VLSI Final Project: Full Adder

David Sheppard 916213658
VLSI Design, Section 3
December 20, 2019
sheppardd1@students.rowan.edu

I. Introduction and Objectives

A. Goals

When constructing any logic circuit at the silicon level, one must be sure to factor in circuit speed amongst other design constraints. The end goal of this project was to design, test, and analyze a full adder circuit. The design was to be implemented with speed as the foremost priority. Furthermore, the circuit must present an input capacitance equivalent to that of 3 unit-transistors (henceforth referred to as 3C of capacitance) and drive a load with 30 times as much capacitance (90C). The linear delay model with logical effort analysis was to be used to determine optimal gate sizing and expected delay.

The full adder circuit was to be implemented as both a schematic and layout. The layout was constructed for the AMI C5N process with a feature size of 600 nm [1]. The schematic-based and extracted (layout-based) designs were simulated and the results compared to each other and to the calculated values. Discrepancies between the results were discussed and their potential causes were summarized.

II. Approach

The design process began by determining possible logical expressions to use for representing the full adder's functionality. A truth table was constructed to determine the circuit's basic logical equations in the sum of products format. These equations were used in conjunction with K-Maps to determine appropriate ways to simplify the overall output expressions. Once this was complete, the principle of bubble pushing was used to further simplify the logical design into one that would be more CMOS-friendly in terms of both construction and speed. During this reduction process, two final designs were decided upon with which to further analyze by means of the linear delay model.

Using the values of logical effort and parasitic delay given in [2], the normalized delay for each of the designs was determined using effort-based calculations. After deciding which of the proposed designs was optimal, the proper number of stages and gate sizes for each circuit was decided upon mathematically. Once this was completed, the circuit was ready for construction.

The full adder circuit and its individual circuit components were constructed using both the Cadence Virtuoso Schematic Editor and the Cadence Virtuoso Layout Editor. Both designs were tested and their equivalence was verified through the layout vs. schematic (LVS) validation tool. After simulating the entire full adder, the results were compared to the previous calculations after denormalization with the simulation results of the unit inverter. The results from the calculations, schematic simulation, and extracted (layout-based) simulation were all compared and the discrepancies discussed.

III. Design Methodology

A. Logical Optimization

To begin the analysis of the circuit, the logic of the circuit was quantified and summarized. The full adder contains three inputs: two standard input bits and a carry-in bit. For simplicity, these inputs will be referred to as A, B, and C, respectively. The full adder uses these three bits of input to generate two outputs: Sum and Carry. The Sum bit represents the single-bit addition of the three bits while the Carry output represents whether or not a digit must be carried into the output. The full adder's truth table can be found in Table I.

TABLE I
Truth Table for Full Adder

Inputs			Outputs	
A	B	C	Sum	Carry
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

As Table I shows, the Sum goes high whenever there are an odd number of high inputs. On the other hand, the Carry output goes high whenever there are at least two high inputs. These observations can be used to construct a logical equation to represent each output. To simplify this process, the K-Maps shown in Fig. 1 were used to create simplified logic expressions as a starting point.

Based on the results from the K-Maps given in Fig. 1, the following logical expressions can be found:

$$Sum = \overline{A}\overline{B}C \;||\; A\overline{B}\overline{C} \;||\; \overline{A}B\overline{C} \;||\; ABC \tag{1}$$

$$Carry = AB \;||\; AC \;||\; BC \tag{2}$$

(see *Appendix A* for a catalog of logical expression symbols). For (1), the K-Map was unable to simplify the logical expression at all. Nonetheless, it was previously observed that the sum output should be high if and only if an odd number of

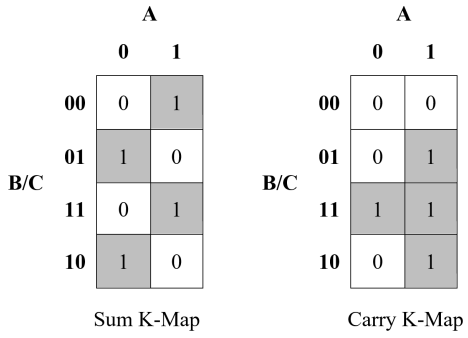


Fig. 1. K-Maps for Sum and Carry Outputs

inputs are high. This can be expressed using the Exclusive OR function, such that

$$Sum = A \oplus B \oplus C = (A \oplus B) \oplus C \tag{3}$$

This allows the value of the Sum to be completed using two XOR gates. Therefore (2) and (3) can be combined to form the logical diagram seen in Fig. 2

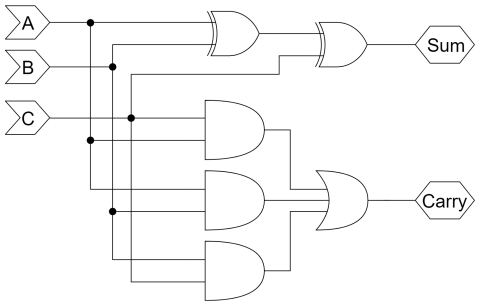


Fig. 2. Basic Full Adder Circuit with Independent Sum and Carry Outputs

Furthermore, it can be noted that an alternative logical expression for Carry can be constructed by reusing the expression $A \oplus B$ from the Sum logic. Since Carry should be high if and only if there are at least two high inputs, the expression can be rewritten as saying that Carry should go high if A and B are high or if one of them is high along with C. The expression for Carry can therefore be given by

$$Carry = AB \;||\; [(A \oplus B)C] \tag{4}$$

This design has the benefit of using one less logic gate than the design in Fig. 2 because the first XOR gate is used to compute both Sum and Carry. Combining (3) and (4) results in the gate-level diagram given in Fig. 3.

