Course project: Hypothetical ALU design

All questions require the use of Cadence or LTSpice (recommended) for full/part of the question to get voltage/current. No layout is needed. For 180nm technology, use **Vdd=1.8V** (unless mentioned otherwise). Screenshot submissions of your outputs must **include date/time when screenshot is taken**. For, input signals, use a rise time of 10ps, fall time of 10ps, frequency >1GHz with 50% duty cycle. Assume *load capacitance of 1fF unless mentioned otherwise*. Circuit (adder, multiplier) diagrams are provided as an example. You are allowed to use other adder/multiplier designs also if they give correct results. If something is not clear, use your best judgement or email me to clarify ASAP.

1. Design a 4-bit register using four pseudo-static latches. Demonstrate that the latches work as intended and you are able to store data reliably.

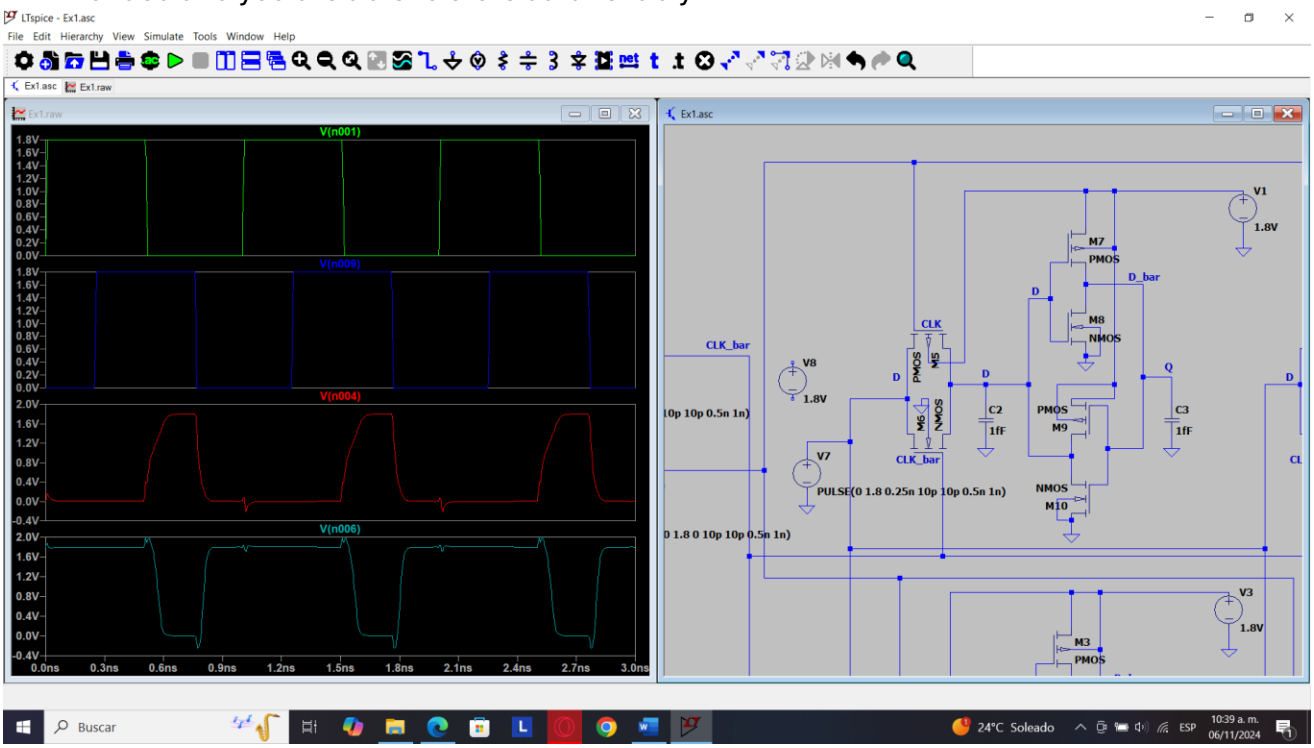


Figure 1A. One cell of 4-bit register with pseudo-static latches. Graphs from top to bottom: Clk (Top), D_input (Top middle), D_data_stored (Bottom middle), D_bar or Q (Bottom)

In Figure 1, an analysis of the 4-bit register in cell one is done, where the value that is intended to be stored in D is 0 and the value in Q is 1, the cell of the 4-bit pseudo-static latch is working as intended as can be seen in the Figure. Nonetheless, it is important to note that when the input is 1 and the clock is 0, the register tries to store the 1 value in D, however, this does not happen due to the change of the input to 0, which is then stored safely in D and Q (Q stores the inverse of 0, in this case, 1).

