ELEC 402 Project A: Fast 8-bit adder

David McPherson 95266128

1. Abstract

Circuit description

The 8-bit full adder takes two 8-bit inputs, and outputs their 8-bit sum, and a carry out signal. This was implemented with a chain of 1-bit full adders, connected by carry signals. The 1-bit full adder was implemented using a truth table and Karnaugh maps. Testbenches were implemented for the inverter, 1-bit full adder and 8-bit full adder.

Optimization methods

PMOS transistors were sized optimally for an inverter, to minimize the worst case propagation delay. Transistors in logic gates were sized to minimize the worst case propagation delays. Adjacent transistors connected to the same input were combined. Output logic for the one-bit full adder was split into two stages, where the first stage computes the carry out signal as quickly as possible to send to the next stage.

Achieved speed

The speed of the circuit was calculated using the worst case propagation delay, measured from the time the last input crosses 0.5V to the time the last output crosses 0.5V. The propagation delay of the entire 8-bit full adder is 631 ps.

2. Circuit implementation and testing

In addition to the 8-bit full adder cell, I implemented an inverter and a 1-bit full adder. I will describe how each module was implemented, and include figures for each schematic and testbench.

2.1 Inverter

The NMOS inverter was implemented with the minimum width.
The PMOS inverter width was optimized to minimize the worst case propagation delay.
By simulation, this width was found to be 140 nm.
See the inverter simulation results section for details.

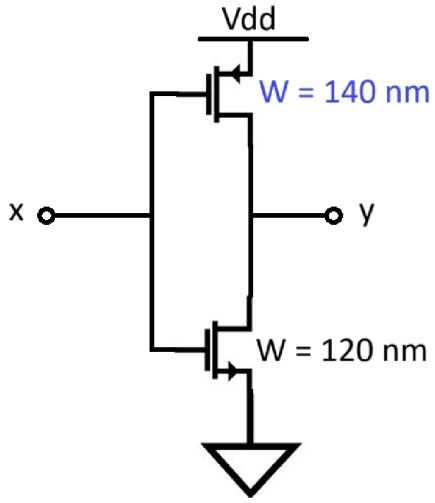


Figure 1: Optimally sized inverter schematic

The above circuit diagram is implemented in this Cadence schematic.

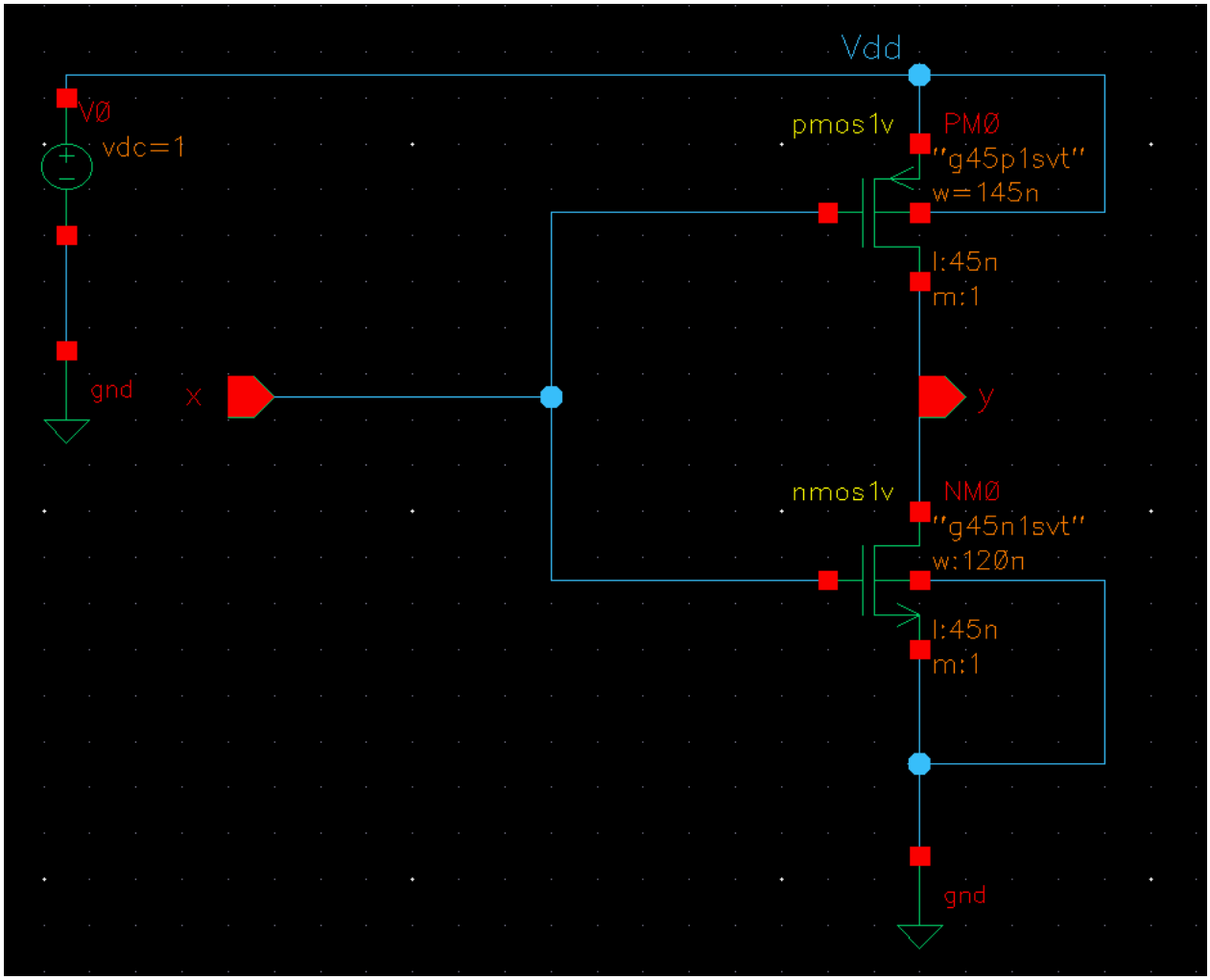


Figure 2: Optimally sized inverter schematic

The inverter testbench module connects the inverter to a voltage source with a trapezoidal signal. The output signal was confirmed to be the negation of the input signal in the steady state. The propagation delay was measured as the time between the input crossing 0.5V and the output crossing 0.5V.

