Ripple Carry Adder Report - ESE 3700

Krishna Karthikeya Chemudupati

Penn ID: 76180033

March 30th, 2025

Table of Contents

Academic Integrity Statement

I, Krishna Karthikeya Chemudupati, certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this project.

March 30th, 2025

Introduction

This project focuses on the circuit-level design, analysis, and optimization of a ripple-carry adder implemented using the High Performance 22nm process node. The ripple-carry adder is a fundamental arithmetic unit in digital systems, composed of cascaded single-bit full-adder cells where the carry output of one stage feeds directly into the carry input of the next. As such, the performance of the overall adder is tightly coupled to the characteristics of a single bit-slice, making the design of the individual full-adder cell a critical concern.

The work is divided into two primary phases: the implementation of a baseline CMOS full-adder design, followed by the development of a delay-optimized version. The baseline design is constructed using static CMOS logic with minimum-sized transistors and operates at a nominal supply voltage of 0.8V. This version serves as a correctness benchmark and establishes a reference point for evaluating delay, energy, and area.

In the optimization phase, the focus shifts to reducing propagation delay through the ripple-carry chain while maintaining acceptable energy and area characteristics. While the primary logic style remains CMOS, alternative circuit topologies—ratioed logic and pass-transistor logic—are also considered as part of the design exploration. These alternatives present distinct tradeoffs in speed, area, output swing, and leakage, and are evaluated through targeted SPICE simulations.

Each design is simulated under realistic driving and loading conditions. Key performance metrics—including worst-case delay, active energy, leakage energy, and total transistor area—are measured and compared. Functional correctness is verified by simulating all eight possible combinations of the three input signals (A, B, Cin). Both the baseline and optimized full-adder cells are cascaded to construct 8-bit ripple-carry adders, which are used to evaluate and compare large-scale performance. Additionally, area and delay trends are analyzed for 1-bit, 2-bit, 4-bit, and 8-bit versions to better understand scaling behavior.

Functional Specification

The target of this project is to implement a single-bit full adder cell that serves as the building block for a ripple-carry adder. The full adder takes three binary inputs—A , B , and Cin (carry-in from the previous stage)—and produces two outputs: the Sum and the CarryOut to the next stage. The behavior of the full adder is described by the following Boolean expressions:

$$\mathrm{Sum} = A \oplus B \oplus \mathrm{Cin}$$
$$\mathrm{CarryOut} = (A \cdot B) + (A \cdot \mathrm{Cin}) + (B \cdot \mathrm{Cin})$$

Alternatively, $\mathrm{CarryOut}$ can be interpreted as the majority function of A, B, and Cin , since it evaluates to 1 whenever at least two of the three inputs are 1. This representation is helpful when analyzing the carry propagation path in the ripple-carry structure.

The functional correctness of the full adder must hold for all $2^3 = 8$ possible input combinations of A, B, and Cin. These combinations are shown in the truth table below:

A	*B*	*Cin*	*Sum*	*Cout*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the ripple-carry architecture, the output $\mathrm{CarryOut}$ of each bit-slice feeds directly into the Cin of the next higher-order stage. Therefore, maintaining proper delay and signal integrity along the carry chain is essential for ensuring correct timing and performance in multi-bit adders. The design must be cascadable and consistent with standard cell-based construction methods so that it can be reused across arbitrary adder widths.

This functional specification provides the logical foundation upon which both the baseline and delay-optimized full adder designs are implemented, tested, and evaluated.

Baseline CMOS Design

Schematics

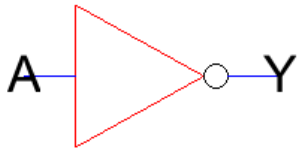
The baseline design is constructed using static CMOS logic with minimum-sized transistors throughout. All gates are implemented following the standard CMOS design discipline, with complementary pull-up and pull-down networks, powered by a 0.8V supply. The full adder bit-slice is constructed entirely from these basic gates.

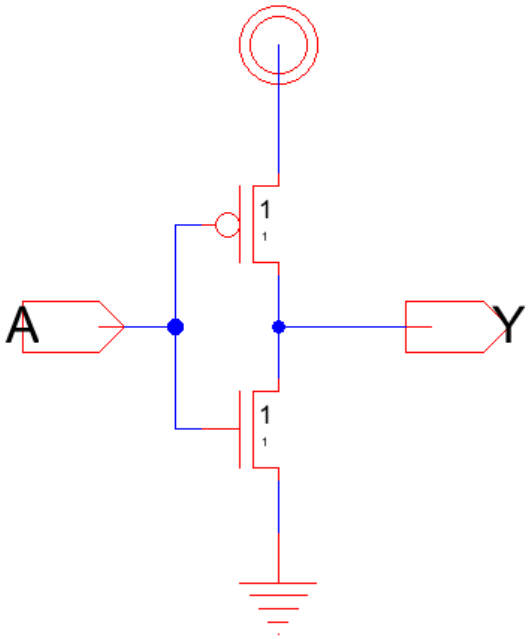
Gate-Level Schematics

The fundamental gates required to implement the full adder logic are the inverter (NOT), 2-input NAND, 2-input NOR, and 2-input XOR gates. Each gate is implemented using minimum-sized transistors with channel length L=22nm and width W=22nm, consistent with the minimum design rules of the 22nm HP process.

Inverter (NOT)

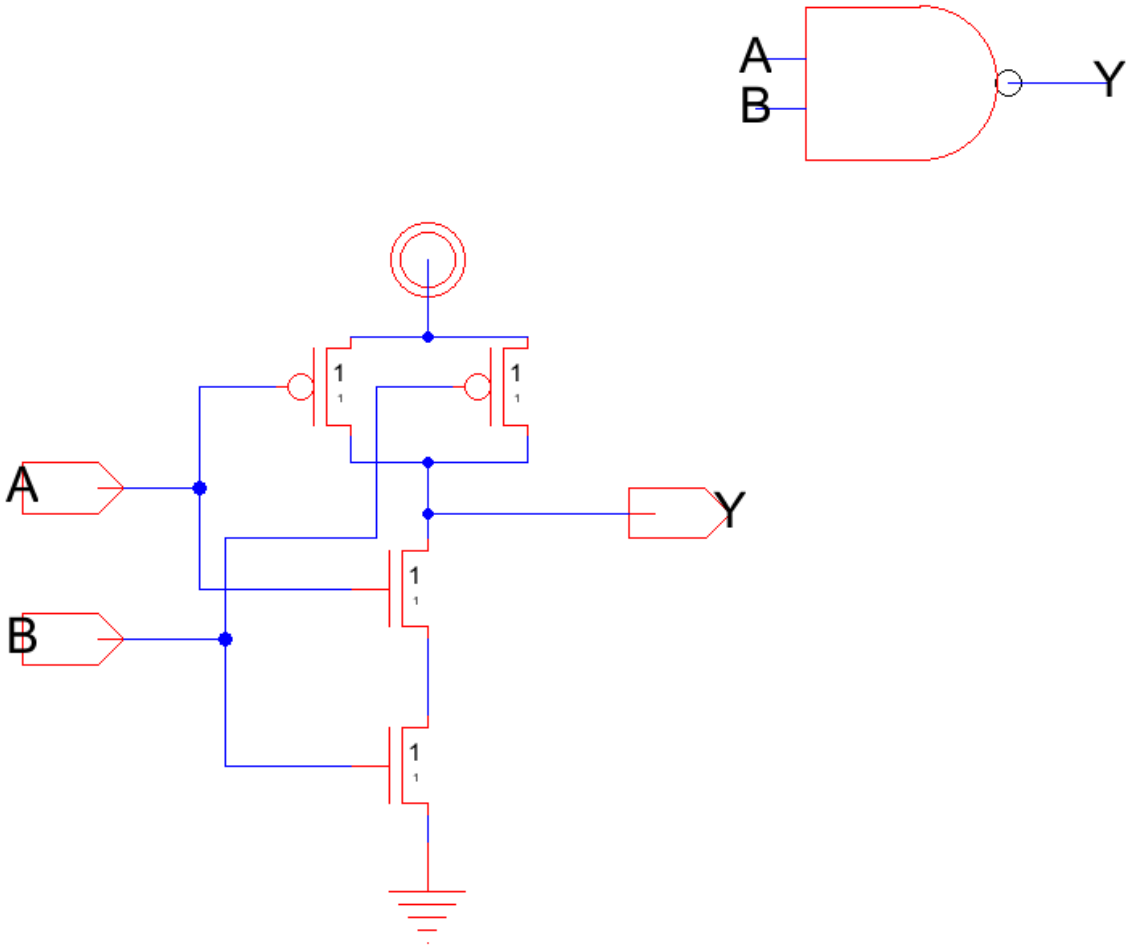
The inverter consists of a single PMOS transistor in the pull-up network and an NMOS transistor in the pull-down network, with the input tied to both gates and the output taken from the common drain node.





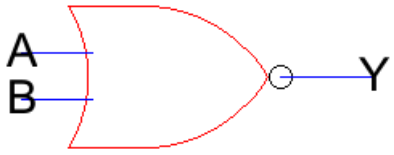
2-Input NAND (NAND2)

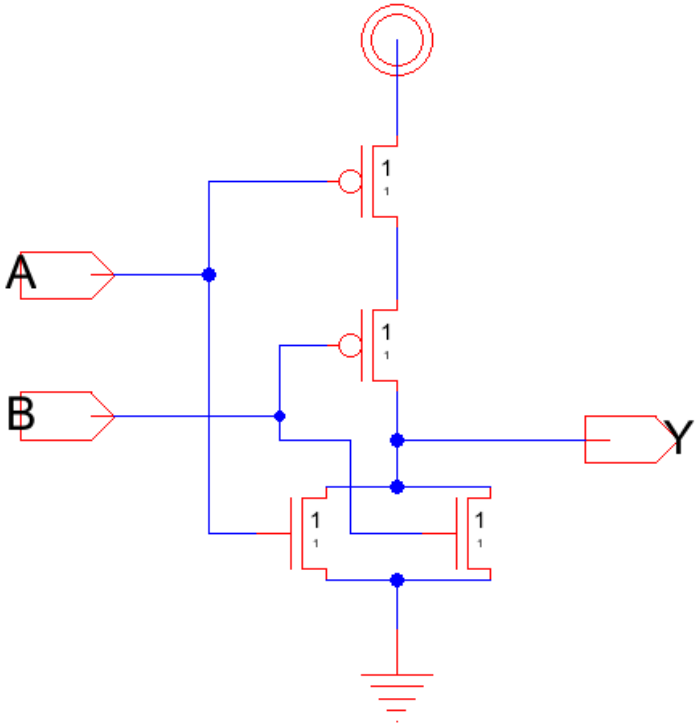
The NAND gate uses two NMOS transistors in series in the pull-down network and two PMOS transistors in parallel in the pull-up network. This structure ensures that the output only goes low when both inputs are high.



2-Input NOR (NOR2)

The NOR gate is constructed with two NMOS transistors in parallel in the pull-down path and two PMOS transistors in series in the pull-up path. The output is high only when both inputs are low.



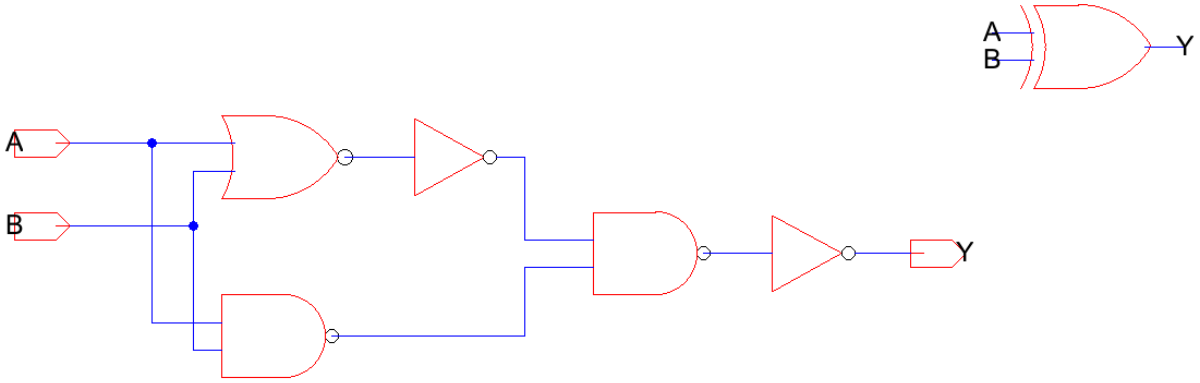


2-Input XOR (XOR2)

In the baseline design, the 2-input XOR gate is implemented using only standard CMOS gates—specifically NAND, NOR, and NOT gates—to preserve static operation and ensure full voltage swing. The logic expression used to derive this implementation is $A \oplus B = (A+B) \cdot \overline{A \cdot B}$, which expresses the XOR function as the logical AND of the OR of the inputs with the negation of their AND. This identity can be fully realized using available CMOS primitives.

To implement this expression, the term $A+B$ is constructed by first passing the inputs through a NOR gate, followed by an inverter to recover the OR function. Similarly, the term $\overline{A \cdot B}$ is obtained by first computing the NAND of the inputs and then inverting the result to recover the AND. These two intermediate signals—one representing $A+B$ and the other $\overline{A \cdot B}$—are then combined through a NAND gate to compute the final AND operation, which is followed by a final inverter to restore the positive polarity and complete the XOR computation.

This structure results in an XOR gate composed entirely of static CMOS gates, with strong pull-up and pull-down paths, ensuring clean signal transitions and full voltage swing. Although this implementation incurs a higher gate count and contributes to critical path delay, it avoids the degraded output levels and contention issues common to pass-transistor logic.



Full Adder Bit-Slice Schematic

The full adder computes two outputs, Sum and $\mathrm{CarryOut}$, from inputs A, B, and Cin. The sum output is 1 when an odd number of the inputs are 1, leading to the expression:

$$\mathrm{Sum} = A \oplus B \oplus \mathrm{Cin}$$

This is implemented in two stages: first compute $S_1 = A \oplus B$, then $\mathrm{Sum} = S_1 \oplus \mathrm{Cin}$. Each XOR is realized using the identity $X \oplus Y = (X + Y)(\overline{XY})$, which maps to a combination of OR, AND, and NOT gates.

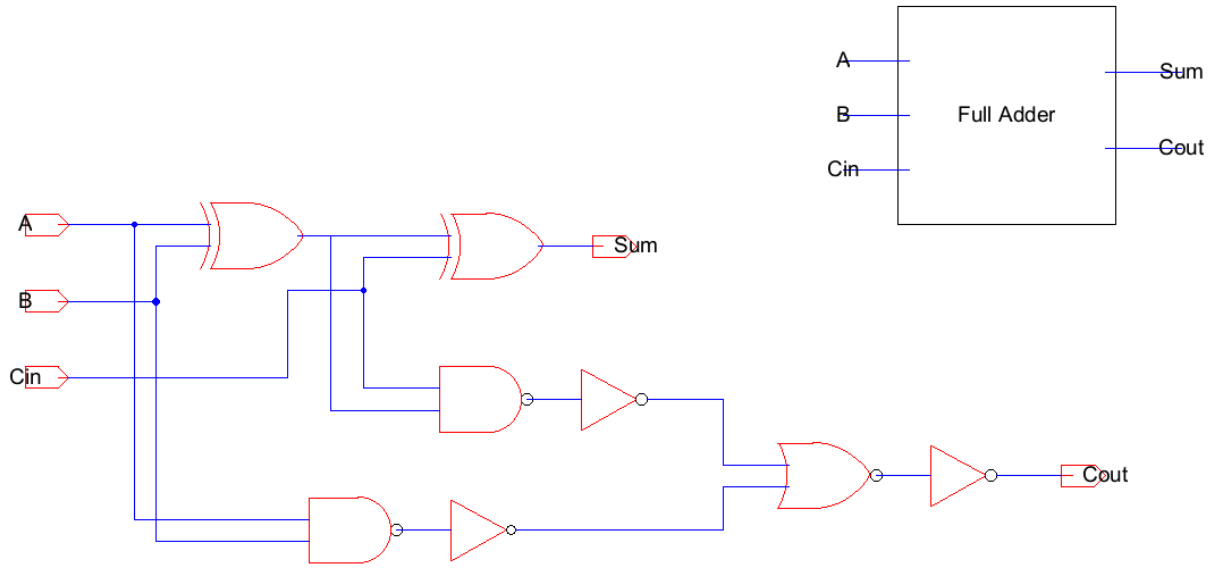
The carry output is 1 when at least two of the inputs are 1. The canonical expression is:

$$\mathrm{CarryOut} = AB + A\mathrm{Cin} + B\mathrm{Cin}$$

This is algebraically reduced to:

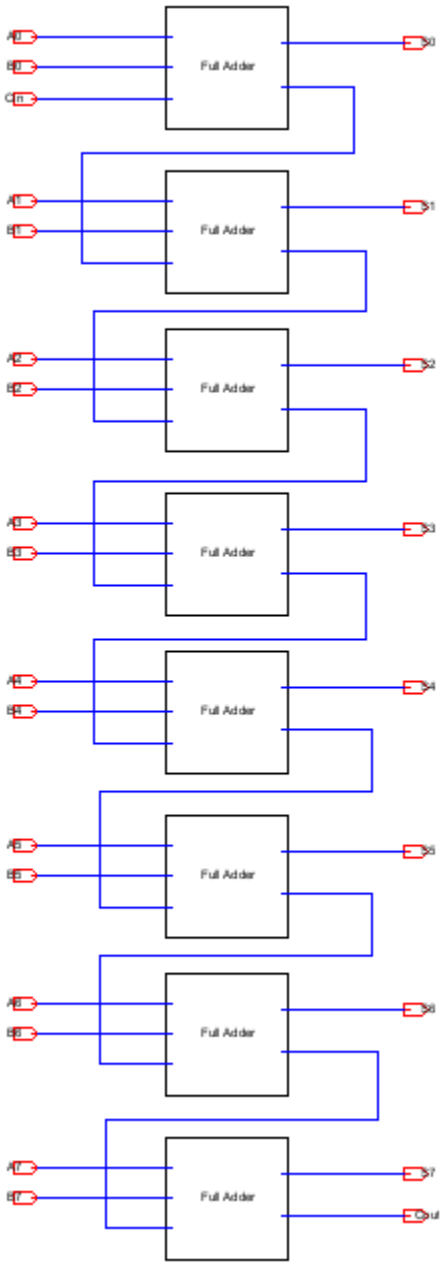
$$\mathrm{CarryOut} = AB + (A \oplus B)\mathrm{Cin}$$

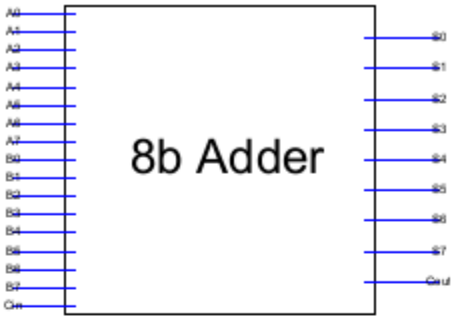
This form is more efficient, as the term $A \oplus B$ is already computed in the sum path. Thus, both outputs are derived from basic Boolean identities using only two-input logic gates and reused intermediate signals.



8-bit Ripple Carry Adder Schematic

An 8-bit ripple-carry adder connects eight 1-bit full adder cells in series. Each cell computes $S_i = A_i \oplus B_i \oplus C_i$ and $C_{i+1} = A_i B_i + (A_i \oplus B_i) C_i$, with C_0 as the initial carry-in. The carry-out of each stage feeds directly into the next stage's carry-in.





Logic Description

The baseline full adder is built using static CMOS gates: two XOR stages compute $\mathrm{Sum} = A \oplus B \oplus \mathrm{Cin}$, and $\mathrm{CarryOut}$ is derived from the factored form $AB + (A \oplus B)\mathrm{Cin}$. All logic is implemented using only NAND, NOR, and inverter gates with signal reuse to reduce gate count.

Delay Analysis

To calculate the overall delay of the delay-optimized CMOS ripple-carry adder using the τ model, I trace the signal propagation from the input A_0 to the final carry-out C_{out}. The first stage begins with the inverter driving A_0, which contributes:

$$2C_0$$

Then, the signal proceeds through the XOR, NAND, inverter, NOR, inverter, and output inverter of a single full adder. Each gate contributes an associated resistance-capacitance product. Across eight stages, the total delay is the sum of all RC terms as follows:

$$R_0 \cdot 2C_0 \rightarrow 2R_0 \cdot 2C_0 \rightarrow R_0 \cdot 2C_0 \rightarrow 2R_0 \cdot 2C_0 \rightarrow$$

$$R_0 \cdot 2C_0 \rightarrow 2R_0 \cdot 2C_0 \rightarrow R_0 \cdot 2C_0 \rightarrow 2R_0 \cdot 2C_0 \rightarrow R_0 \rightarrow C_{\text{out}} \rightarrow 6C_0$$

The final load of C_{out} is modeled as:

$$R_0 \cdot 6C_0$$

Thus, summing up the coefficients, the total delay becomes:

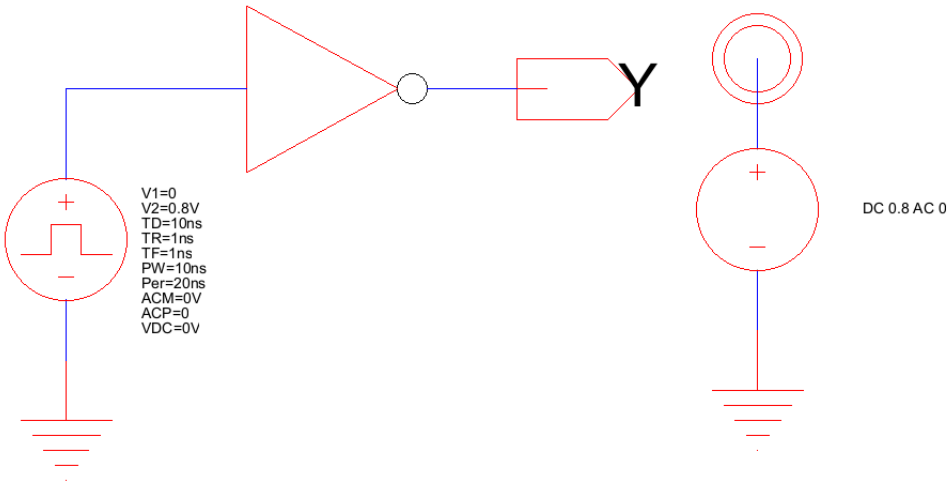
$$(2+4+2+4+2+4+2+4+6)\tau = 30\tau$$

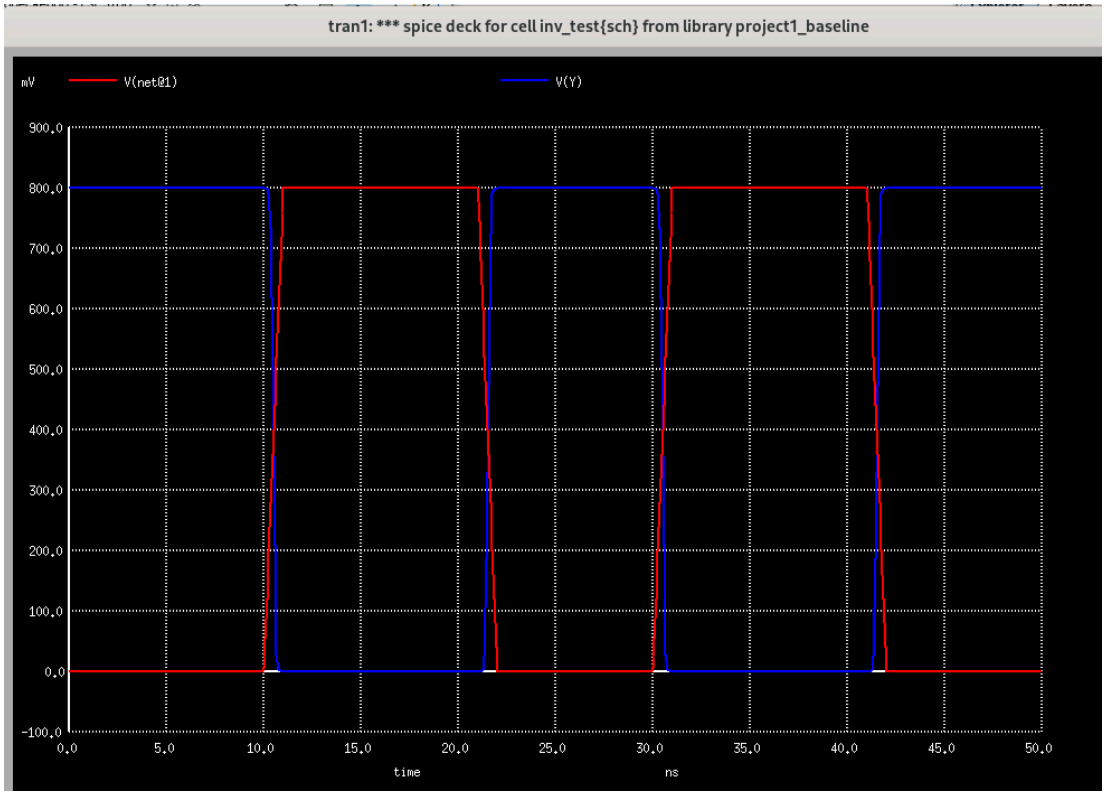
where each term corresponds to the RC delay of a specific gate along the path from input to output, and τ is the normalized delay unit $\tau = R_0 C_0$.

$$t_{\text{delay, RCA}} = 8 \cdot t_{\text{delay, FA}} = 8 \cdot 30\tau = 240\tau$$

Functional Verification

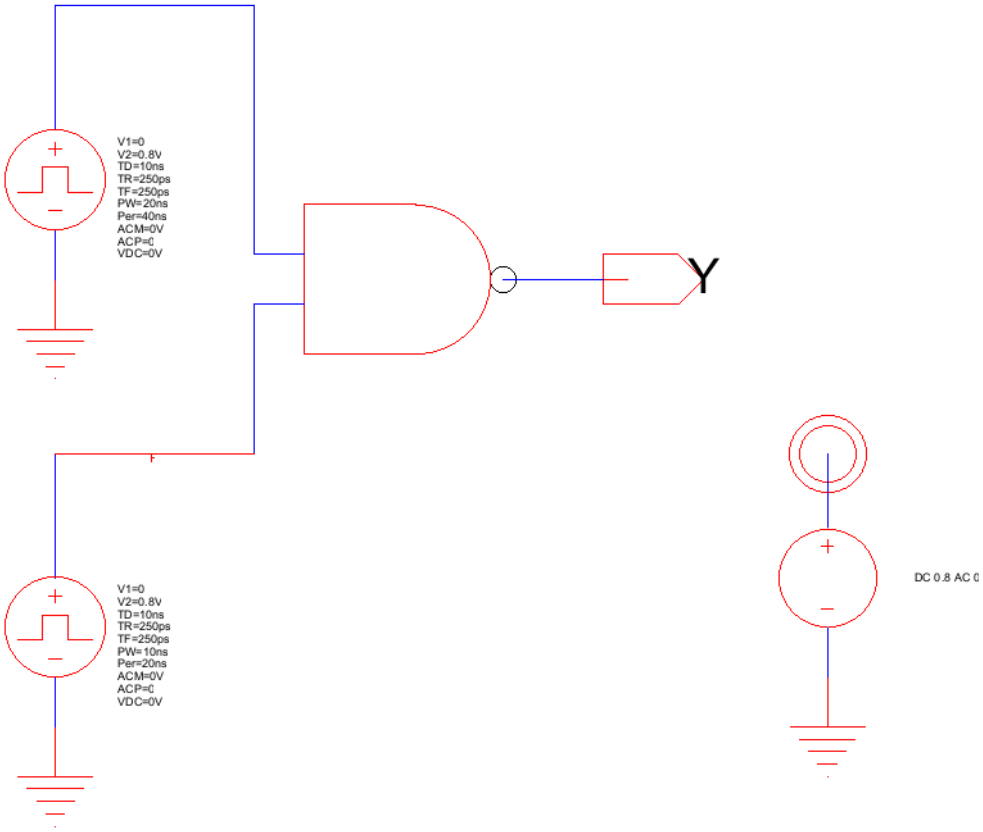
Inverter

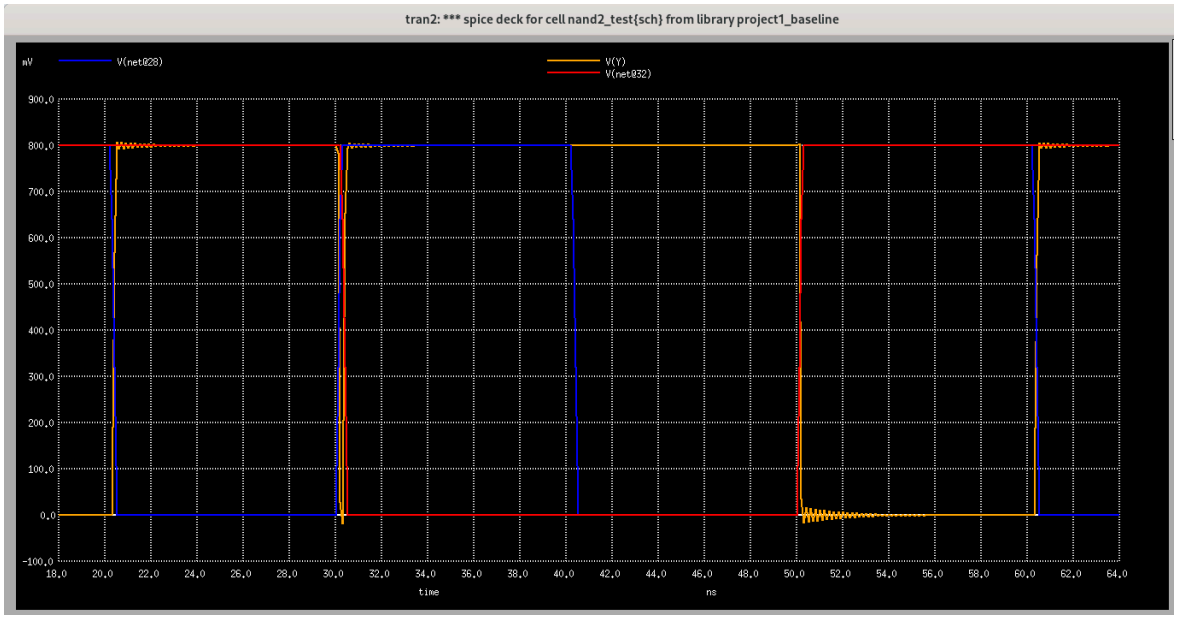




Input(red) is clearly inverted at output (blue).

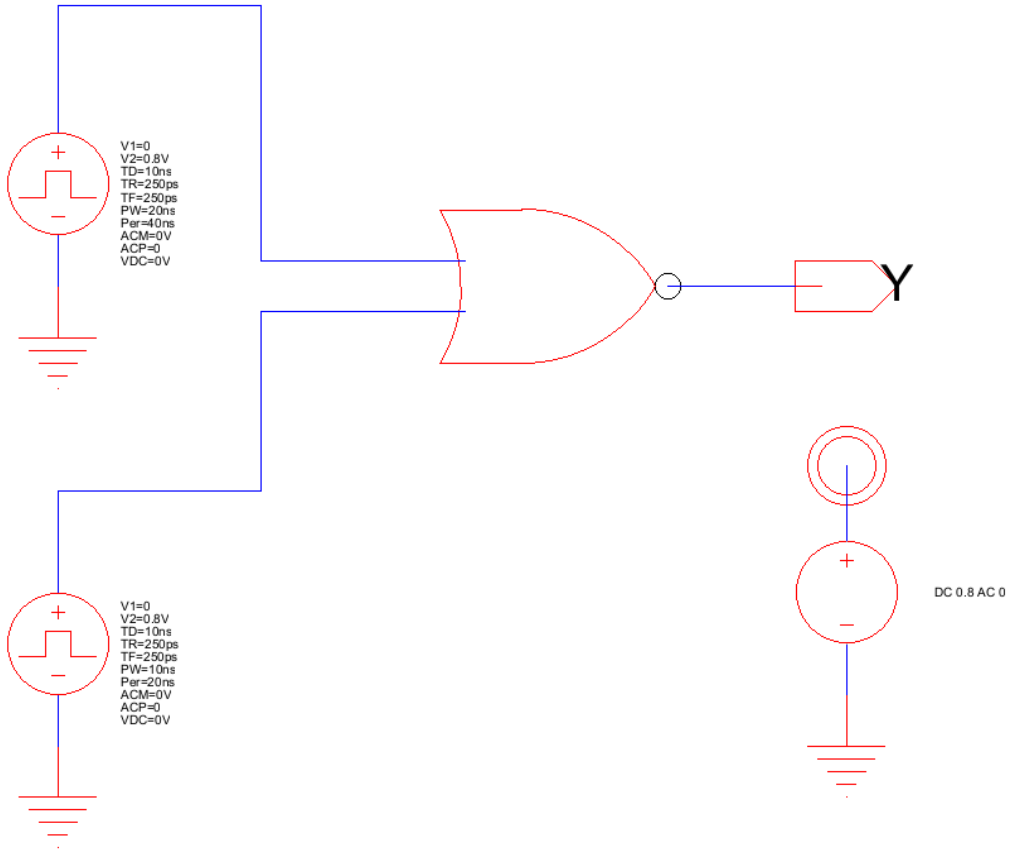
NAND2

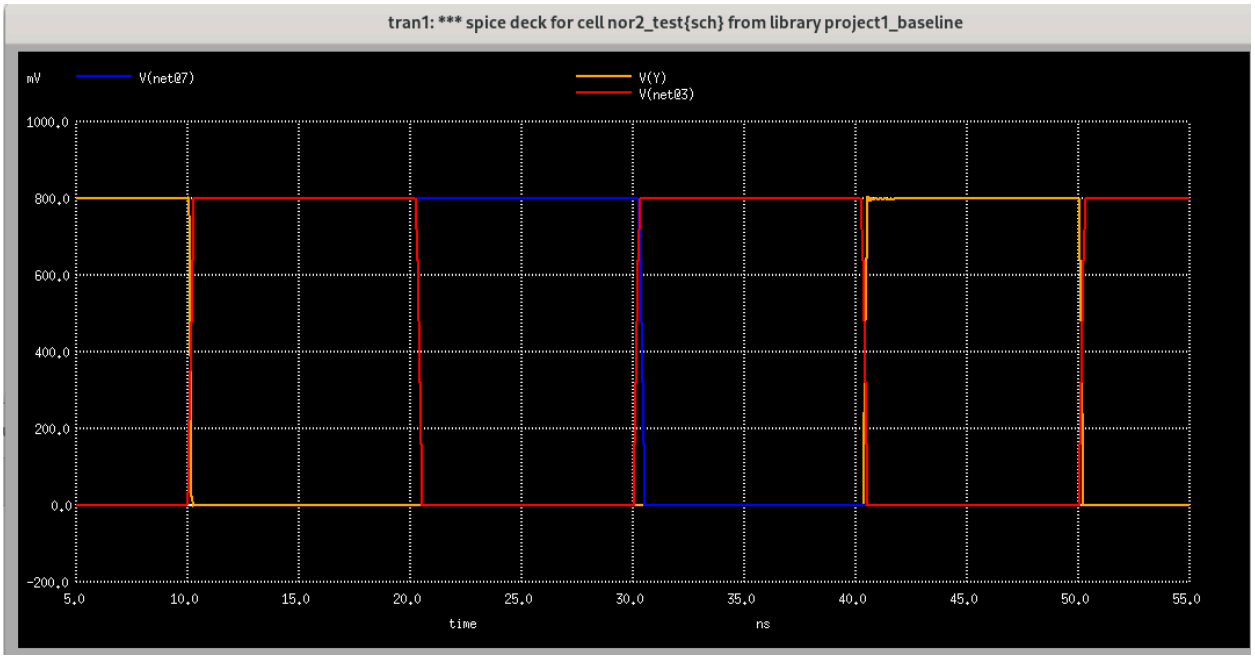




Output (yellow) is only low when both inputs (red and blue) are high, as we expect from a NAND2.