Following the determination of the designs in Fig. 2 and Fig. 3, the property of bubble pushing was used to simplify the gate-level designs even further. As Weste and Harris explain in [2], bubble pushing allows for complicated logic gates such as AND and OR gates to be represented more simply as NOR and NAND gates. The basic premise of bubble pushing is rooted in DeMorgan's Rule, which allows for conversion between conjunctive (AND) statements and disjunctive (OR) statements by altering the distribution of negations in an

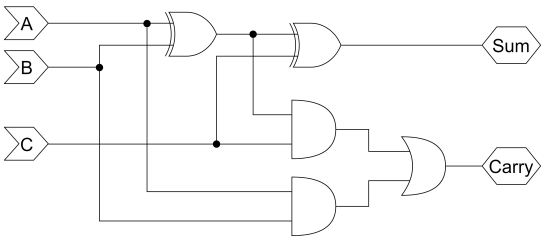


Fig. 3. Basic Full Adder Circuit

expression [3]. Using these properties, the OR gate in Fig. 3, can be represented as a NAND gate with bubbles (signifying inversion) at each of its inputs [2]. Making this conversion then allows for the new bubbles at the new NAND gate's inputs to be pushed backward to the preceding AND gates, thus converting them into NAND gates as well. The resulting gate-level logic design can be seen in Fig. 4. This design is much simpler to implement in CMOS because it requires fewer transistors. Additionally, it is likely faster than the design in Fig. 3 because NAND gates have notably lower logical effort than OR and AND gates [2].

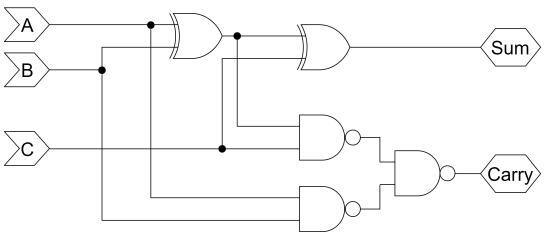


Fig. 4. Design 1: Full Adder Improved with Minimal Logic Gates

As with the design shown in Fig. 3, the design given in Fig. 2 can also be minimized by bubble pushing. In a virtually identical manner, the OR gate can be converted into a NAND gate with bubbles at its inputs. These bubbles can then be pushed to the preceding AND gates to convert them into NAND gates as well. The resulting design is shown in Fig. 5. For the sake of simplicity, the design shown in Fig. 4 will be referred to as Design 1 and the design in Fig. 5 will be referred to as Design 2.

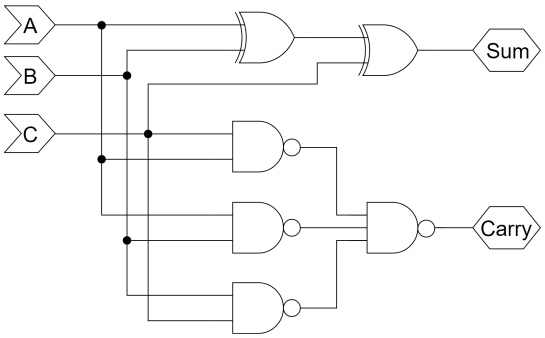


Fig. 5. Design 2: Full Adder with Independent Sum and Carry Outputs

B. Effort Delay Optimization

To determine which full adder design would result in the shortest delay from input to output, the logical efforts of the previously discussed designs were calculated. Weste and Harris define logical effort of a gate as "the ratio of the input capacitance of the gate to the input capacitance of an inverter that can deliver the same output current" [2]. The basis of this linear delay model is the idea that a unit inverter can be taken as the basis for normalization so that all other logic gates can be modeled with respect to the inverter's delay. The delay of the unit inverter can then be found in simulations so that the estimated delay of the entire circuit can be calculated based on this information [2]. The base equation used to approximate the overall circuit's delay is given as

$$D = F + P = GBH + P = NF^{\frac{1}{N}} + P = N\hat{f} + P \quad (5)$$

where F is the effort delay of the circuit, P is the parasitic delay of the circuit, G is the path logical effort, B is the path branching effort, H is the electrical effort of the circuit, N is the number of stages in the path, and $\hat{f} \equiv F^{\frac{1}{N}}$ is the optimum effort per stage [2]. These properties refer to those of the entire path that is being analyzed. To determine these values, their equivalent values for each stage (signified as $f,\ p,\ g,\ b$, and h) must be determined. The values of stage logical effort and stage parasitic delay for the gates relevant to this analysis are presented in Table II. The values given here are normalized to the logical effort and parasitic delay of the inverter.

TABLE II
NORMALIZED LOGICAL EFFORT AND PARASITIC DELAYS OF RELEVANT LOGIC GATES

Gate	Logical Effort (g)	Parasitic Delay (p)
Inverter	1	1
NAND2	$\frac{4}{3}$	2
NAND3	$\frac{5}{3}$	3
XOR2	4	4

While the values of g and p are given in Table II, the values of b and h must be determined based on the circuit's construction itself. The value of b refers to the branching effort of a stage. This depends on the number of logic gates that a given gate outputs to. For example, the first XOR gate in Fig. 4 would have a stage branching effort of 2 because its output is connected to two gates: the second XOR gate and a NAND gate. Similarly, the value of h represents the stage electrical effort which quantifies the output capacitance of a gate divided by its input capacitance. The relations between path and stage values for a path with N stages are summarized in the following equations:

$$\text{Logical Effort} \equiv G = \prod_{i=1}^{N} g_i \quad (6)$$

$$\text{Branching Effort} \equiv B = \prod_{i=1}^{N} b_i \quad (7)$$

$$\text{Electrical Effort} \equiv H = \frac{C_{path\ out}}{C_{path\ in}} \quad (8)$$

$$\text{Parasitic Delay} \equiv P = \sum_{i=1}^{N} p_i \quad (9)$$

$$\text{Effort Delay} \equiv F = \sum_{i=1}^{N} f_i = \sum_{i=1}^{N} g_i b_i h_i = GBH \quad (10)$$

Using these properties in conjunction with (5), the theoretical delay of each logic device can be calculated. As Fig. 4 and Fig. 1 show, the logical effort taken starting at input A or input B will be greater or equal to the logical effort taken starting at input C for each case. For this reason, the analyses will be preformed by calculating logical effort from input A to each output (i.e. the worst case logical path).

As previously mentioned, the design constraints dictate that the circuit must present a capacitance of 3C at each input where C is the capacitance of a unit transistor. As a result, each input of the full adder will be immediately followed by an input buffer to ensure that this design constraint is made. The first inverter of each input buffer will be the unit inverter with a pMOS width of 3 μm and an nMOS width of 1.5 μm.