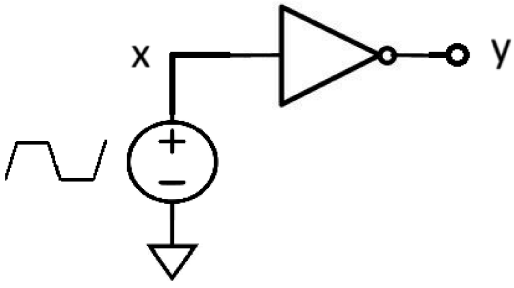


Figure 3: Inverter testbench

The inverter testbench on the previous page is implemented in the following schematic.

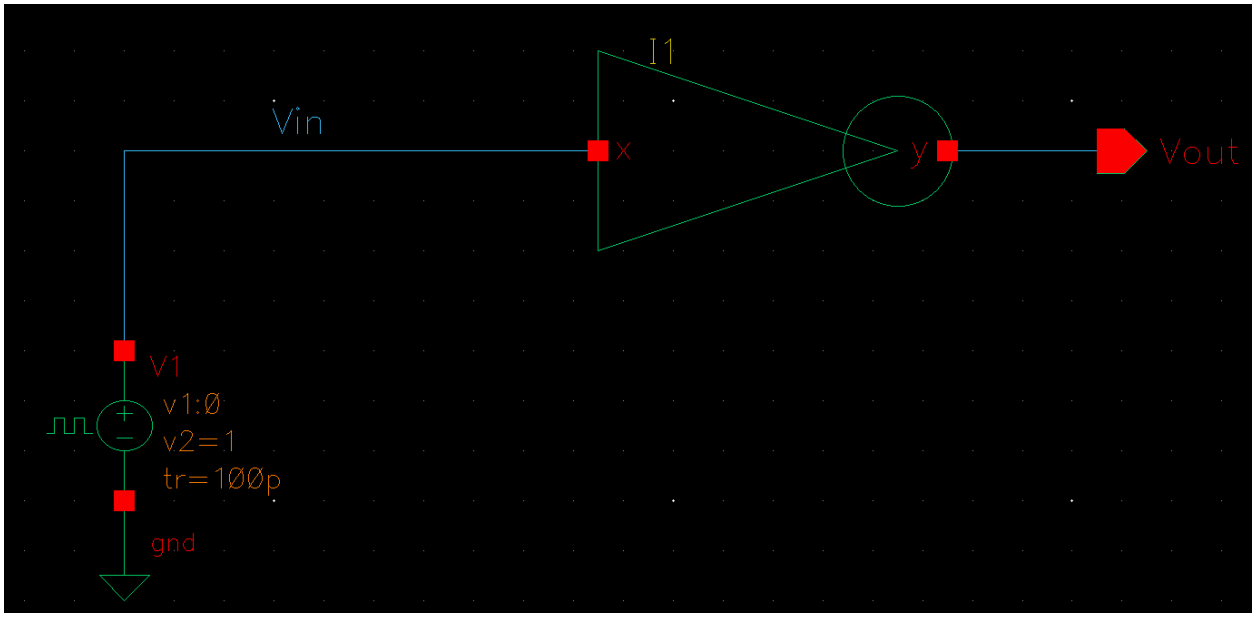


Figure 4: Inverter testbench

2.2 One-bit full adder

The one-bit full adder has three inputs: *A, B,* C_{in}, and two outputs: *Cout, S*.

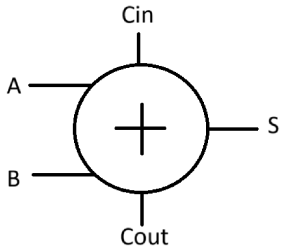


Figure 5: FA1 symbol

Here is the module's truth table.

A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 1: FA1 truth table

Output $\sim C_{out}$ can be represented by the logical function $(AB) + (AC_{in}) + (BC_{in})$. This is equivalent to the more optimal function $\sim C_{out} = (AB) + C(A + B)$. See section 3.3 for justification of this optimization.

Output S is implemented by the logic function $\sim C_{out}(A+B+C) + (ABC)$.
The intermediate value $\sim C_{out}$ is used as one of four inputs for the S output logic.

The C_{out} output was implemented by a 3-input logic gate followed by an inverter.
All transistors are sized relative to the minimum widths computed for the inverter, where NMOS width Wn = 120 nm and PMOS width Wp = 140 nm. These widths were set as the finger widths, and the coefficients on the schematic are the numbers of fingers.