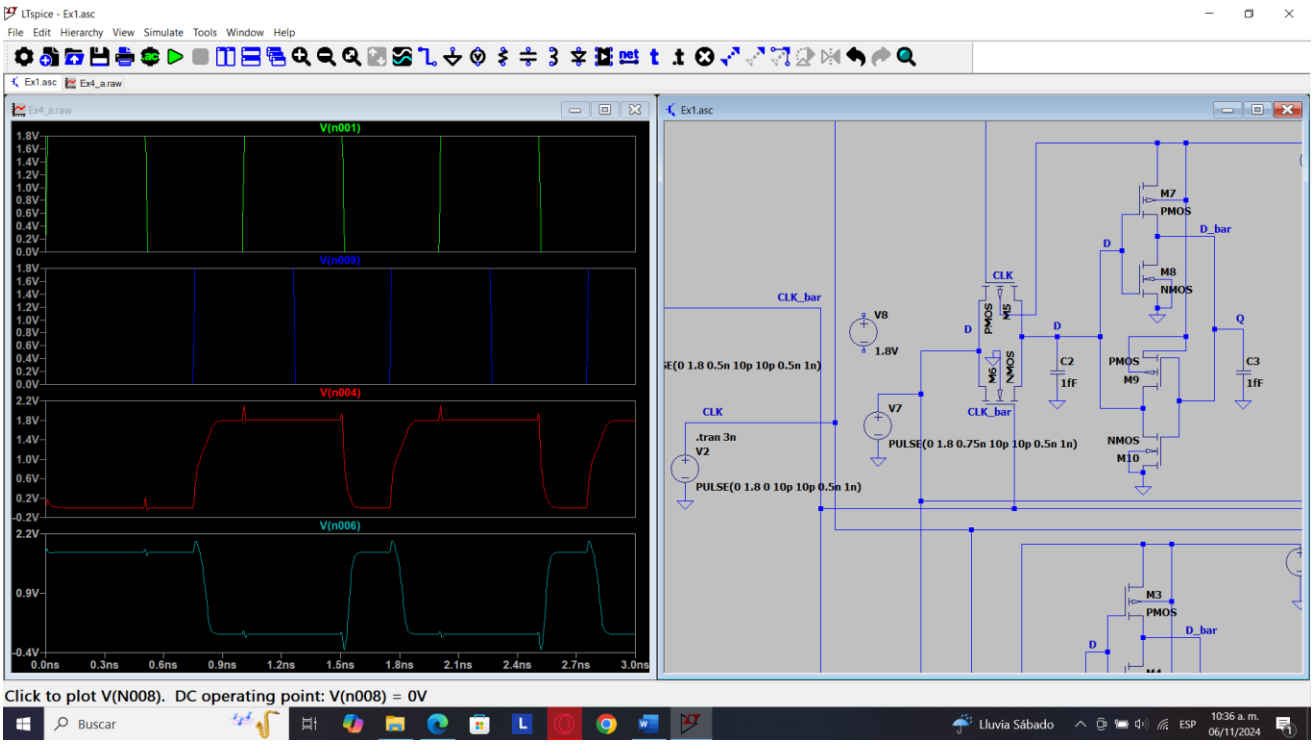


Figure 2A. One cell of 4-bit register with pseudo-static latches, when the value desired to be recorded in D is 1, and 0 in Q.

Figure 2A shows that when the input desired to be recorded in D is 1, it also functions as desired.

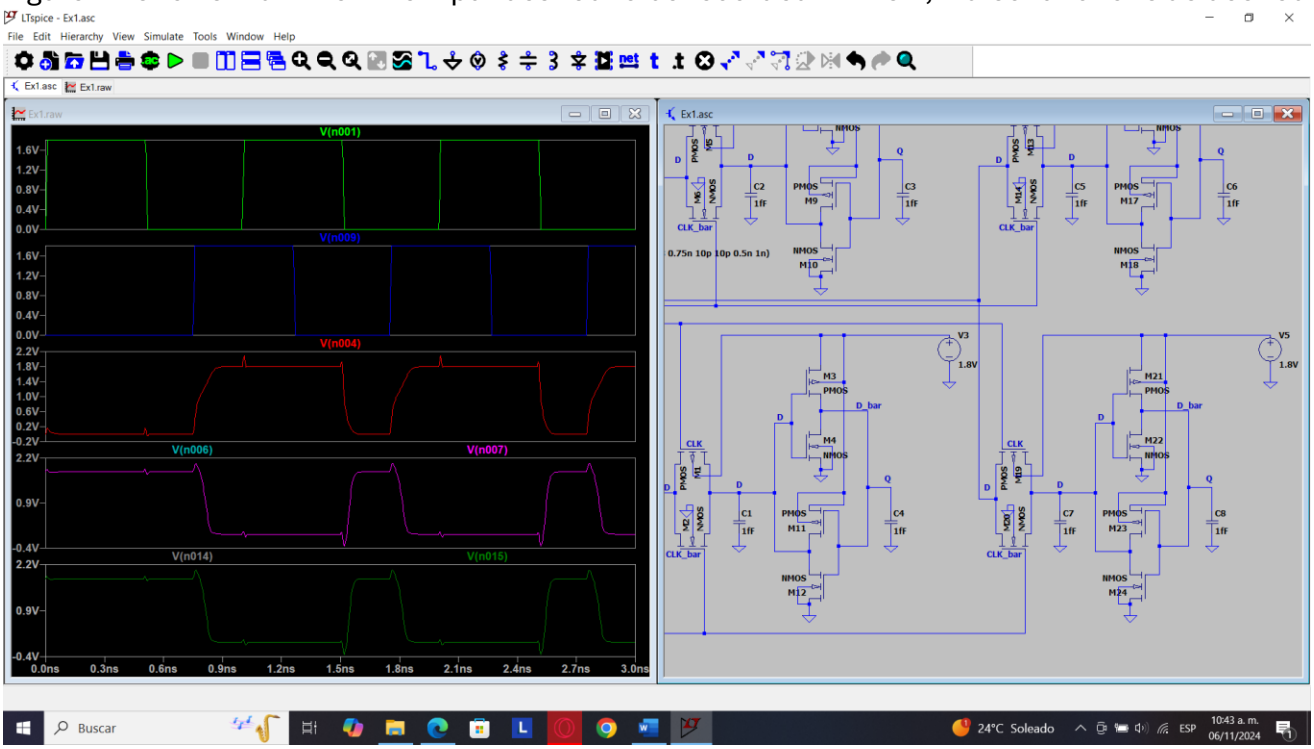


Figure 3A. The output of all pseudo-static cells (Bottom 2 graphs)

As shown in Figure 3A all 4 cells have the same output, meaning all 4-bits are being saved reliably.

2. Next, design a *half* and *full* adder using transmission gates and static CMOS. Compare the area, worst case power and worst-case performance. Which is the better design here and why?