Additionally, the design constraints also specify that each of the full adder's outputs will be driving 90C of capacitance. This gives the overall circuit an electrical effort of 30, in accordance with (8). As Jaeger and Blalock point out in [4], circuits with a high electrical effort can often benefit from added output buffers so that the path electrical effort is better distributed among each stage. As a general rule of thumb, the optimal number of stages $\hat{N}$ in a circuit can be approximated as

$$\hat{N} \approx \log_4(F) \quad (11)$$

Using an Excel spreadsheet, the normalized delays for Design 1 and Design 2 with 0-2 output buffers were computed. The results of these computations can be seen in Table III, with the optimal delays shown in bold.

TABLE III
COMPARISON OF OPTIMAL NORMALIZED DELAYS FOR DESIGN 1 AND DESIGN 2 WITH VARYING NUMBERS OF OUTPUT BUFFERS

		A to Sum	A to Carry
Design 1	No Buffer	36.48	29.28
	1 Buffer	33.15	30.36
	2 Buffers	34.58	33.05
Design 2	No Buffer	34.64	**22.04**
	1 Buffer	**32.16**	23.5
	2 Buffers	33.86	26.51

As Table III shows, the fastest design should be Design 2 with one output buffer at Sum and no output buffer at Carry. Comparing this with the rule in (11) shows that the results were sound. The effort delay of A to Sum is $F = GBH = (4 \times 4)(3)(30) = 1440$ and A to Carry is $F = GBH(\frac{4}{3} \times \frac{5}{3})(3)(30) = 200$. Using (11) to calculate according to the fan-out of 4 rule produces $\log_4(1440) \approx 5.25$ and $\log_4(200) \approx 3.82$. Since the number of stages in any path

must be an integer, the value of N for the path from A to Sum must be 5 or 6 and the value of N for the path from A to Carry must be 3 or 4. As Table III shows, this was indeed the case. The path from A to Sum requires 6 stages and the path from A to Carry demonstrates a need for 4 stages. The final circuit design can be seen in Fig. 6 with gate size variables listed at each logic gate.

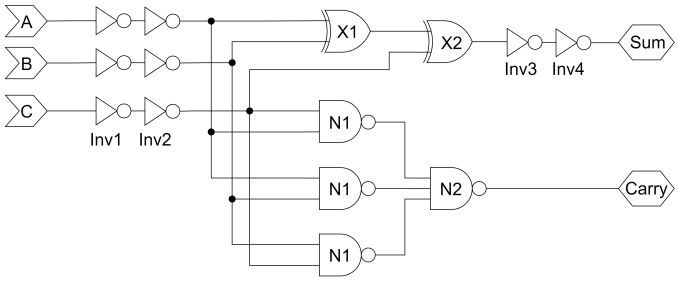


Fig. 6. Final Full Adder Design

From these results, the optimal gate sizes can be calculated. This was done by giving each gate an effort delay equal to the optimal effort delay per stage, $\hat{f}$. For the path from A to Sum, the optimal stage effort delay would be $\hat{f} = F^{\frac{1}{N}} = 1440^{\frac{1}{6}} \approx 3.36$. Similarly, the path from A to Carry would have an optimal stage effort delay of $\hat{f} = F^{\frac{1}{N}} = 200^{\frac{1}{4}} \approx 3.76$. Using the equation $\hat{f} = gh$ for each stage, the optimal size of each gate was determined for each path.

During the gate sizing process, it must be remembered that this analysis assumes that each gate in a single stage has the same size. As a result, there would be conflicting sizes found for the first XOR2 gate and the NAND2 gates. For the sake of simplicity, the XOR2 gates was constructed with the size determined using the analysis for the path from A to Sum while the sizes for the NAND2 gates were determined using the analysis for the path from A to Carry. Additionally, each gate size had to be rounded to the nearest 0.15 µm due to design constraints. A summary of the final gate sizes is given is Table IV.

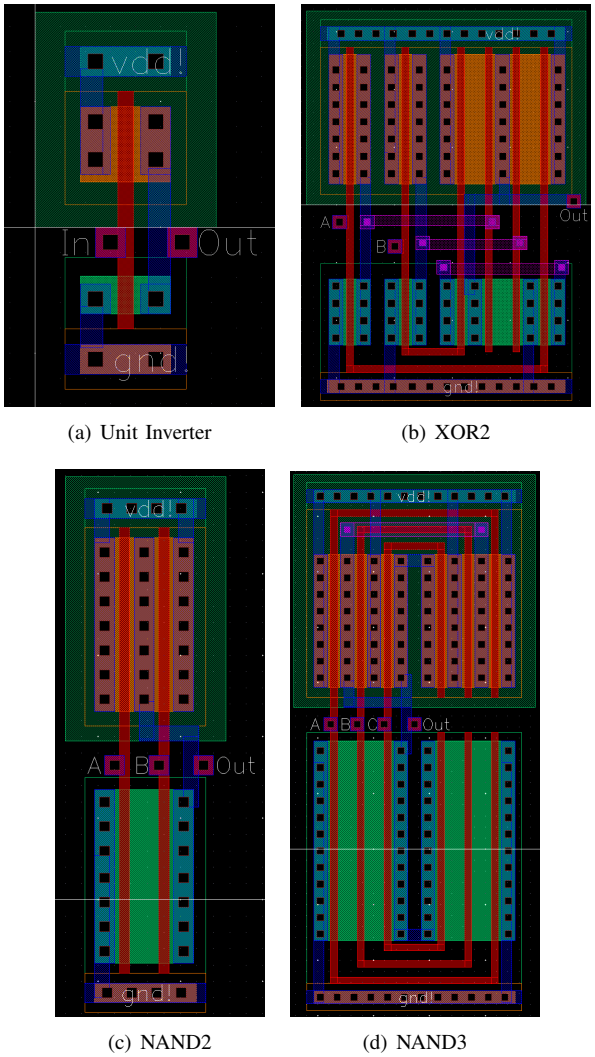
TABLE IV
FINALIZED LOGIC GATE SIZES

Logic Gate	Quantity	Normalized Gate Size	P:N Ratio	pMOS Size (µm)	nMOS Size (µm)
Inv1	3	1	2:1	3.00	1.50
Inv2	3	3.36	2:1	10.05	5.10
Inv3	1	2.66	2:1	7.95	4.05
Inv4	1	8.93	2:1	26.85	13.35
N1	3	4.71	1:1	10.65	10.65
N2	1	13.3	2:3	24.00	35.85
X1	1	3.76	1:1	11.25	5.70
X2	1	3.16	1:1	9.45	4.80

Despite the limitations of gate sizing, each stage effort remains well within a reasonable range. As [2] explains, a stage effort in the range of 2.4 to 6 will result in a design "within 15% of minimum delay." Using the real gate sizes given in Table IV, a spreadsheet was used to determine that the real stage efforts ranged from 3.36 to 3.92 - well within the range needed to be $\pm 15\%$ of the optimal delay.