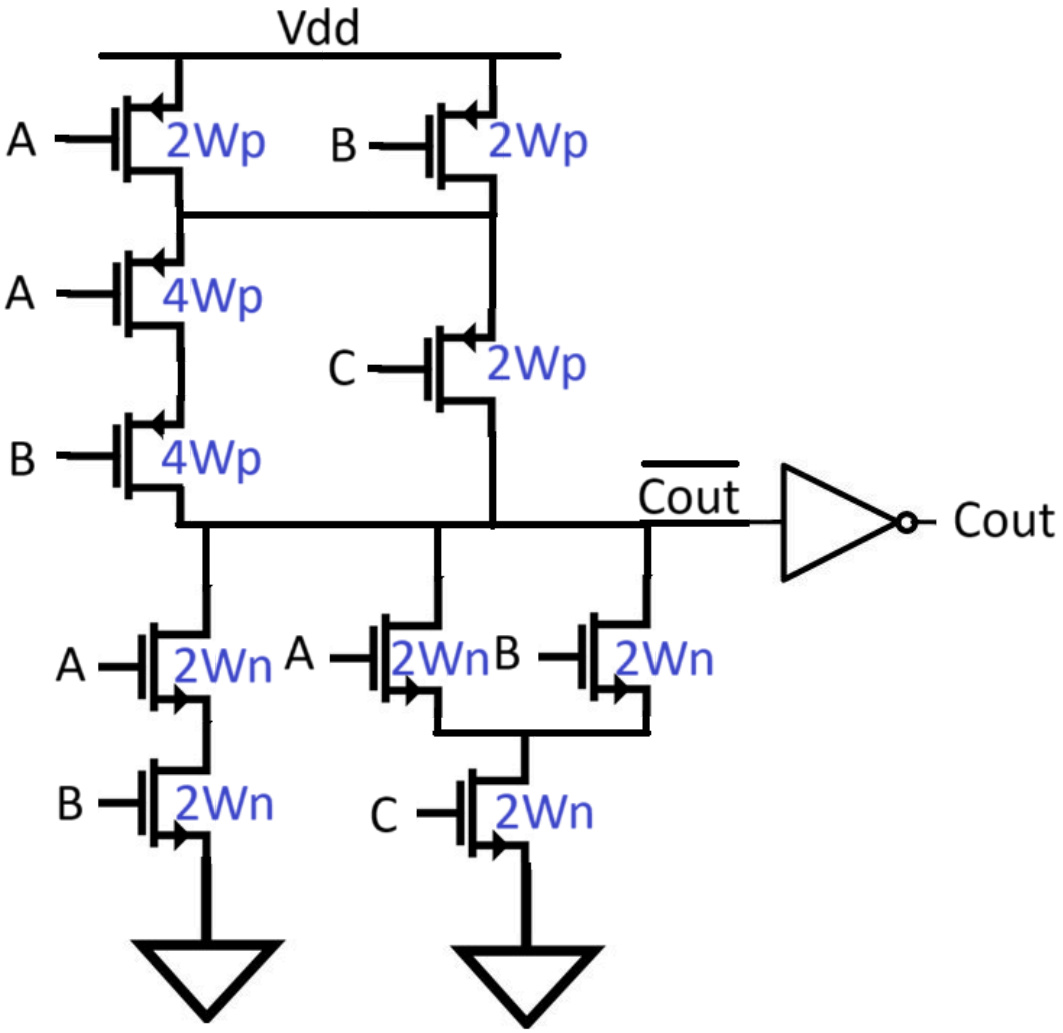


Figure 6: FA1 C_{out} logic

The schematic for $S = \sim(\sim C_{out}(A+B+C) + (ABC))$ is shown below.

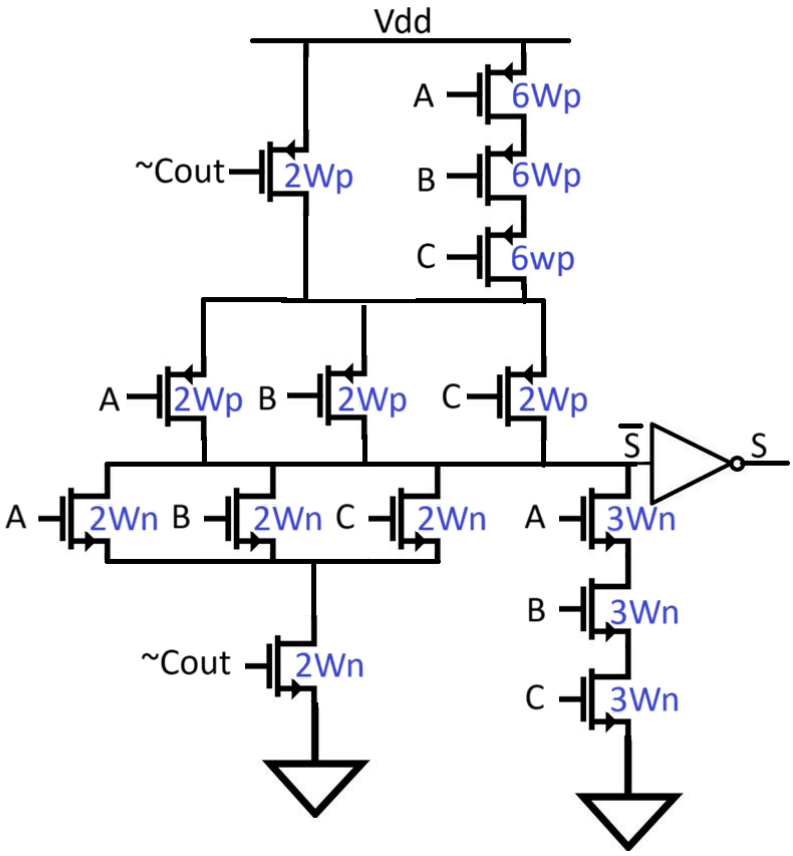


Figure 7: FA1 S output logic

Here is a screenshot of the full Cadence schematic for the one-bit full adder.
The left half is the carry-out logic and the right half is the sum logic.