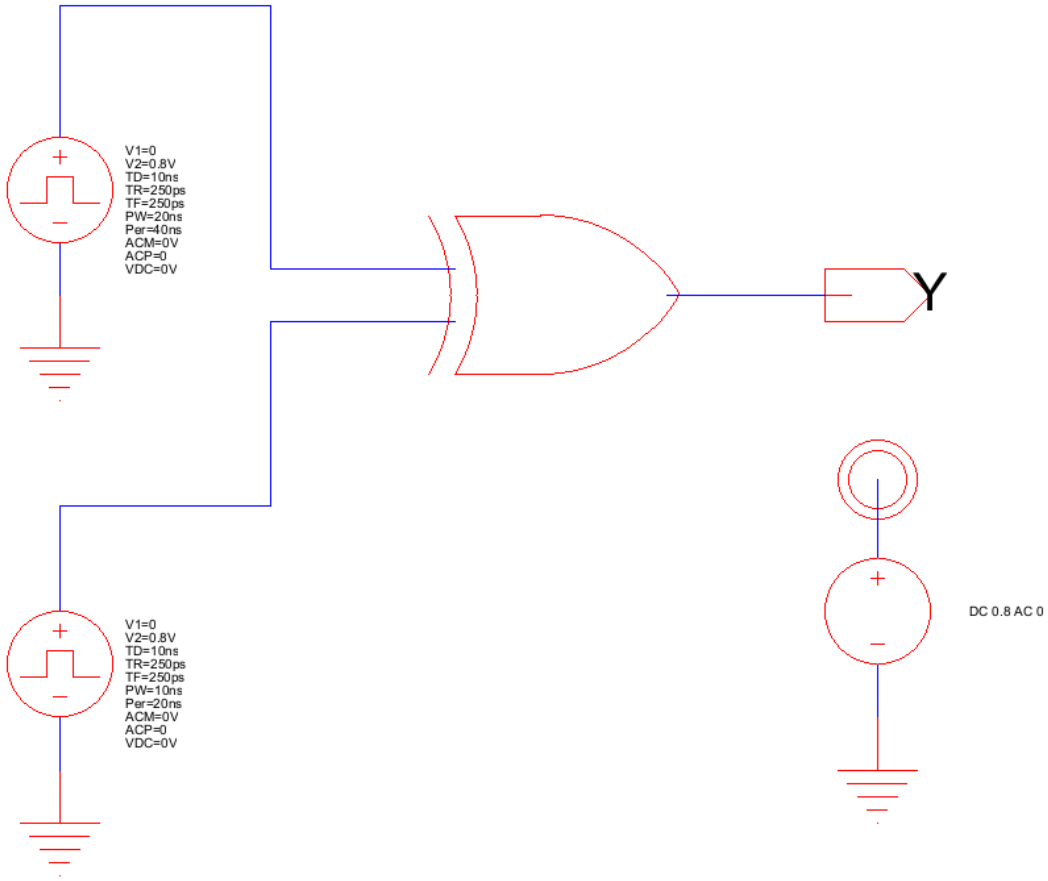
NOR2

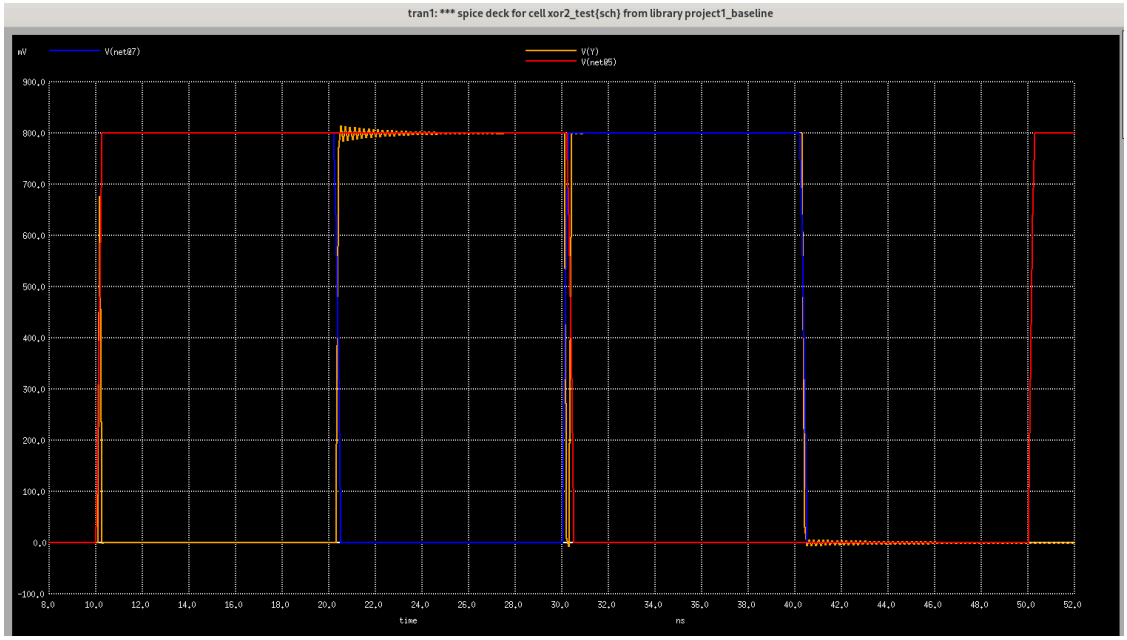




Output (yellow) is high only when both inputs are low (as expected from a NOR2).

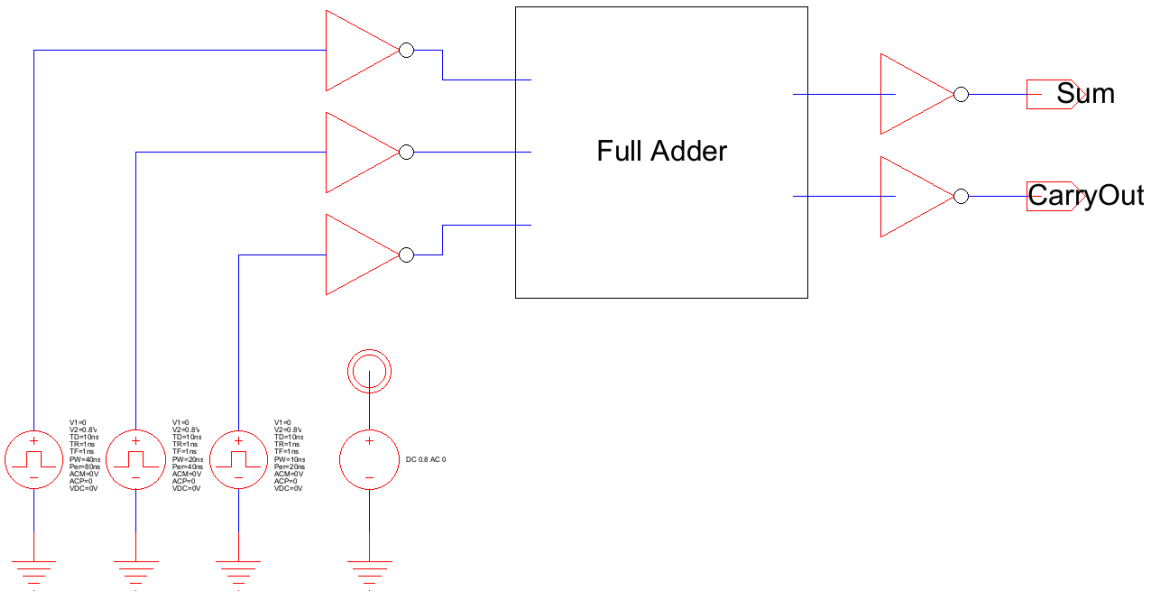
XOR2





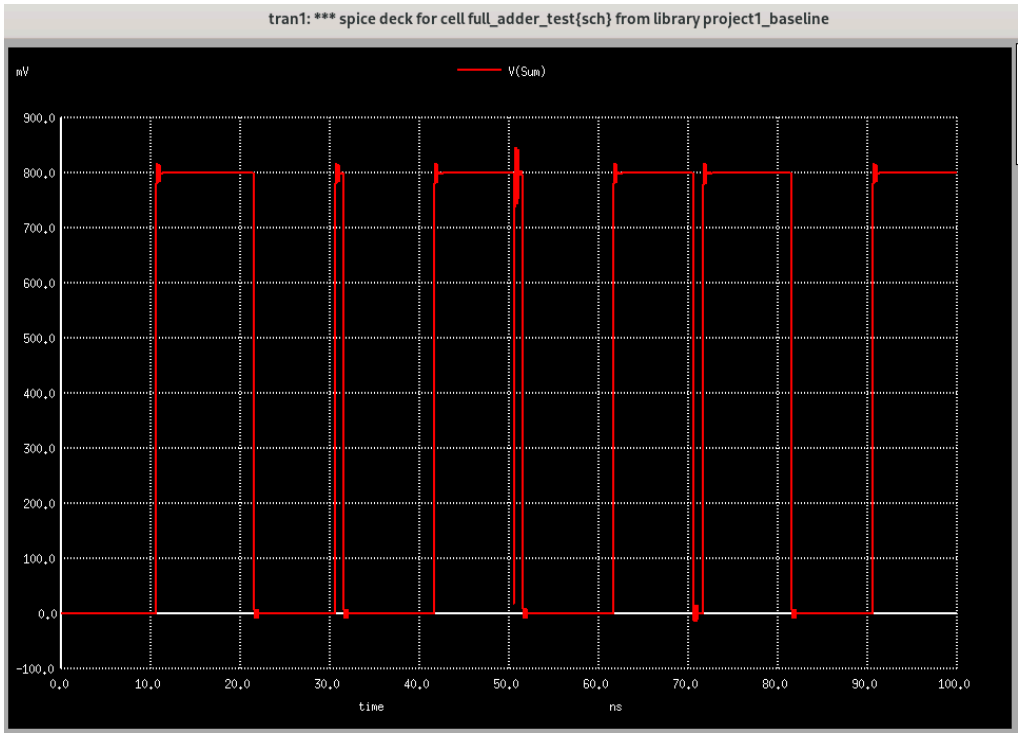
Output (yellow) is high only when one of red or blue inputs is high and the other is low, as we expect from the exclusive OR gate with 2 inputs.

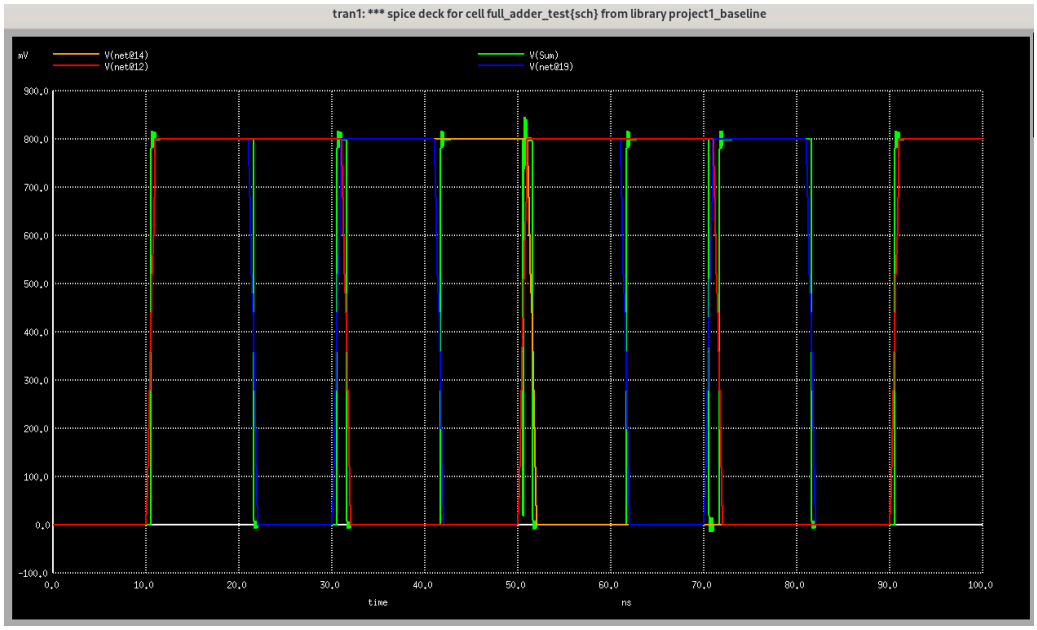
Full Adder



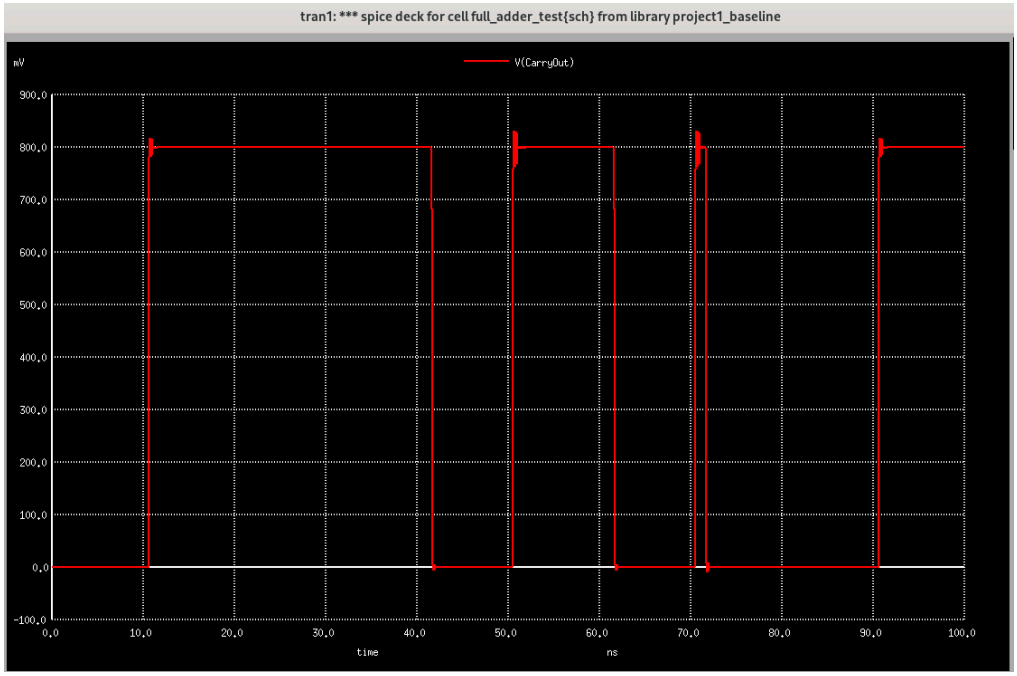
Sum Functionality Testing

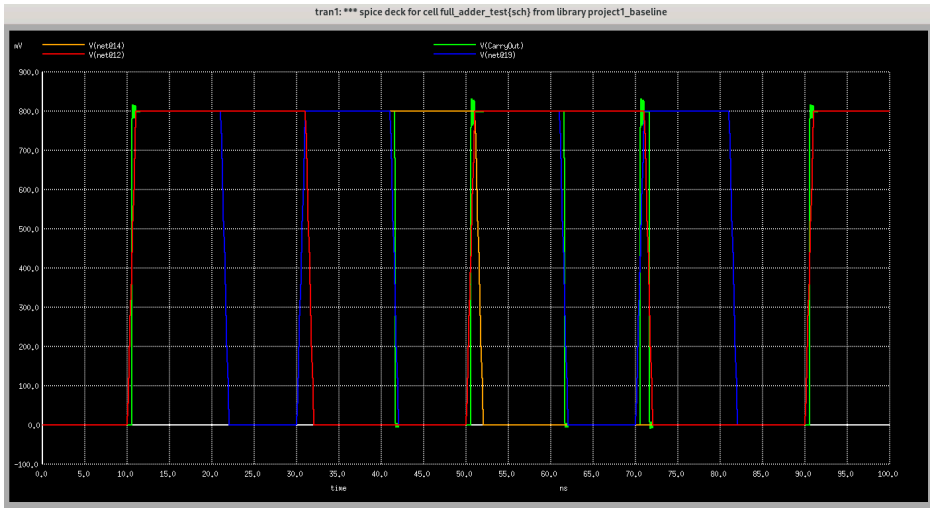
A = 14, B = 12, Cin = 19





CarryOut Functionality Testing





The following truth table was constructed based on the plots above, dissecting for each input case and output Sum and CarryOut values

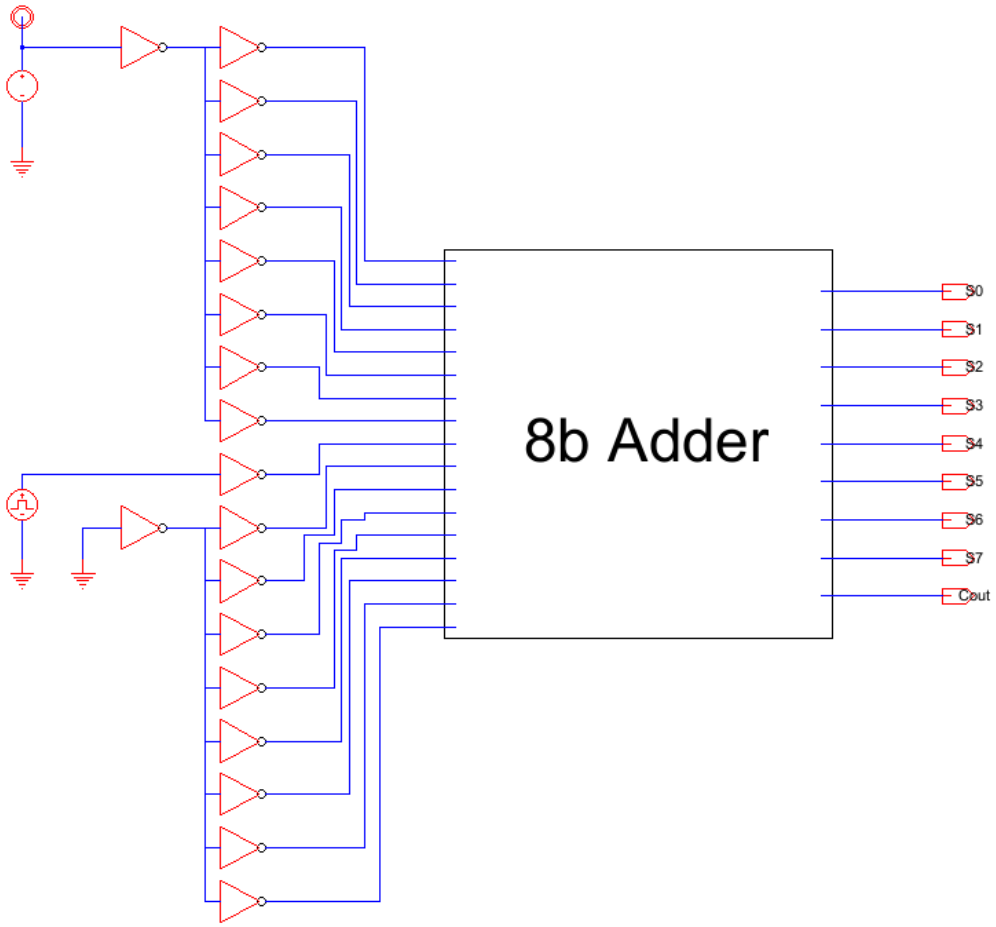
A	B	Cin	Sum	$Cout$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

This adheres to the functionality we expect from the RCA, verifying it’s correctness.

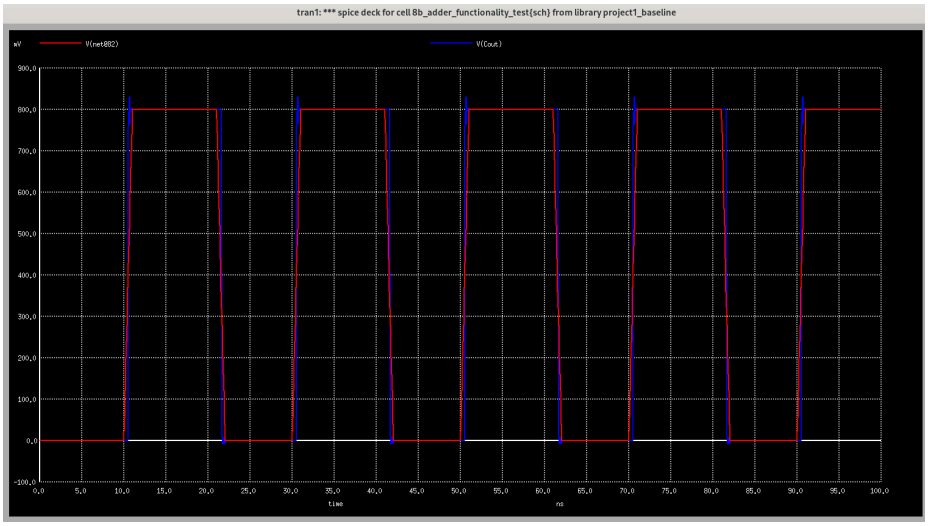
8-bit Ripple Carry Adder

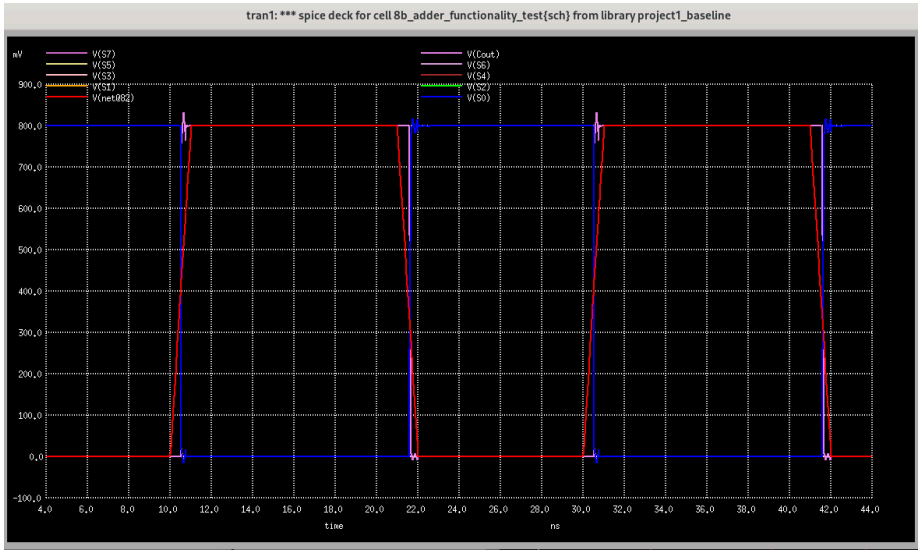
To verify the functionality of the 8-bit ripple-carry adder without exhaustively testing all input combinations, I selected two specific test cases that exercise the most critical behaviors of the design. In the first case, I set input A=255A = 255, B=1B = 1, and Cin=0\text{Cin} = 0. This forces a carry to propagate through all eight full adder stages, verifying correct wiring, chaining, and carry generation logic across the entire adder. In the second case, I used A=3A = 3, B=1B = 1, and Cin=0\text{Cin} = 0, which produces a brief internal carry starting at bit 0 and terminating at bit 2, resulting in only bit 2 of the sum being high. This case confirms proper handling of localized carry propagation and validates internal sum computation. Together, these two test cases cover full carry propagation and partial carry absorption, demonstrating the correctness of the ripple-carry adder's operation.

Test 1: Carry Propagated through all 8 Stages



Red = B0, Blue = Cout

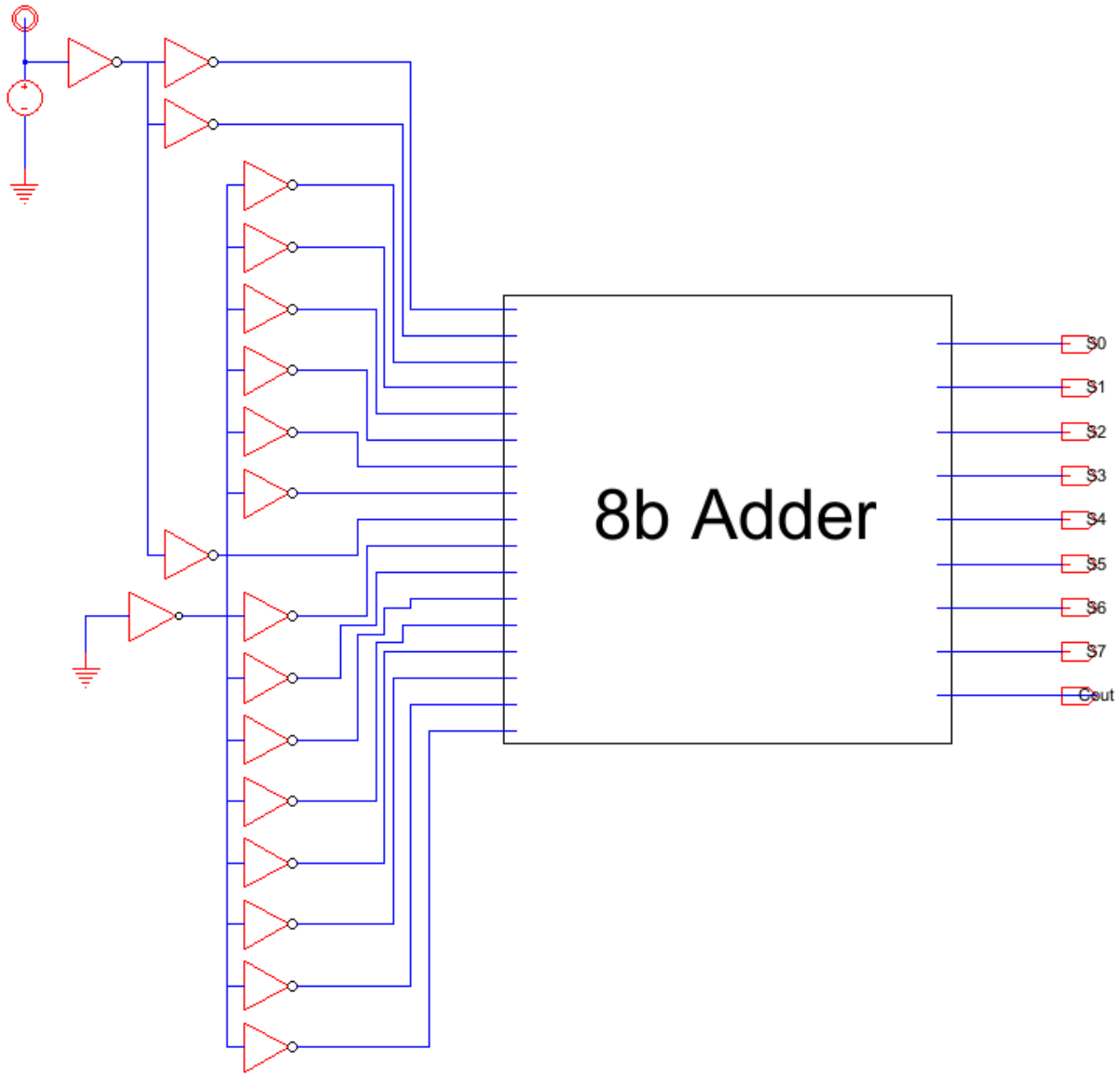




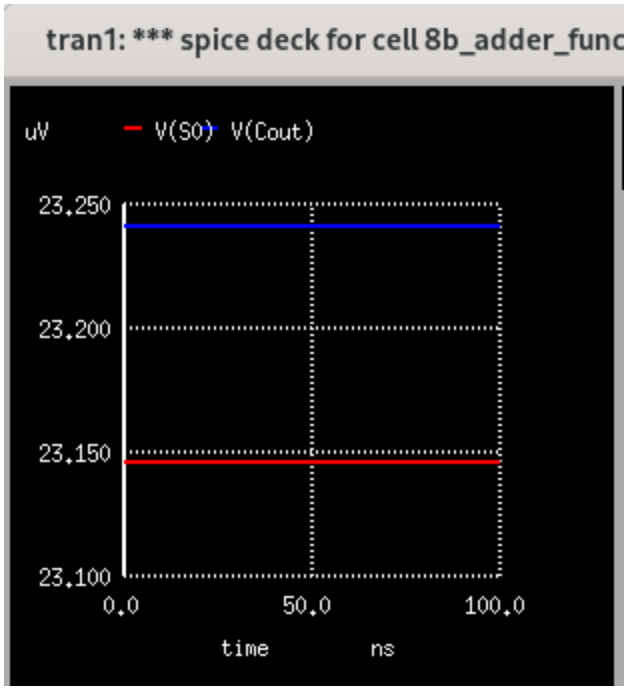
The Cout appropriately switches, and the sum outputs are at the expected levels.

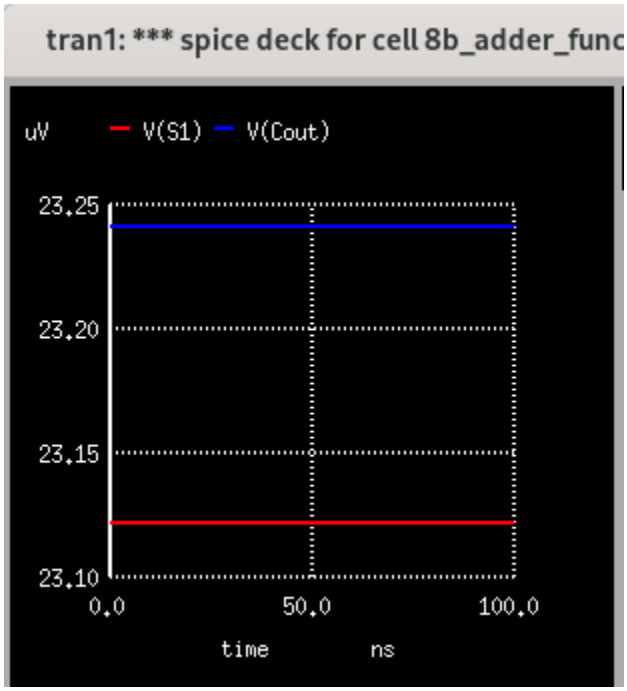
Test 2: Internal Carry with Only 1 Sum as High

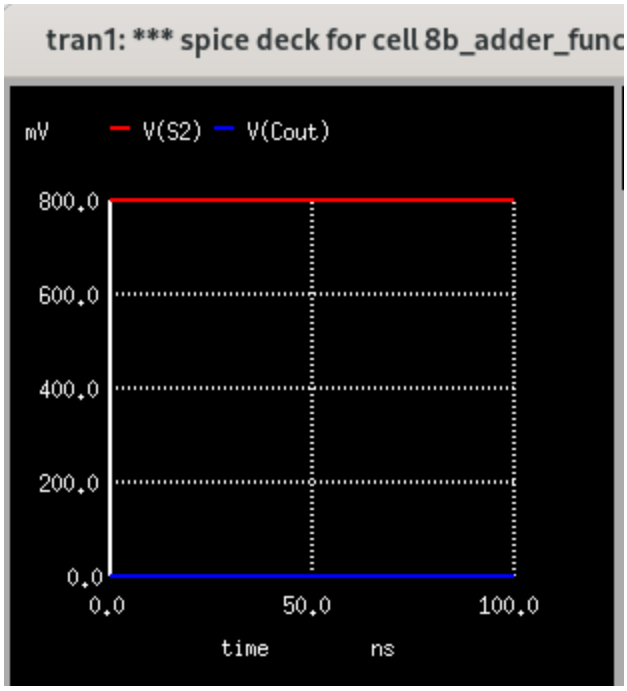
DC Input to Test Final Result

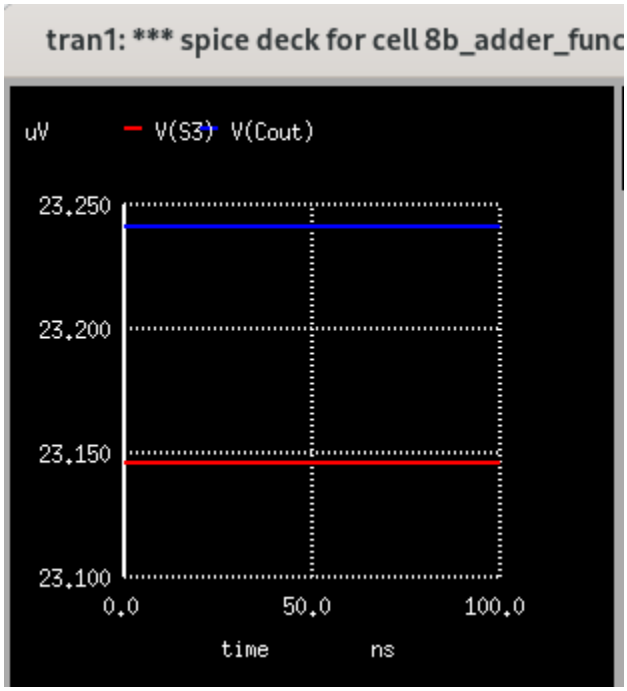


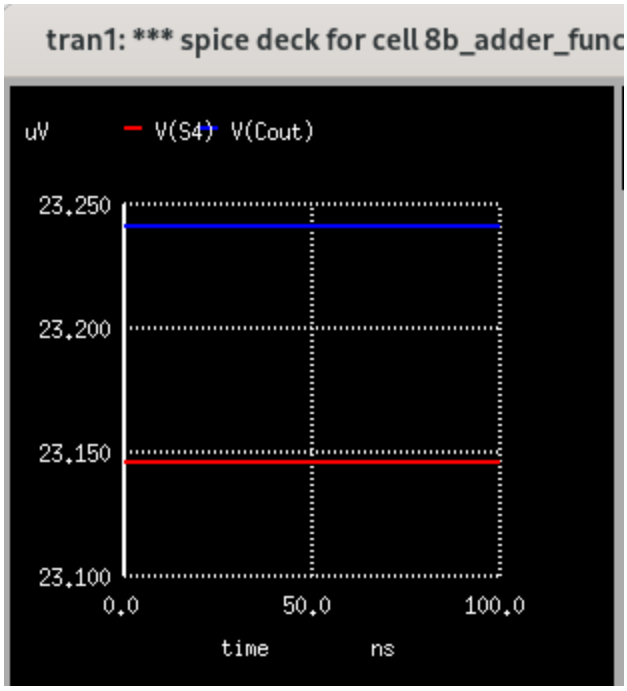
We can see from the plots below that in this DC test, only S2 is at a logic high. This is the expected behaviour we were seeking. This confirms the functionality of the 8-bit RCA baseline design.