C. Standard Cell Library

Once these values were obtained, each logic gate was constructed as both a schematic and layout. After each layout was constructed, it was extracted and compared with the schematic via the LVS verification tool. The LVS tool ensured that the netlists of both the extracted layout and schematic matched, i.e. it affirmed that the electrical connections of the layout were the same as those in the schematic. The layouts of the unit inverter, the NAND2 gate, the NAND3 gate, and the first XOR2 gate are shown in Fig. 7. The inverters not shown vary only in width from the unit inverter, as does the other XOR2 gate from the one shown. It is important to note that the NAND3 gate was folded for the sake of compactness; this was done to allow the overall full adder to be more densely constructed.



(a) Unit Inverter (b) XOR2

(c) NAND2 (d) NAND3

Fig. 7. Standard Cell Library Layouts

D. Denormalization of Calculated Delays

One the unit inverter was built, its logical effort and parasitic delay could be determined by testing the device with a varying number of loads. The parasitic delay of the inverter is simply the intrinsic delay of the inverter itself [2]. To determine the parasitic delay, the inverter was simulated with no load; the resulting delay was its parasitic delay. From the simulations, this was found to be $t_{pd} \approx 63.5$ ps. This was determined by finding the average rising and falling propagation delay. The rising propagation delay t_{pdr} is defined as the time between the input signal rising to $\frac{1}{2}V_{DD}$ and the output signal hitting $\frac{1}{2}V_{DD}$ during its transition. The falling propagation t_{pdf} is identical to rising propagation delay except that it is found when the input is falling instead of rising [2]. The results are summarized as follows: $t_{pdr} \approx 74$ ps, $t_{pdf} \approx 53$ ps, and $t_{pd} \approx 63.5$ ps.

The inverter was also tested with 1-3 loads at its output. In each case, another copy of the inverter was used as the load. For these testing purposes, the extracted version of the inverter was used. The results of the simulations were recorded, normalized to the unloaded value of t_{pd}, and plotted to determine the inverter's logical effort. The logical effort corresponds to the slope of the plotted data [2]. Ultimately, it was found that the logical effort of the inverter is about $g \approx 0.46$. The results are given in Fig. 8, where the slope of the data-fitted line is the inverter's logical effort.

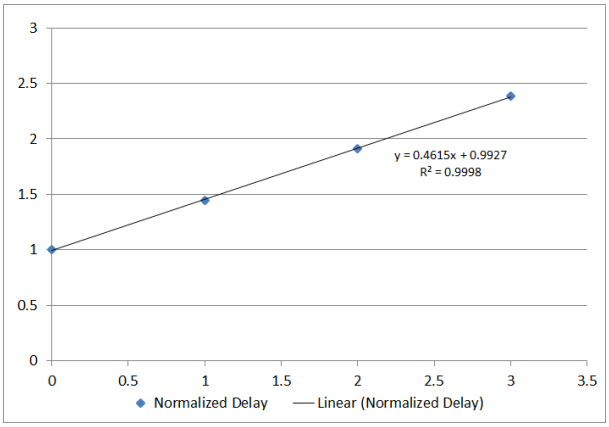


Fig. 8. Normalized Delay of Unit Inverter with Respect to Loading

From this new data, the theoretical delay was then denormalized and calculated in terms of nanoseconds. To do this, the logical effort values given in Table II were multiplied by 0.46 and the parasitic delays given in Table II were multiplied by 63.5 ns. These results are given in Table V.

TABLE V
CALCULATED PROPAGATION DELAYS IN NANOSECONDS

	A to Sum (ns)	A to Carry (ns)
t_{pdr}	1.596	1.024
t_{pdf}	1.143	0.733
t_{pd}	1.369	0.878

E. Complete Schematic and Layout Designs

After the previous analyses were finished, the complete full adder circuit was constructed. The schematic view was made by wiring together the symbols created for each standard cell library element. Each of these symbols refer back to their own respective schematics. The full adder schematic can be seen in Fig. 9.

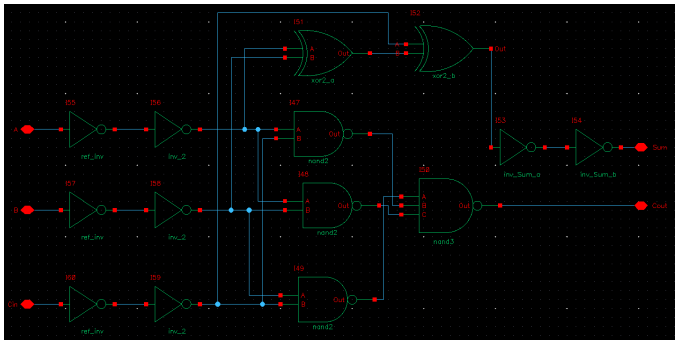


Fig. 9. Full Adder Schematic

This design was then created as a layout using 600 nm technology [1]. The layout view can be seen in Fig. 10 (see *Appendix C* for an enlarged image of the layout). As the image shows, the device is laid out with the inputs on the left and the outputs on the right. The circuit begins with the input buffers (from left to right: A, B, and C input buffers). These are followed by the three NAND2 logic gates which are followed by the NAND3 gate. At this point, the Carry output is finished and is wired on a Metal 2 track to the right edge of the design. Following the NAND3 is the first XOR2 gate which outputs to the second XOR2 gate. This ends the Sum computation which then passes through a buffer before going to the Sum output pin on the right.