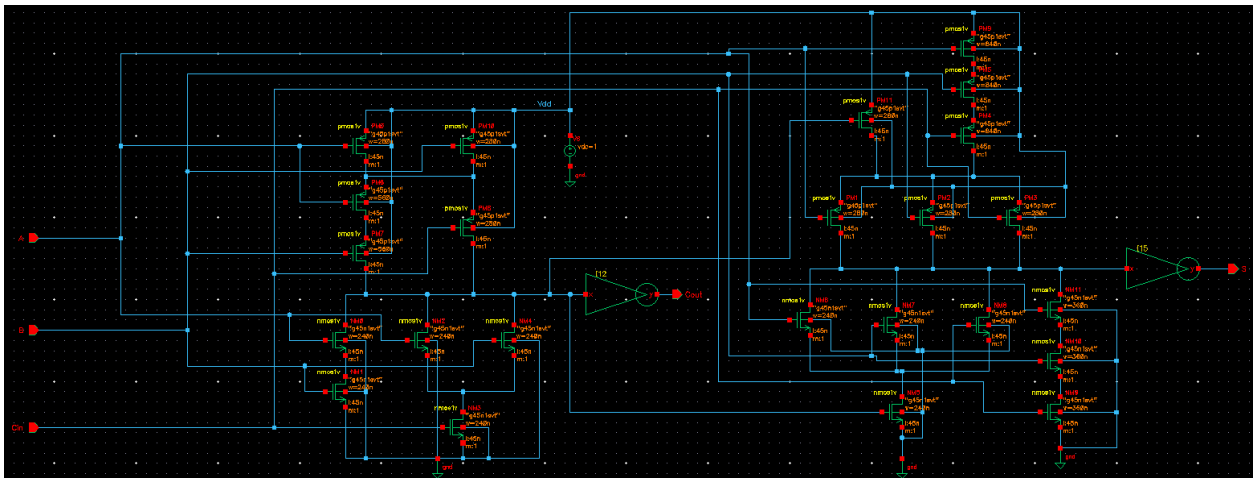


Figure 8: FA1 full schematic

The testbench was designed to generate all possible inputs. This was achieved by generating trapezoidal waveforms for each of the inputs, and by making the inputs 1 ns, 2 ns and 4 ns. Correctness was confirmed for all eight possible inputs.
Speed was determined by measuring the propagation delay between the input crossing 0.5V and output *S* crossing 0.5V. Since *Cout* is an input for *S*, its propagation delay was smaller, so it was not useful for determining the overall speed of the circuit.

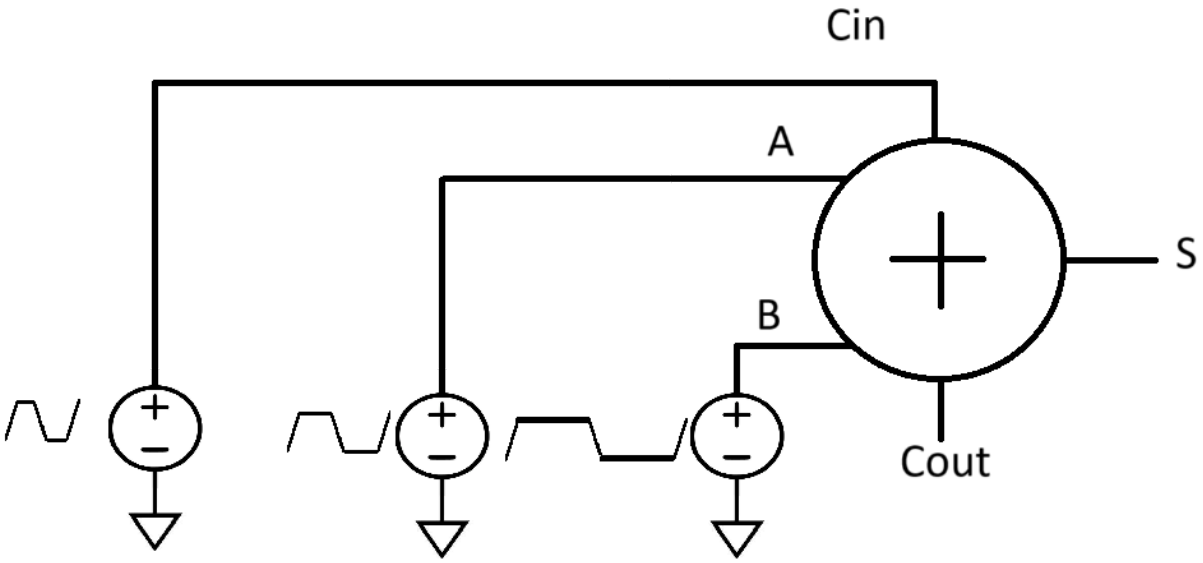


Figure 9: FA1_tb testbench schematic

Here is the Cadence schematic of the above circuit.

