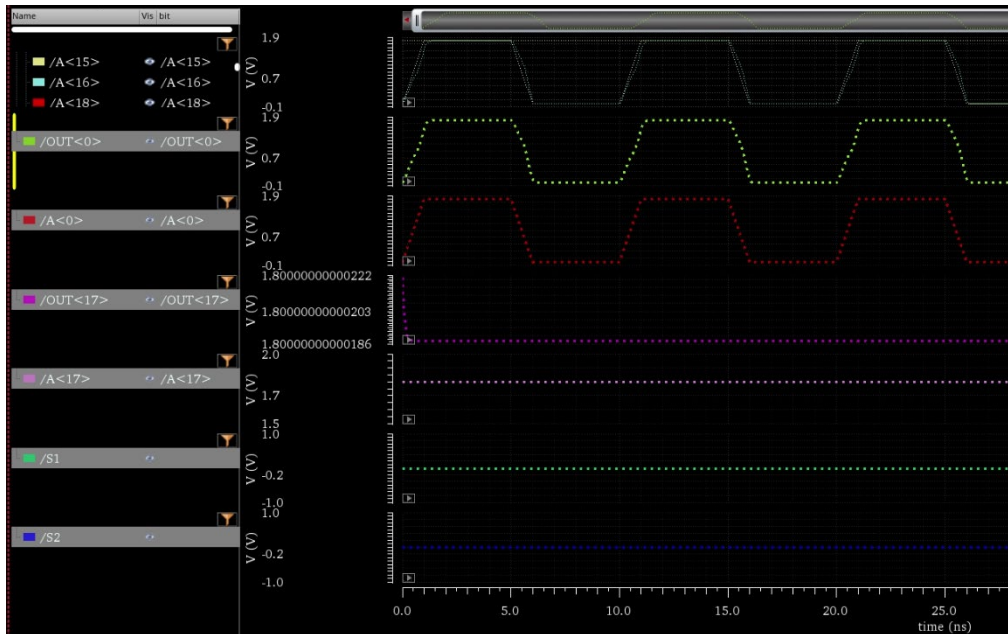
The layout is a 8x4 array of 1 – bit 3 to 1 MUXes. The area is 19680 μm^2 .

Post-Layout Simulation

The *AV_Extracted* file for the 32 – Bit 3 to 1 MUX (and therefore subsequent 1-bit MUX building blocks) is included below to confirm that all components used in this 32 – Bit 3 to 1 MUX has passed DRC, LVS, and Quantas.







The same signals as the test bench simulation are used, and it is worth noting that there is a very small spike for OUT<17> where the output is a high signal.

Lessons Learned

From making the building blocks for the 32 – bit 3 to 1 MUX, I realized how important it is in layout how to have a good grasp of what you want your layout to look like even before you start. This helps with making the layout as a building block for larger implementations much easier, because there is much less breaking out of traces and finagling to connect two points that could have been much easier to connect if a better design had been planned. There were multiple times where I went back to my 1 – bit block and adjusted it so that it made more sense in my 32 – bit block.

Another thing I realized was the importance of doing several steps of small building blocks before the final implementation, so that it will be much easier to diagnose an LVS error and much easier to do layout in general.

Lesson's Learned

The 32 – Bit ALU was selected as the project's design implementation. However, the final 32 – Bit ALU was not completed by the deadline set. This can be attributed to the lack of experience using *Cadence Virtuoso*. Because of the initial design methodology chosen, each member was able to create a different functional block of the 32 – Bit ALU. We were able to complete all functional blocks of the 32 – Bit ALU defined in the Design Requirements in this document. These functional blocks were laid out and simulated to confirm correct functionality.

In the final step, the designed functional blocks would be integrated and simulated to verify full functionality of the 32 – Bit ALU as one functional block. In the final steps of integration, a few technical notes must be kept in mind.

Because each bit of the A and B input will need to drive at least 3 Arithmetic Functional Blocks, the input capacitance will be high. Externally, a buffer inverter chain should be used to drive this high capacitive load.

Because each functional arithmetic block has 32 x 2 inputs, parasitic capacitances due to wire-to-wire capacitance should be kept in mind when laying out the 32-Bit ALU.

Because of the size of the final 32 – Bit ALU, the length of all inputs, outputs, and internal signal interconnects should be considered. If any input, output, or internal signal interconnect's length causes parasitic capacitance and resistance to stack, and in turn affect propagation delay or normal operation, inverter buffer chains should be used to prevent this from happening.

Lastly, the output driving capacity of the 32 – Bit ALU was not defined in this project or document. If this ALU were to be used as a functional block of a larger design, the ALU outputs may need to be buffered to drive higher capacitive output loads.

Bibliography

- [1 C. Today, "Ripple Carry Adder," [Online]. Available: <https://www.circuitstoday.com/ripple-carry-adder#:~:text=A%20ripple%20carry%20adder%20is%20a%20logic%20circuit%20in%20which,rippled%20into%20the%20next%20stage..>
- [2 Wikipedia, "Carry-Lookahead Adder," 3 November 2020. [Online]. Available: https://en.wikipedia.org/wiki/Carry-lookahead_adder.
- [3 L. H., "High Speed binary Parallel Adder," *IEEE Transactions on Electronic Computers*, Vols. EC-15, no. 5, pp. 799-802, 1966.
- [4 M. Mishra and S. Akashe, "High Performance, Low Power 200 Gb/s 4:1 MUX with TGL in 45nm Technology," *Springerlink.com*, Feb 2013.

Appendix: Recorded Delays and Areas

Gate	Pre Layout			Post Layout			Size		
	Rise Time (us)	Fall Time (us)	Propagation Delay (us)	Rise Time (us)	Fall Time (us)	Propagation Delay (us)	Height (um)	Width (um)	Area (um ²)
INV	259.505	239.539		263.563	241.239	129.72			0
AND	123.93	100.816	200.81	126.434	102.133	202.33			0
OR	117.38	110.512	49.3438	120.198	112.327	54.538			0
XOR	270.592	194.92	213.368	278.319	197.425	224.763	19	9.88	187.72
P&G							31.995	13.68	437.6916
P&G 4 bit							34.955	67.1	2345.4805
P						235.4			
G						135.9			
Sum Carry						429.1	102.655	96.94	9951.3757
CLA 4 bit						591.3	138.61	99.935	13851.99035
32 CLA						1027.3	1144.78	99.935	114403.5893