Half adder using transmission gates:

Graph order for all following Figures in this section is Input (Top graph), S output (Middle graph), and C_out (Bottom graph)

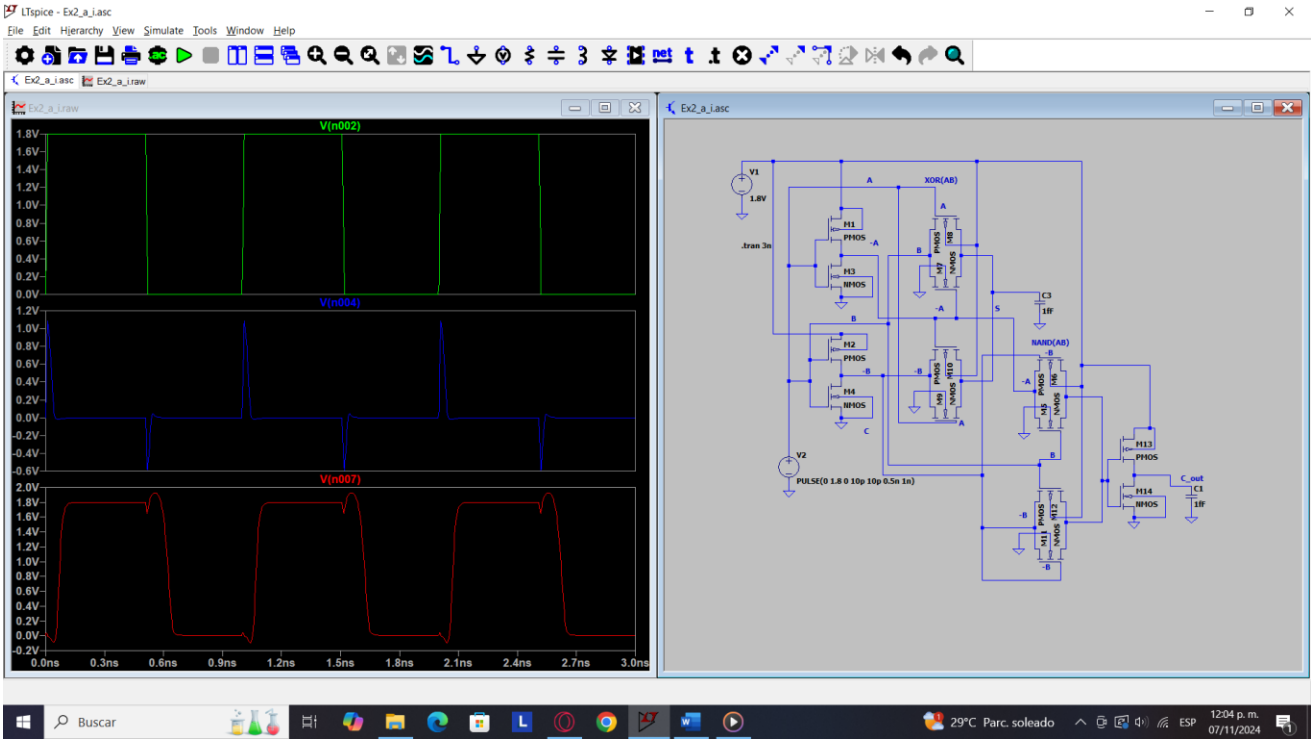


Figure 1Bi. Half adder with transmission gates with inputs A and B switching between 1 and 0.

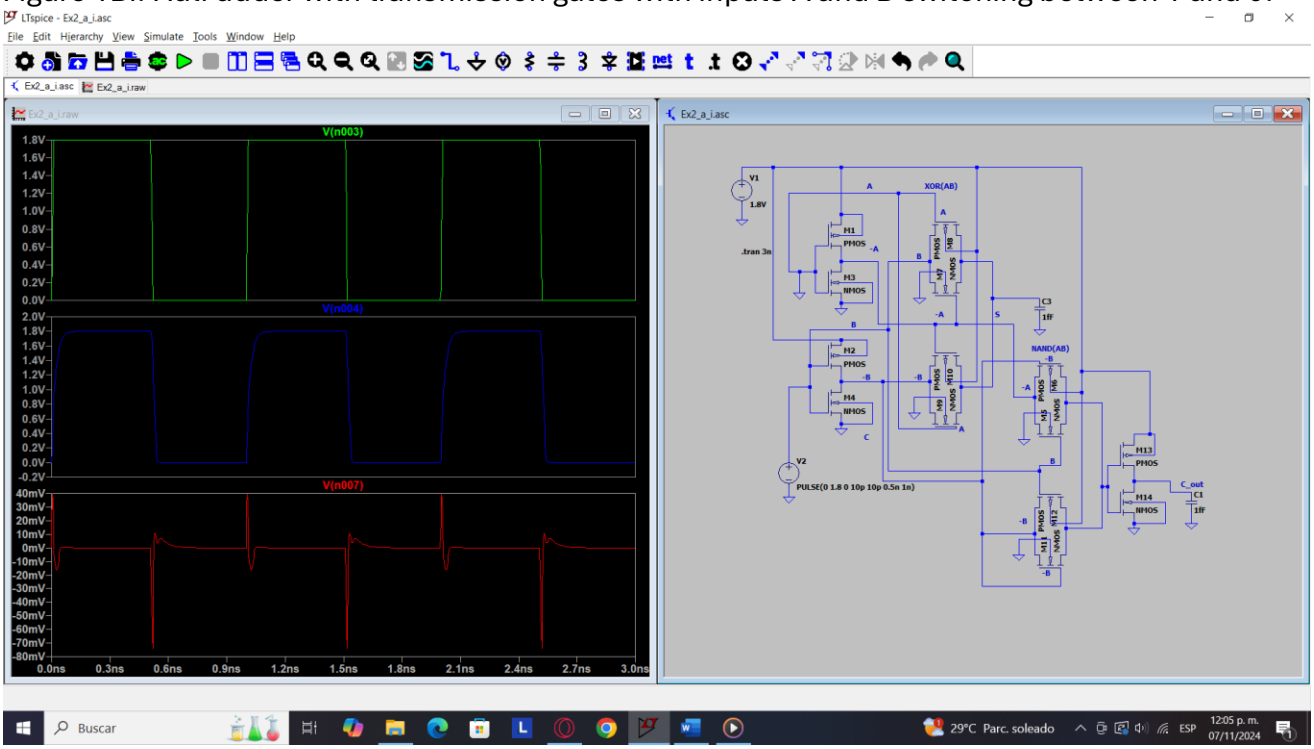


Figure 2Bi. Half adder with transmission gates with input A at 0 and B switching between 1 and 0.