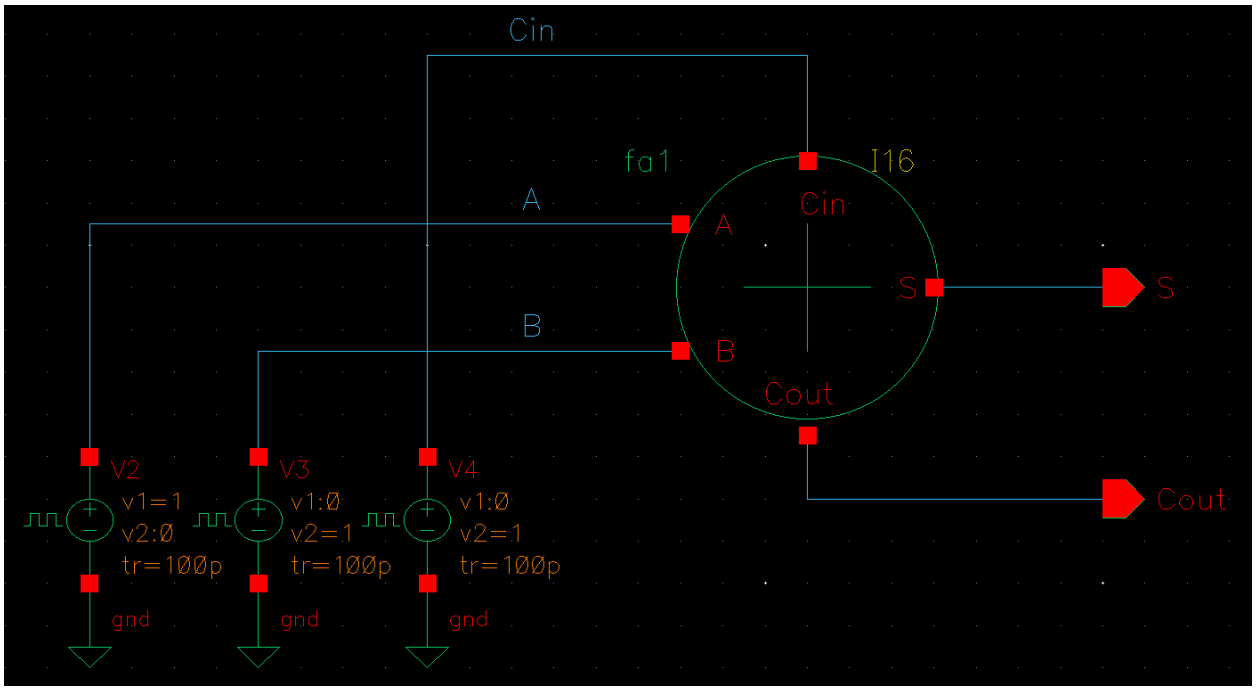


Figure 10: FA1_tb testbench schematic

2.3 Eight-bit full adder with output carry

The 8-bit full adder was created by connecting eight *FA1* modules together.
There are 16 inputs: *A0, A1, …, A7, B0, B1, …, B7*
There are 9 outputs: *S0, S1, …, S7,* C_{out}
The initial module's input carry C_{in} was connected to ground.
The final module's output carry is the 8-bit adder's C_{out} output.

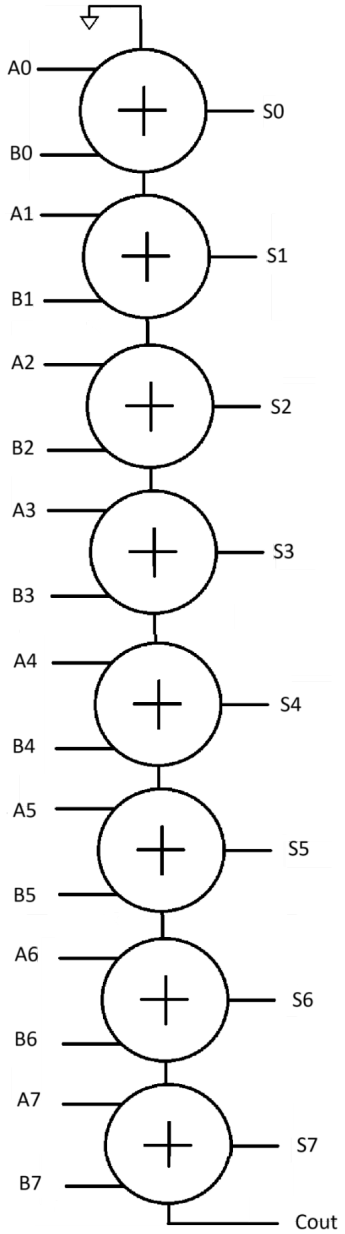


Figure 11: FA1 circuit diagram

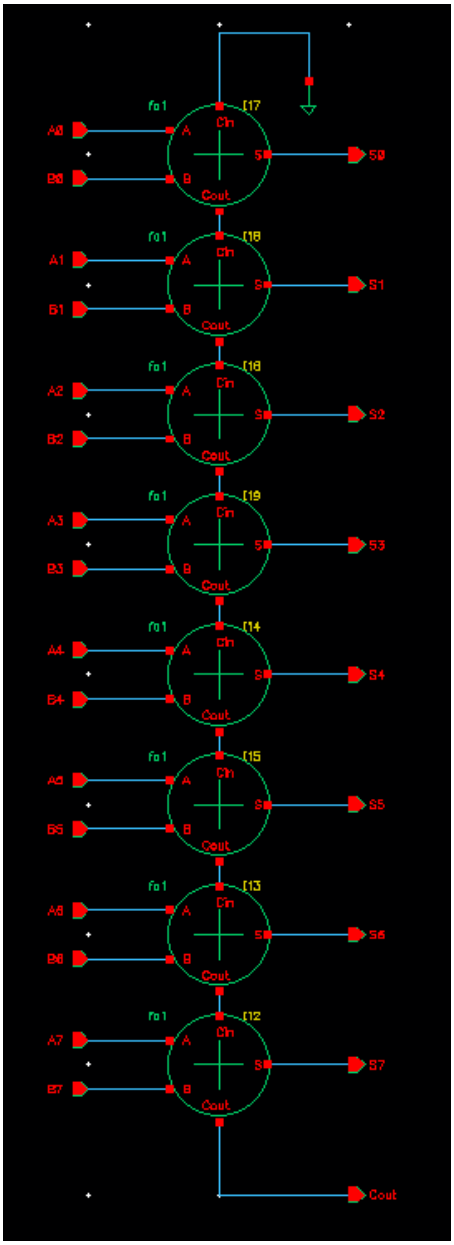


Figure 12: FA1 circuit schematic

Unlike previous modules, it was not feasible to test all possible inputs for the *FA8* cell. Instead, the worst case was tested: A[7:0] alternating between 254 and 255, and B[7:0] alternating between 0 and 1. This testbench causes all outputs to change value, except for the fastest output *S0*. It also has the longest propagation delay in the first *FA1* module. Importantly, it causes all of the intermediate carry signals between *FA1* modules to flip, maximizing the total propagation delay.

To test correctness, the circuit must output S[7:0] = 8'b1111_1110 and C_{out} = 0 when *A0* = 0 and *B0* = 0. And it must output S[7:0] = 8'b0000_0000 and C_{out} = 1 when *A0* = 1 and *B0* = 1.

The last output in the circuit to change is S7. So the propagation delay is the time between B0 crossing 0.5V and S7 crossing 0.5V.

There is no point generating variable signals for any inputs other than *A0* and *B0* because each *FA1* instance's *A* and *B* inputs will be stable by the time the new C_{in} signal arrives. So all inputs other than A0 and B0 are effectively static.