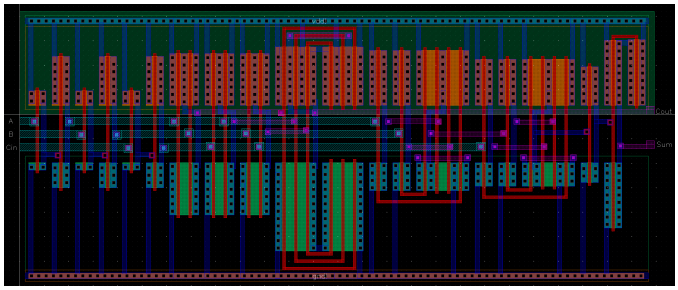


Fig. 10. Full Adder Layout

The overall dimensions of the full adder were 129.6 µm by 55.2 µm, or 432λ by 184λ. This compactness was made possible in part by the folded designs of the NAND3 gate and final inverter. Without this folding, the design would have been much larger. This layout was extracted along with its parasitic capacitances for use in simulations. The extracted layout (with the parasitic capacitors hidden for simplicity) can be seen in Fig. 11.

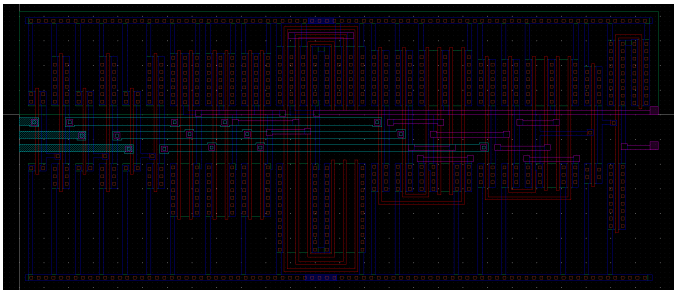


Fig. 11. Extracted Full Adder Layout

Following construction of the layout, it was compared to the full adder schematic by means of the LVS tool to ensure that their netlists did match. After making some minor corrections, the netlists did in fact match. (A screen capture of the result can be seen in Fig. 17 in *Appendix B*.) Once this verification was complete, the testing stage began.

IV. Results and Analyses

A. Results

The testing phase began by simulating the circuit with all possible logical input values. This was achieved by setting inputs A, B, and C as pulses such that their periods were successively doubled. Input A was set to alternate between 0 V and 5 V with a period of 10 ns, a 50% duty cycle, and rise and fall times of 100 ps. Inputs B and C had the same properties, save that they had periods of 20 ns and 40 ns, respectively. This allowed for the full adder to have inputs that represented binary values 000 through 111 that incremented by 1 every 5 ns. The simulation was run with these properties for 45 ns for both the schematic-based and extracted-based designs.

In order to meet the design specifications, the full adder was tested with an inverter with 90C of capacitance at its load. Since the unit inverter was constructed of an nMOS and pMOS with widths of 1.5 µm and 3 µm, the 90C inverter was given nMOS and pMOS widths of 45 µm and 90 µm, respectively. This resulted in an inverter with 30 times as much capacitance as the unit inverter (which has 3C of capacitance). The schematic used for testing is shown in Fig. 12.

The simulation waveform from the extracted simulation is shown in Fig. 13. In the figure, the waveforms represent (from top to bottom): A, B, C, Sum, and Carry. As the figure shows, the full adder did indeed compute the logical results correctly. Additionally, the figure also shows that the Carry output was computed faster than the Sum because the Carry waveform responded faster to the inputs and appears to be placed further to the left than the Sum output waveform. This aligns well with the expectations set forth in Table V.

To better determine the delay from the simulations, a more simplistic waveform than the one shown in Fig. 13 was used. The new waveform had input C grounded while inputs A and B toggled on and off one at a time. Unlike the more comprehensive logic simulations, this one helped to isolate the various responses of the circuit because only one input

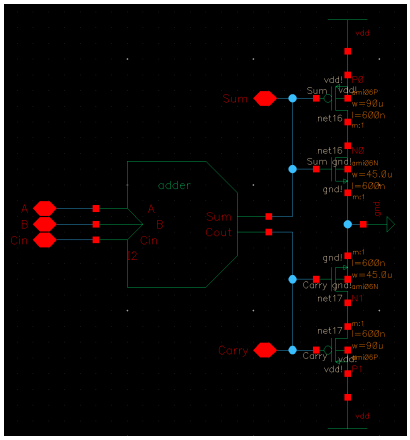


Fig. 12. Schematic Used for Simulations

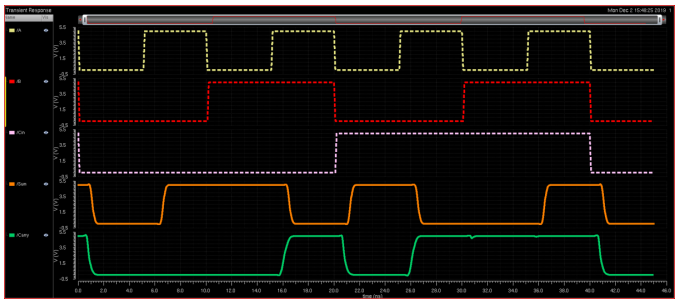


Fig. 13. Full Adder Extracted-Based Simulation (Top to Bottom: A, B, C, Sum, Carry)

toggled at a time. In this simulation, input A had a period of 10 ns with rise and fall times of 100 ps. Input B had a period of 20 ns with rise and fall times of 100 ps and a delay of 2.5 ns. The simulation was run for 25 ns. The Marker Toolbox Assistant was used to place a horizontal bar at the 2.5 V level of each waveform to easily see the time values at which the waveforms crossed the 2.5 V mark. These waveforms can be found in Fig. 14.

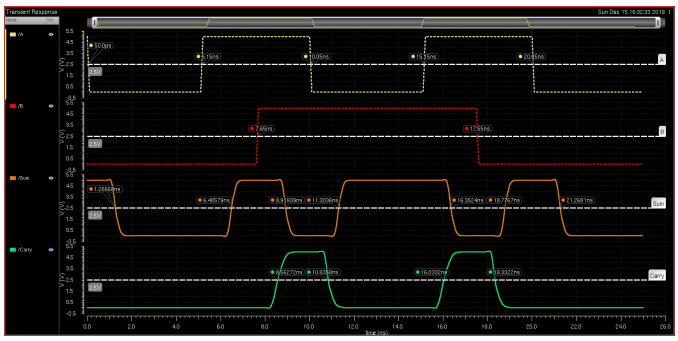


Fig. 14. Full Adder Extracted-Based Simulation with Delay Details (Top to Bottom: A, B, Sum, Carry)

This type of analysis was used to determine the propagation delays for both the extracted and schematic simulations. The results are summarized in Tables VI, VII, and VIII. In each of these cases, the delay was taken from input A to the Sum

and Carry outputs. The simulation shown in Fig. 14 was the source for each of the results.