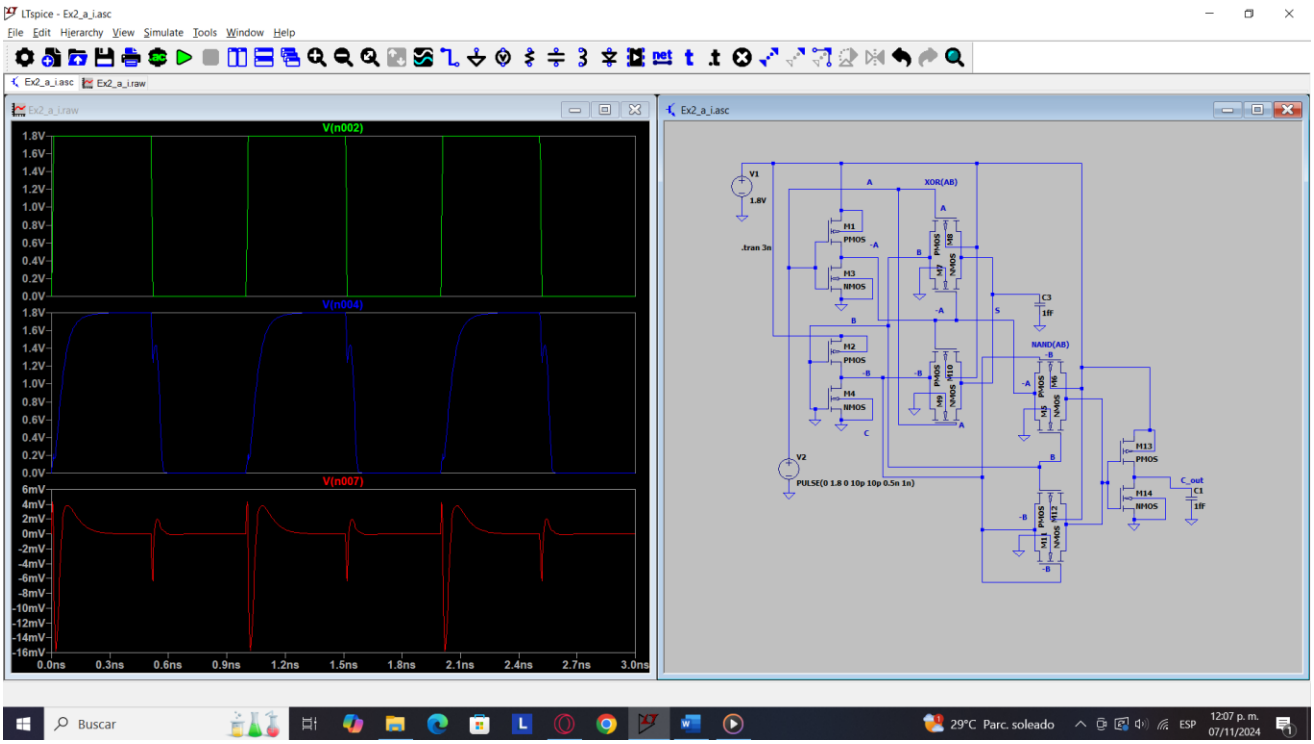


Figure 3Bi. Half adder with transmission gates with input B at 0 and A switching between 1 and 0.

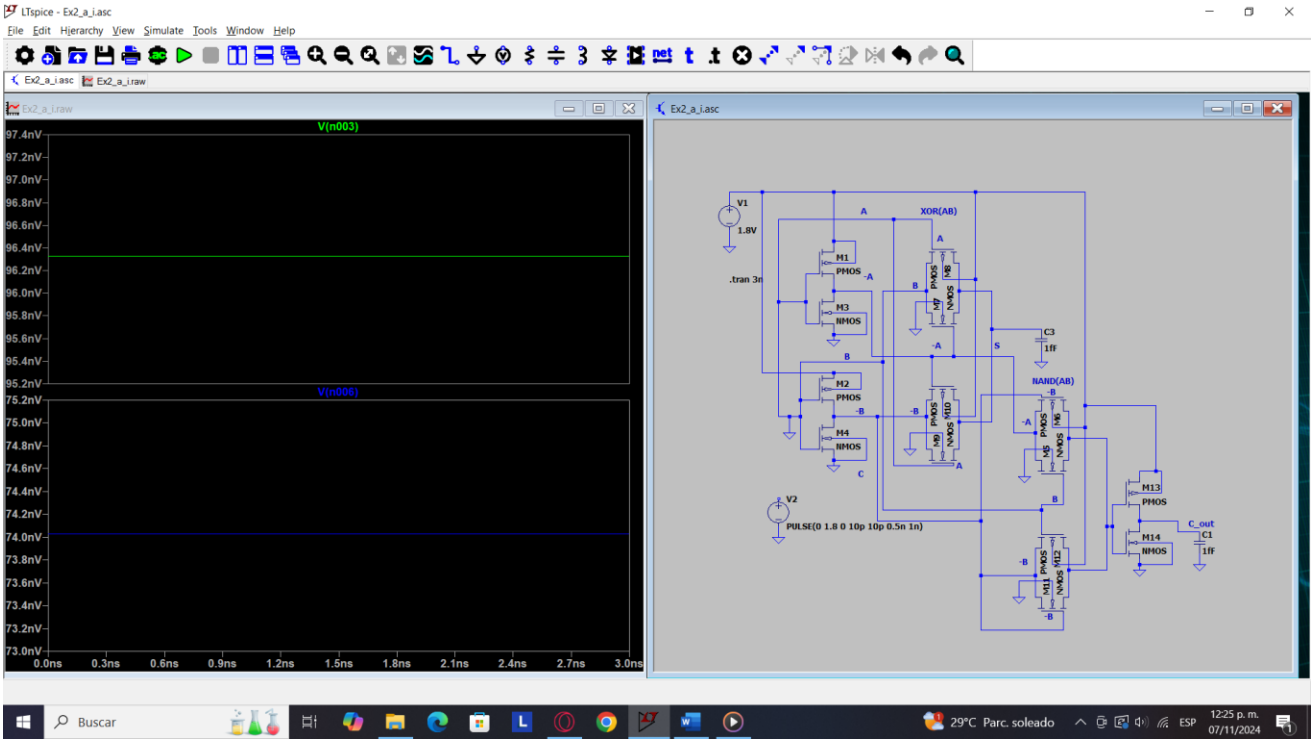


Figure 4Bi. Half adder with transmission gates with inputs A and B at 0. Graph order is S (Top graph), and C_out (Bottom graph).