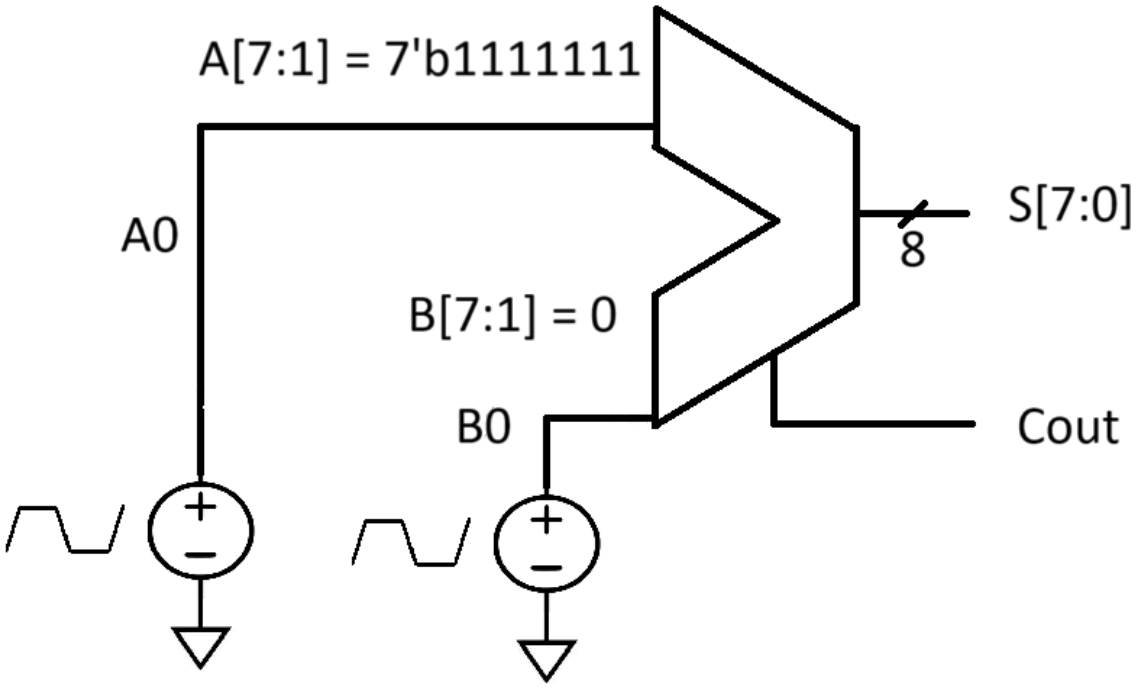
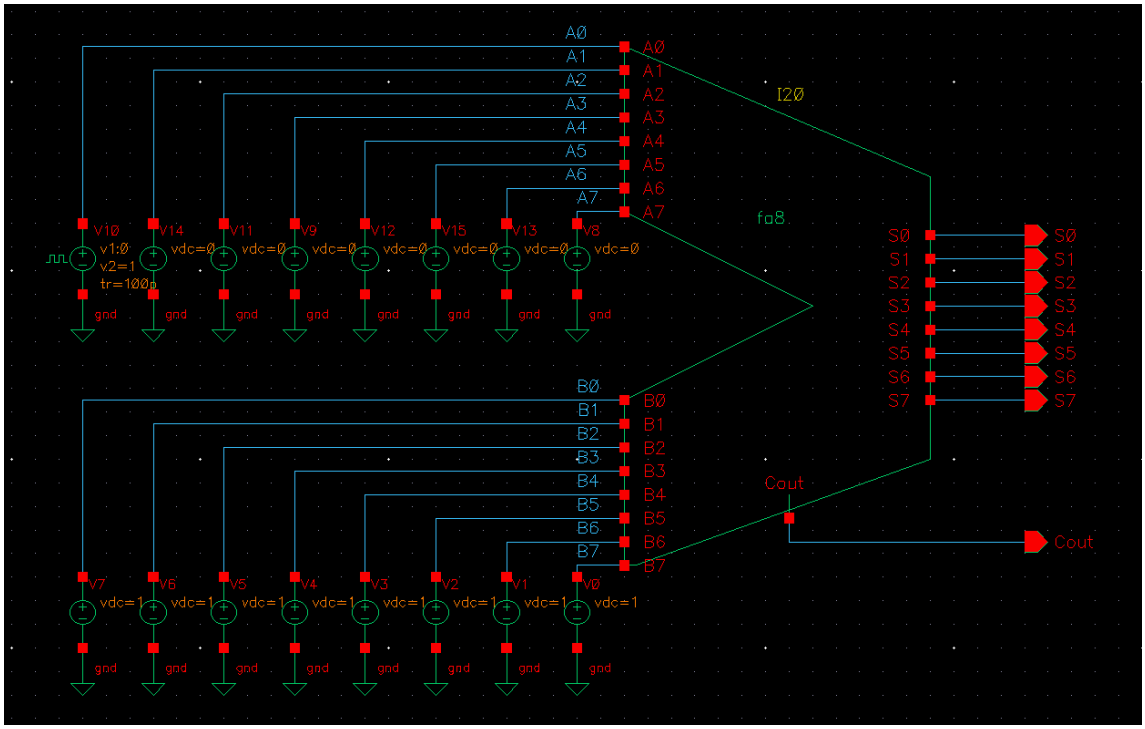


Figure 13: FA8 testbench

Here is the Cadence schematic of the final testbench.



3. Techniques used

This section describes all of the optimization techniques used, including an idea that was explored but not implemented. The full simulation results used to justify the decisions made in this section are in section 4.

3.1 Transistor sizing

All transistors use the minimum width as their finger width, based on an optimal inverter. Based on the simulation results in section 4.1, I selected the following widths:

- NMOS: 120 nm
- PMOS: 140 nm

These widths improve performance by minimizing the worst-case propagation delay.