Table VI shows the relationship between the calculated delay values and the delays taken from the schematic simulation. From this data, it can be concluded that the schematic simulation yielded smaller rising propagation delays than predicted and larger falling propagation delays than predicted. This was the case for both the Sum and Carry outputs. On average, the schematic responses were a bit slower than the calculated ones. Taking the simulation as the true value, the calculations yielded percent errors of 6.18% and 0.62% for the Sum and Carry average delays, respectively.

TABLE VI
CALCULATED DELAY VS. SCHEMATIC DELAY

	Value	Calculated Delay (ns)	Schematic Delay (ns)	Percent Error
Sum	tpdr	1.596	1.510	5.67%
	tpdf	1.143	1.409	18.88%
	Average	1.369	1.460	6.18%
Carry	tpdr	1.024	0.907	12.85%
	tpdf	0.733	0.861	14.82%
	Average	0.878	0.884	0.62%

Unlike the schematic simulations, the extracted simulations demonstrated a faster average propagation delay than the calculations. As Table VII shows, the rising propagation delay of the extracted layout proved to be notably faster than the calculated values with percent errors of 19.47% and 15.91% for the Sum and Carry calculated t_{pdr} values, respectively. While the falling propagation delays were still slower than the calculated ones, the overall average speed of the extracted device was faster than the calculations predicted.

TABLE VII
CALCULATED DELAY VS. EXTRACTED DELAY

	Value	Calculated Delay (ns)	Extracted Delay (ns)	Percent Error
Sum	tpdr	1.596	1.336	19.47%
	tpdf	1.143	1.218	6.17%
	Average	1.369	1.277	7.24%
Carry	tpdr	1.024	0.883	15.91%
	tpdf	0.733	0.786	6.70%
	Average	0.878	0.835	5.26%

The difference between the schematic and extracted simulations varied quite a bit, with percent errors ranging from less than 3% to just over 15% (if taking the extracted simulation as the true value). As Table VIII displays, the variations for the Sum propagation delays were greater than those associated with the delays to the Carry output. The greatest disparity was seen between the schematic and extracted falling propagation delays of the Sum output. The most similar value between the two simulations was for the rising Carry propagation delay with a percent error of only 2.71%.

While not the focus of the analysis, the delay from C to Sum was also determined. Since the path from C to Sum only involves one XOR2 gate, it was expected that the delay would be shorter than from A or B to Sum. Using the logical effort

TABLE VIII
SCHEMATIC DELAY VS. EXTRACTED DELAY

	Value	Schematic Delay (ns)	Extracted Delay (ns)	Percent Error
Sum	tpdr	1.510	1.336	13.06%
	tpdf	1.409	1.218	15.68%
	Average	1.460	1.277	14.31%
Carry	tpdr	0.907	0.883	2.71%
	tpdf	0.861	0.786	9.53%
	Average	0.884	0.835	5.92%

of the path, it was determined that the normalized average propagation delay is theoretically 1.011 ns. From the extracted simulation, this was found to be 0.973 ns, making the error only 3.9%.

Lastly, the rise and fall times of the extracted and schematic simulations were recorded and compared. The rise time (T_R) of the circuit represents the time taken for the output waveform to rise from 20% of its final value to 80% of the final value. The fall time (T_F) of the circuit is the inverse of this, measuring the time spent dropping from 80% to 20% of the starting value. The data is shown in Table IX. As the table shows, the differences between the schematic and extracted rise and fall times were relatively small for the Sum output and larger for the Carry output. Taking the extracted rise and fall times as the true values, the percent errors were between 2% and 4% for the Sum output and between 12% and 19% for the Carry output. With the exception of the Carry rise time, the extracted circuit had steeper rising and falling slopes than the schematic-based circuit.

TABLE IX
RISE AND FALL TIMES IN PICOSECONDS

		Schematic	Extracted	Percent Error
Sum	T_R (ps)	245.86	240.16	2.37%
	T_F (ps)	247.98	239.64	3.48%
Carry	T_R (ps)	317.63	393.6	19.30%
	T_F (ps)	320.9	284.4	12.83%

B. Analysis

The simulations of both the schematic-based design and the extracted (layout-based) design demonstrated both the limitations and accuracy of the effort-based delay calculations. During the simulation processes, the percent errors between simulated and calculated delays (from input A to the Sum and Carry outputs) ranged from 5.67% to 19.47%. By summarizing these results in Fig. 15, it becomes apparent that the schematic tended to produce delays higher than calculated while the extracted layout produced less delay than expected.

These differences can be attributed to a number of factors that are outlined in [2]. One parameter that is hard to account for when predicting delay of a circuit is wiring. The methods by which a circuit is constructed can vary widely among designers. The use of different metal layers, different wire lengths, and folded designs results in changes in parasitic capacitance and resistance that are hard to quantify. Since

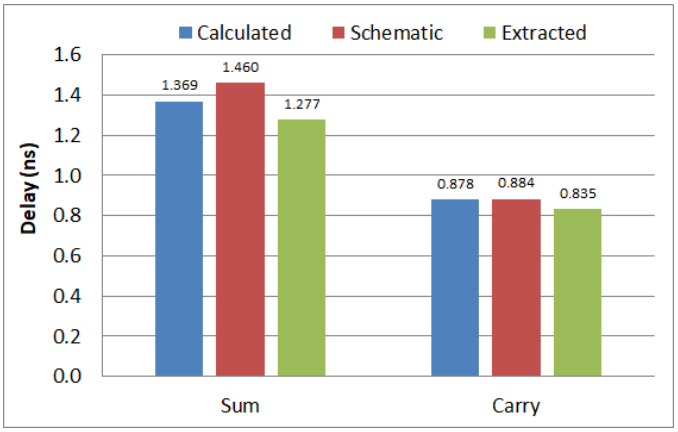


Fig. 15. Summary of Average Propagation Delay Results

the delay of a circuit is highly dependent upon the internal resistance and capacitance of the design, any changes in these parameters can help or hinder a circuit's speed. One source of discrepancy between the full adder's schematic delay and extracted delay could be the folded design used to save space when constructing the NAND3 and one of the inverters (see Fig. 7 and Fig. 10).