Figures 1Bi to 4Bi demonstrate the correct functioning of the half adder with transmission gates.

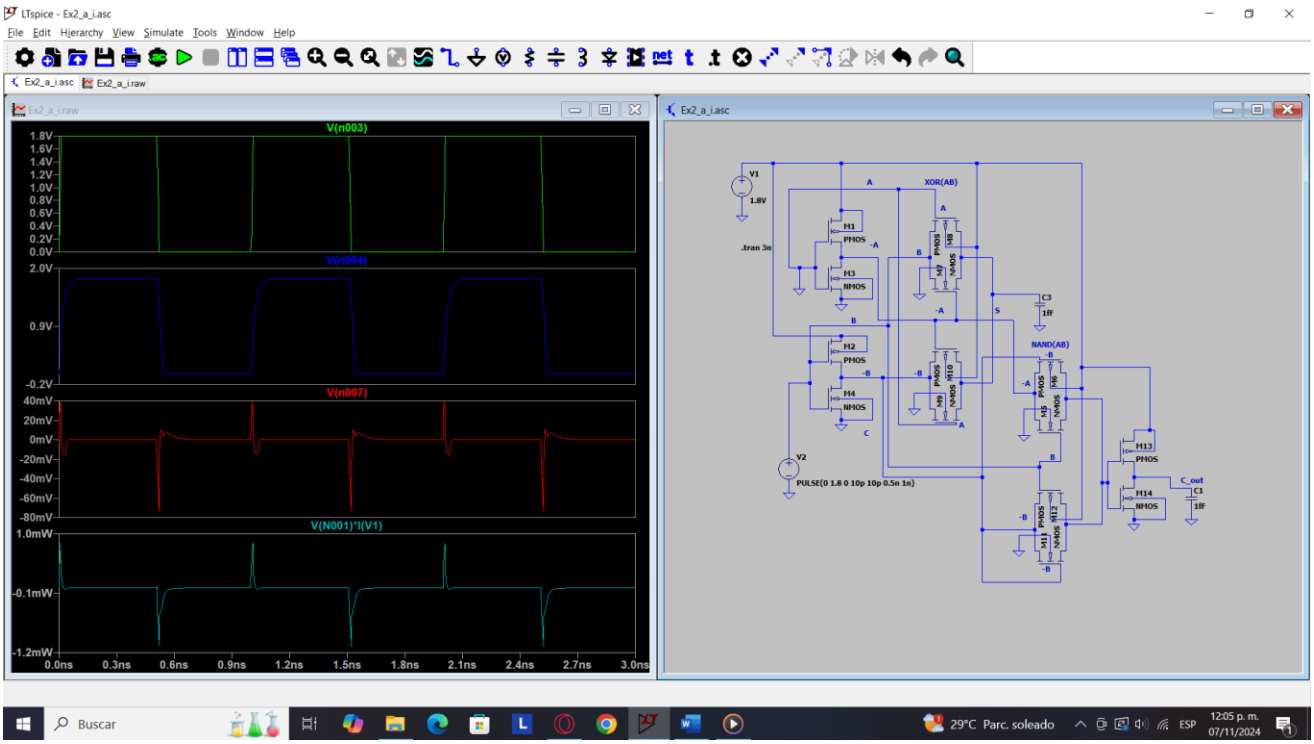


Figure 5Bi. Half adder with transmission gates with input A at 0 and B switching between 1 and 0 power consumption (Power consumption on bottom graph).

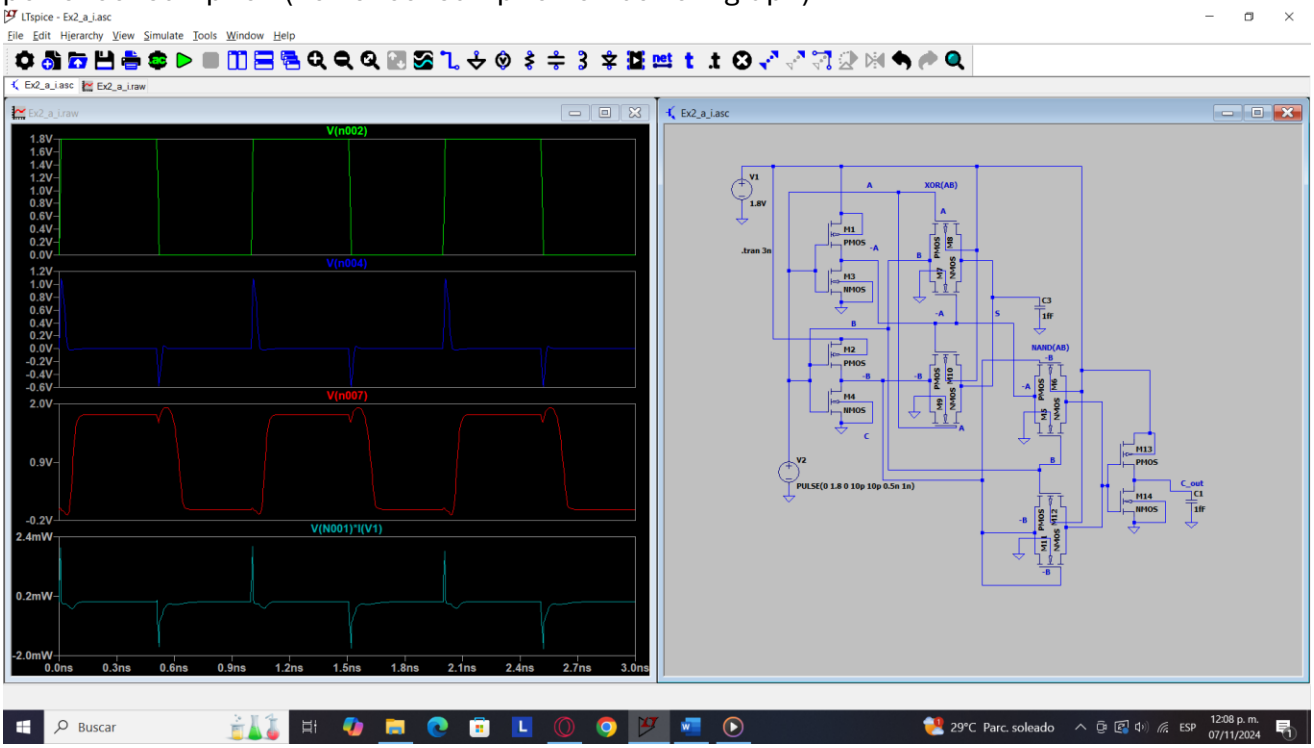


Figure 6Bi. Half adder with transmission gates with inputs A and B switching between 1 and 0 power consumption.