3.2 Rejected idea: carry out logic with NAND gates

Using a Karnaugh map, the carry-out logic for the *FA1* module could be implemented by the sum-of-products boolean function $(AB) + (AC_{in}) + (BC_{in})$. This could be implemented with three NAND gates connected to a NAND3 gate. After testing both circuits, I found that the *NAND* circuit below has a worst-case propagation delay of 67.7 ps while the single gate transistor implementation has a worst case propagation delay of 66.4 ps. So the idea to split the gate into smaller *NAND* gates was rejected. See section 4.2 for the full simulation results.

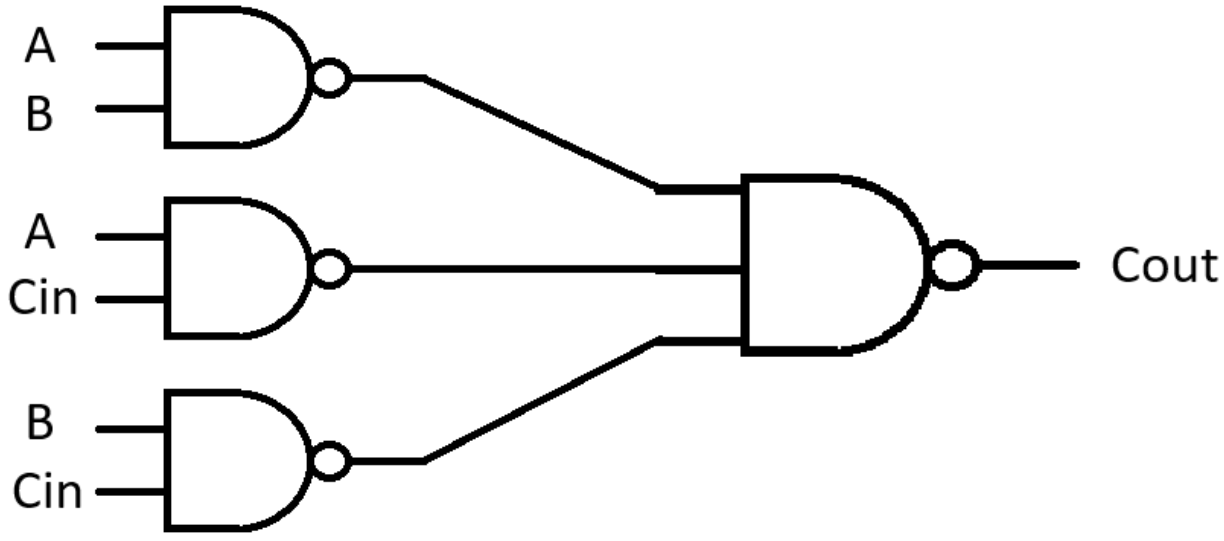


Figure 14: NAND carry-out idea

3.3 Carry-out logic: combining transistors with the same input

The logic gate for the boolean function $C_{out} = (AB) + (AC_{in}) + (BC_{in})$ has two PMOS transistors in series, both connected to the same input C_{in}. Also, there are two NMOS transistors in parallel in the pull down network, both connected to C_{in}. These transistors were combined to improve speed. Effectively, the implemented equivalent boolean function is $C_{out} = (AB) + C_{in}(A + B)$.

This optimization improves performance by reducing the transistors and properly sizing them.

3.4 Computing S after C_{out}

The direct boolean function for the sum output logic is,

$$S = (\sim A \sim B\ C_{in}) + (\sim A\ B \sim C_{in}) + (A \sim B \sim C_{in}) + (A\ B\ C_{in})$$

This function would require inverters for all three inputs, effectively creating a six input gate. Instead, I decided to use the intermediate value C_{out} to create a simpler function,

$$S = C_{out}(A + B + C_{in}) + (A\ B\ C_{in})$$

This optimization improves performance by computing C_{out} as quickly as possible, before computing S. This is very important because C_{out} is an input for the next one-bit full adder instance in the chain.

4. Simulation results

This section has tables and plots for each cell's propagation delays, as well as data used to make optimization decisions.

4.1 Inverter

Different PMOS widths were simulated. The propagation delays were calculated for both the output rising to 1 (rise delay) and the output falling to 0 (fall delay). For greater clarity, these are not rise times and fall times, they are propagation delays.

PMOS width (nm)	Rise delay (ps)	Fall delay (ps)	V_M (mV)
130.00	22.48	21.87	497.30
135.00	22.32	22.04	498.27
140.00	**22.18**	**22.20**	**499.18**
145.00	22.04	22.37	500.05
150.00	21.91	22.53	500.94

Table 2: Inverter results

Interestingly, 145 nm is the PMOS width with the switching voltage closest to 0.5V. So 140 nm is not the optimal choice for noise tolerance. But my objective is to optimize for speed, so I selected 140 nm for my minimum PMOS width. The table data is displayed in the plot below. The final inverter design has a propagation delay of 22.20 ps.