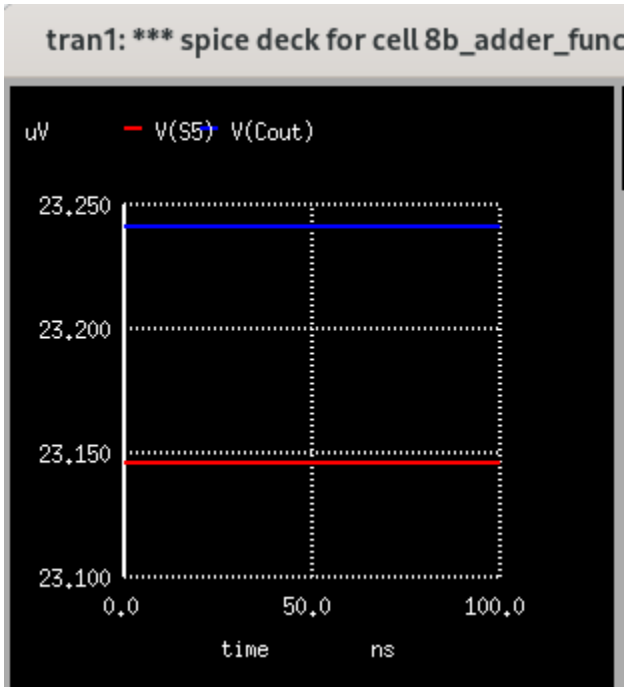

tran1: *** spice deck for cell 8b_adder_func
uV
V(S0)
V(Cout)
23.250
23.200
23.150
23.100
0.0
50.0
100.0
time
ns

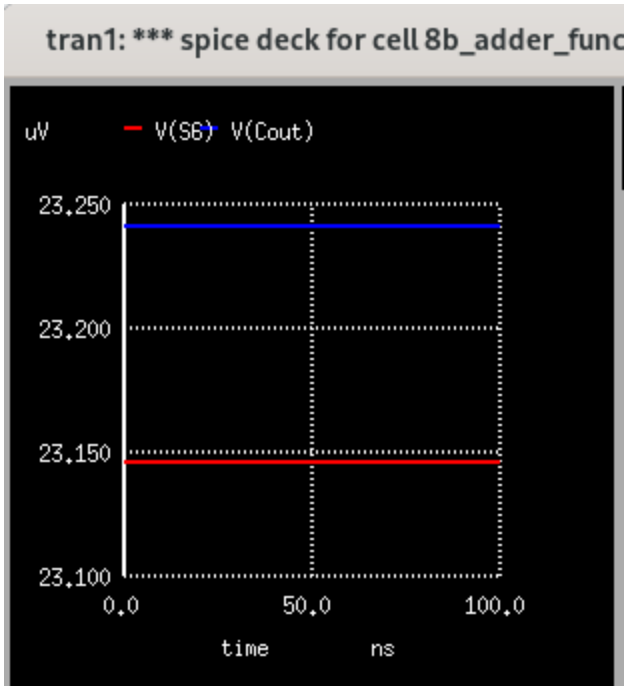

tran1: *** spice deck for cell 8b_adder_func
uV
V(S1)
V(Cout)
23.25
23.20
23.15
23.10
0.0
50.0
100.0
time
ns

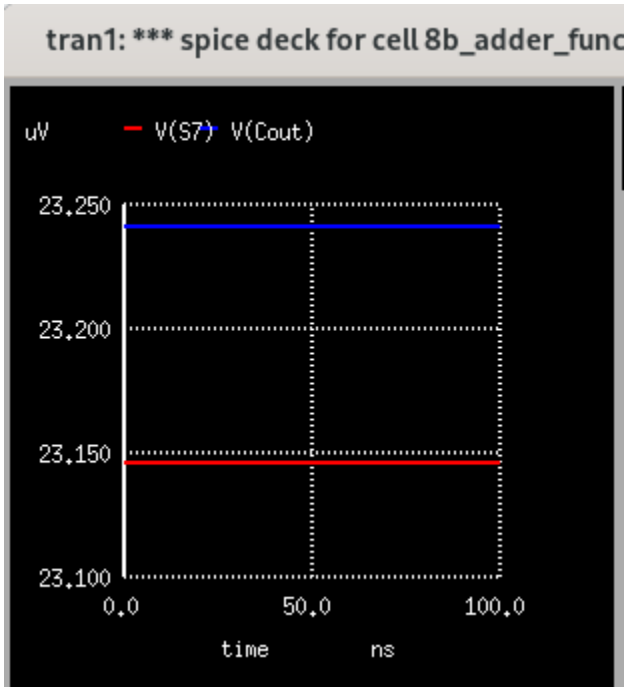

tran1: *** spice deck for cell 8b_adder_func
mV
V(S2)
V(Cout)
800.0
600.0
400.0
200.0
0.0
0.0
50.0
100.0
time
ns


tran1: *** spice deck for cell 8b_adder_func
uV
V(S3)
V(Cout)
23.250
23.200
23.150
23.100
0.0
50.0
100.0
time
ns


tran1: *** spice deck for cell 8b_adder_func
uV
V(S4)
V(Cout)
23.250
23.200
23.150
23.100
0.0
50.0
100.0
time
ns


tran1: *** spice deck for cell 8b_adder_func
uV
V(S5)
V(Cout)
23.250
23.200
23.150
23.100
0.0
50.0
100.0
time
ns

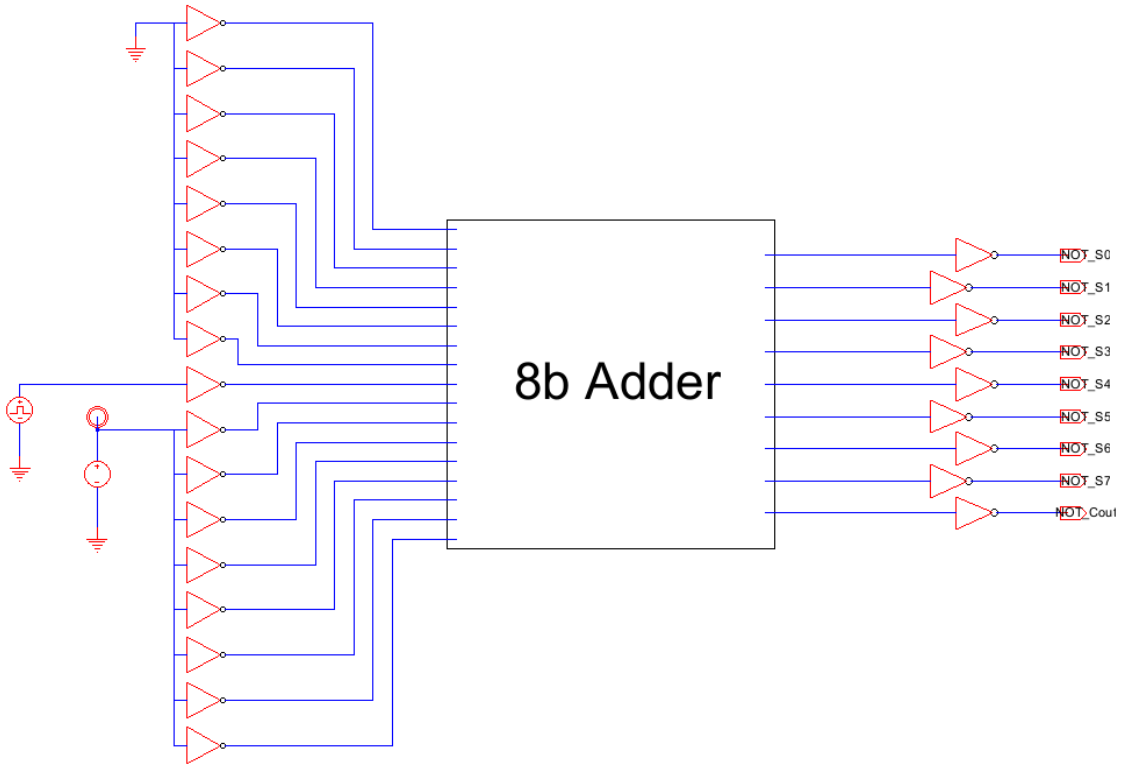

tran1: *** spice deck for cell 8b_adder_func
uV
V(S6)
V(Cout)
23.250
23.200
23.150
23.100
0.0
50.0
100.0
time
ns


tran1: *** spice deck for cell 8b_adder_func
uV
V(S7)
V(Cout)
23.250
23.200
23.150
23.100
0.0
50.0
100.0
time
ns

Evaluation Metrics

Delay

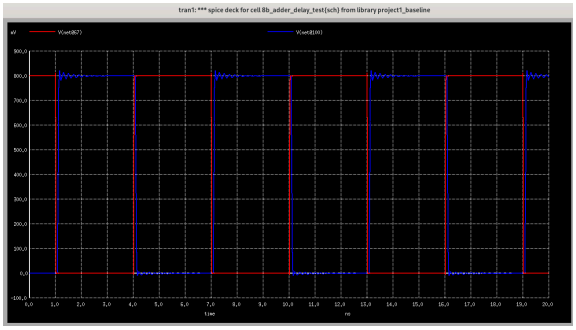
To accurately evaluate the worst-case delay of the 8-bit ripple carry adder, a test circuit was constructed to replicate realistic datapath conditions, consistent with how the adder would be used in a larger arithmetic unit such as an adder tree. All input bits A0-A7 and B0-B7 were driven through minimum-sized inverters to model the drive strength of upstream logic, specifically the Sum output of another full adder, which in this design is driven by a unit inverter. On the output side, each Sum bit and the final CarryOut were loaded with an inverter scaled to three times minimum size to emulate the capacitive load presented by the A or B input of a downstream full adder. This sizing is based on a gate-level analysis of the full adder circuit, which showed that an input such as A fans into one XOR2 (internally composed of a NOR2 and a NAND2) and another NAND2 gate, resulting in an estimated input capacitance of approximately six times the unit gate capacitance; since a unit inverter has a gate capacitance of twice that value, a three-times minimum inverter was used to match this load. The worst-case input was carefully selected as A equal to all ones, B equal to all zeros except for the least significant bit, and carry-in equal to zero. This configuration causes a carry to propagate from the least significant bit through every full adder stage to the most significant bit, ultimately producing a final carry-out of one. Measuring the delay from the transition of the least significant bit of B to each output allows us to capture the timing behavior across the full carry chain, thereby revealing the true worst-case delay path of the design.

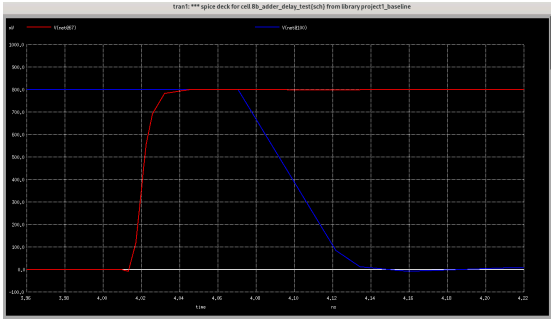


To accurately capture the worst-case delay of the 8-bit ripple carry adder, measurements were made by triggering on the rising and fallin =g edge of the B_0 input and observing the resulting transitions at each output (propagation delay rise and fall at S7 and Cout). This is because in the selected worst-case test values—where A is all ones, B is all zeros except for a pulse on B_0, and carry-in is held at zero—the ripple carry path is fully activated. Initially, the output of the adder produces all ones on the sum lines and zero on the carry-out. When B_0 transitions from logic zero to one, it causes the sum outputs to flip from one to zero and the carry-out to rise from zero to one, following a full carry propagation from the least to the most significant bit. Therefore, the delay to each sum output (S_0 through S_7) is measured from the rising transition of B_0 to the falling transition at that sum output, while the delay to CarryOut is measured from the same rising transition of B_0 to the rising transition of the carry-out signal. This direction-specific measurement approach ensures that the observed delays correspond exactly to the transitions caused by the carry chain triggered by B_0. It is sufficient to measure only S_7 and CarryOut to determine the worst-case delay of the 8-bit ripple carry adder. The lower-order sum bits (S_0–S_6) settle earlier and do not reflect the full carry propagation delay. In contrast, S_7 and CarryOut are both dependent on the longest internal carry chain and therefore represent the latest possible output transitions. Since the goal is to capture the maximum delay from input to output, the longer of the two delays—from the rising edge of B_0 to the falling edge of S_7 or the rising edge of CarryOut—can be reported as the worst-case. The following plots show the empirical derivation of the aforementioned delays through SPICE simulation.

B0 = Net@67, S7 = Net@100, Cout = Net@101

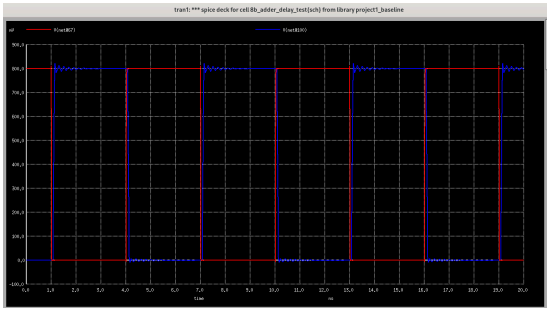
Propagation Delay Rise B0 -> S7

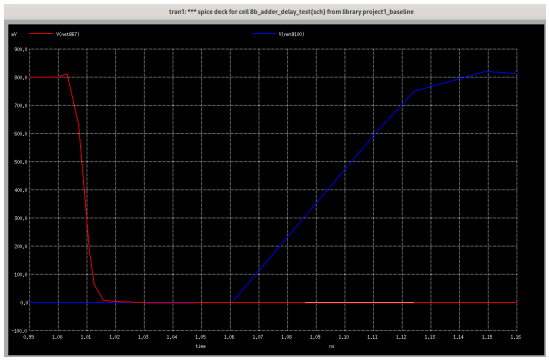




```
dx = 7.85e-11,
```

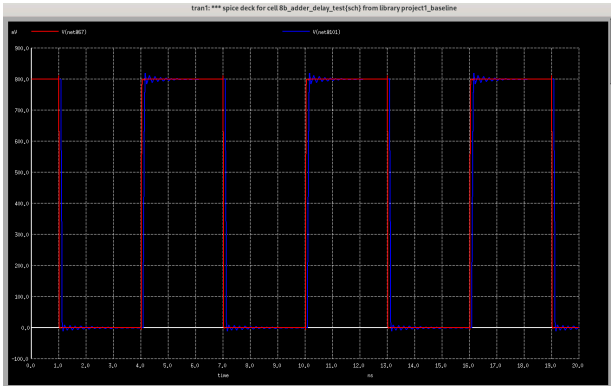
Propagation Delay Fall B0 -> S7

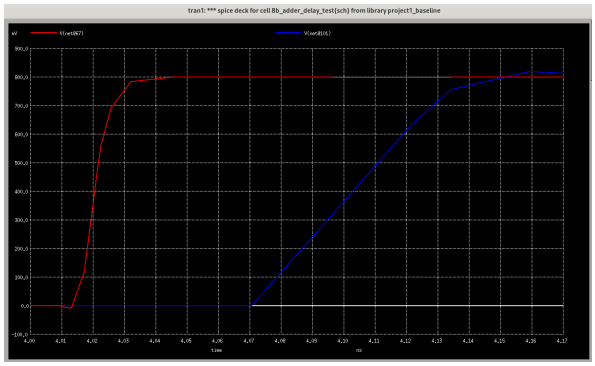





```
dx = 1.14925e-10, c
```

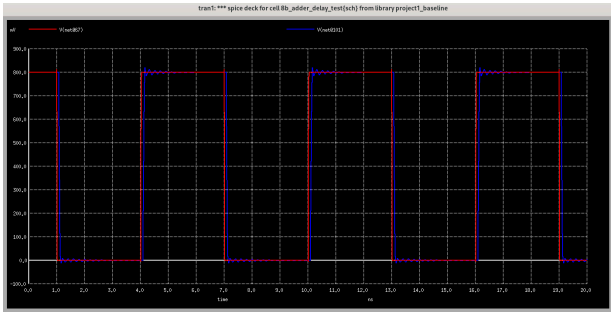
Propagation Delay Rise B0 -> Cout

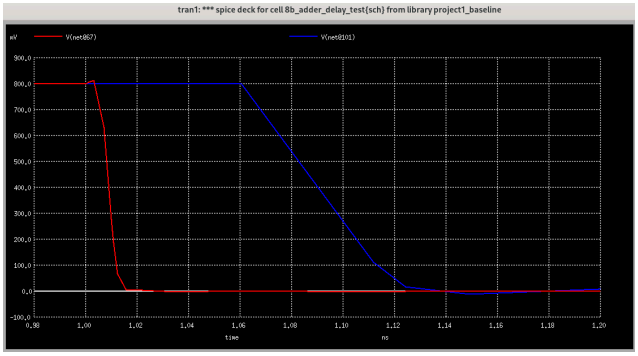




dx = 8.24691e-11,

Propagation Delay Fall B0 -> Cout





dx = 1.50438e-10, c

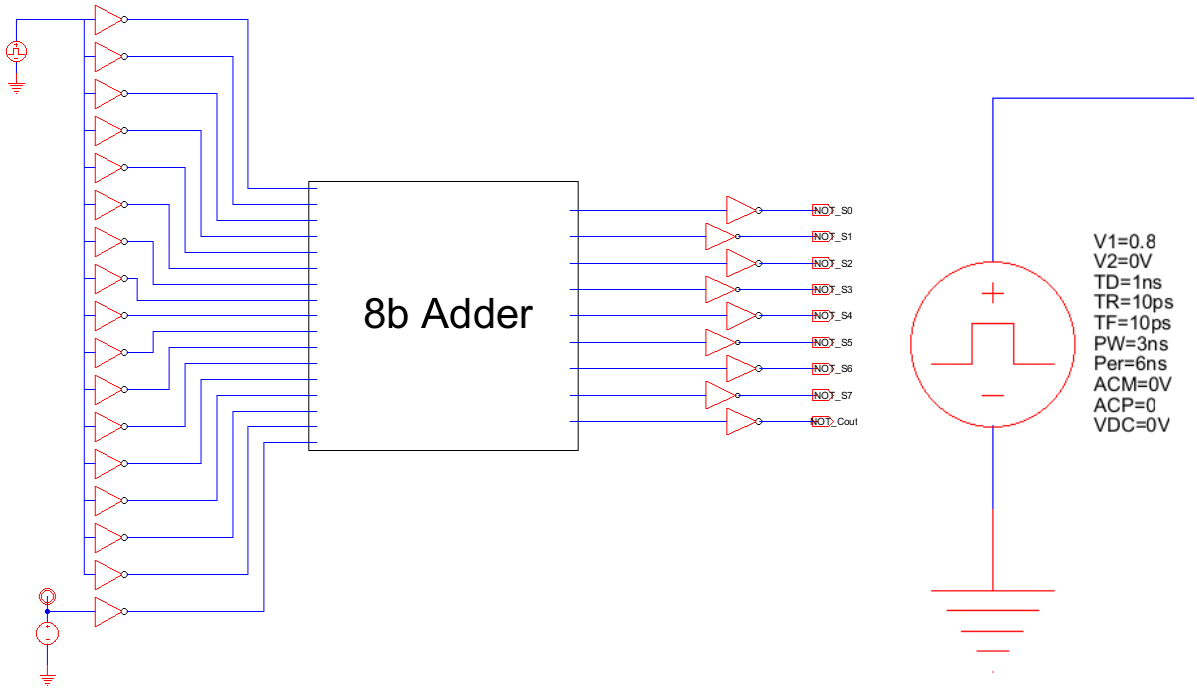
$\frac{7.85}{0.69} \cdot 10^{-11}$	$= 1.13768116 \times 10^{-10}$
$\frac{1.14925}{0.69} \cdot 10^{-10}$	$= 1.66557971 \times 10^{-10}$
$\frac{8.24691}{0.69} \cdot 10^{-11}$	$= 1.19520435 \times 10^{-10}$
$\frac{1.50438}{0.69} \cdot 10^{-10}$	$= 2.18026087 \times 10^{-10}$

Dividing each of these dx values 0.69, we find the the worst case propagation delay is seen in the fall time propagation delay fall from B0 to Cout (dx = 1.50438e-10s), with a delay of **218ps**. This is the delay I will use for my leakage energy calculations and the value I will try to optimize.

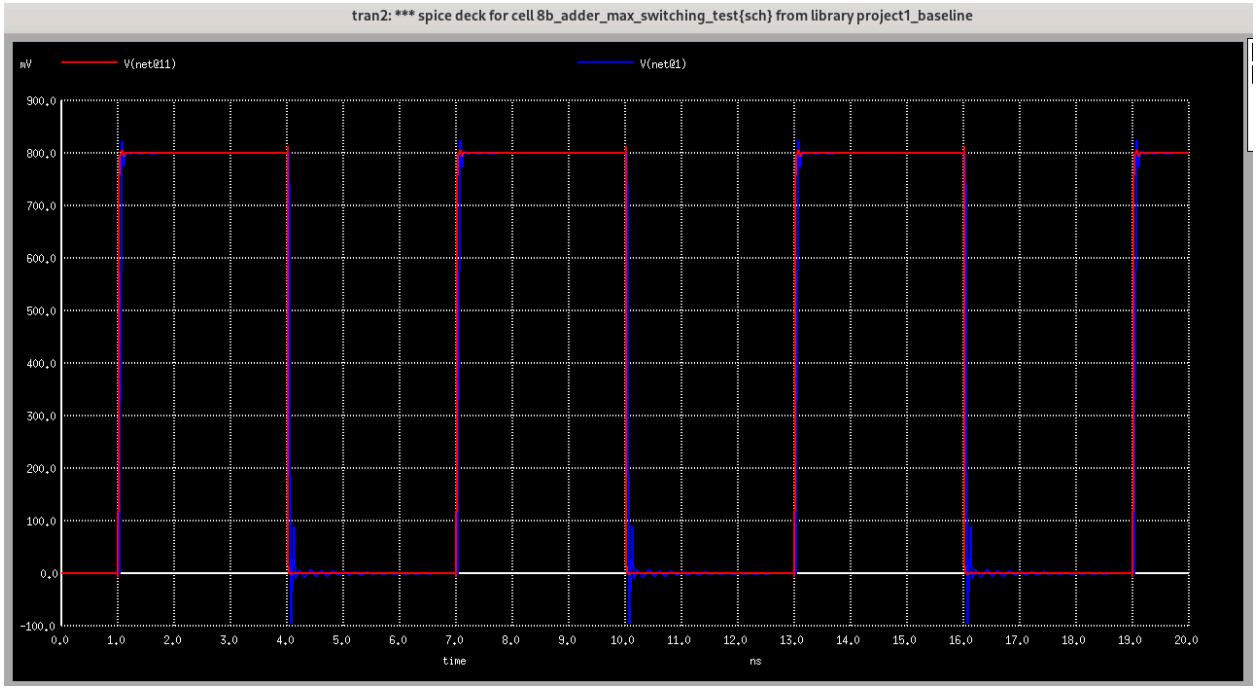
Active Energy

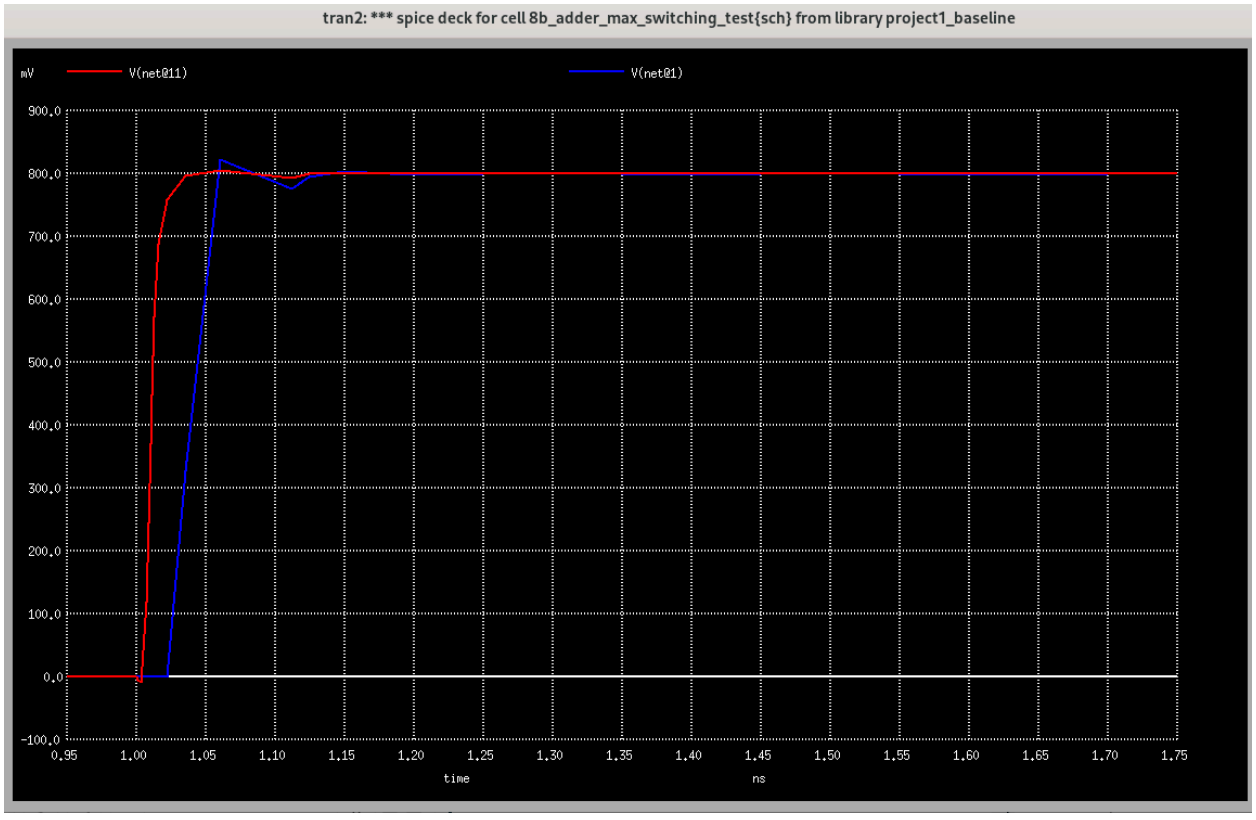
Maximum Switching Energy Case

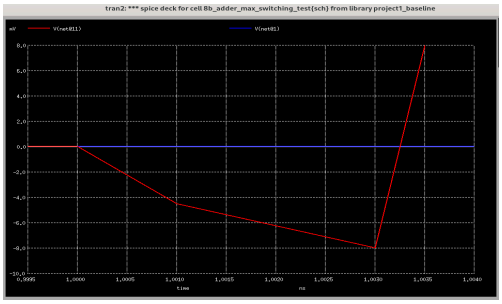
To measure the maximum switching energy of the 8-bit ripple-carry adder, I configured all inputs A, B, and internal node drivers to switch simultaneously from 0 V to 1 V, while keeping Cin tied to 0 V. This transition from A = 0, B = 0 to A = 1, B = 1 across all bits activates every full adder stage and produces the highest switching activity within the circuit. Nearly all internal nodes—including sum and carry generation paths—undergo a transition, leading to maximum dynamic energy dissipation. I calculated the total switching energy by measuring the charge drawn from the power supply during this transition and multiplying it by VDD = 0.8V.

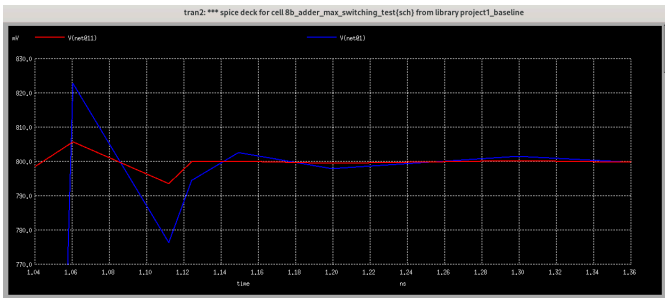


A0 is net@11 and my Cout is net@1


tran2: *** spice deck for cell 8b_adder_max_switching_test{sch} from library project1_baseline
mV
V(net011)
V(net01)
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
-100.0
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0 20.0
time
ns


tran2: *** spice deck for cell 8b_adder_max_switching_test{sch} from library project1_baseline
mV
V(net011)
V(net01)
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
-100.0
0.95 1.00 1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50 1.55 1.60 1.65 1.70 1.75
time
ns





x0 = 1e-09, y0 = 0

x0 = 1.35e-09, y0 = 0.8

Q-Switch Calculation

```
ngspice 116 -> meas tran qswitch integ I(VV_Generi@0) from=1ns to=1.35ns
qswitch              =  -5.84685e-15 from=  1.00000e-09 to=  1.35000e-09
```

Max Switching Energy (ignoring the negative sign [explained below]) = 0.8 * Q-Switch = 4.67748 * 10^-15 = **4.67748 fJ**

We ignore the negative sign in the Qswitch value because it simply reflects the direction of current flow with respect to the chosen current probe polarity in SPICE. In ngspice, when measuring the current drawn from a voltage source (such as VDD), the tool treats current flowing *into* the positive terminal of the source as negative—since it corresponds to energy being delivered *from* the source to the circuit.

However, when calculating switching energy, we care only about the *magnitude* of the energy drawn from the power supply, which is always a positive quantity. Energy is a scalar, and the negative sign is an artifact of convention, not of physical meaning. Thus, the correct switching energy is computed as:

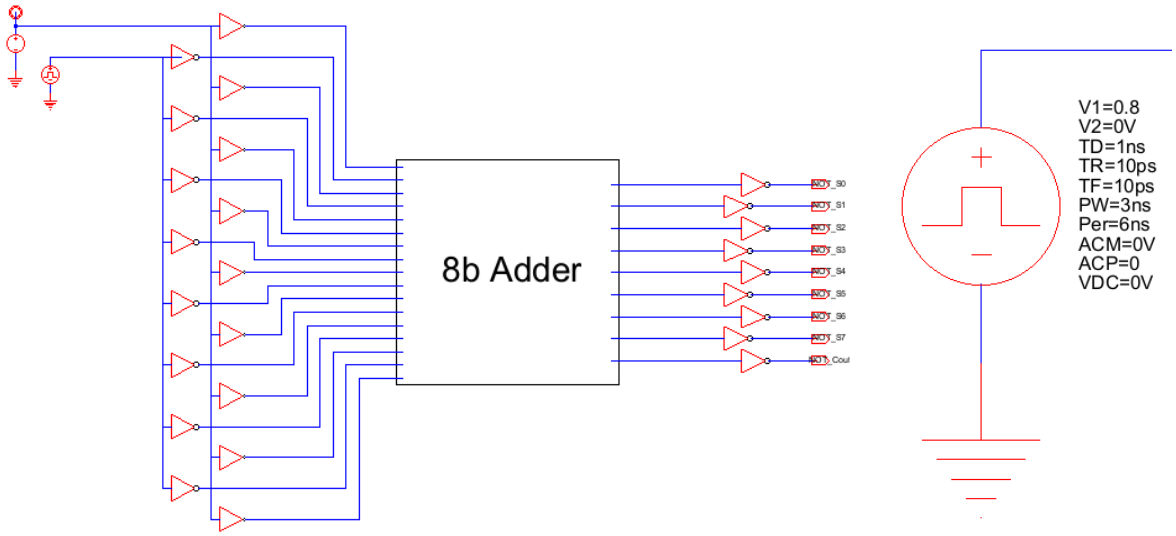
E switch=VDD* |Q Switch| = 0.8*5.84685*10^−15 = 4.67748 fJ

This represents the total energy consumed during the switching event, independent of sign.