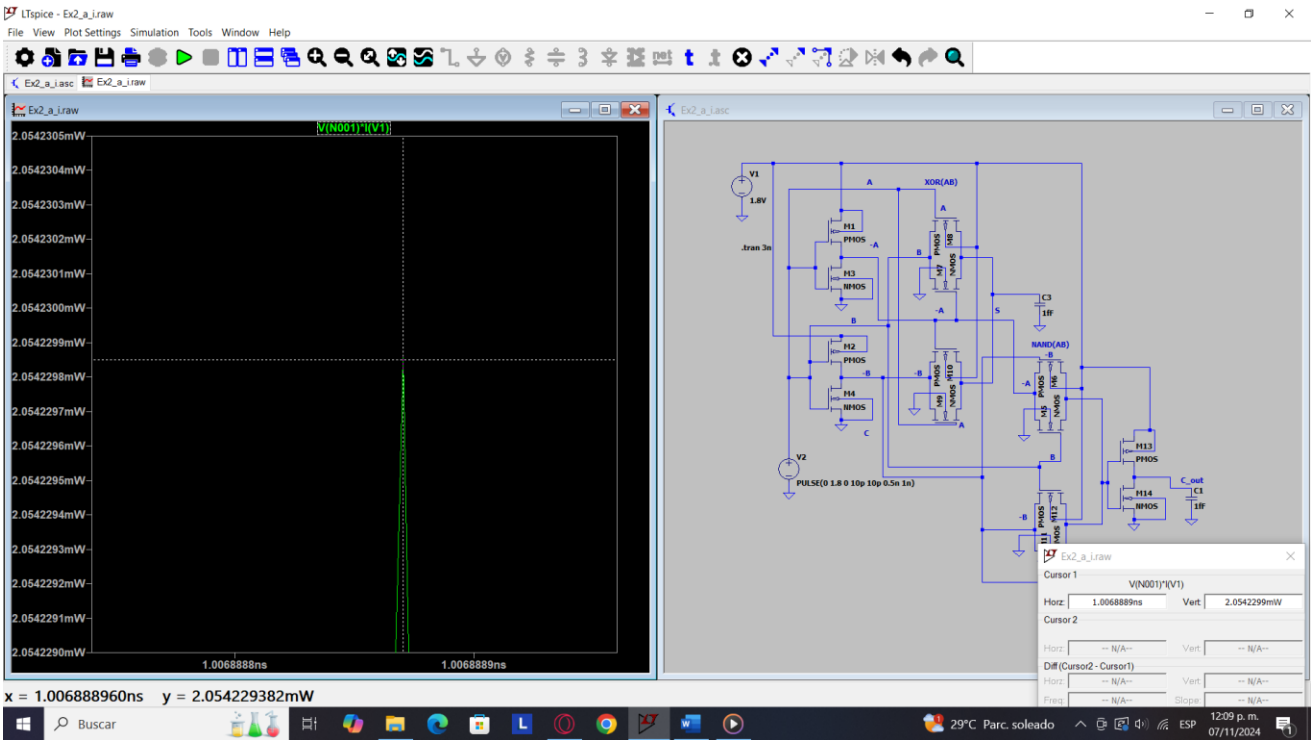


Figure 7Bi. Worst-case power consumption with inputs A and B switching between 1 and 0.