Another factor that may come into play is the approximations assumed in the linear delay model as a whole. As Sakurai and Newton explain in [5], placing n transistors with resistance R in series does not necessarily produce an equivalent resistance of nR. In fact, the resistance can often be less than nR. This disparity between assumptions and reality would most affect the NAND3 gate used in the full adder. The NAND3 device itself was constructed with a P:N ratio of 2:3 with the assumption that placing n transistors of resistance R in series would in fact produce nR of resistance. This principle may have led to differences not only between the extracted design and calculations, but also between the extracted design and the schematic.

In addition, the linear delay model does not account for all capacitances within each device. As [2] points out, the model does not anticipate the gate to drain capacitance (C_{gd}) that exists within transistors. This capacitance leads to an increase in delay primarily because of the bootstrapping behaviors that it produces. Bootstrapping refers to the tendency of a transistor to draw additional current to charge C_{gd} even before the output begins to fall. This results in a small voltage spike above the supply voltage value because C_{gd} briefly contributes to a charging of the output while it is still high [2]. A clear example of bootstrapping was seen when simulating the first XOR2 gate. A portion of the output waveform is shown in Fig. 16. Bootstrapping leads to an increased delay because it gives the output a higher charge just when it is set to fall. As a result, the sub-circuit's output will take longer to discharge and so produce a longer delay. While bootstrapping is not a major source of delay, its effects can still be seen by looking closely at the full adder's output waveforms in Fig. 13 and Fig. 14.

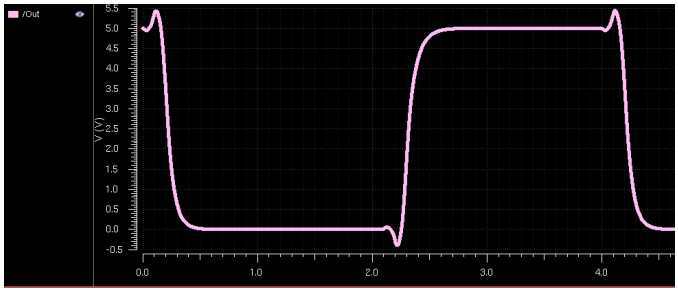


Fig. 16. Demonstration of Bootstrapping when Simulating an XOR2 Gate

Despite these small differences, the simulations were generally consistent with one another and with the calculations. The average propagation delay to the Carry output behaved the most consistently across the board. This is likely due to the uniformity of the path and the NAND gates' reduced complexity compared to the XOR gates. The overall analysis showed that the extracted simulation yielded the fastest response for both the Carry and Sum outputs.

While the Carry output did give the fastest response for each simulation, it is important to note that it can still be the bottleneck of the design if the full adder is used in a carry ripple adder. In a carry ripple adder, the Carry output of each full adder is fed as the "Carry In" (C) input of a successive full adder. As a result, each successive adder is dependent upon the previous one, leading to a multi-bit adder with a delay dependent upon the C to Carry and C to Sum propagation delay. As a result, it was important to take each output into account when optimizing the overall circuit for speed.

V. Conclusions

Ultimately, the analyses and simulations proved to resemble one another with minor differences. When comparing the average propagation delay from input A to each output, the simulation of the extracted layout performed the best with propagation delays of about 1.28 ns and 835 ps to the Sum and Carry outputs, respectively. The linear delay model was able to give reasonable estimates for each scenario and demonstrated percent errors that never exceeded 20%. While the linear delay model is simply an estimation [2], it gave insight that helped to design the optimal logical paths without performing any simulations beforehand.

During the design process, the fan-out of 4 rule was confirmed by calculations to determine the optimal number of stages for each path within the circuit. This rule - in conjunction with a logical effort analysis - allowed for the delay from the inputs to the Sum to be reduced by adding a buffer to help better drive the high output capacitance. By sizing the gates properly, the effort delay of each logic gate was distributed almost evenly among each gate to provide the shortest delay possible. Despite the numerous downsides of the linear delay model that were discussed, the calculated and simulated delays matched well. Overall, the linear delay model proved to be a very useful starting point for CMOS circuit design.

REFERENCES

[1] "SCMOS Restrictions," *mosis.com*, May 11, 2009. [Online]. Available: https://www.mosis.com/files/scmos/scmos.pdf. [Accessed Dec. 13, 2019].

[2] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA: Addison-Wesley, 2011.

[3] P. Hurley, *A Concise Introduction to Logic*, 12th ed. Stamford, CT: Cengage Learning, 2008.

[4] R. Jaeger and T. Blalock, *Microelectronic Circuit Design*, 2nd ed. New York, NY: McGraw Hill, 2004.

[5] T. Sakurai and A. R. Newton, "Delay analysis of series-connected MOSFET circuits," in *IEEE Journal of Solid-State Circuits*, vol. 26, no. 2, pp. 122-131, Feb. 1991.