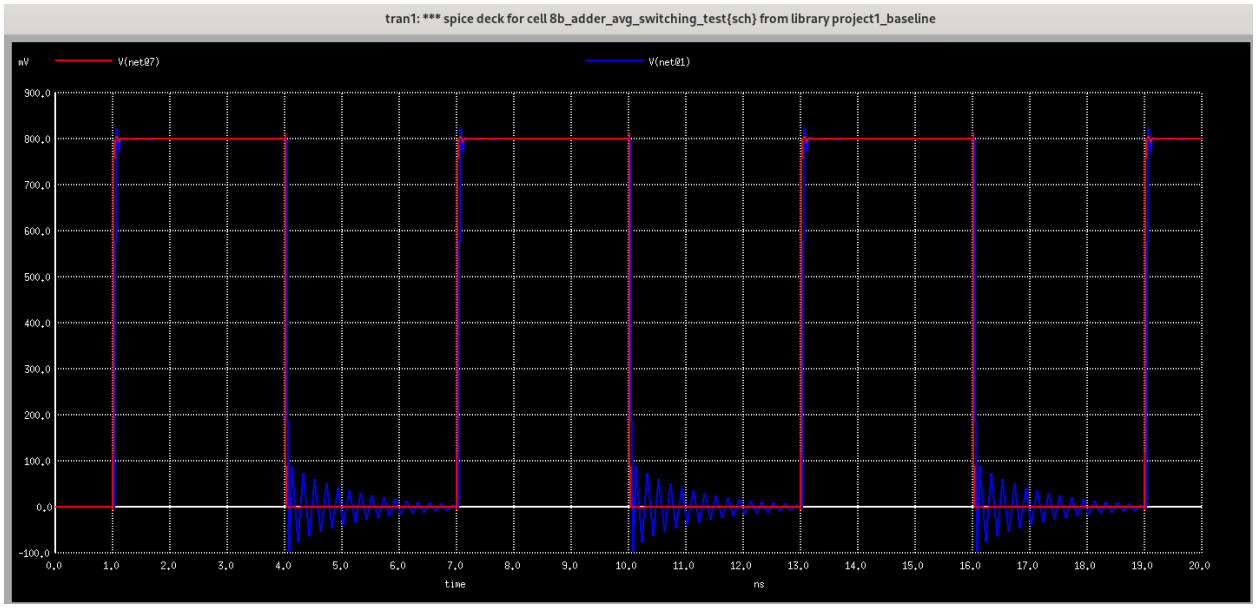
Average Switching Energy Case

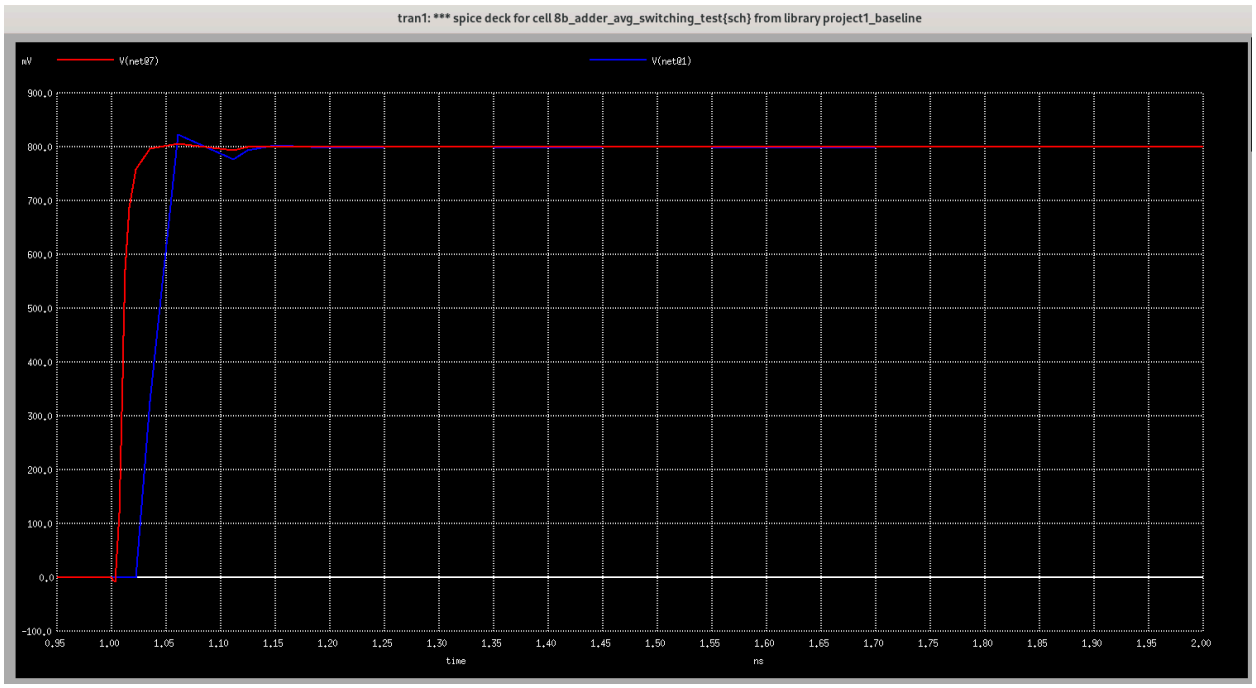
For the average switching energy case, I selected a realistic input configuration where exactly half of the input bits toggle from 0 to 1. Specifically, I applied rising edge pulses to the odd-indexed bits: A1,A3,A5,A7,B1,B3,B5,B7, while holding the remaining inputs—including all even-indexed A and B bits and the carry-in Cin—at ground. This ensures that out of the 17 total input bits (8 A, 8 B, and 1 Cin), exactly 8 bits are actively switching, simulating the average case where 50% of the inputs toggle.

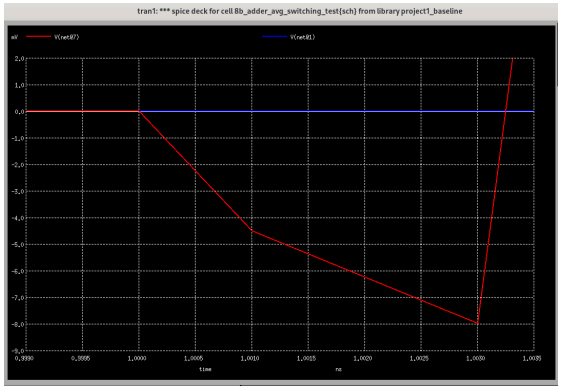
This configuration was chosen to provide a representative measurement of typical dynamic behavior during real-world operation, where not all bits switch simultaneously. Moreover, by spreading the toggling bits across both operands and alternating indices, the configuration avoids any biased switching cluster and fairly exercises internal logic across multiple adder stages. It also ensures that the final carry-out still switches, allowing a valid energy measurement reflective of meaningful computation within the adder.

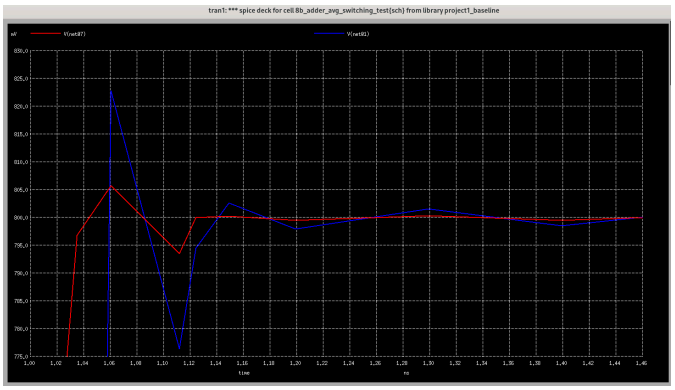


A1 is net@7 and Cout is net@1


tran1: *** spice deck for cell 8b_adder_avg_switching_test{sch} from library project1_baseline
mV
V(net07)
V(net01)
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
-100.0
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0 20.0
time
ns


tran1: *** spice deck for cell 8b_adder_avg_switching_test{sch} from library project1_baseline
mV
V(net07)
V(net01)
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
-100.0
0.95 1.00 1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50 1.55 1.60 1.65 1.70 1.75 1.80 1.85 1.90 1.95 2.00
time
ns


tran1: *** spice deck for cell 8b_adder_avg_switching_test{sch} from library project1_baseline


tran1: *** spice deck for cell 8b_adder_avg_switching_test{sch} from library project1_baseline

```
x0 = 1e-09, y0 = 0          x0 = 1.45e-09, y0 = 0.8
```

Q-Switch Calculation

```
ngspice 118 -> meas tran qswitch_avg integ I(VV_Generi@0) from=1ns to=1.45ns
qswitch_avg          =  -2.93051e-15 from=  1.00000e-09 to=  1.45000e-09
```

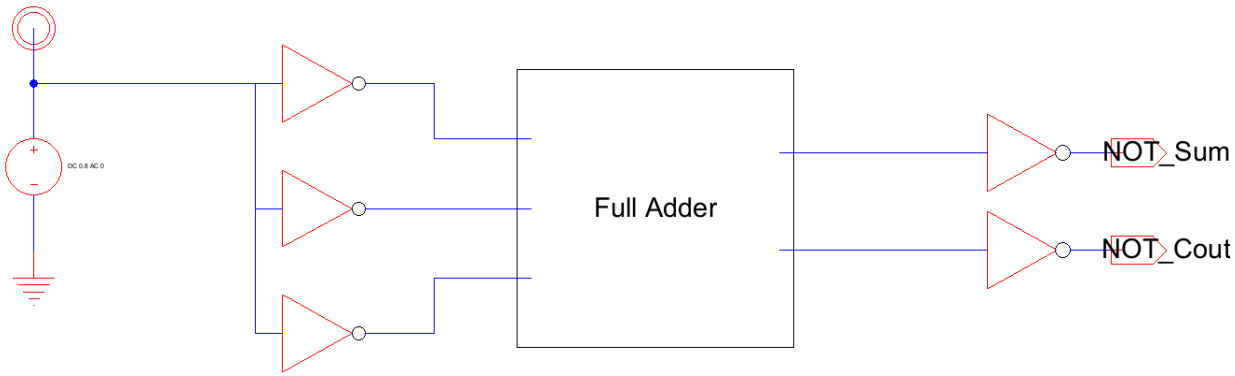
Max Switching Energy (ignoring the negative sign for same reason as in maximum case) = 0.8 * Q-Switch = 2.93051 * 10^-15 = **2.344408 fJ**

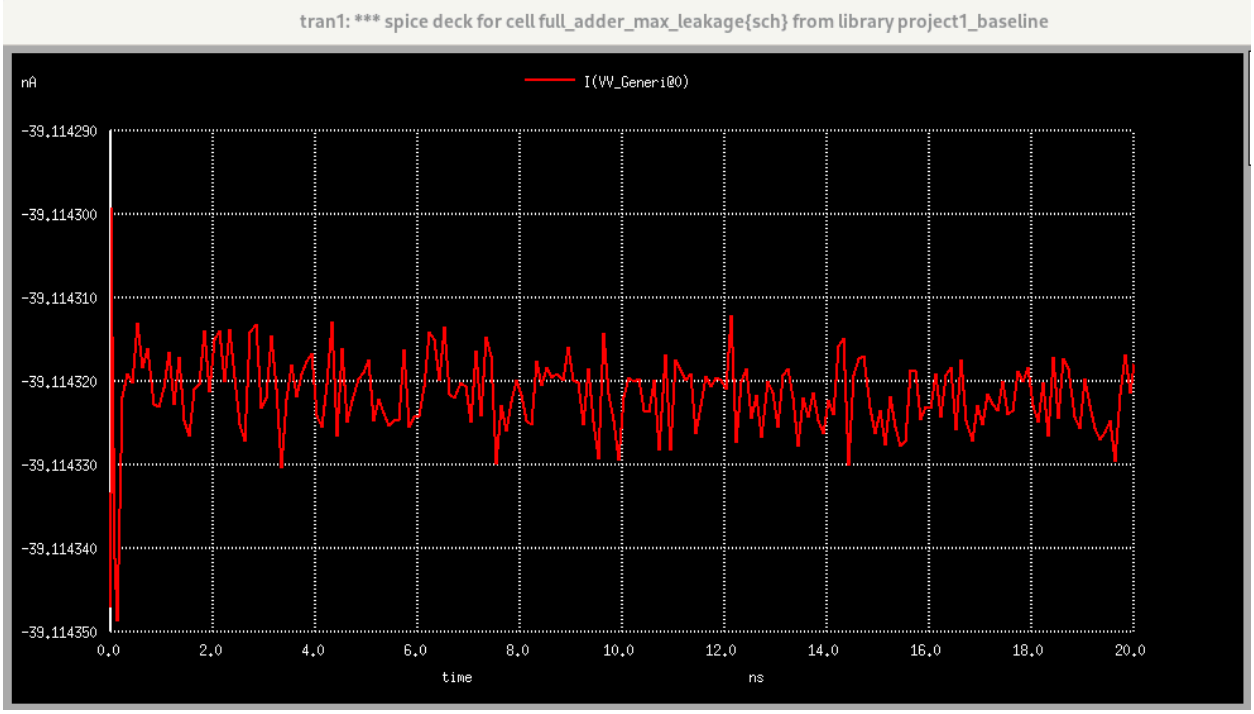
Leakage Energy

Maximum Leakage Energy Case

To identify the maximum leakage energy case for the 8-bit ripple carry adder (RCA), I began by analyzing a single full adder (FA) in isolation. Since leakage energy depends on the static input conditions and the resulting subthreshold and gate leakage through transistors, I examined all eight possible input combinations of the full adder's three inputs (A, B, Cin), where each input was held constant either at logic 0 (0 V) or logic 1 (0.8 V). Each combination was applied using inverters driven by DC voltage sources to ensure a fixed, non-switching configuration. For each case, I ran a transient simulation over one full adder delay period (218 ps), consistent with the evaluation time window prescribed for the project. I then computed the leakage energy using the integral of the current drawn from the VDD supply over this interval, multiplied by the supply voltage (0.8 V). By comparing the leakage energies for all eight input cases, I was able to identify the input configuration that resulted in the highest energy consumption due solely to leakage currents. This maximum-leakage configuration for a single full adder will later be scaled appropriately across the 8-bit RCA, taking into account the required logic levels of the carry-in signals for each bit-slice stage.

Case 1: A = B = Cin = 0

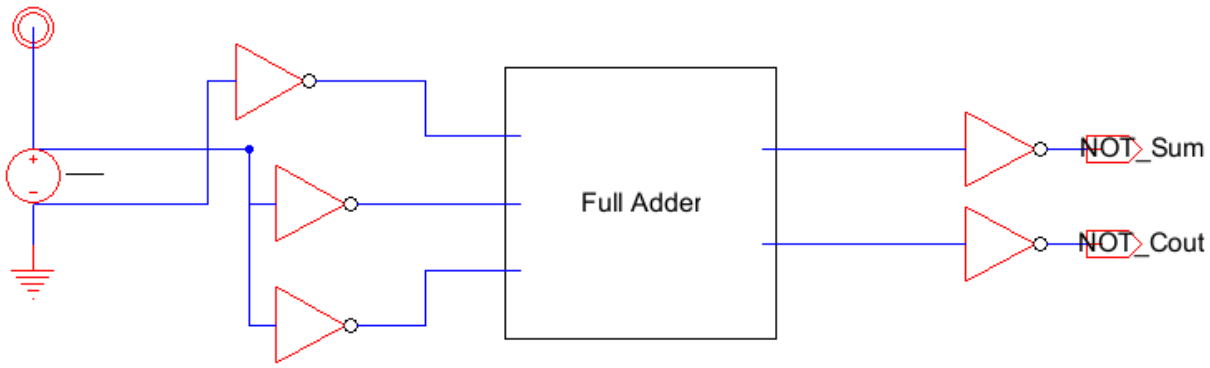


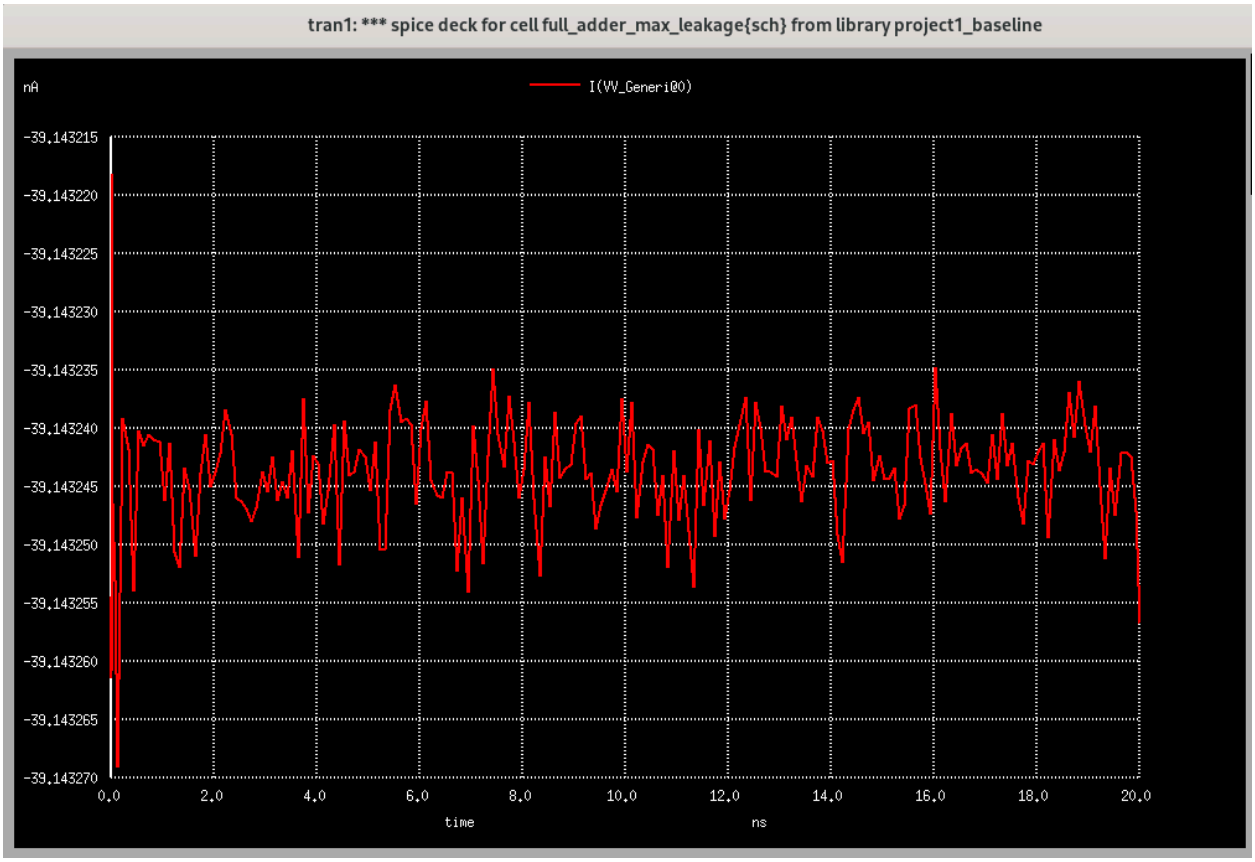



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                _     =  -8.52692e-18 from=  1.00000e-09 to=  1.21800e-09
```

Leakage Energy = 6.821536 * 10^-18 J

Case 2: A = 0.8, B = Cin = 0

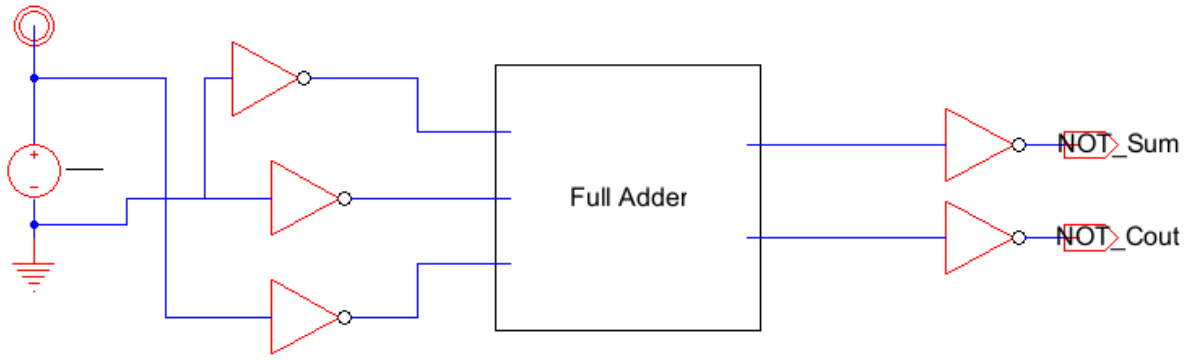


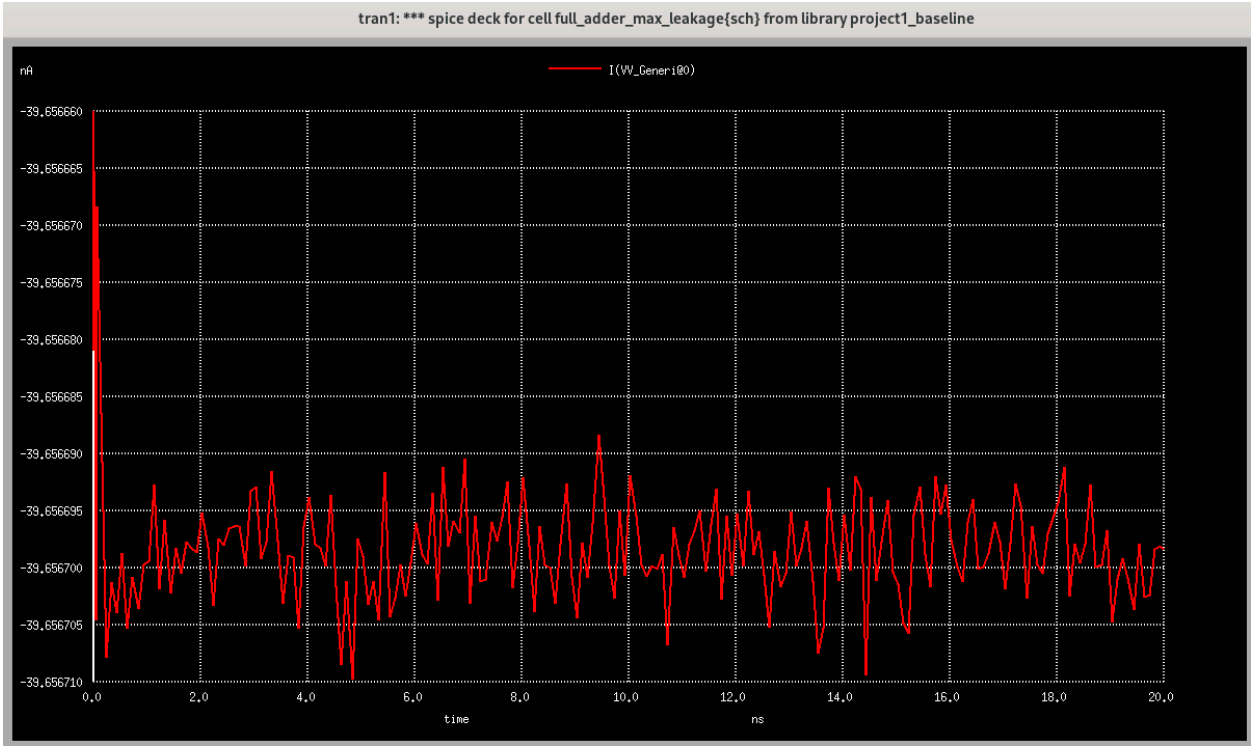



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -8.53323e-18 from=  1.00000e-09 to=  1.21800e-09
```

Leakage Energy = 6.826584 * 10^-18 J

Case 3: A = B = 0.8, Cin = 0

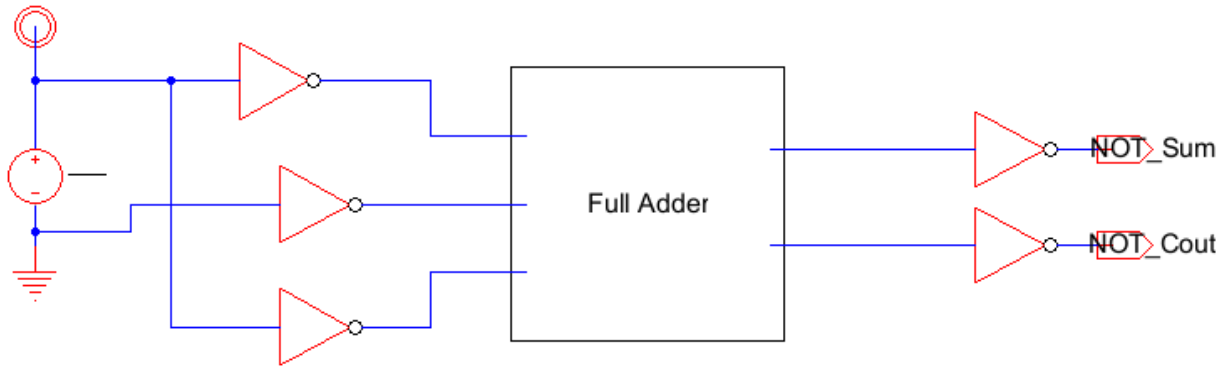


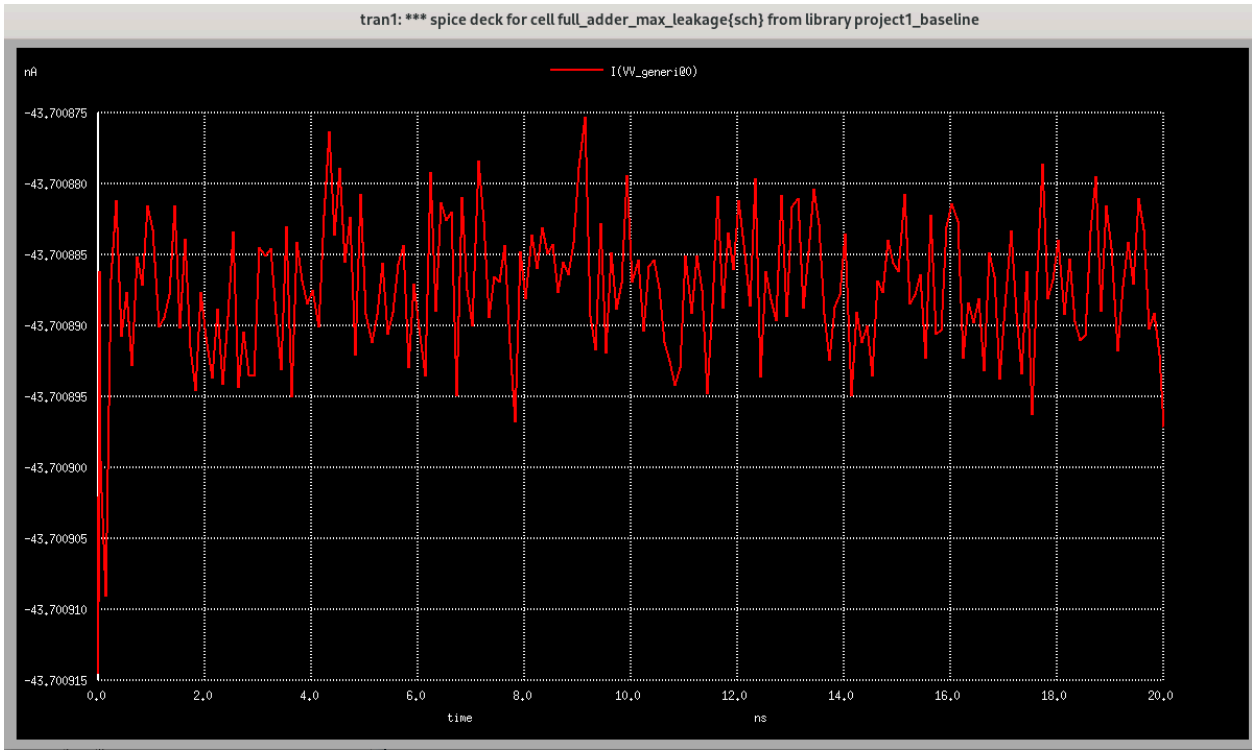



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -8.64516e-18 from=  1.00000e-09 to=  1.21800e-09
```

Leakage Energy = 6.916128* 10^-18 J

Case 4: A = 0, B = 0.8, Cin = 0 (theoretically the same as A = 0.8, B = 0, Cin = 0, but testing to ensure symmetry in case of longer wires, etc.)

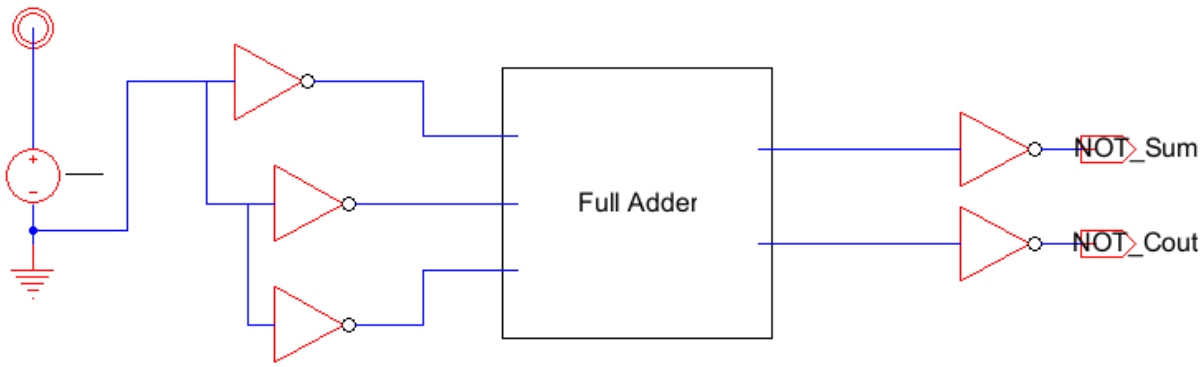


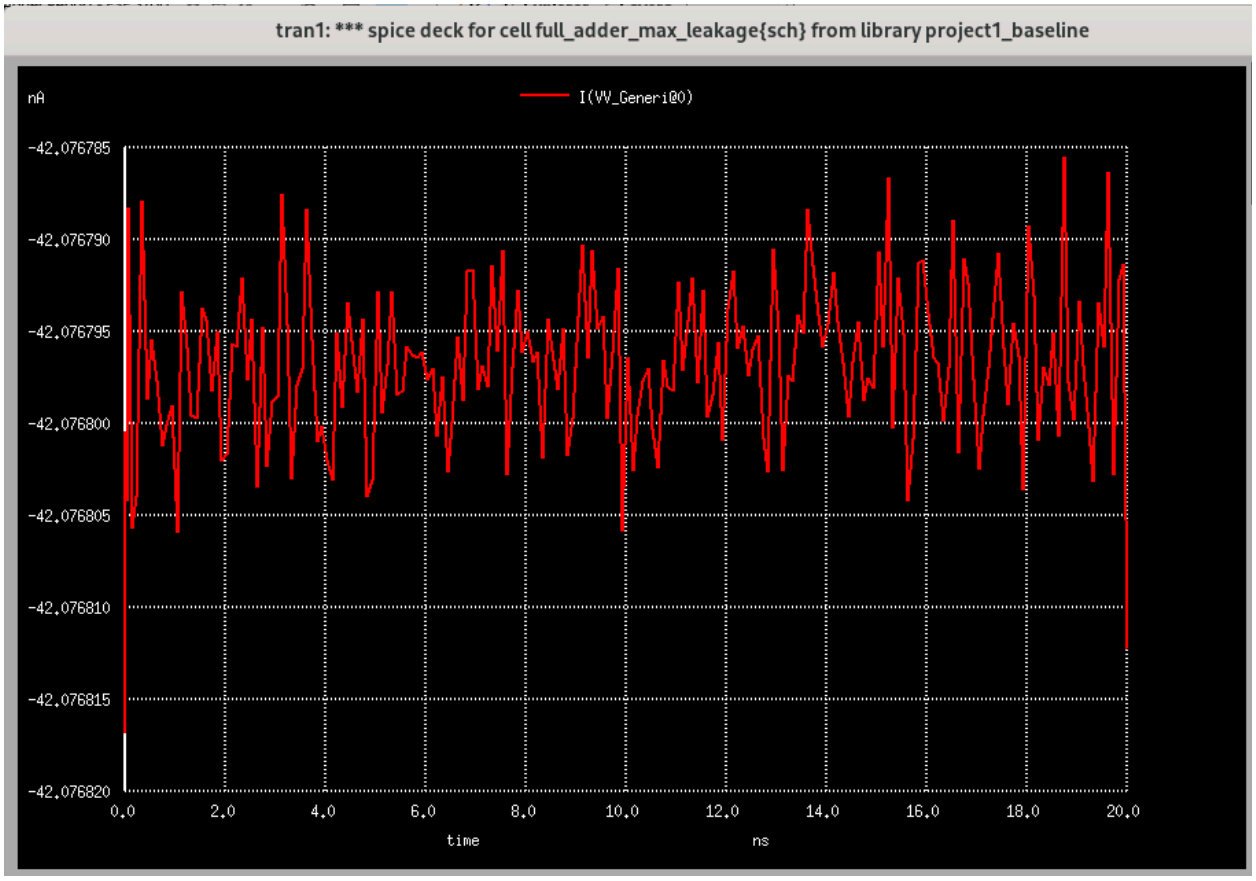



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -9.52679e-18 from=  1.00000e-09 to=  1.21800e-09
```