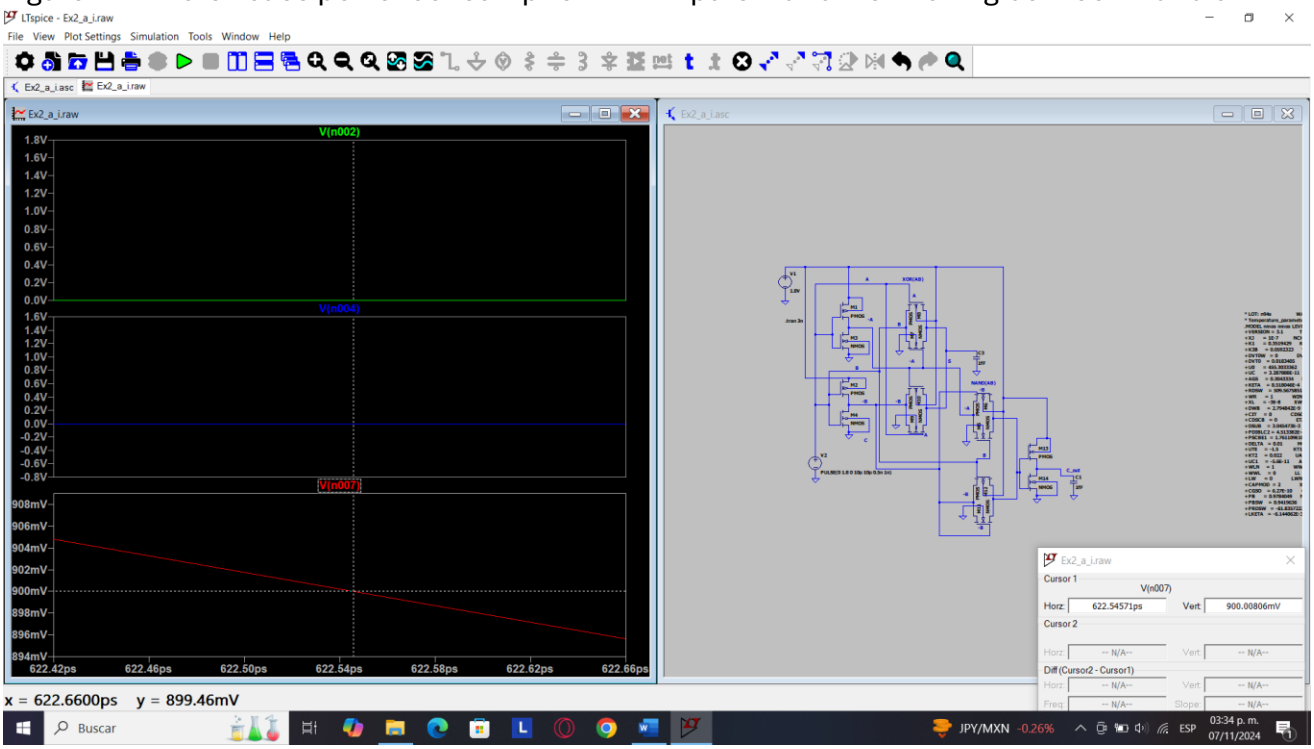


Figure 8Bi. Delay at C_out when inputs A and B switch from 1 to 0.

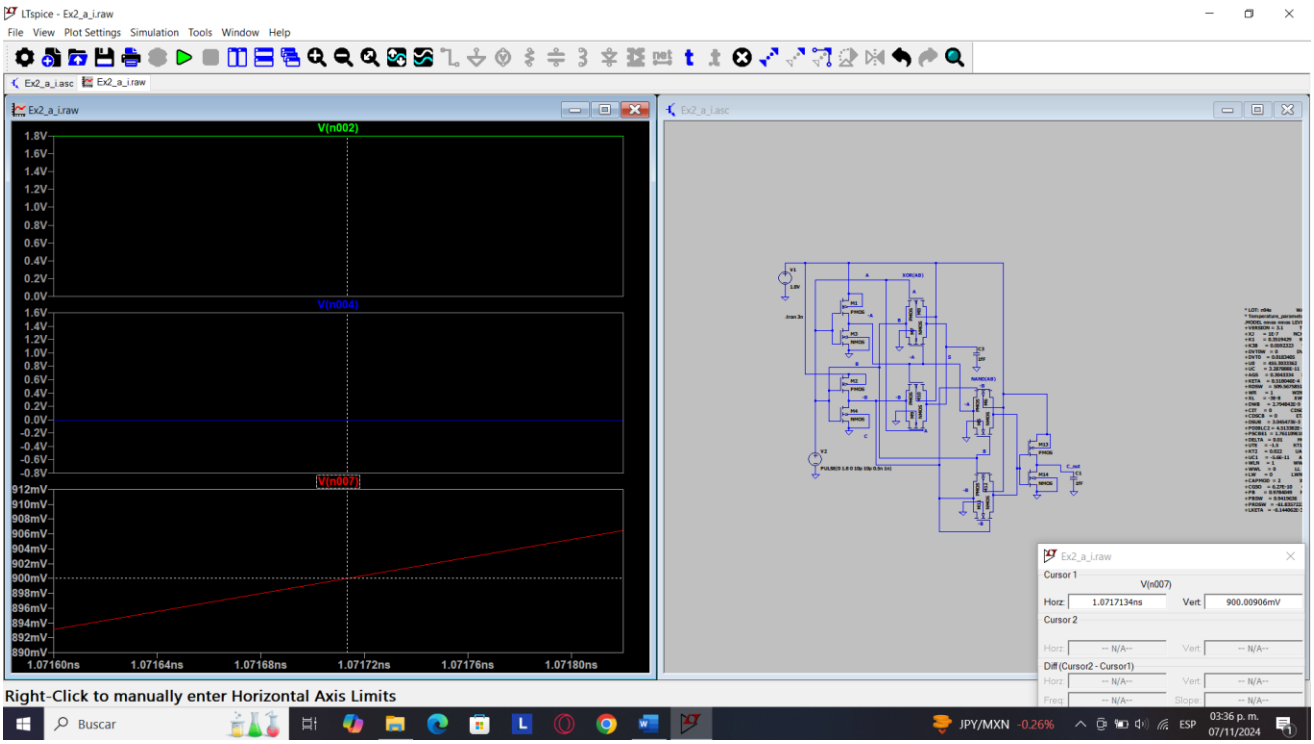


Figure 9Bi. Delay at C_out when inputs A and B switch from 0 to 1.