VI. APPENDIX

A. Catalog of Logical Expression Symbols

TABLE X
LOGIC EXPRESSIONS

Operation	Written Expression	Symbol
NOT	not X	$\overline{X}$
AND	X and Y	XY
OR	X or Y	$X \mid\mid Y$
XOR	X xor Y	$X \oplus Y$

B. LVS Verification Screen Capture

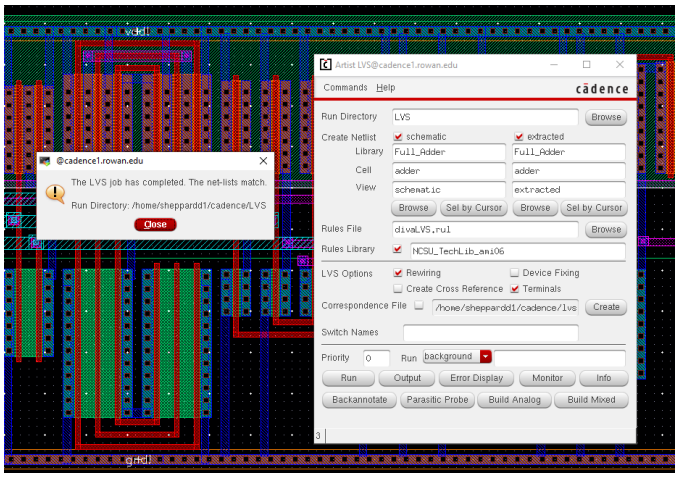


Fig. 17. Full Adder LVS Result

C. Enlarged Image of Final Circuit Layout

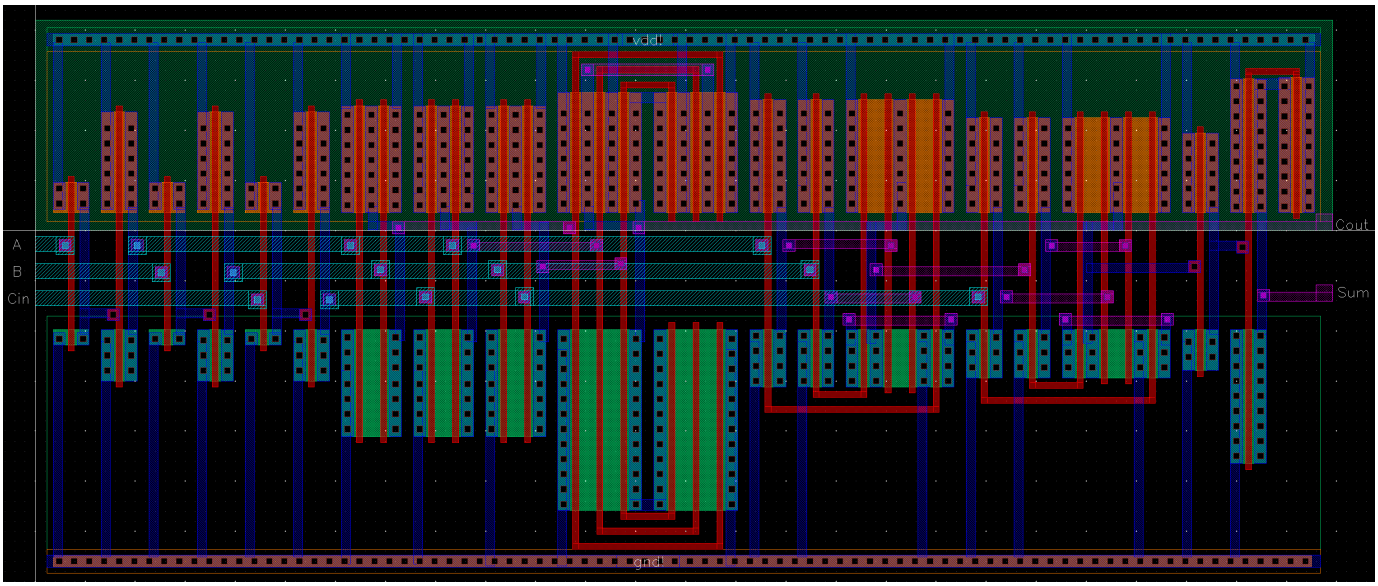


Fig. 18. Full Adder Layout

D. Enlarged Image of Extracted Layout with Labels and Parasitic Capacitors Hidden

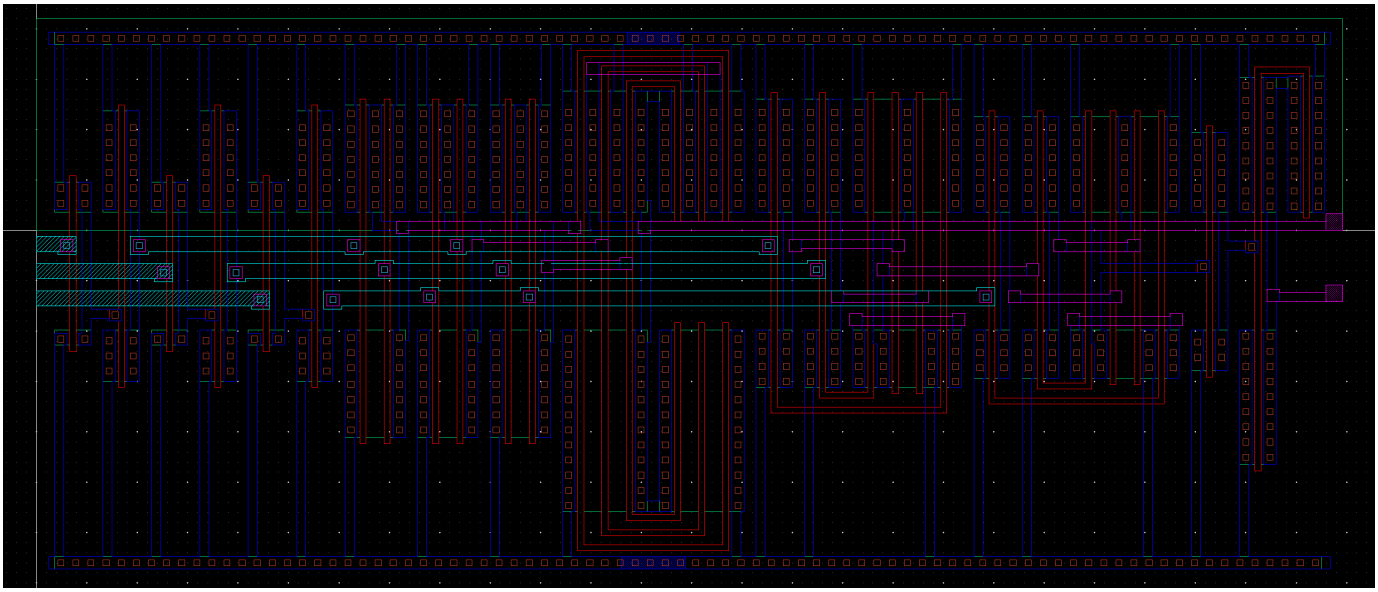


Fig. 19. Extracted Full Adder Layout