Leakage Energy = 7.621432 * 10^-18 J

Case 5: A = B = Cin = 0.8

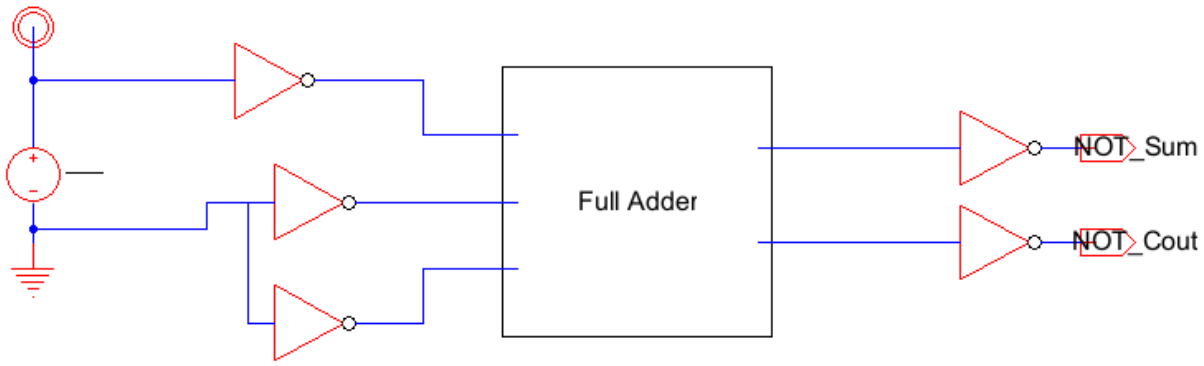


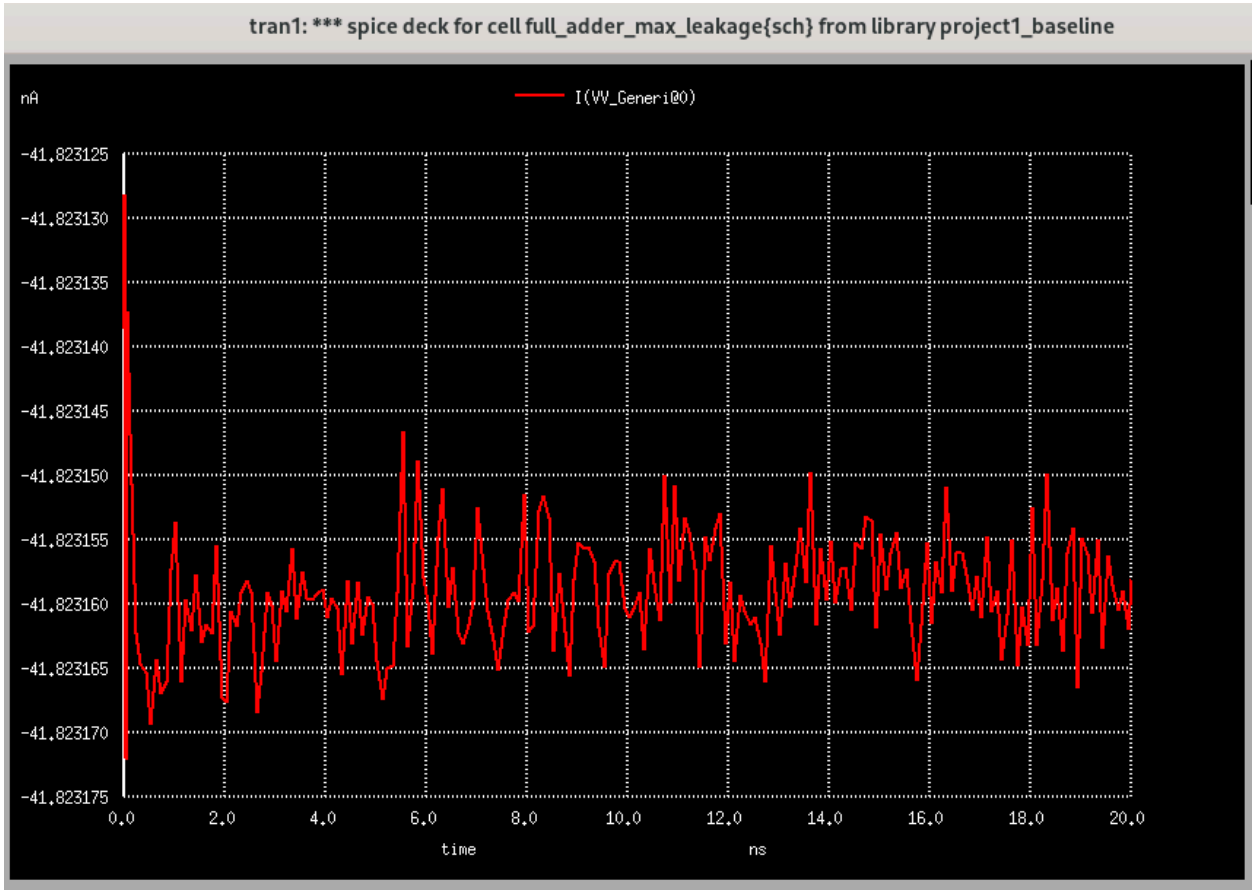



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -9.17274e-18 from=  1.00000e-09 to=  1.21800e-09
```

Leakage Energy = 7.3382 * 10^-18 J

Case 6: A = 0, B = Cin = 0.8

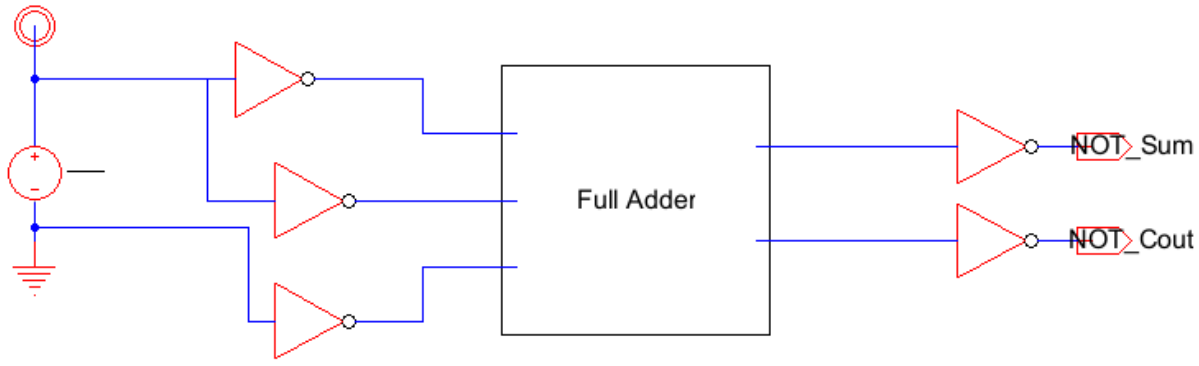


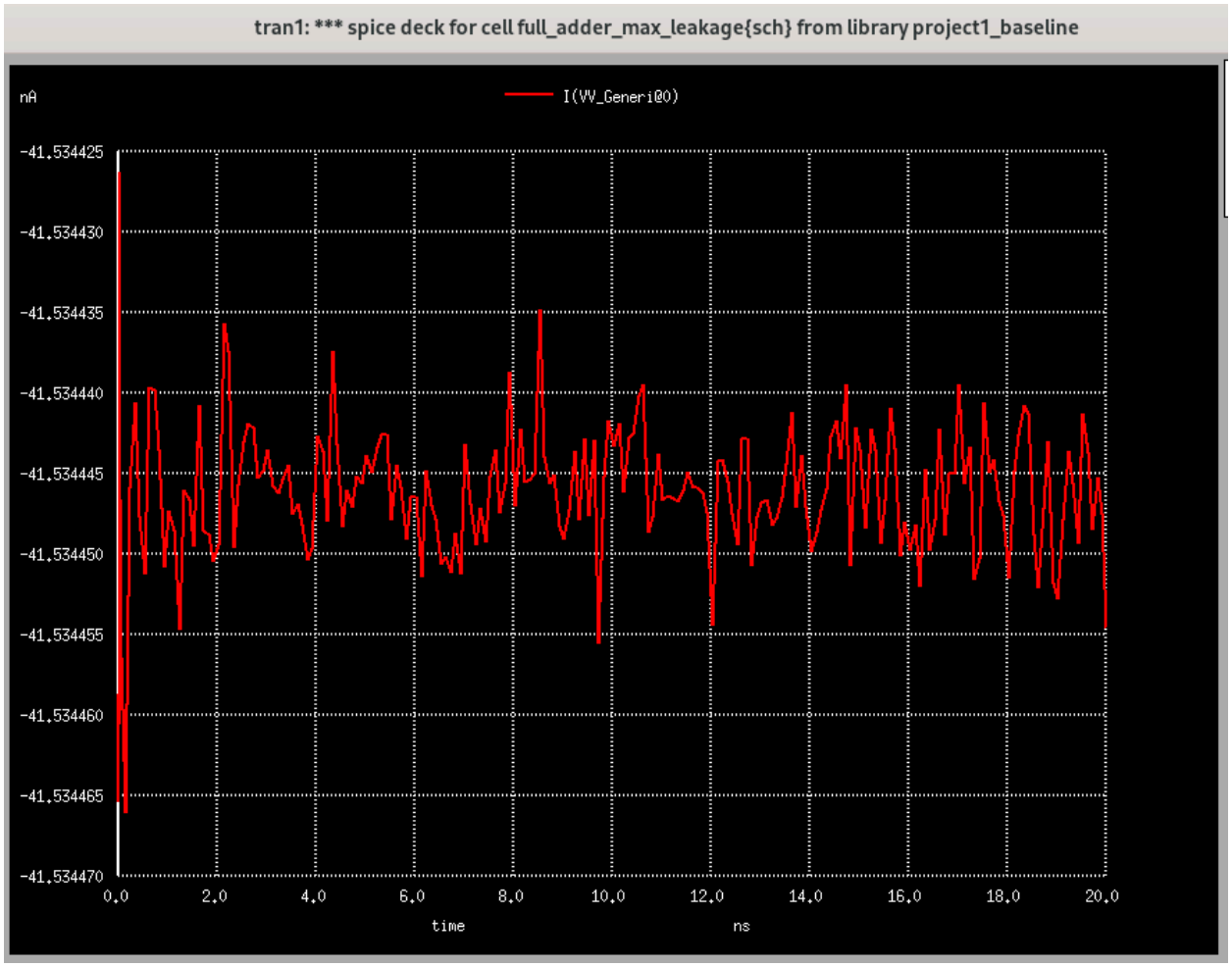



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -9.11745e-18 from=  1.00000e-09 to=  1.21800e-09
```

Leakage Energy = 7.29396 * 10^-18 J

Case 7: A = B = 0, Cin = 0.8

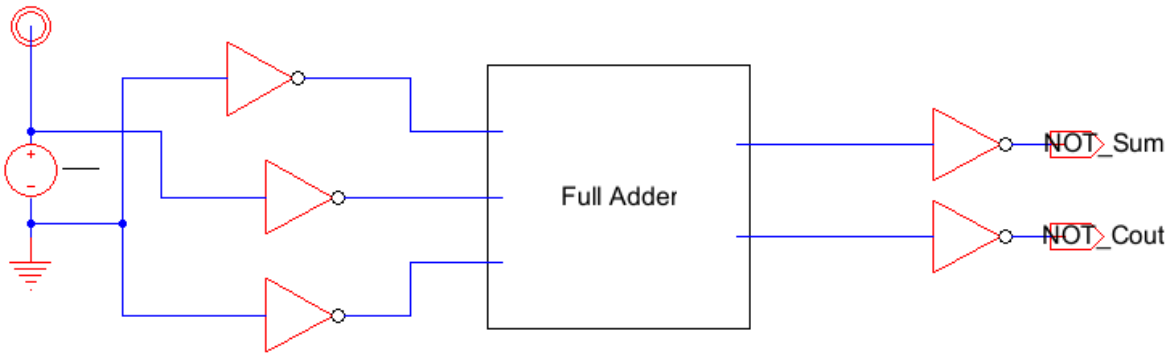


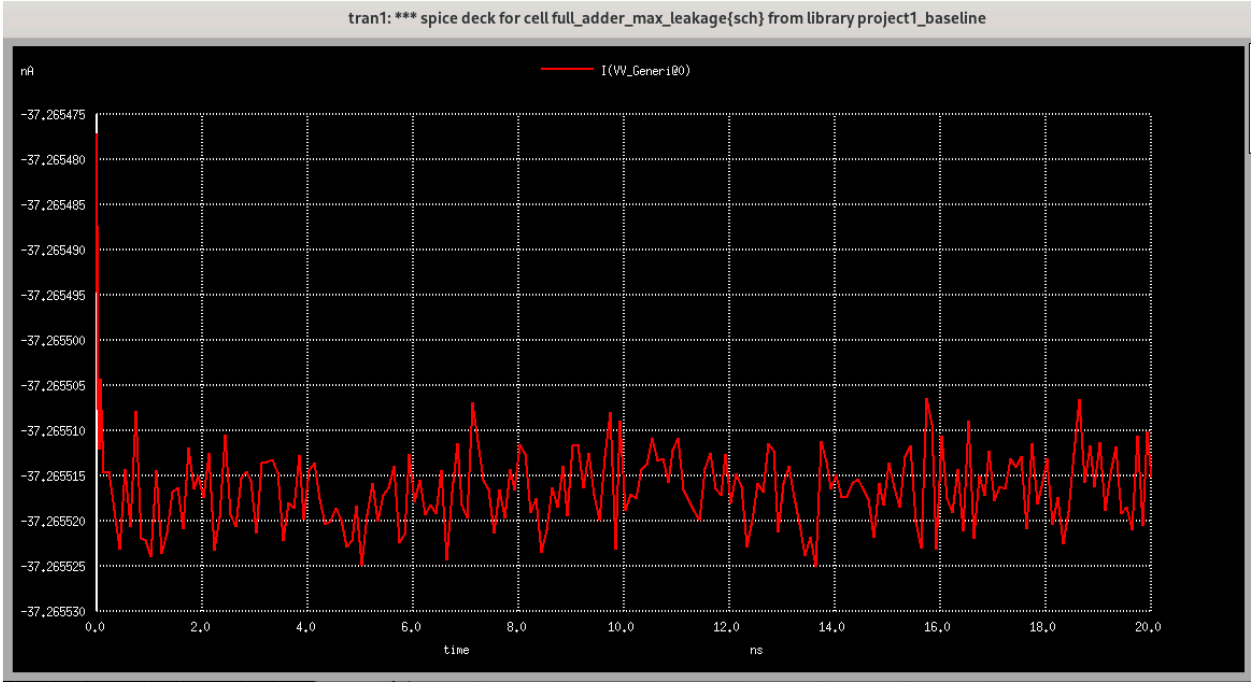



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -9.05451e-18 from=  1.00000e-09 to=  1.21800e-09
```

Leakage Energy = 7.243608 * 10^-18 J

Case 8: A = 0.8, B = 0, Cin = 0.8



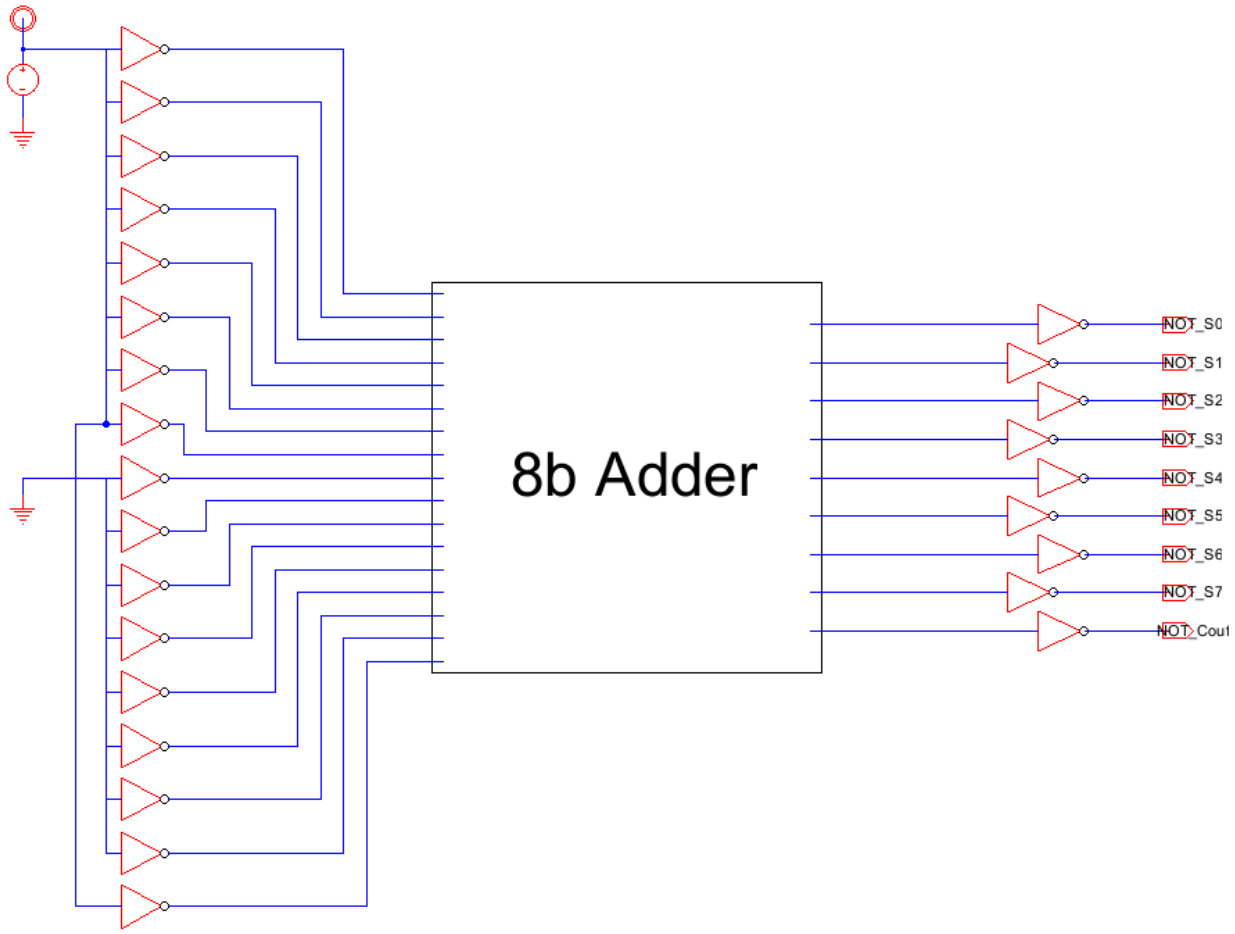



```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -8.12388e-18 from=  1.00000e-09 to=  1.21800e-09
```

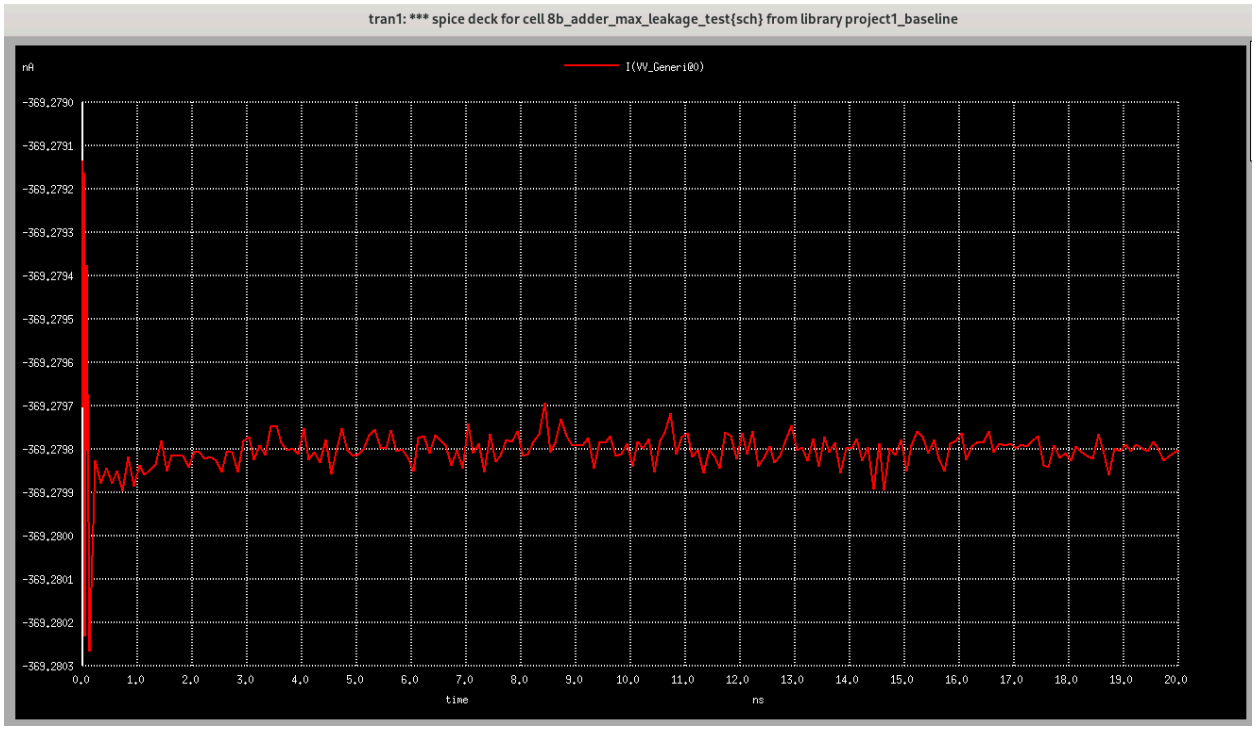
Leakage Energy = 6.499104 * 10^-18 J

We can see that we achieved maximum leakage energy for at case 4 (leakage energy = 7.621432 * 10^-18 J). To extrapolate the maximum leakage energy condition from a single full adder to the entire 8-bit ripple carry adder (RCA), I replicated the input combination that produced the highest leakage in the isolated test: A = 0, B = 1, Cin = 0. Since leakage energy arises from static input conditions and is determined primarily by the transistor states within each bit-slice, it is reasonable to apply the same high-leakage configuration across all eight full adder stages. This results in a full RCA input of A = 00000000 and B = 11111111, which ensures

that every full adder receives the same input conditions that previously maximized subthreshold and gate leakage. Importantly, because this test is entirely static—with no switching or carry propagation—Cin can be uniformly held at 0 for all eight stages without affecting correctness. This simplifies the test setup significantly and guarantees consistency across the adder, allowing for accurate measurement of worst-case leakage energy.



All eight A inputs (A0–A7) and the global Cin were configured to logic 0, while all B inputs (B0–B7) were set to logic 1. In the schematic, this was implemented by connecting the A inputs and Cin through inverters to a constant 0.8 V DC source (so they resolve to logic 0 at the adder inputs), and the B inputs through inverters to GND (so they resolve to logic 1 at the adder inputs). This configuration maintains the highest-leakage condition consistently across each bit slice while preserving Cin = 0 for the RCA as required. Since none of the inputs switch during this test, the energy consumed is solely due to leakage, and the integration of supply current over one measured adder delay period (218 ps) provides the total leakage energy.

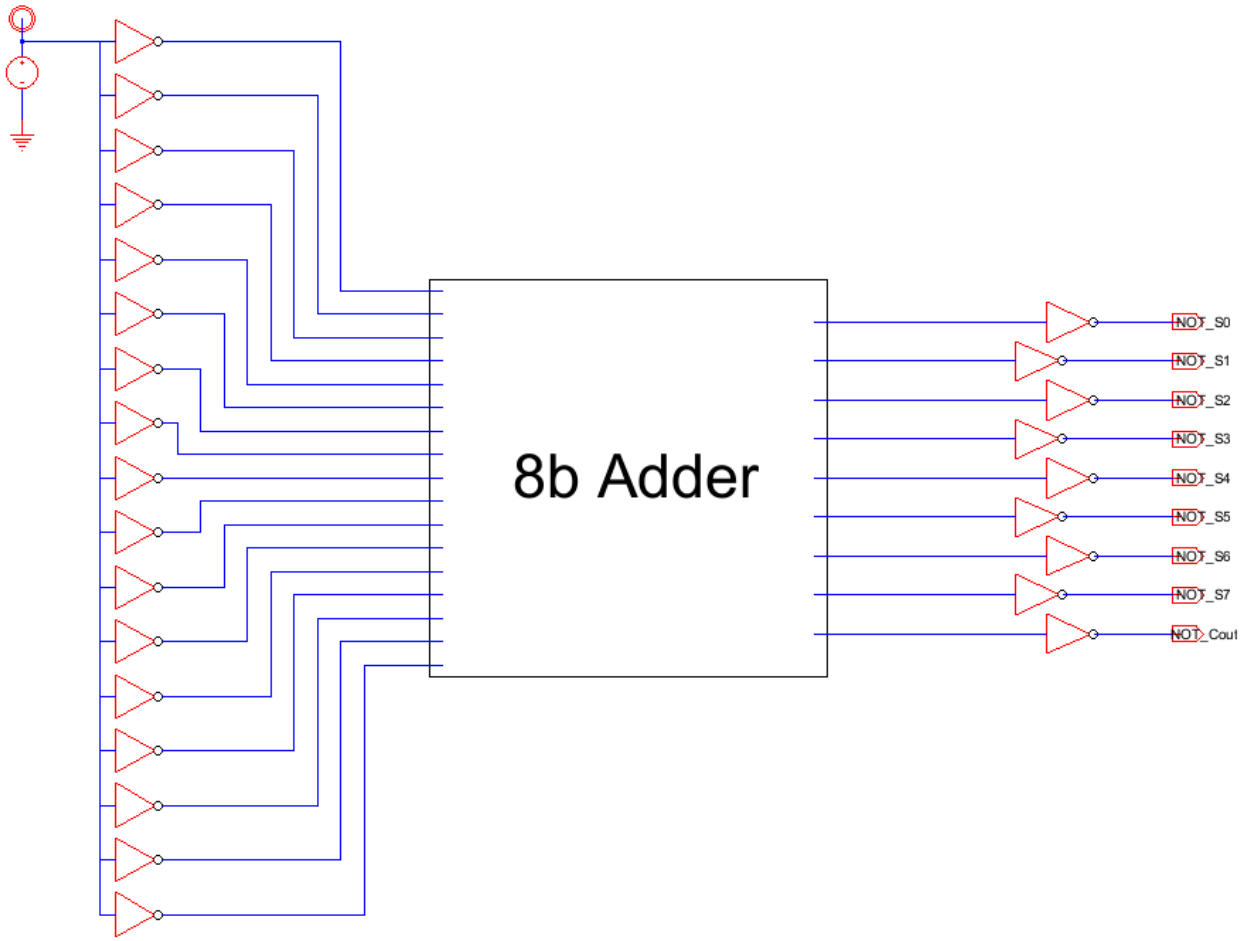


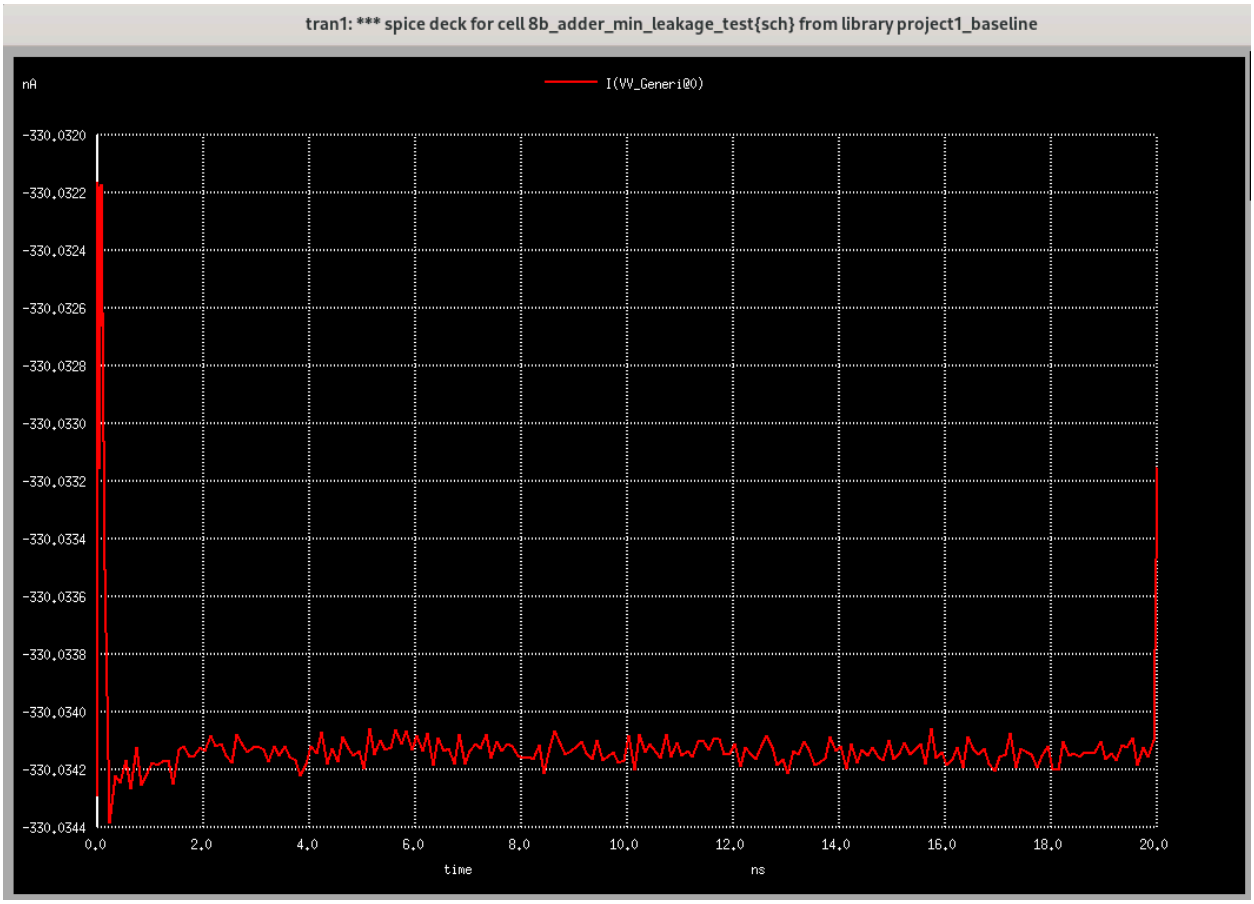

```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -8.05030e-17 from=  1.00000e-09 to=  1.21800e-09
```

Maximum Leakage Energy = **6.44024 * 10^-17 J**

Minimum Leakage Energy Case

Although Case 8 (A = 1, B = 0, Cin = 1) yielded the lowest leakage energy (6.499104 * 10^-18 J) in the single full adder simulations, this configuration could not be applied directly to the full ripple carry adder (RCA) due to structural constraints. Specifically, during leakage energy testing, all inputs must remain static and the initial carry-in (Cin) must be held at logic 0 for the entire adder, consistent with the RCA architecture and the absence of dynamic carry propagation. Since Case 8 required Cin = 1, it was incompatible with this requirement. Instead, the lowest-leakage configuration among the valid Cin = 0 cases - Case 1 (A = 0, B = 0, Cin = 0) with leakage energy of 6.821536 * 10^-18 - was selected for the full RCA. This case preserves the correct RCA structure while still minimizing leakage, and was therefore used to determine the minimum leakage energy.