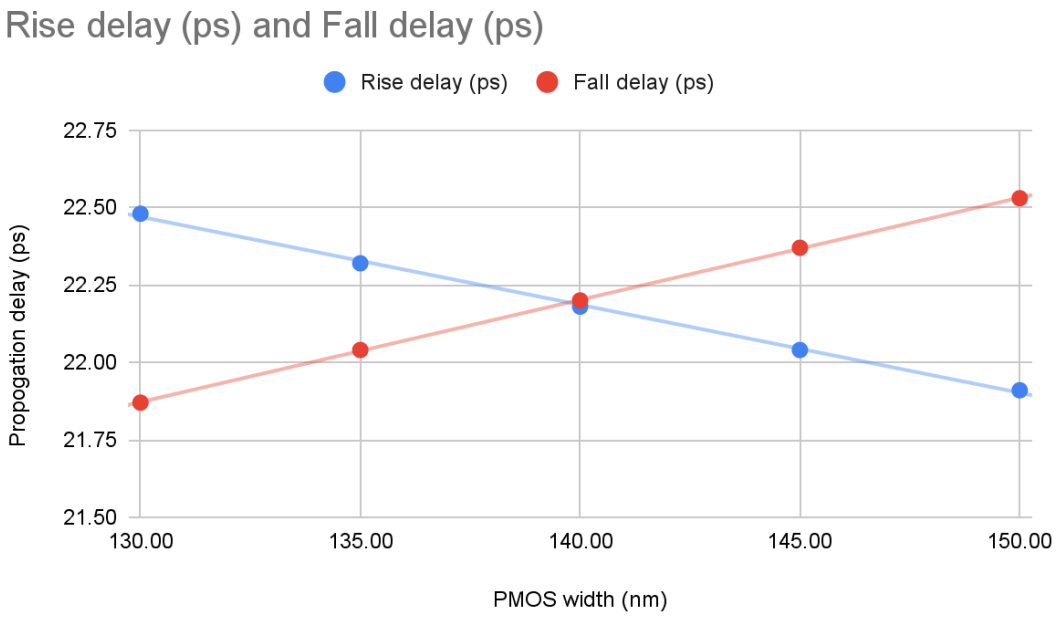


Figure 15: Inverter propagation delays.

4.2 One-bit full adder

Two designs for the C_{out} logic were tested. *fa1_LG* is the implemented design, using a single logic gate and an inverter. *fa1_NAND* was an equivalent circuit I simulated, and described in section 3.2. I simulated the transitions between inputs 000 and 111, as well as 100 and 011. The results are that *fa1_LG* has a worst case propagation delay of 66.4 ps and *fa1_NAND* has a worst case propagation delay of 67.7.

Switch	fa1_LG (ps)	fa1_NAND (ps)	Type
000 -> 111	48.9	50.9	rise
111 -> 000	54.4	44.1	fall
100 -> 011	63.4	**67.7**	rise
011 -> 100	**66.4**	51.7	fall

Table 3: Inverter results

4.3 Final simulation: 8-bit full adder

The plot below shows the full waveform. Within each bunch of signals, the left-most signals are inputs *A0* and *B0* while the rightmost signal is *S7*, preceded by C_{out}. So the overall propagation delay of the circuit is the time between the inputs *A0, B0* crossing 0.5V and the output *S7* crossing 0.5V. When *S7* is falling from 1 to 0, this delay is 576 ps. When *S7* is rising from 0 to 1, this delay is 631 ps. So 631 ps is the actual propagation delay of the entire circuit.

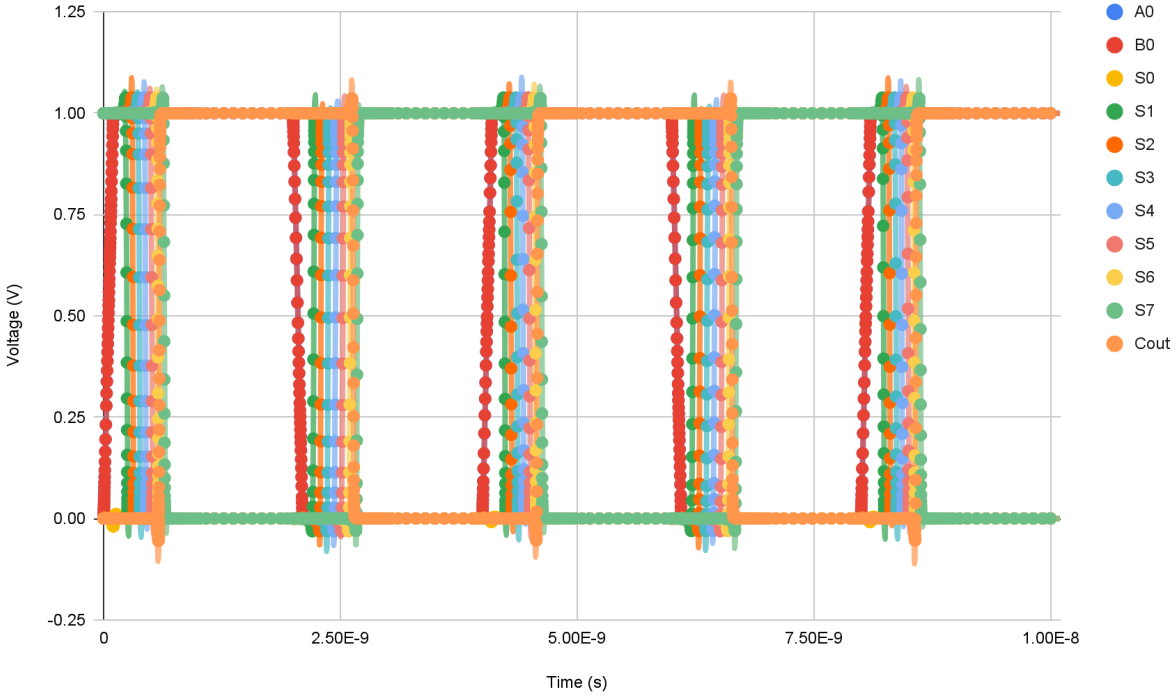


Figure 16: Full transient simulation.

5. Conclusion

The full circuit has a propagation delay of 631 ps. This is equivalent to a chain of 28 minimum-sized inverters, each with a delay of 22.2 ps.

Areas of future improvement

The last one-bit full-adder instance could be modified to compute both outputs at the same time. The one-bit full adders were optimized for minimum delay on the carry out signals, to speed up the next full adder instance in the chain. But the slowest signal in the entire circuit is *S7*. So implementing a custom full adder optimized for minimum delay on both outputs could improve speed.

The first one-bit full adder instance could be optimized to have just two inputs *A* and *B*, without a carry-in signal. I could have replaced the C_{out} logic with an *AND* gate and the *S* output logic with an *XOR* gate. I chose not to do this to reduce complexity in the design process.

Final remarks

My design is very simple, and pretty fast. I simply optimized a single one-bit full adder and instantiated eight of them. Although not perfectly optimal, my implementation reduces complexity in the design process, and is simple to test and simulate.