```
ngspice 117 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.218ns
qleak                =  -7.19475e-17 from=  1.00000e-09 to=  1.21800e-09
```

Minimum Leakage Energy = **5.7558 * 10^-17 J**

Area

The purpose of the area metric is to evaluate the intrinsic complexity and resource usage of the ripple carry adder implementation. Since drive and load inverters were introduced solely for evaluation purposes (e.g., to create realistic drive strength or to provide standardized load capacitance), they are not representative of the adder's actual logic implementation and are thus excluded from the area calculation.

To evaluate the area of the baseline CMOS 8-bit ripple carry adder (RCA), I sum the total transistor widths for the core design, excluding any load or drive inverters used solely for simulation purposes. Each full adder in the RCA is composed of 2 XOR2 gates, 2 NAND2 gates, 1 NOR2 gate, and 3 inverters. Each XOR2 gate is itself built from 1 NOR2 gate, 2 NAND2 gates, and 2 inverters. Therefore, the two XOR2 gates within each full adder contribute a total of 2 NOR2s, 4 NAND2s, and 4 inverters. Including the remaining 2 NAND2s, 1 NOR2, and 3 inverters from the rest of the full adder logic, the full gate breakdown per full adder becomes 3 NOR2s, 6 NAND2s, and 7 inverters. Since each NOR2 and NAND2 gate consists of 4 transistors, and each inverter consists of 2 transistors (with W = L = 1), this yields $3\times4+6\times4=36$ transistors from the NOR2s and NAND2s, and $7\times2=14$ transistors from the inverters, for a total of 50 transistors per full adder. Multiplying by the 8 full adder stages in the RCA, I find that the complete 8-bit RCA uses $50\times8=400$ transistors in total.

In the 22nm HP PTM process, each transistor has a width and length of 22 nm, resulting in an area of $22\,\mathrm{nm}\times22\,\mathrm{nm}=484\,\mathrm{nm}^2$ per transistor. With a total of 400 transistors used in the 8-bit ripple carry adder design, the overall area is calculated as $400\times484\,\mathrm{nm}^2=193600\,\mathrm{nm}^2$. Converting to square micrometers, the total area amounts to

$$193600\,\mathrm{nm}^2=0.1936\,\mu\mathrm{m}^2.$$

Summary Table of Baseline Metrics

Metric	Value	Units	Description
Delay	218	ps	Worst-case delay from input (B_0) to final Carry-Out across 8 full adders
Max Switching Energy	4.6775×10^{-15}	J (4.6775 fJ)	Measured for case with full carry propagation through all stages
Average Switching Energy	2.3444×10^{-15}	J (2.3444 fJ)	Measured for internal carry and only one sum output toggling
Max Leakage Energy	6.4402×10^{-17}	J (64.4 aJ)	Based on static input case A = 0, B = 1, Cin = 0 replicated across RCA
Min Leakage Energy	5.7558×10^{-17}	J (57.56 aJ)	Based on static input case A = 0, B = 0, Cin = 0 replicated across RCA
Total Transistor Count	400	transistors	50 transistors per full adder × 8 full adders
Area	0.1936	μm^2	Based on 400 transistors with 22nm × 22nm dimensions in HP 22nm PTM process

Delay Optimized Design

Schematics

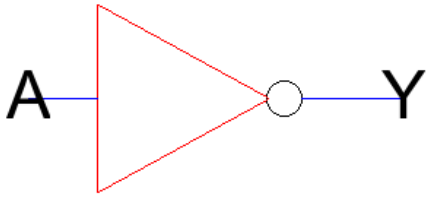
The delay-optimized design is constructed entirely using static CMOS logic, but with two key modifications compared to the baseline. First, all transistors in the design—both PMOS and NMOS—are sized with a width of W = 1.22 (in units of the minimum feature size), based on tau-model optimization to reduce propagation delay. Second, the XOR2 gate has been redesigned using a logic expression derived from a K-map-minimized form, allowing for more efficient implementation. The circuit is powered by a 1 V supply voltage to further reduce delay, as permitted in the delay-optimized design constraints. The full adder bit-slice is composed of inverter (NOT), 2-input NAND, 2-input NOR, and the new K-map-reduced XOR2 gate.

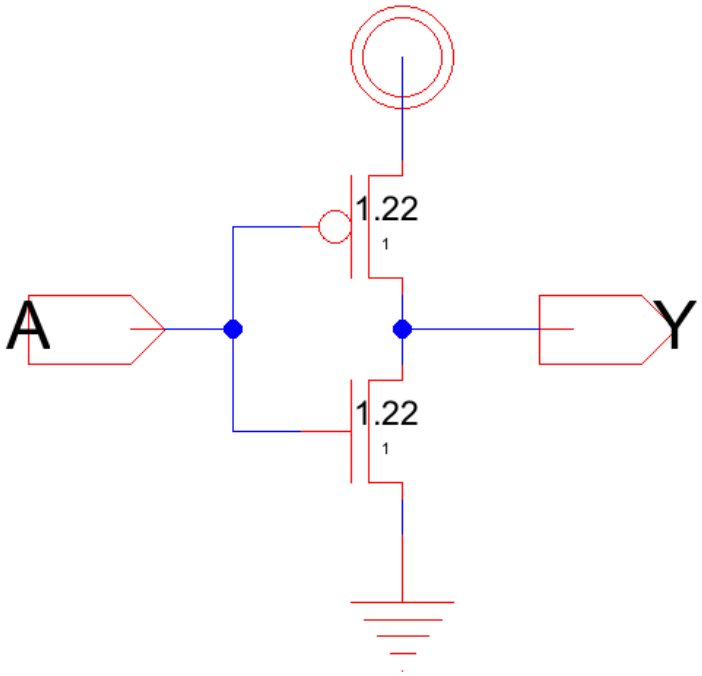
Gate-Level Schematics

The fundamental gates required to implement the full adder logic are the inverter (NOT), 2-input NAND, 2-input NOR, and 2-input modified XOR gates. Each gate is implemented using transistors with channel length L=22nm and width W=22 * 1.22 nm, consistent with the minimum design rules of the 22nm HP process.

Inverter (NOT)

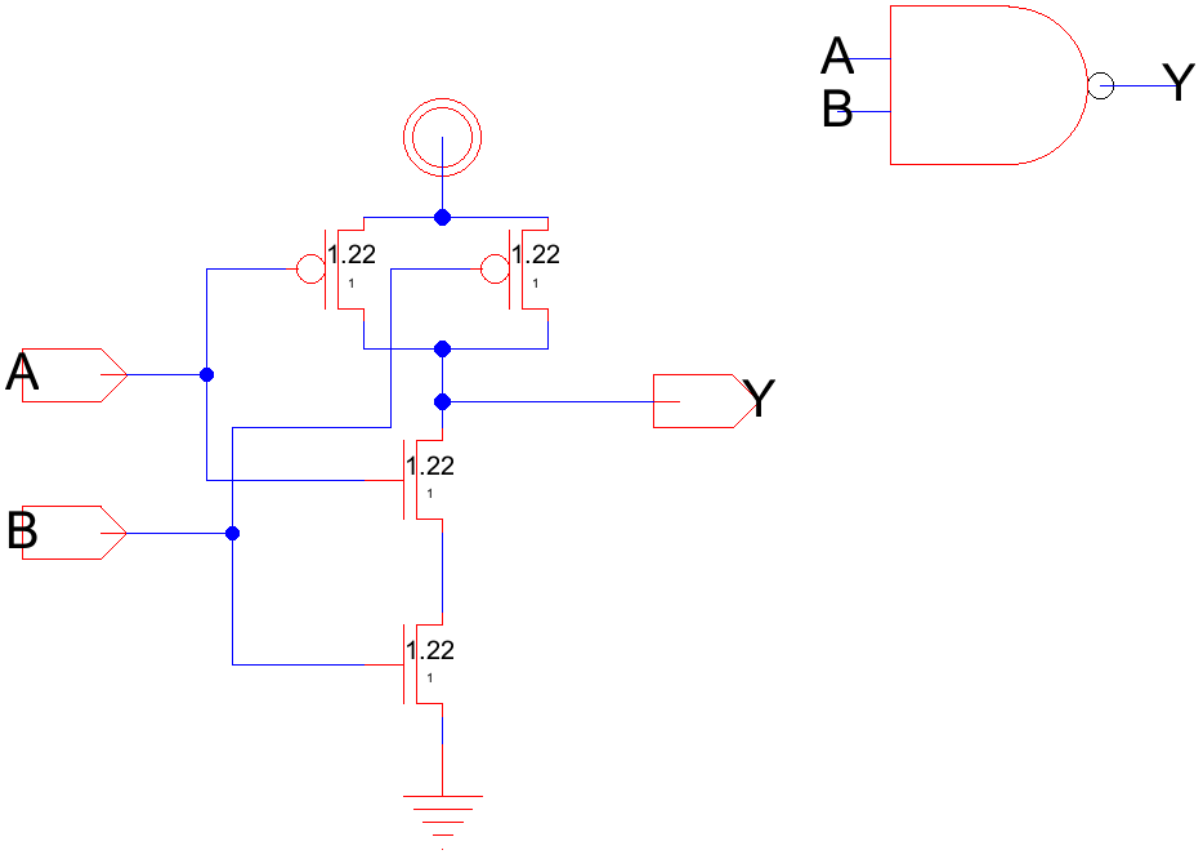
The inverter consists of a single PMOS transistor in the pull-up network and an NMOS transistor in the pull-down network, with the input tied to both gates and the output taken from the common drain node.





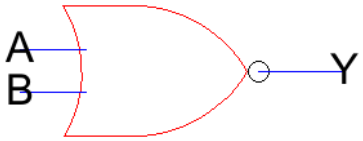
2-Input NAND (NAND2)

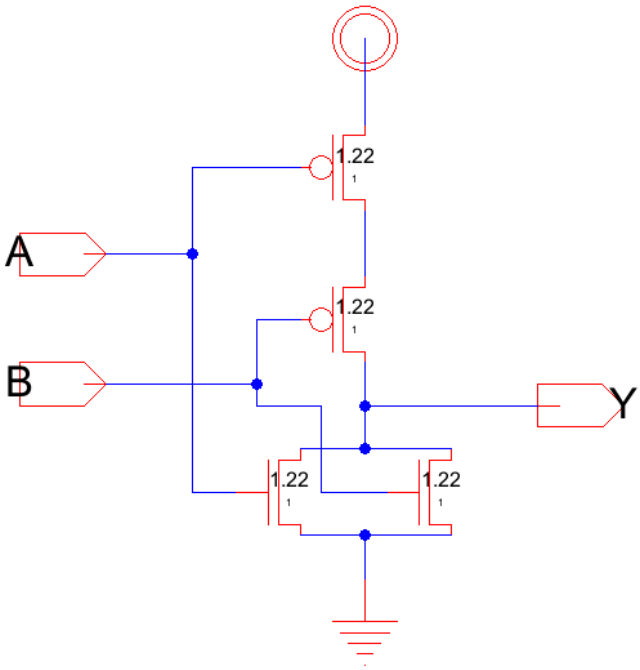
The NAND gate uses two NMOS transistors in series in the pull-down network and two PMOS transistors in parallel in the pull-up network. This structure ensures that the output only goes low when both inputs are high.



2-Input NOR (NOR2)

The NOR gate is constructed with two NMOS transistors in parallel in the pull-down path and two PMOS transistors in series in the pull-up path. The output is high only when both inputs are low.



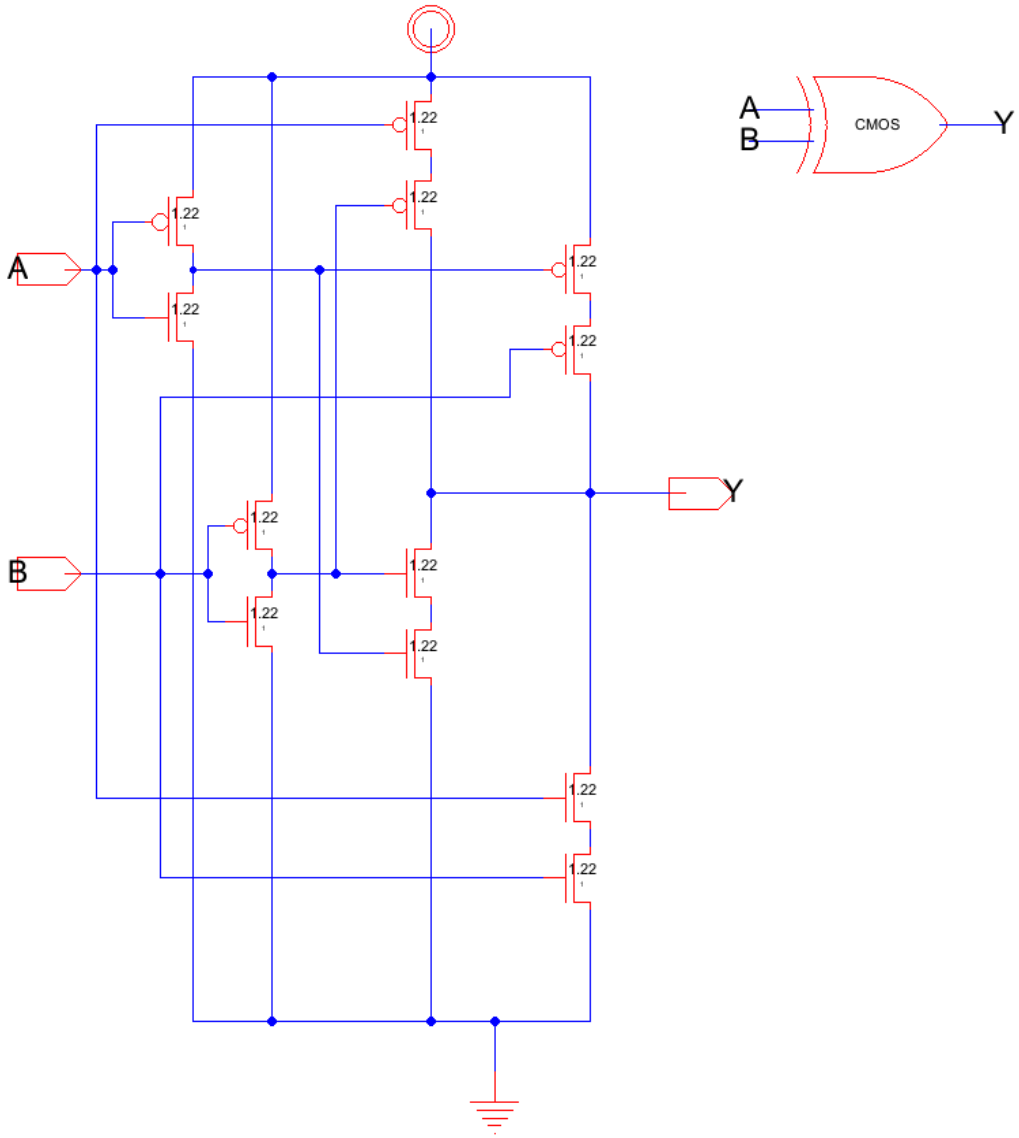


2-Input XOR (XOR2)

The XOR2 gate in this implementation is optimized based on a minimized sum-of-products (SOP) expression from its K-map. The expression:

$$A \oplus B = \overline{A}B + A\overline{B}$$

is implemented directly using complementary CMOS logic rather than through composition of NAND, NOR, and inverter gates. This version uses 12 transistors, forming two transmission branches that combine the partial product terms and an output stage that restores signal strength and logic level. All transistors in the XOR are uniformly sized at W = 1.22 to balance delay across series and parallel paths. This logic structure reduces gate depth and critical path resistance compared to the baseline XOR, contributing significantly to the reduced delay of the overall design.



Full Adder Bit-Slice Schematic

The full adder computes two outputs, Sum and $\mathrm{CarryOut}$, from inputs A, B, and Cin. The sum output is 1 when an odd number of the inputs are 1, leading to the expression:

$$\mathrm{Sum} = A \oplus B \oplus \mathrm{Cin}$$

This is implemented in two stages: first compute $S_1 = A \oplus B$, then $\mathrm{Sum} = S_1 \oplus \mathrm{Cin}$. Each XOR is realized using the identity $X \oplus Y = (X + Y)(\overline{XY})$, which maps to a combination of OR, AND, and NOT gates.

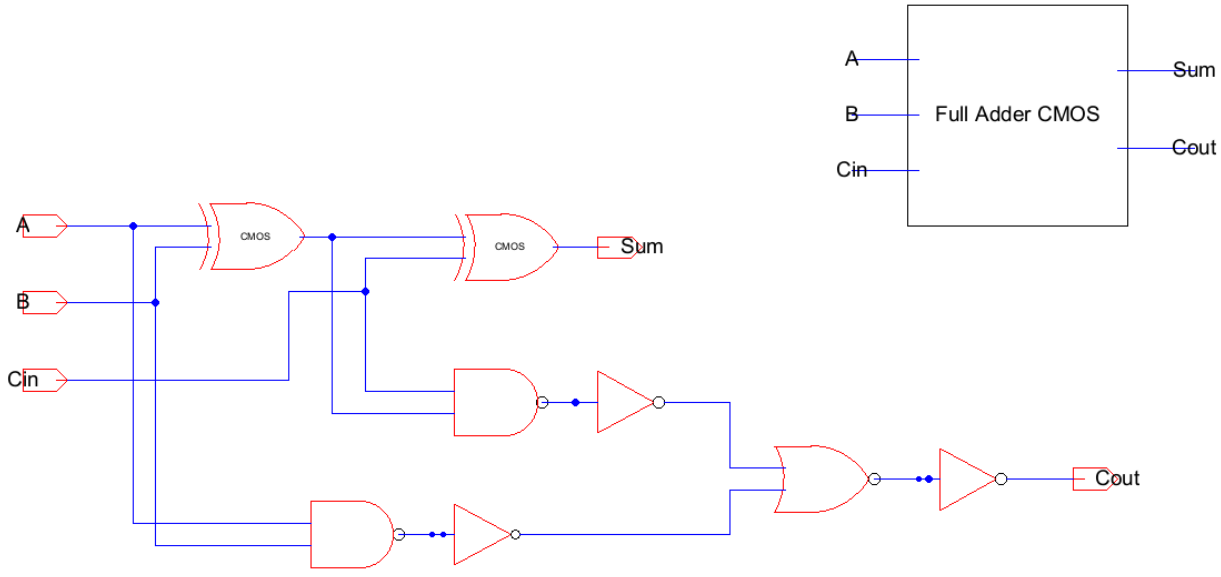
The carry output is 1 when at least two of the inputs are 1. The canonical expression is:

$$\mathrm{CarryOut} = AB + A\mathrm{Cin} + B\mathrm{Cin}$$

This is algebraically reduced to:

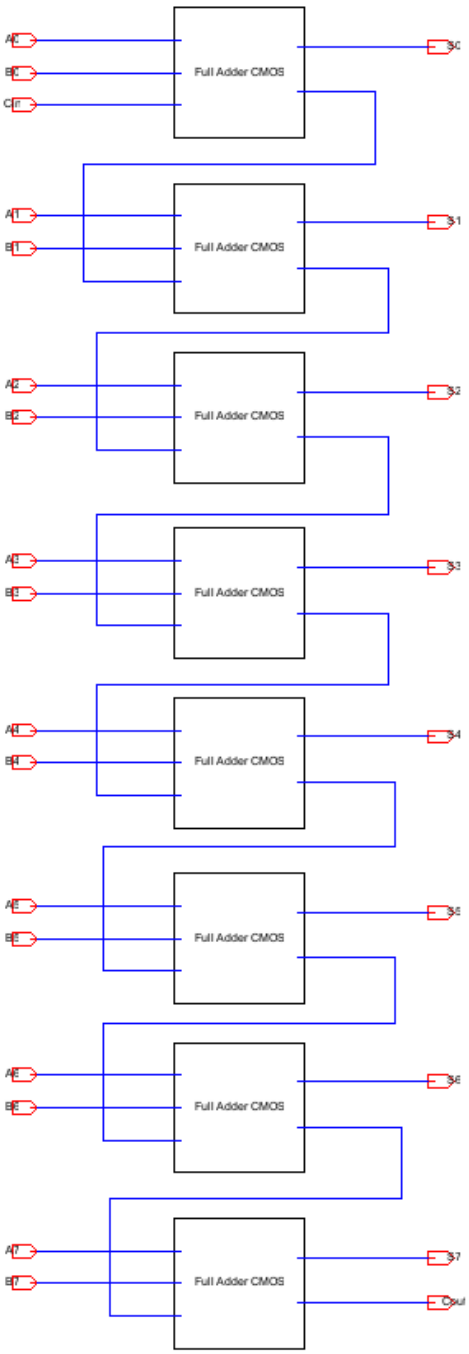
$$\mathrm{CarryOut} = AB + (A \oplus B)\mathrm{Cin}$$

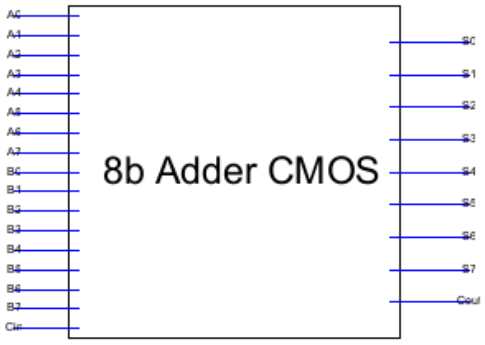
This form is more efficient, as the term $A \oplus B$ is already computed in the sum path. Thus, both outputs are derived from basic Boolean identities using only two-input logic gates and reused intermediate signals.



8-bit Ripple Carry Adder Schematic

An 8-bit ripple-carry adder connects eight 1-bit full adder cells in series. Each cell computes $S_i = A_i \oplus B_i \oplus C_i$ and $C_{i+1} = A_i B_i + (A_i \oplus B_i) C_i$, with C_0 as the initial carry-in. The carry-out of each stage feeds directly into the next stage's carry-in. The increased supply voltage of 1 V and wider transistors work in tandem to improve drive strength, reduce internal node delays, and improve the overall delay performance of the RCA.





Logic & Performance Rationale

This delay-optimized design strategically adopts uniform static CMOS logic, but with key optimizations focused on reducing propagation delay across the ripple-carry path. The most significant change from the baseline is the substitution of the original logic-gate-based XOR2 implementation with a minimized form derived from the K-map of the XOR truth table. The truth table for a 2-input XOR gate is:

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

From this, the corresponding Karnaugh map is:

	$B=0$	$B=1$
$A=0$	0	1
$A=1$	1	0

This map yields the minimized sum-of-products (SOP) expression:

$$A \oplus B = \overline{A}B + A\overline{B}$$

This expression allows a more direct and efficient implementation using only 12 transistors in static CMOS. The two product terms are implemented using parallel branches of series-connected NMOS and PMOS transistors, each evaluating either $\overline{A}B$ or $A\overline{B}$, and merged at the output. The result is a balanced pull-up and pull-down structure that restores full logic swing, avoiding the complexity and delay overhead of the gate-level XOR built from NAND, NOR, and inverters used in the baseline design.

To further optimize delay, I increased all transistor widths uniformly to $W = 1.22$, as calculated from the tau model to minimize gate delay. This width balances the trade-off between resistance and capacitance in the gate, achieving faster switching times across the critical path. Additionally, the supply voltage was raised from 0.8 V to 1.0 V to exploit increased overdrive and reduce intrinsic gate delays, as permitted in the delay-optimized design specification.

I considered alternative logic styles such as ratioed logic and pass-transistor logic but ultimately chose not to use them. Ratioed logic introduces output voltage degradation and sizing constraints due to its dependence on asymmetric pull-up and pull-down paths, which can compromise noise margins and require careful transistor sizing to maintain logic levels. Pass-transistor logic offers a reduction in transistor count but suffers from voltage degradation due to threshold drops and requires additional buffers or level restorers to maintain full logic

swing, especially in multi-stage circuits like the ripple-carry adder. These drawbacks would have added complexity and potentially counteracted delay improvements.

By adhering to static CMOS and incorporating minimized logic expressions, analytical transistor sizing, and increased supply voltage, this design achieves significant delay reduction while preserving robustness, signal integrity, and full swing operation—making it a strong candidate for high-performance arithmetic logic.

Delay Analysis

To estimate the delay of the optimized full adder design, I applied the τ-model, which expresses delay as the product of resistance and capacitance at each stage. For each gate, the resistance is modeled as $\frac{R_0}{W}$ and the capacitance as WC_0, where W is the transistor width, and R_0, C_0 are the unit resistance and capacitance for minimum-size transistors. The critical path from input A to the carry-out C_{out} traverses the XOR2, NAND2, an inverter, NOR2, another inverter, and finally the output inverter. The total delay along this path is expressed as:

$$D = \frac{R_0}{1} \cdot 4WC_0 + \frac{2R_0}{W} \cdot 2WC_0 + \frac{2R_0}{W} \cdot 2WC_0 + \frac{R_0}{W} \cdot 2WC_0 + \frac{2R_0}{W} \cdot 2WC_0 + \frac{R_0}{W} \cdot 6C_0$$

$$D = 4W\tau + 14\tau + \frac{6}{W}\tau$$

To minimize this total delay, I differentiated D with respect to W and set the derivative to zero:

$$\frac{dD}{dW} = 4\tau - \frac{6\tau}{W^2} = 0$$

Solving this equation yields:

$$4\tau = \frac{6\tau}{W^2} \Rightarrow W^2 = \frac{3}{2} \Rightarrow W \approx 1.22$$

Thus, I chose a transistor width of approximately 1.22 for all transistors in the delay-optimized CMOS design to minimize delay according to the -model. This balances the trade-off between increasing gate drive strength and limiting capacitive loading along the critical path.

Plugging this into our equation for delay gives us approximately

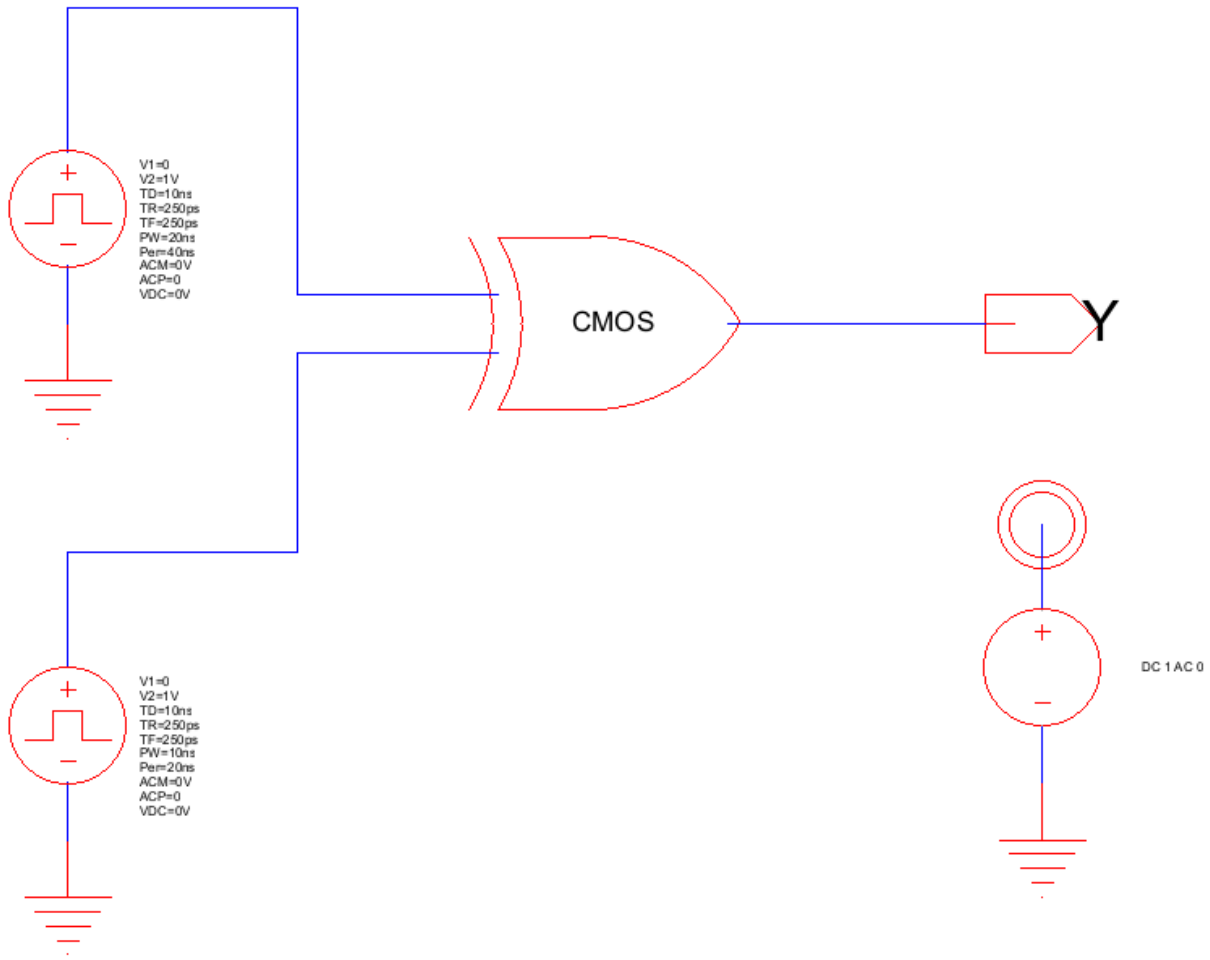
$$4(1.22) + 14 + \frac{6}{1.22}| \qquad = 23.79803279$$

= 23.80 tau for or estimate of delay.

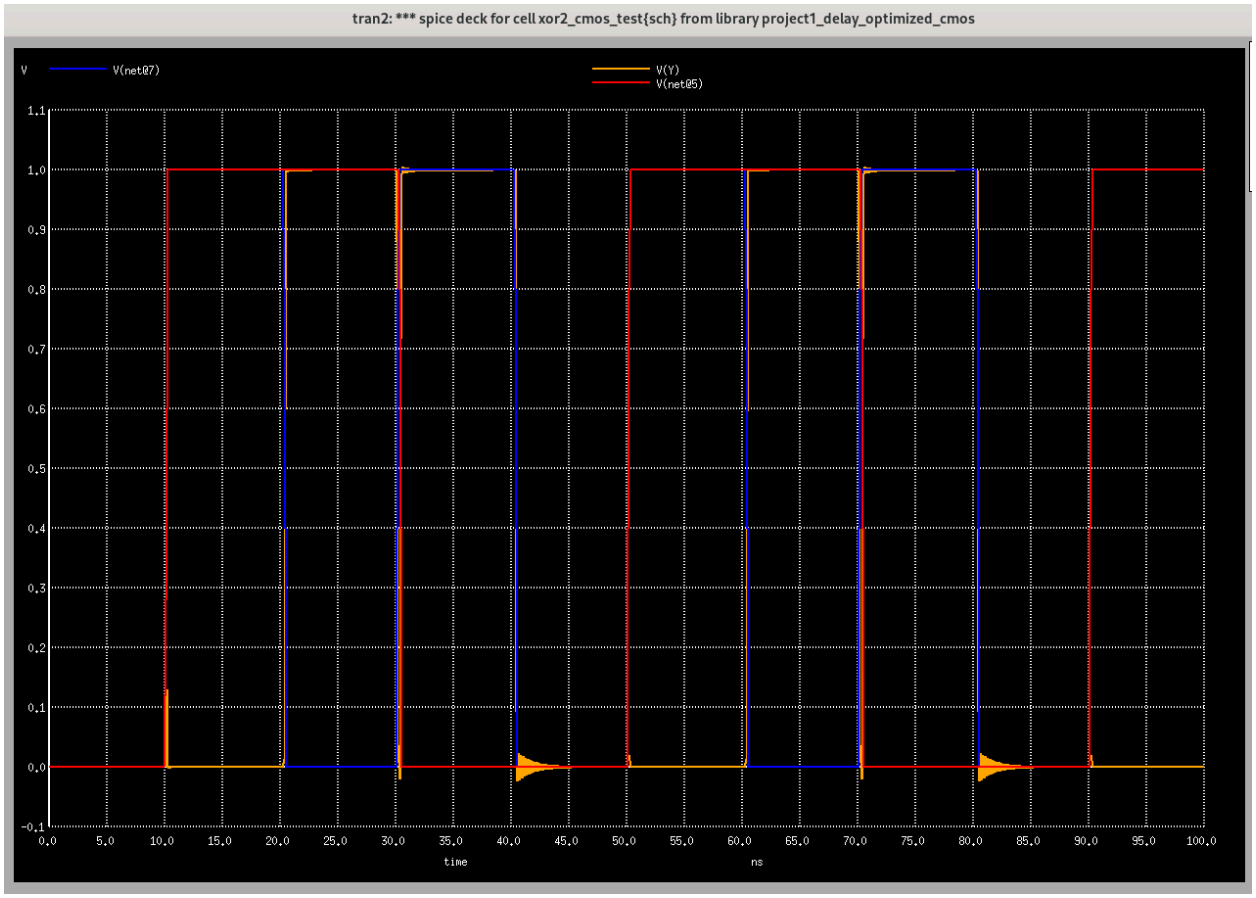
Functional Verification

To verify the correctness of the delay-optimized design, I tested the functionality of the newly implemented XOR gate, which was the only structural change made compared to the baseline design. All other modifications—such as increased supply voltage and uniform transistor sizing—were process-level optimizations that did not alter the logic behavior of the circuit. Since the full adder and ripple-carry architecture remained logically identical, validating the correctness of the optimized XOR gate was sufficient to ensure that the overall functional behavior of the adder remained unchanged.

XOR Verification



I set up the given circuit with two VPulse sources at the inputs and plot the entire response of the gate as shown below:

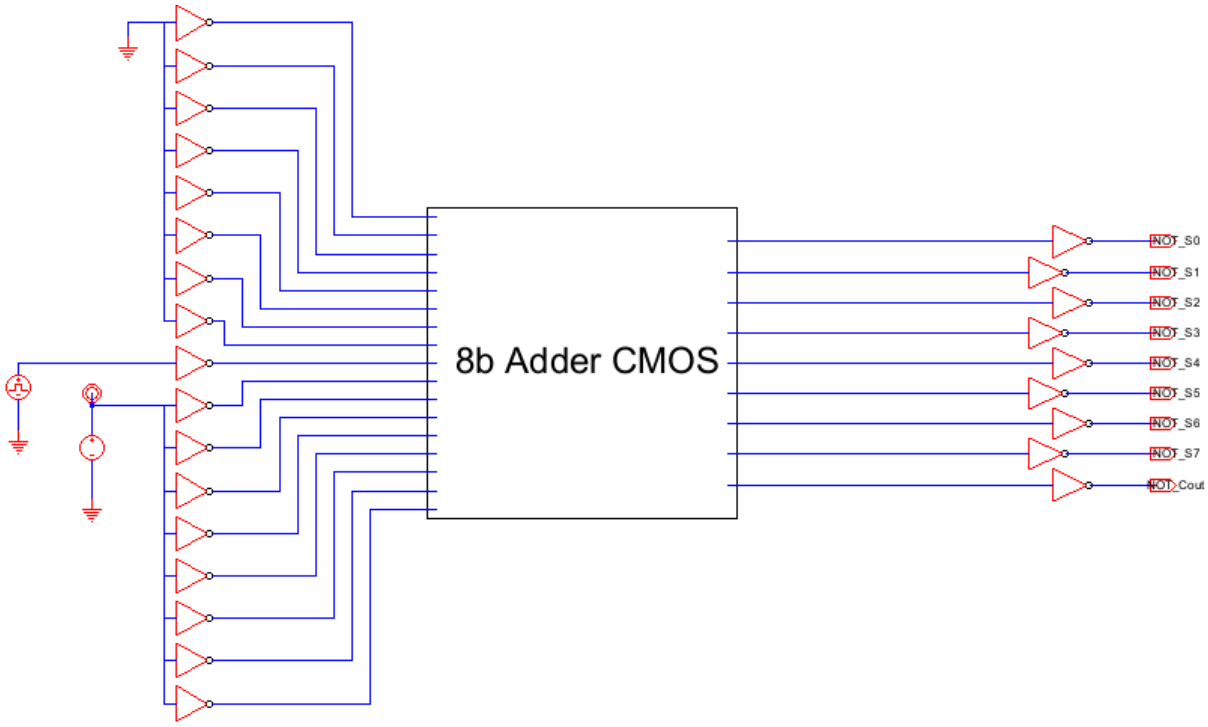


As we can see, the gate only takes on a logic high only when one of the inputs is at a logic high and the other is at a logic low. This is the exact functionality we sought out, and this verification confirms the integrity of our design.

Evaluation Metrics

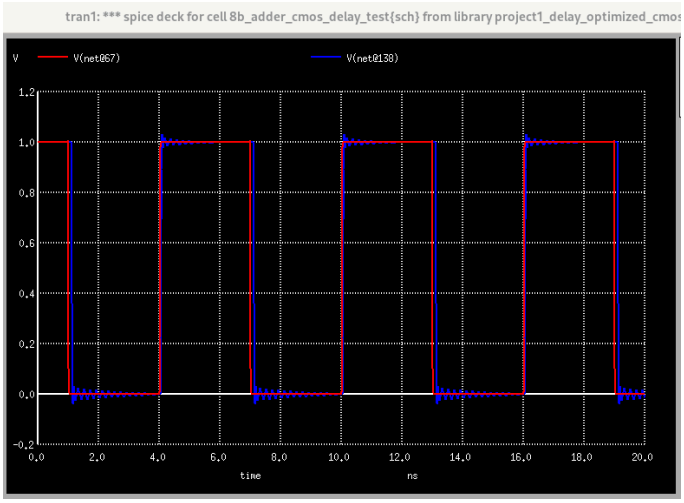
Delay

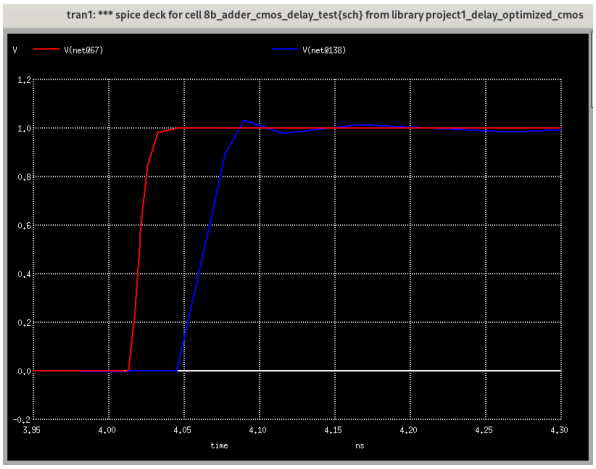
To evaluate and compare delay performance, I measured the propagation delay through the full 8-bit ripple carry adder. I constructed the complete 8-bit adder using the CMOS delay-optimized implementation and measured delay from a rising or falling transition on input B0 to the resulting transition on the final output—either S7 or Cout. I held A and Cin constant and applied a single rising or falling pulse at B0, driving all inputs with minimum-size inverters and loading each output with width-3 inverters to reflect realistic conditions. I captured both rising and falling propagation delays from B0 to S7 and from B0 to Cout, for a total of four measurements. The maximum delay among these four was taken as the worst-case delay for that implementation. I applied this consistent measurement strategy across the baseline and delay optimised setup, which would then be used for further evaluation of energy and area.



B = net@167, S7 = net@146, Cout = net@138

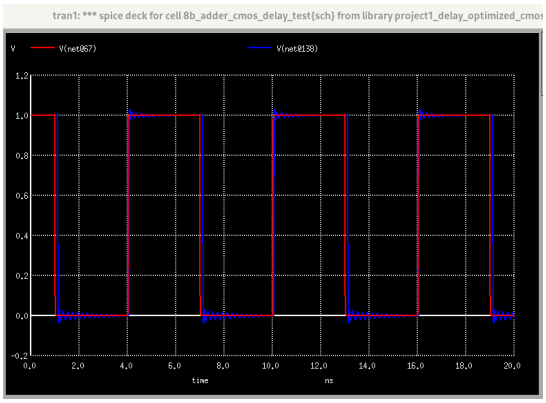
Propagation Delay Rise B0 -> Cout

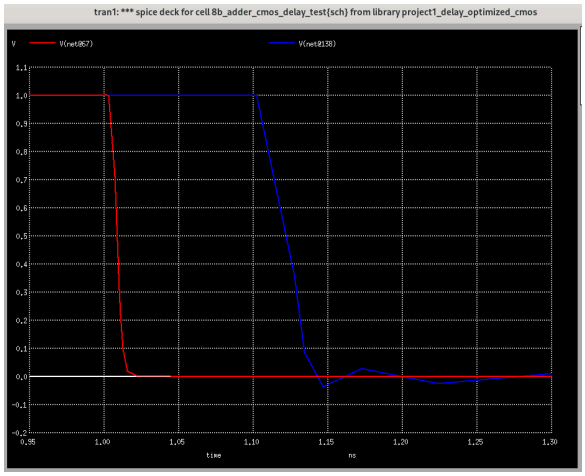




dx = 4.35644e-11, dy = 0

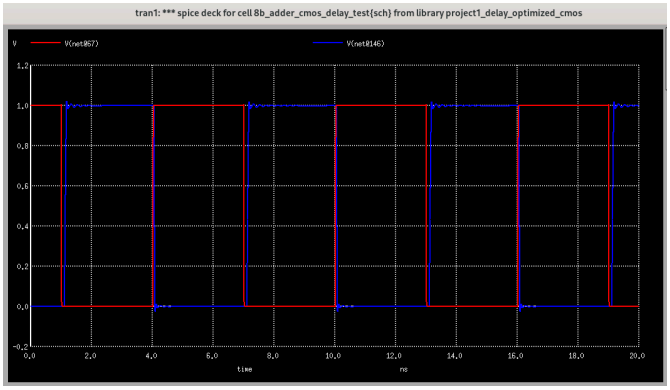
Propagation Delay Fall B0 -> Cout

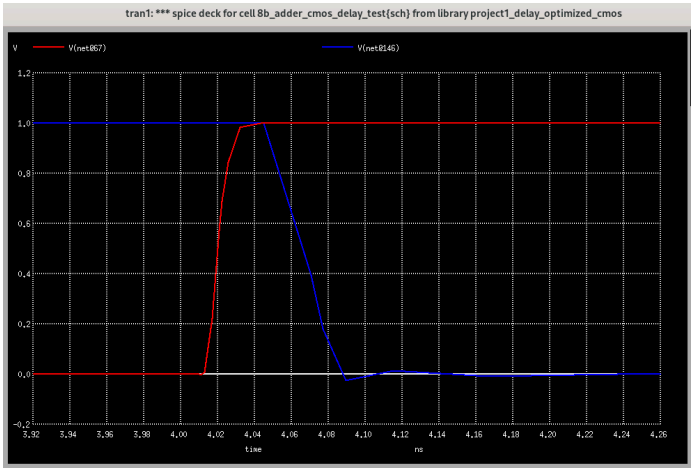




dx = 8.13187e-11,

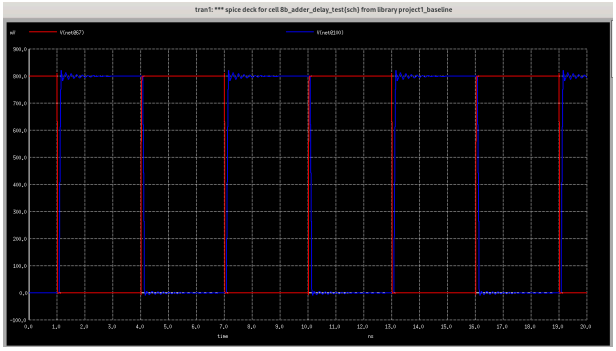
Propagation Delay Rise B0 -> Cout

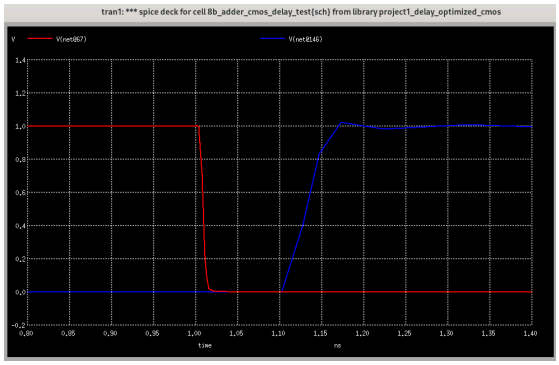




dx = 5.07463e-11,

Propagation Delay Fall B0 -> Cout





$$dx = 8.5e\text{-}11,$$

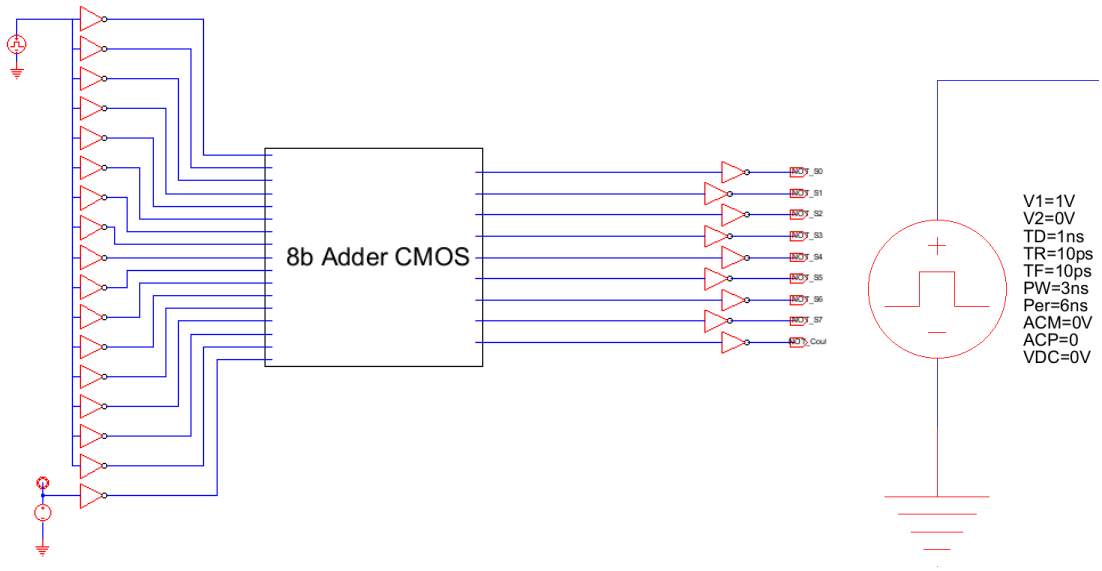
$\frac{4.35644}{0.69} \cdot 10^{-11}$	$= 6.31368116 \times 10^{-11}$
$\frac{8.13187}{0.69} \cdot 10^{-11}$	$= 1.17853188 \times 10^{-10}$
$\frac{5.07463}{0.69} \cdot 10^{-11}$	$= 7.35453623 \times 10^{-11}$
$\frac{8.5}{0.69} \cdot 10^{-11}$	$= 1.23188406 \times 10^{-10}$

Dividing each of these dx values 0.69, we find the the worst case propagation delay is seen in the fall time propagation delay fall from B0 to Cout (dx = 1.50438e-10s), with a delay of **123ps**. This is the delay I will use for my leakage energy calculations and the value I will try to optimize.

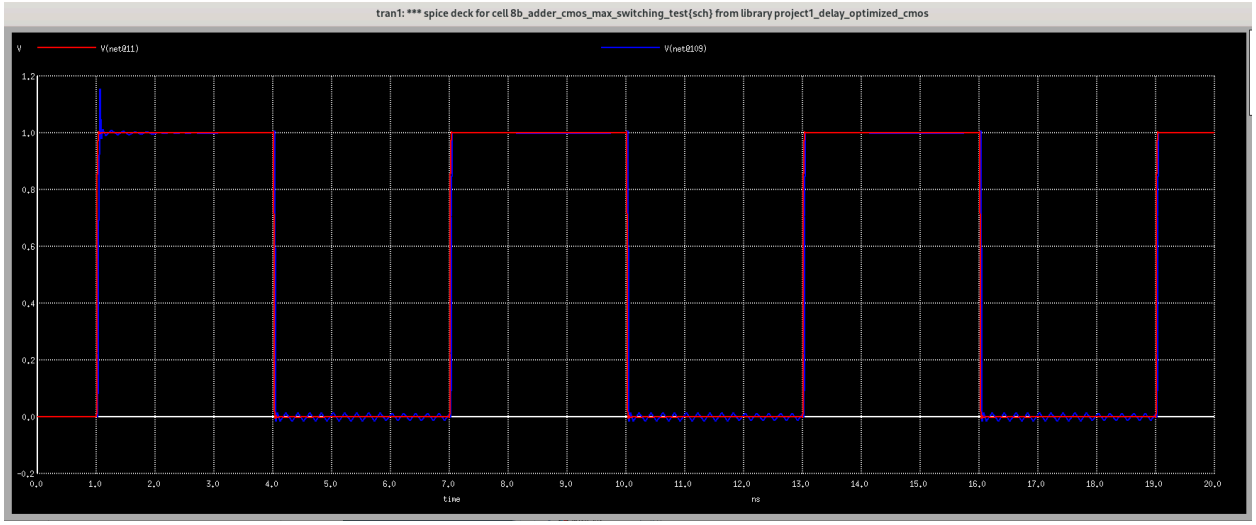
Active Energy

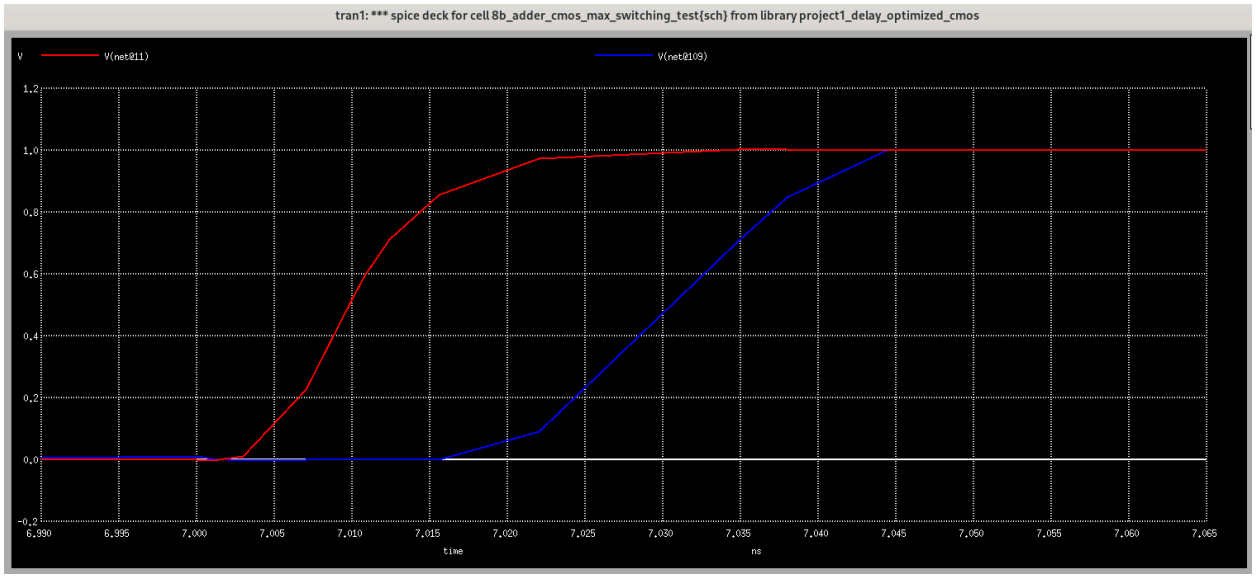
Maximum Switching Energy Case

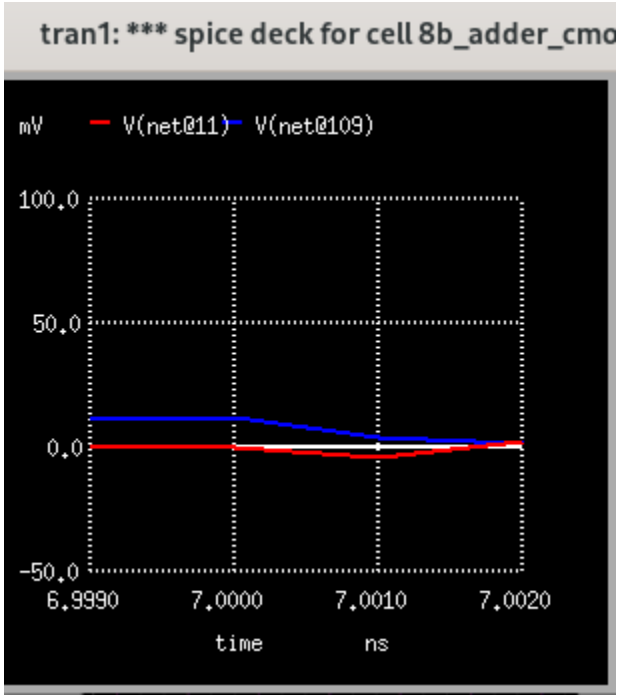
Using the same reasoning as for the baseline case, we can test the maximum switching energy case as the case where all inputs switch from 0 to 1, with Cin kept at ground.

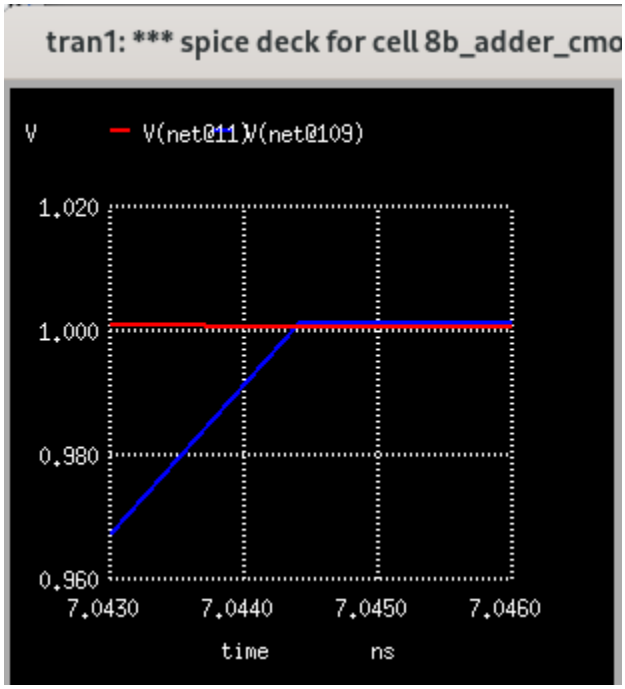


A0 is net@11 and my Cout is net@109









x0 = 7e-09, y0 = 2.77556e-17

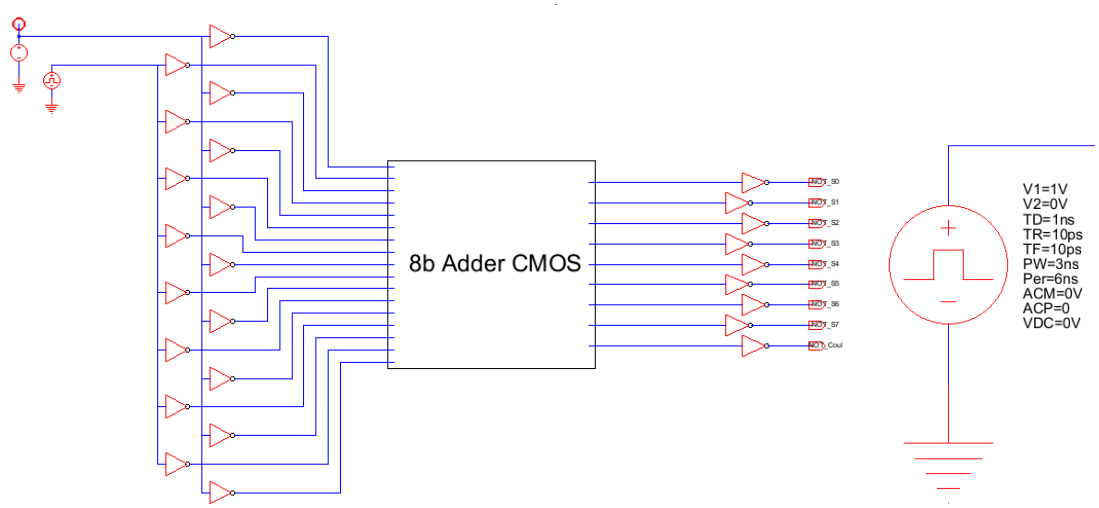
x0 = 7.04452e-09, y0 = 1.00267

```
ngspice 120 -> meas tran qswitchsmos integ I(VV_Generi@0) from=7
ns to=7.04452ns
qswitchsmos          =  -7.17441e-15 from=  7.00000e-09 to=  7.04
452e-09
```

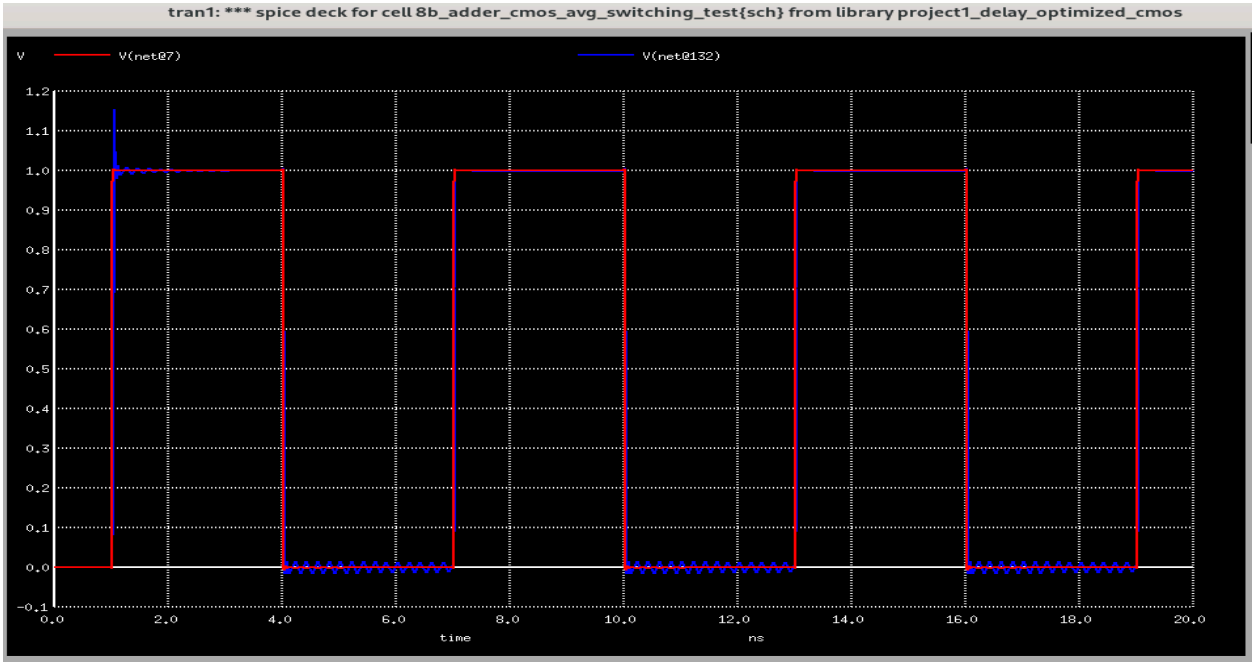
This implies that the maximum switching energy is **7.17 fJ,** as our VDD = 1V, and as we continue with the same calculation methods as in the baseline approach.

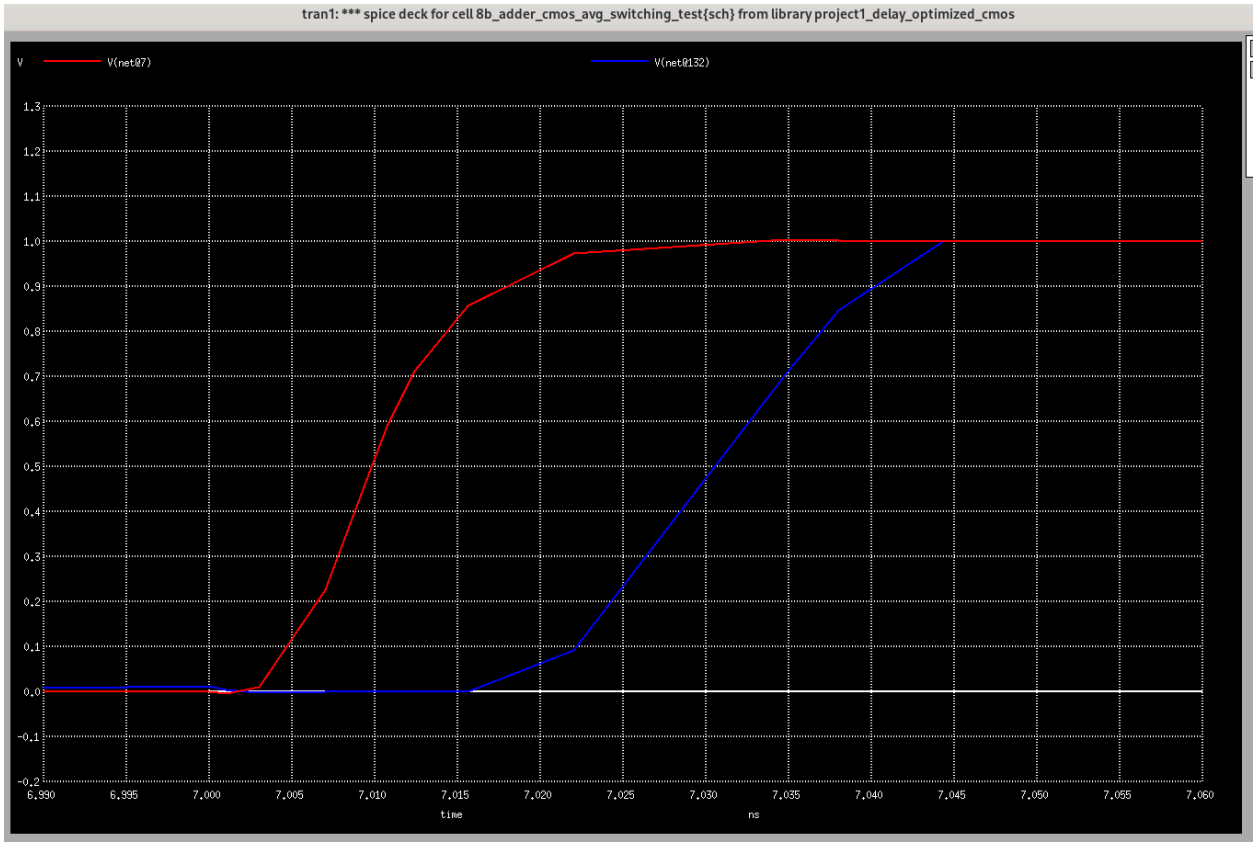
Average Switching Energy Case

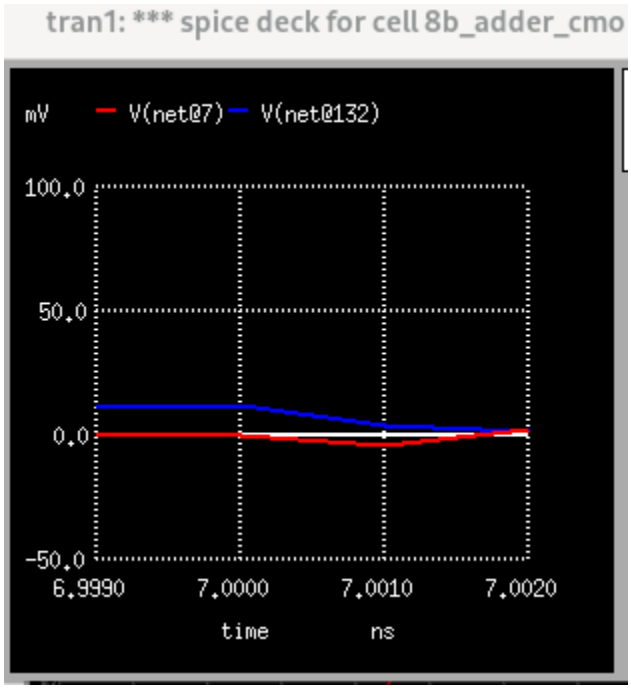
Using the same reasoning as for the baseline case, we can test the maximum switching energy case as the case where half the inputs switch from 0 to 1, with Cin kept at ground.. Specifically, A1, A3, A5, A7, B1, B3, B5, and B7 are left as switching inputs and the remaining inputs are tied to ground.

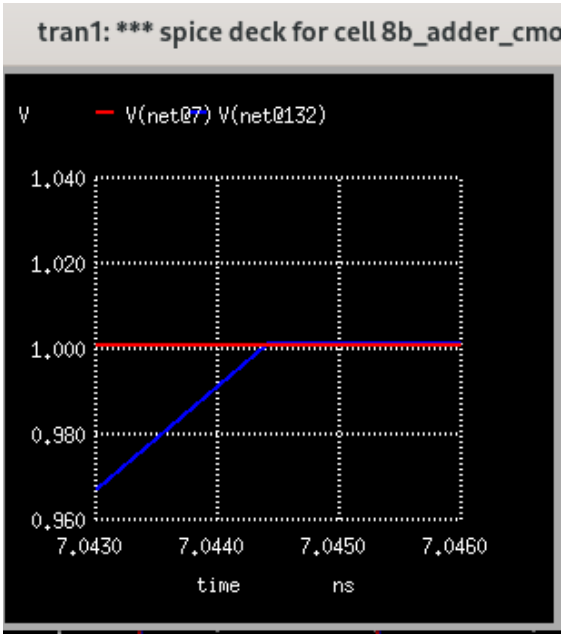


A1 is net@7 and Cout is net@132










```
x0 = 6.99999e-09, y0 = 0.0016129
```

```
x0 = 7.04446e-09, y0 = 1.0017
```

```
ngspice 120 -> meas tran qswitchcmos integ I(VV_Generi@0) from=7
ns to=7.04446ns
qswitchcmos          =  -3.56254e-15 from=  7.00000e-09 to=  7.04
446e-09
```

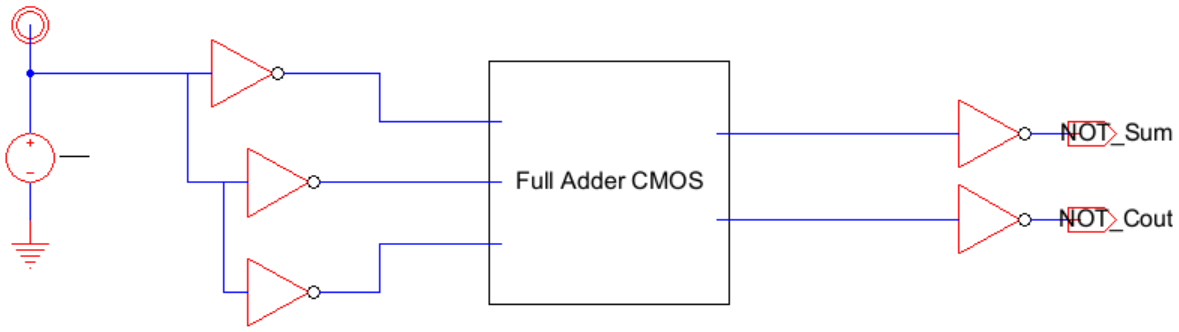
This implies that the average switching energy is **3.56 fJ,** as our VDD = 1V, and as we continue with the same calculation methods as in the baseline approach.

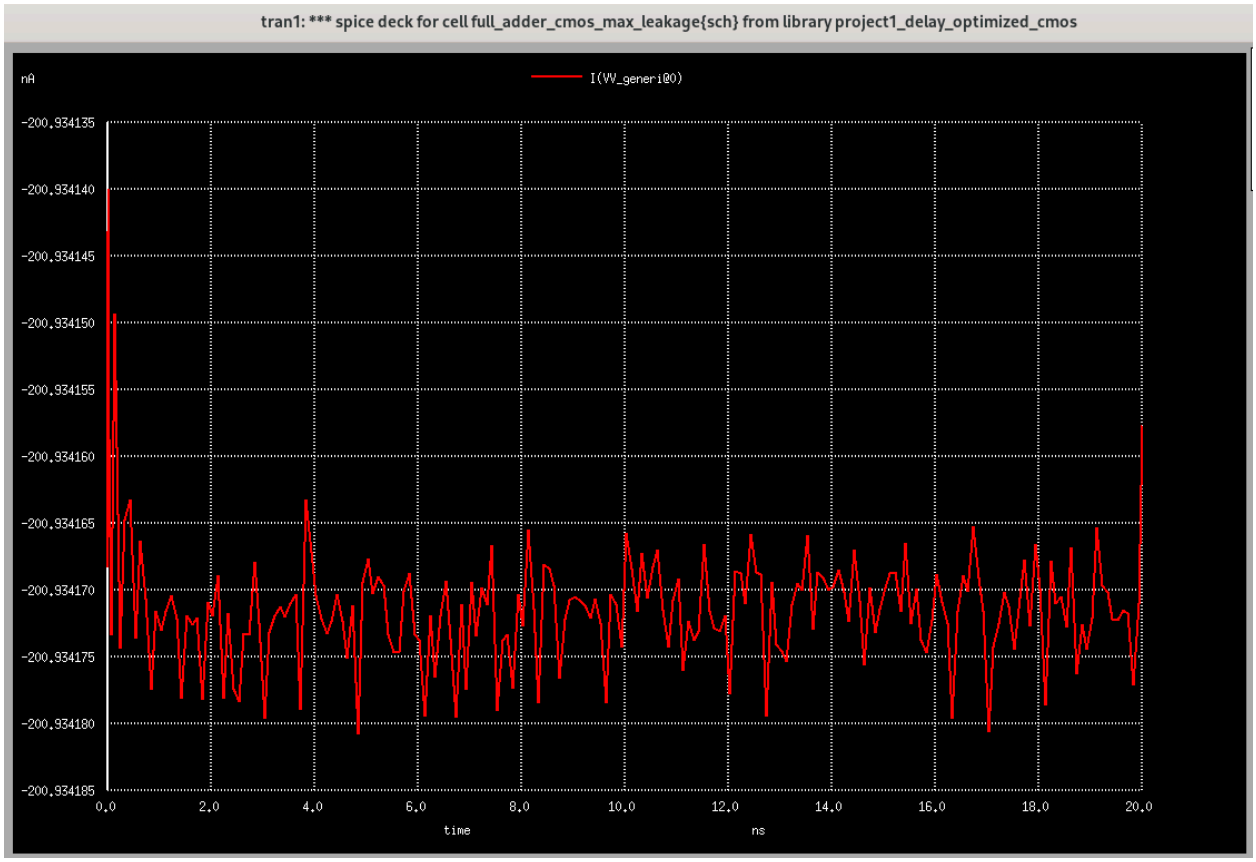
Leakage Energy

Maximum Leakage Energy Case

To identify the maximum leakage energy case for the 8-bit ripple carry adder (RCA), I began by analyzing a single full adder (FA) in isolation. Since leakage energy depends on the static input conditions and the resulting subthreshold and gate leakage through transistors, I examined all eight possible input combinations of the full adder's three inputs (A, B, Cin), minus those where Cin was to be set at a logic high, as we tie Cin to ground, so essentially 4 cases) where each input was held constant either at logic 0 (0 V) or logic 1 (1V). Each combination was applied using inverters driven by DC voltage sources to ensure a fixed, non-switching configuration. For each case, I ran a transient simulation over one full adder delay period (123ps), consistent with the evaluation time window prescribed for the project. I then computed the leakage energy using the integral of the current drawn from the VDD supply over this interval, multiplied by the supply voltage (1V). By comparing the leakage energies for all four input cases, I was able to identify the input configuration that resulted in the highest energy consumption due solely to leakage currents. This maximum-leakage configuration for a single full adder will later be scaled appropriately across the 8-bit RCA, taking into account the required logic levels of the carry-in signals for each bit-slice stage.

Case 1: A = B = Cin = 0

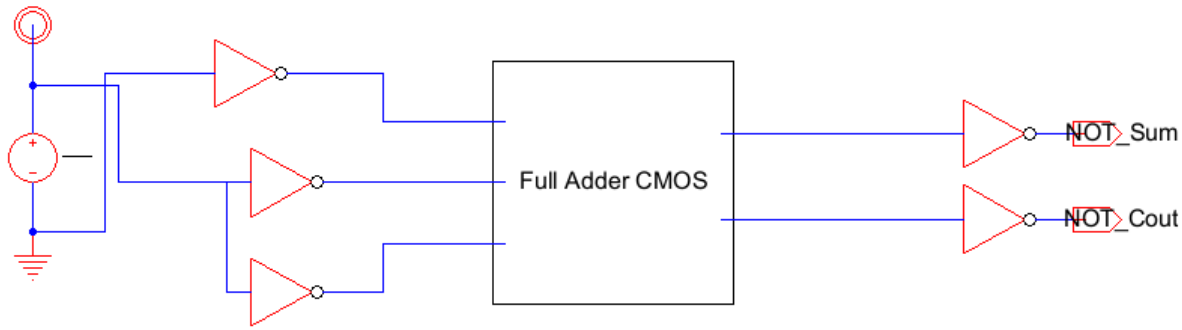


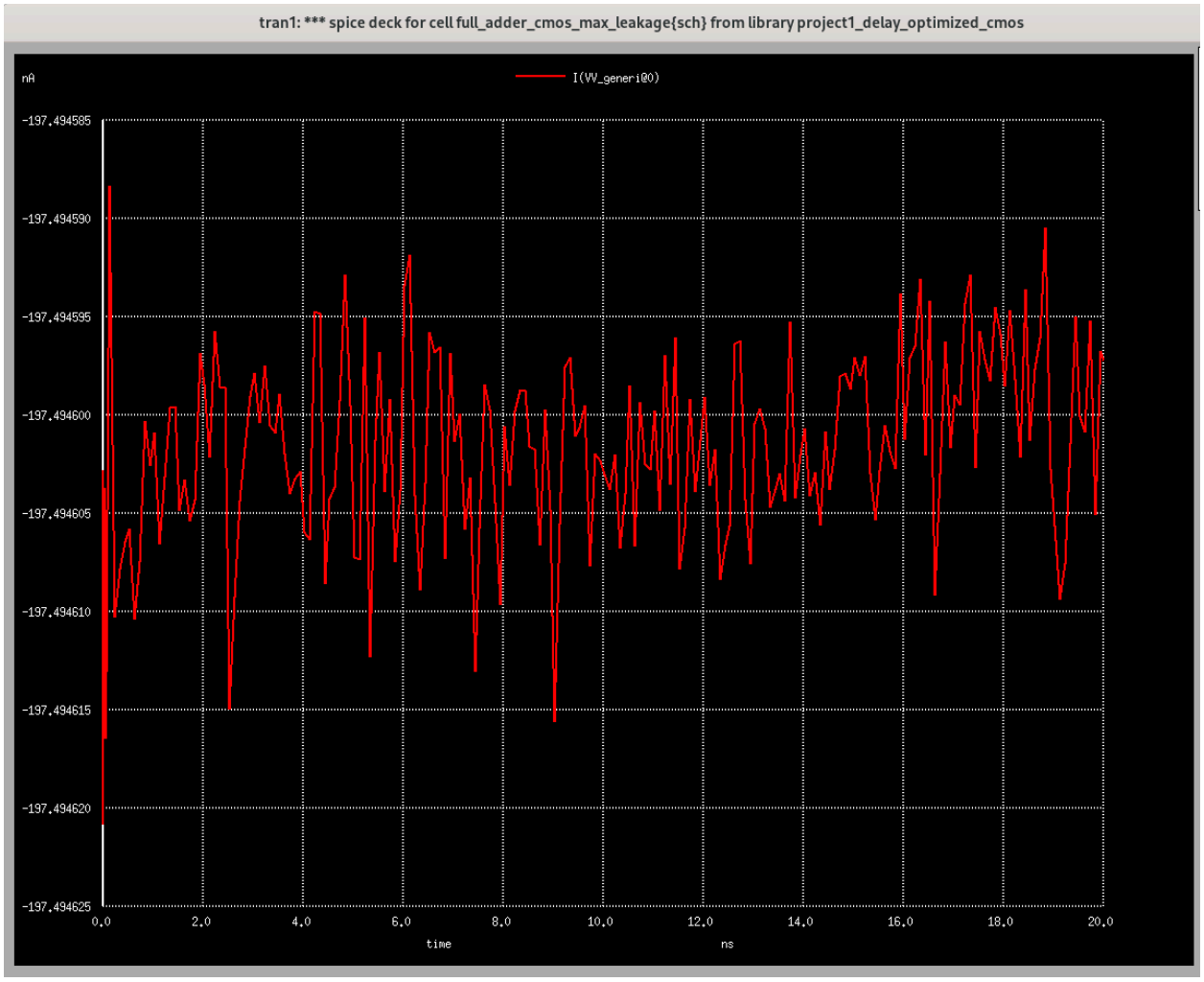



```
ngspice 121 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.123ns
qleak                =  -2.47149e-17 from=  1.00000e-09 to=  1.12300e-09
```

Leakage Energy = 2.47149* 10^-17 J

Case 2: A = 1, B = Cin = 0

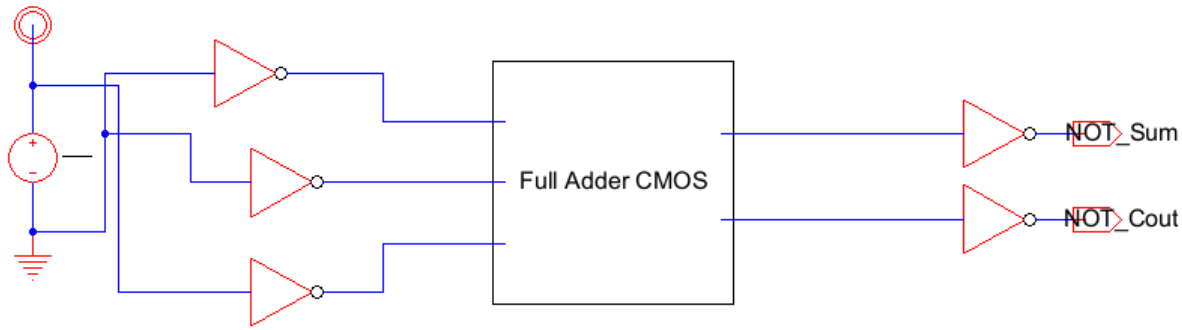


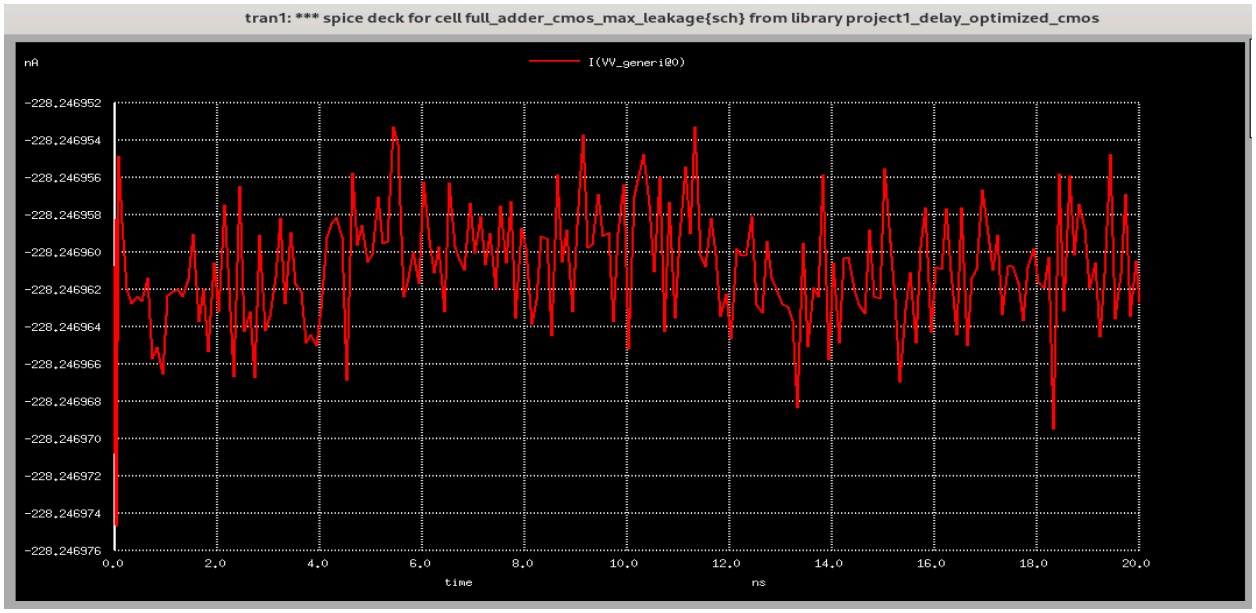


```
ngspice 121 -> plot I(VV_generi@0)
ngspice 122 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.123ns
qleak                =  -2.42918e-17 from=  1.00000e-09 to=  1.12300e-09
```

Leakage Energy = 2.42918* 10^-17 J

Case 3: A = B = 1 Cin = 0

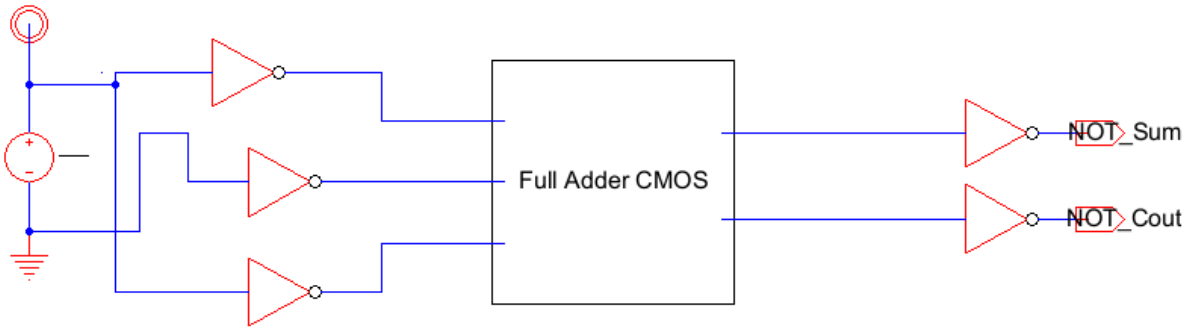


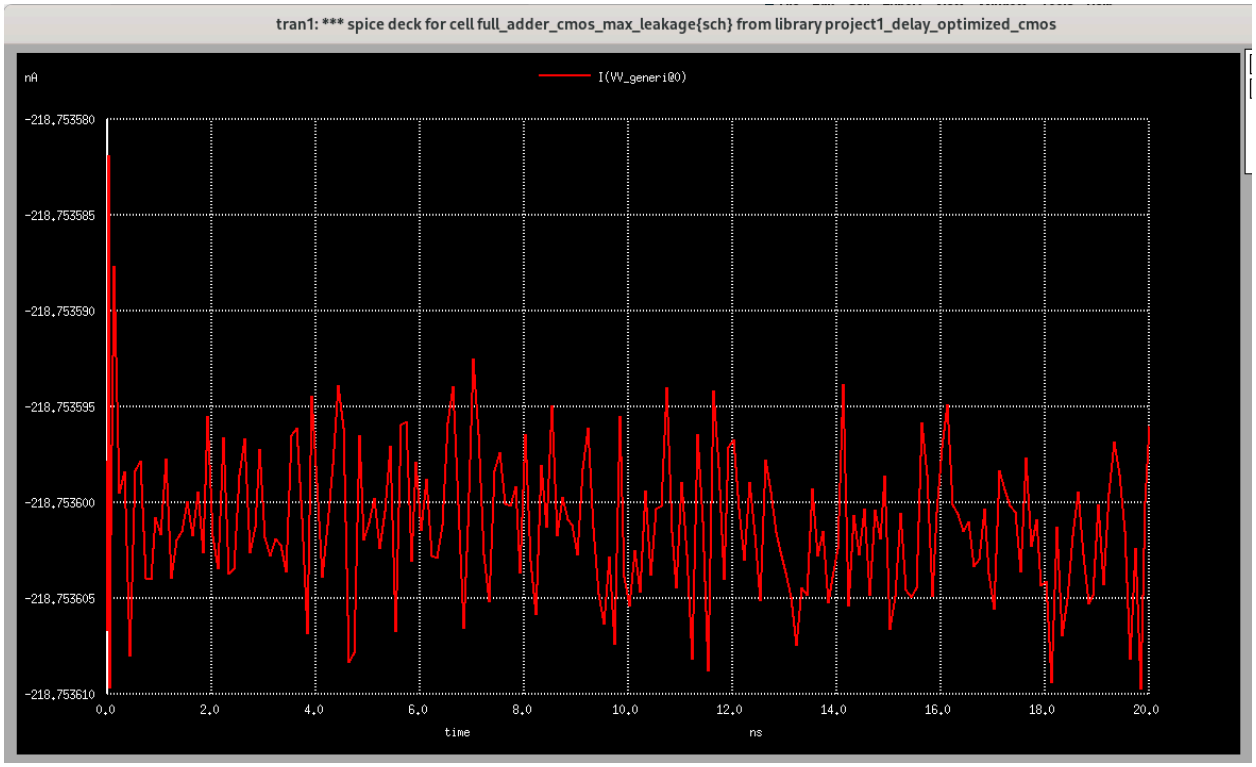



```
ngspice 119 -> plot I(VV_generi@0)
ngspice 120 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.123ns
qleak                =  -2.80744e-17 from=  1.00000e-09 to=  1.12300e-09
```

Leakage Energy = 2.80744* 10^-17 J

Case 4: A = 0, B = 1, Cin = 0



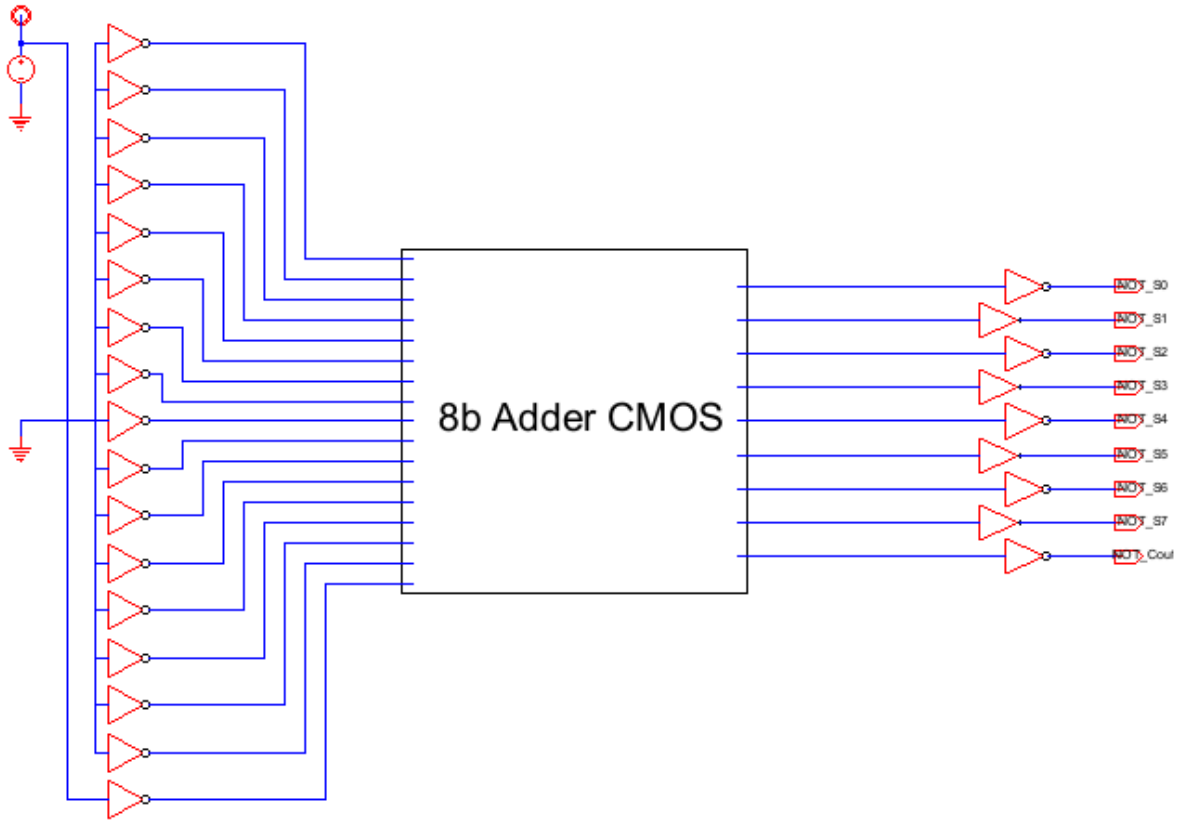



```
ngspice 119 -> plot I(VV_generi@0)
ngspice 120 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.123ns
qleak                =  -2.69067e-17 from=  1.00000e-09 to=  1.12300e-09
```

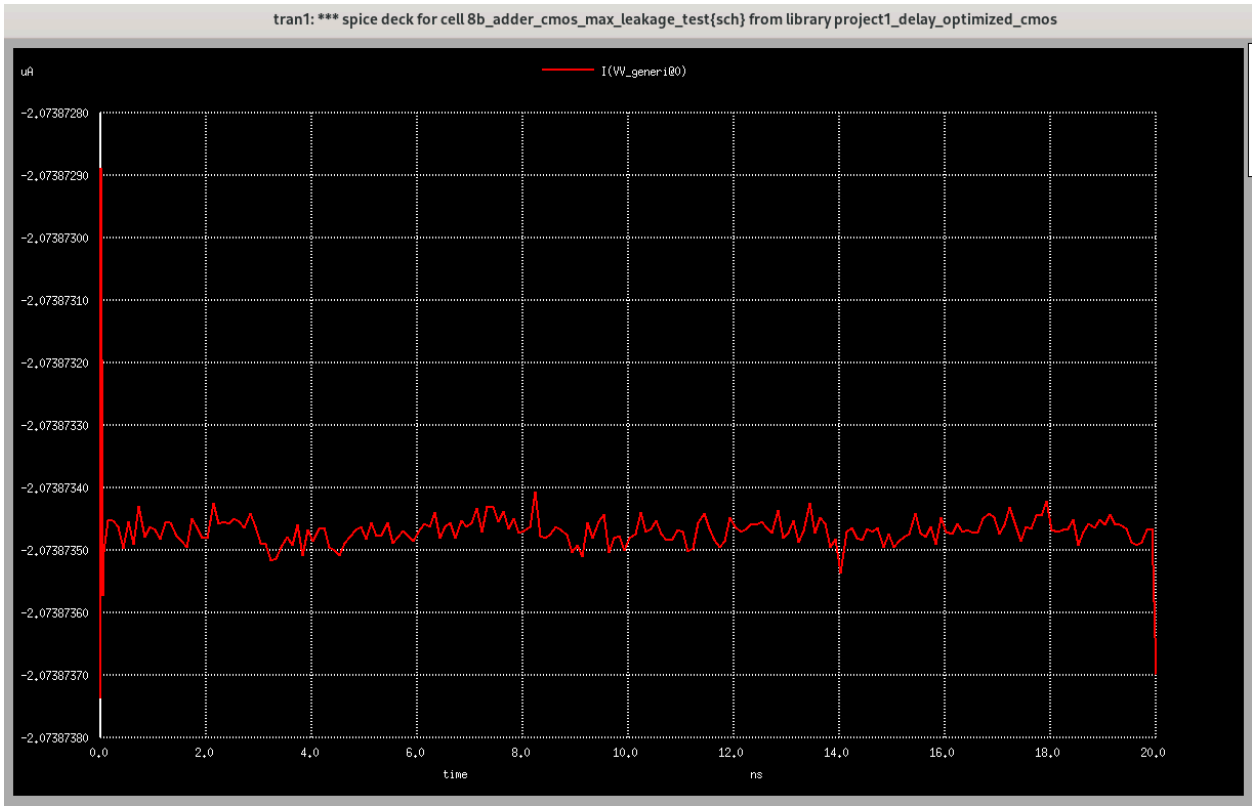
Leakage Energy = 2.69067 * 10^-17 J

We can see that we achieved maximum leakage energy for at case 3 (leakage energy = 2.80744* 10^-17 J). To extrapolate the maximum leakage energy condition from a single full

adder to the entire 8-bit ripple carry adder (RCA), I replicated the input combination that produced the highest leakage in the isolated test: A = 01 B = 1, Cin = 0. Since leakage energy arises from static input conditions and is determined primarily by the transistor states within each bit-slice, it is reasonable to apply the same high-leakage configuration across all eight full adder stages. This results in a full RCA input of A = 11111111 and B = 11111111, which ensures that every full adder receives the same input conditions that previously maximized subthreshold and gate leakage. Importantly, because this test is entirely static—with no switching or carry propagation—Cin can be uniformly held at 0 for all eight stages without affecting correctness. This simplifies the test setup significantly and guarantees consistency across the adder, allowing for accurate measurement of worst-case leakage energy. Note that this is a different case as from the baseline, reflecting the impact of change in design.



Since none of the inputs switch during this test, (All As and Bs are set to logic HIGHs through the inverters) the energy consumed is solely due to leakage, and the integration of supply current over one measured adder delay period (123ps) provides the total leakage energy.

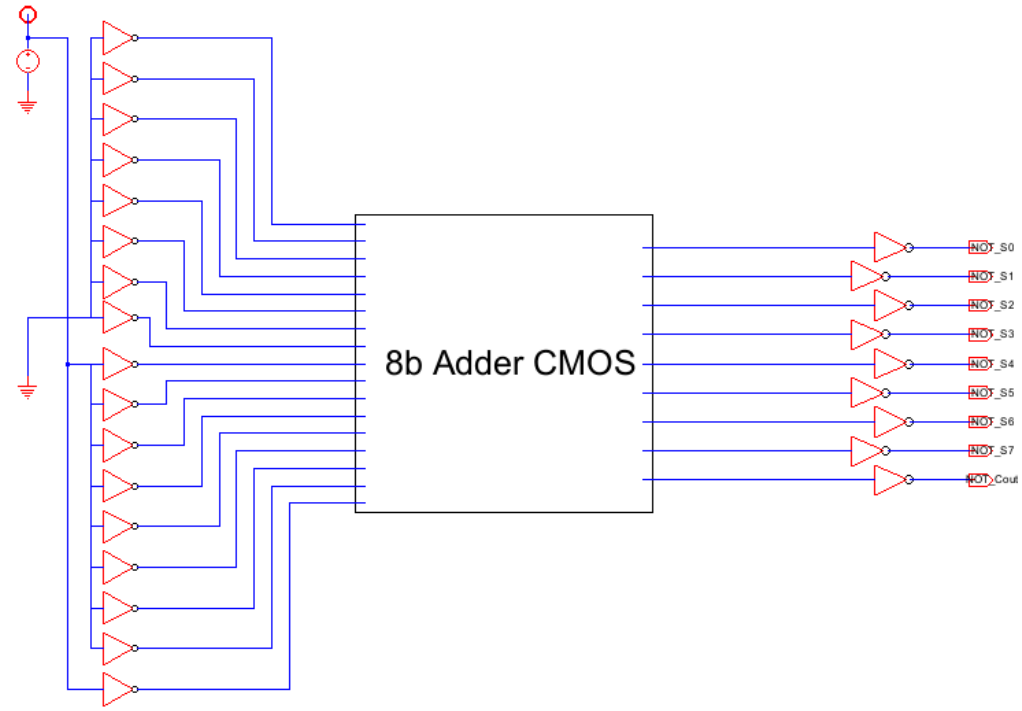


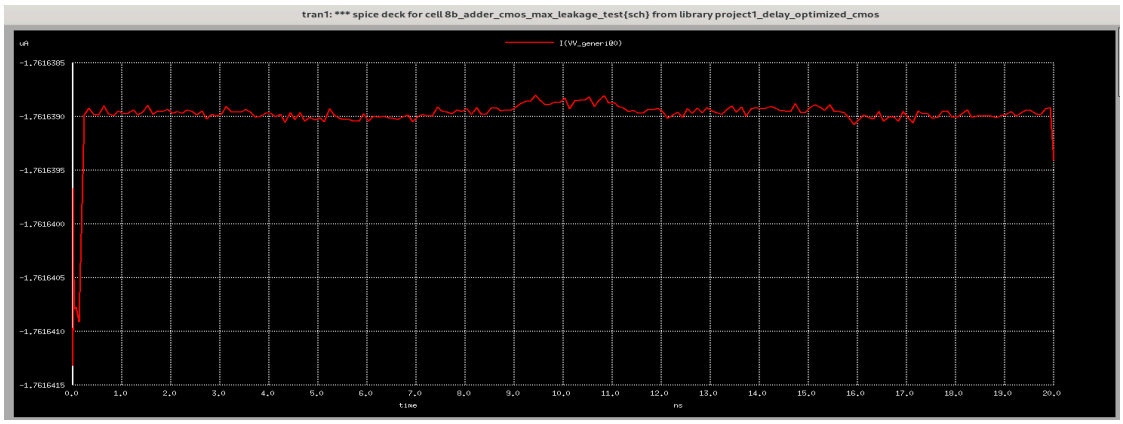

```
ngspice 119 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.123ns
qleak                =  -2.55086e-16 from=  1.00000e-09 to=  1.12300e-09
ngspice 120 -> plot I(VV_generi@0)
```

Maximum Leakage Energy = **2.55086 * 10^-16 J**

Minimum Leakage Energy Case

Case 2 (A = 1, B = Cin = 0) yielded the lowest leakage energy over one full adder, and thus will be used to extrapolate for the minimum leakage energy case.






```
ngspice 121 -> meas tran qleak integ I(VV_Generi@0) from=1ns to=1.123ns
qleak                =  -2.16682e-16 from=  1.00000e-09 to=  1.12300e-09
```

Minimum Leakage Energy = **2.16682 * 10^-16 J**

Area

Area was calculated in the same manner as before for the baseline. Each 8 bit adder of the KMAP optimised CMOS XOR version consists of 8 full adders, each of which consist of 2 optimised XOR2s, 2 NAND2s, 1 NOR2s, and 3 inverters. Each of the optimised XOR2s contain 12 transistors, each of the NAND2s contain 4, each of the NOR2s contain 4, and each inverter contains 2. Therefore, for each full adder, we have 2(12) + 2(4) + 1(4) + 3(2) = 24 + 8 + 4 + 6 = 42 transistors per full adder. Multiplying this by 8 for each of the full adders yields a total of 336 transistors. Each of these transistors has W=1.22 and L=1, where W and L are stated with respect to the 22nm process used for these transistors. Therefore, the total area can be calculated as

$$(1.22 \cdot 0.022)(1 \cdot 0.022) \cdot 336 = 0.19840128$$

In square micrometers. This value is around 50 square micrometers greater than our baseline model due to the introduction of an optimal width, which has in turn paid dividends through significantly improved delay.

Summary Table of Optimised Metrics

Metric	Value (Delay-Optimized CMOS)
Worst-Case Propagation Delay	123 ps
Maximum Switching Energy	7.17 fJ
Average Switching Energy	3.56 fJ
Maximum Leakage Energy	25.51 fJ
Minimum Leakage Energy	21.67 fJ
Area	0.1984 μm^2

Design Options Explored

Optimisations Implemented

To reduce the worst-case propagation delay in the 8-bit Ripple Carry Adder (RCA), I implemented two primary optimizations: logic restructuring using a Karnaugh map (K-map) optimized CMOS XOR2 gate and uniform transistor sizing guided by analytical delay modeling using the tau model. These changes were targeted specifically at the critical path, which traverses through two XOR2 gates, a NAND2, a NOR2, and several inverters in each full adder stage.

The first and most substantial optimization was the replacement of the baseline XOR2 gate, which had been constructed from multiple levels of NAND, NOR, and inverter gates, with a 12-transistor static CMOS implementation derived directly from K-map simplification. Using the truth table of the XOR function and applying Karnaugh map minimization, I obtained a minimal sum-of-products (SOP) expression, which was then mapped to a custom transistor-level schematic that preserved the full swing and robustness of static CMOS. This reduced the logical depth of the XOR2 gate and decreased the number of gate transitions required to compute the output, thereby reducing intrinsic gate delay and critical path length. Since each full adder contains two XOR2 gates on its critical path—one for computing the sum and another for computing the carry-out—the impact of this optimization scaled across all eight stages of the RCA.

The second optimization focused on sizing the transistors to minimize delay using the tau delay model. I modeled each logic gate's delay as the product of the effective resistance of the pull-up or pull-down network and the capacitance of the driven load. Using this first-order model, I wrote the total delay of the full adder's carry-out path in terms of transistor width W, assuming that input inverters were minimum-sized (width = 1) and each output was loaded with a 3-width inverter to standardize the load capacitance. Solving analytically, I determined that the optimal transistor width that minimizes the full adder's delay is approximately $W = \sqrt{3/2} \approx 1.22$. I then used this width for all transistors in the XOR2, NAND2, NOR2, and inverter gates of the final design. This consistent sizing ensures uniform drive strength across the entire adder, reducing resistive delay without introducing excessive parasitic capacitance.

Additionally, I increased the supply voltage VDD from 0.8V (used in the baseline design) to 1V in the optimized design. This change improves the gate overdrive voltage for both NMOS and PMOS transistors, which increases current drive and accelerates both rising and falling transitions. Since the delay of CMOS gates is inversely related to VDD - Vth, where Vth is the threshold voltage, this change contributes a further reduction in delay while remaining within acceptable operating conditions for the 22nm high-performance (HP) technology node.

These optimizations were chosen because they preserved the use of robust static CMOS logic and avoided the signal degradation and design complexity associated with ratioed logic and

pass-transistor logic. By maintaining full voltage swing and avoiding contention or weak drive conditions, the new design remains reliable under process variations and scalable for synthesis or layout in future work.

When evaluated using SPICE simulations, the delay-optimized design achieved a worst-case propagation delay of 123 ps, compared to 218 ps in the baseline RCA. This 43.6% improvement surpassed the threshold generally considered significant (20–30%) in logic performance optimization, validating the effectiveness of the combined K-map restructuring, transistor sizing, and voltage scaling strategies.

Alternate Designs Explored and Rejected

In the process of optimizing the 8-bit ripple carry adder (RCA) for delay, I considered three major circuit-level design approaches: ratioed logic, pass-transistor logic, and optimized static CMOS logic using Boolean minimization. The primary goal was to achieve a significant reduction in propagation delay relative to the baseline design, while maintaining functional robustness and full voltage swing.

Ratioed logic was initially considered due to its potential for reduced transistor count and simplified logic paths. However, the technique relies on resistive pull-up networks and unbalanced pull-down structures, which can result in degraded high logic levels, particularly under high fan-out or capacitive loading conditions. This degradation compromises noise margins and introduces contention risks, especially problematic in a multi-bit structure like the RCA where each stage feeds into the next. Additionally, implementing ratioed logic would require careful and possibly uneven transistor sizing, further complicating the design and analysis. Given that this project did not prioritize area or power as critical constraints, the marginal gains in delay offered by ratioed logic did not justify its potential drawbacks in signal integrity and robustness.

Pass-transistor logic was also evaluated as a candidate for delay optimization. It is known for achieving compact implementations of logic functions, particularly multiplexers and XOR gates, due to its ability to bypass gate-level stacking. However, pass-transistor logic suffers from threshold voltage loss, particularly in NMOS-only paths, which can cause logic levels to fall short of the full supply range. This necessitates additional restoring buffers or transmission gates to achieve full logic swings. In this design context, the use of transmission gates was not permitted, and restoring inverters would partially negate the delay and area advantages. Furthermore, integrating pass-transistor logic with standard CMOS stages such as NAND2 and NOR2 would introduce challenges in maintaining compatible logic levels and ensuring consistent drive strength across mixed logic families. Due to these limitations, pass-transistor logic was not pursued further.

Instead, I chose to pursue an optimization approach entirely within the standard static CMOS design discipline. This choice preserved signal reliability and full logic swing, while allowing for performance improvements through logic-level restructuring. The key optimization involved deriving a minimized implementation of the XOR2 gate using a Karnaugh map (K-map). Rather than relying on a high gate count structure composed of NAND, NOR, and NOT gates, the XOR2 function was implemented directly as a 12-transistor static CMOS gate derived from the K-map expression of the output. This approach reduced the logical depth and internal capacitance of the XOR stage, which is critical since each full adder cell contains two XOR gates on the critical path.

To further reduce delay, I applied the tau delay model to the full adder circuit and analytically determined the optimal transistor width for the design. By modeling the delay as a function of effective resistance and capacitance, and assuming minimum-sized input inverters and a fixed load of a size-three inverter, I derived an optimal transistor width of W = 1.22 for all transistors in

the adder. This width minimized the total propagation delay through the circuit, balancing reduced resistance with acceptable increases in internal node capacitance.

To assess the effectiveness of this optimized design, I measured the worst-case propagation delay through the 8-bit adder using SPICE simulations. The delay was defined as the maximum delay among the four rise and fall paths from the input bit B0 to either the output sum bit S7 or the final carry-out Cout. The result for the optimized CMOS design was 123 picoseconds, a significant reduction from the baseline delay of 218 picoseconds. This represents a delay improvement of approximately 43.6%, calculated as:

$$\frac{(218 - 123)}{218} \cdot 100\% = 43.6\%$$

This exceeds the typical 20% to 30% threshold improvement we were seeking in our optimisation. Since this reduction was achieved using a standard CMOS implementation with only a redesigned XOR gate and uniform transistor sizing, there was no need to further complicate the design using less robust logic styles. The performance benefits, combined with the simplicity, noise resilience, and layout regularity of static CMOS, justified the exclusion of ratioed and pass-transistor logic from the final implementation.

How These Alternatives Informed the Final Design

In the process of optimizing the 8-bit Ripple Carry Adder (RCA) for delay, I carefully evaluated multiple logic design alternatives—namely static CMOS logic, ratioed logic, and pass-transistor logic—to determine the most effective strategy for improving performance while maintaining signal integrity and design robustness. Each discipline offered unique tradeoffs, and understanding these helped guide my final implementation choices.

Initially, I considered ratioed logic, which reduces transistor count by eliminating complementary pull-up and pull-down paths in favor of a single conducting network. However, ratioed logic requires careful sizing of transistors to avoid logic level degradation and static power consumption due to direct current paths to ground. This introduces a strong dependency on exact transistor ratios and risks non-rail-to-rail output swings, especially under process variation or when driving capacitive loads. Since my design objective prioritized timing improvements without compromising robustness, the need to avoid degraded logic levels and static power led me to deprioritize this option.

Pass-transistor logic was also investigated as a potential alternative. While it offers extremely compact logic implementations and reduces overall capacitance by minimizing the number of transistors switched during transitions, it introduces substantial challenges with threshold voltage loss and signal degradation. Especially in cascaded designs like an RCA, degraded logic levels due to threshold drops can accumulate, leading to unreliable logic evaluation and increased sensitivity to noise. Restoring full swings would require additional circuitry such as output buffers, effectively reintroducing the very complexity and overhead I was attempting to avoid.

Through comparative analysis, I determined that logic restructuring within the static CMOS discipline offered the best balance of speed, noise margin, and reliability. In particular, I observed that the majority of delay in the RCA critical path arose from the two XOR2 gates in each full adder bit-slice. By applying Karnaugh Map (K-map) optimization, I derived a minimal SOP expression for the XOR function and implemented it directly using a 12-transistor CMOS structure. This version preserved strong pull-up and pull-down paths for every output transition, ensuring full voltage swing while reducing logical depth and improving transition sharpness.

To evaluate these choices quantitatively, I implemented CMOS, ratioed, and pass-transistor variants of the XOR2 gate and integrated each into a full adder schematic using their corresponding NAND2 and NOR2 counterparts. After extracting the worst-case delay through ngspice simulation for each configuration, I found that the CMOS implementation with the K-map optimized XOR structure consistently outperformed the others.

Area and Delay Scaling with Adder Width

The area and delay characteristics of a ripple carry adder (RCA) scale linearly with the number of bits, as each bit requires its own dedicated full adder stage and must wait for the carry to propagate from the previous stage. This predictable structure makes the RCA simple to analyze and extend, but also highlights its performance limitations at larger bit widths.

From an area standpoint, each full adder in the baseline implementation consists of 50 transistors. This includes two XOR2 gates, two NAND2 gates, one NOR2 gate, and three inverters, with each logic gate built entirely from minimum-sized static CMOS transistors. Therefore, the total number of transistors for an n-bit RCA in the baseline design is:

$$\text{Total Transistors}_{\text{baseline}} = 50 \times n$$

In the delay-optimized design, the structure of the XOR2 gate was redesigned based on Karnaugh map minimization, resulting in a more compact 12-transistor implementation. This reduced the total transistor count per full adder to 42, giving:

$$\text{Total Transistors}_{\text{optimized}} = 42 \times n$$

Thus, although area was not a primary design constraint in the optimized design, the logic restructuring led to a reduction in overall transistor count by 16%, which in turn reduces dynamic capacitance and contributes to improved performance.

In terms of delay, the ripple carry adder exhibits a worst-case critical path that spans from the least significant bit (LSB) carry-in to the most significant bit (MSB) carry-out. Because each full adder stage must wait for its carry-in to resolve before computing its own outputs, the total delay increases linearly with adder width:

$$t_{\text{RCA}} = n \cdot t_{\text{FA,carry}}$$

In the baseline implementation, the delay through one full adder stage’s carry path can be interpolated as approximately 27.25 ps, resulting in a total delay of 218 ps for the 8-bit RCA. In contrast, the delay-optimized design's interpolation has a significantly lower per-stage delay of approximately 15.4 ps, resulting in a total delay of 123 ps for the 8-bit adder.

This linear scaling trend means that while the RCA is well-suited for small operand widths due to its structural simplicity and efficient area usage, its delay becomes a limiting factor at higher bit widths. In such cases, more advanced architectures like carry lookahead adders may be favored. Such a design would be interesting to implement next, although it was said to be out of the scope of this partifular project. Nonetheless, the RCA remains a useful and instructive structure for studying delay-area tradeoffs in digital arithmetic design.