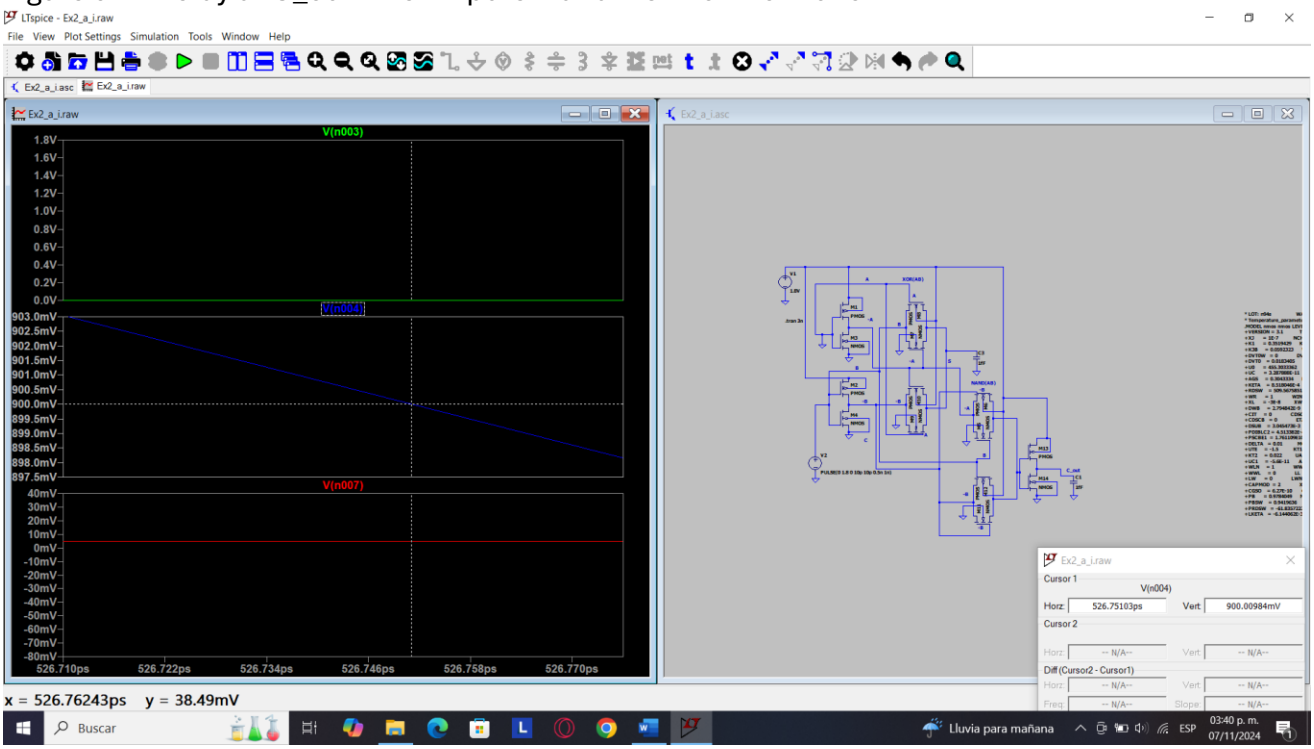


Figure 10Bi. Delay at S when input A stays at 0 and B switches from 1 to 0.

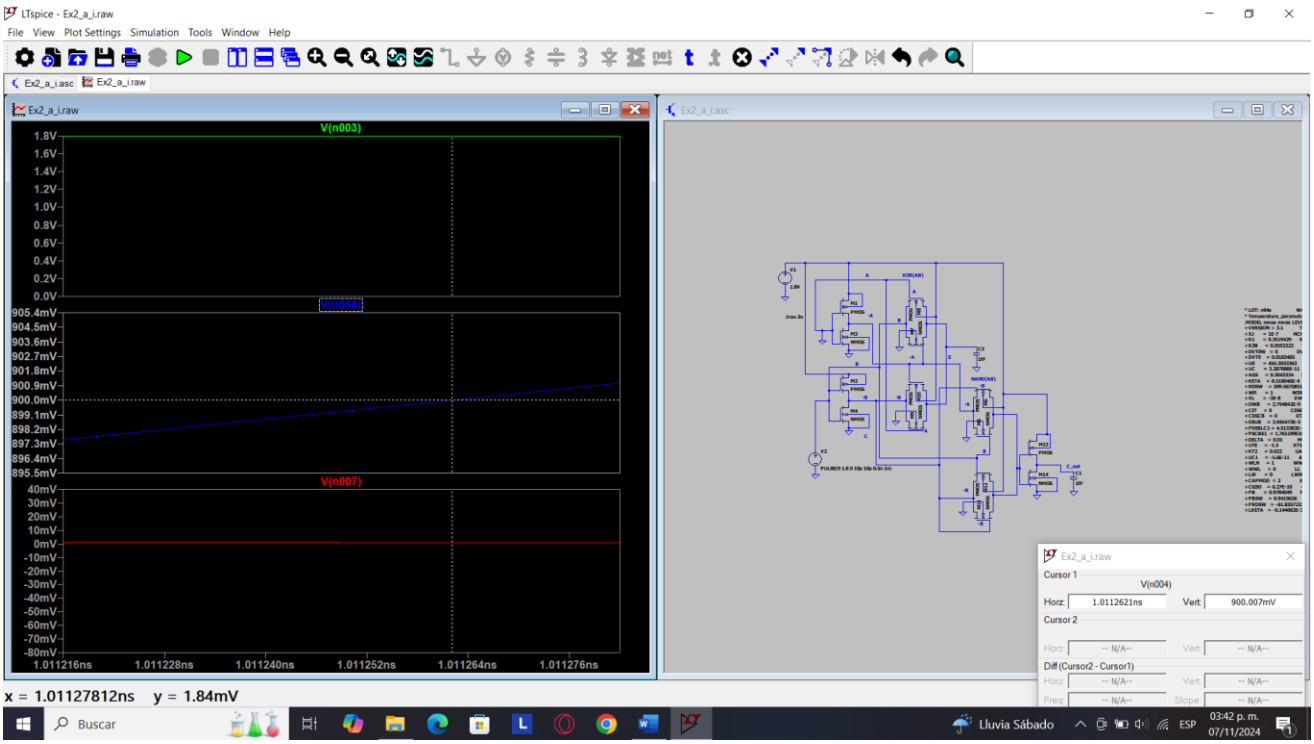


Figure 11Bi. Delay at S when input A stays at 0 and B switches from 0 to 1.

$$t_0 = 0.515\ ns$$
$$t_1 = 0.62254571\ ns$$
$$t_2 = 1.005\ ns$$
$$t_3 = 1.0717134\ ns$$
$$Worst\ Case\ Propagation\ delay =$$
$$t_{phl} = t_1 - t_0 = 0.62254571\ ns - 0.515\ ns = 0.10754571\ ns$$
$$t_{plh} = t_3 - t_2 = 1.0717134\ ns - 1.005\ ns = 0.0667134\ ns$$
$$t_p = \frac{t_{phl} + t_{plh}}{2} = (0.10754571\ ns + 0.0667134\ ns)/2 = 0.087129555\ ns$$

The worst-case propagation delay for charging and discharging is when A and B are changing from 0 to 1 to 0, as can be seen from Figures 8Bi and 9Bi.

$$Area =$$
$$A = A_{PMOS} + A_{NMOS} = 3 * 180 * 880 + 11 * 180 * 440$$
$$= 1{,}346{,}400\ nm^2$$
$$Worst\ Case\ Power = 2.0542\ mW$$

The worst power consumption scenario is when A and B are both switching from 0 to 1 because it is the situation with most transistors with changing input, which leads to dynamic power consumption (Seen in Figures 6Bi and 7Bi).

Full adder using transmission gates:

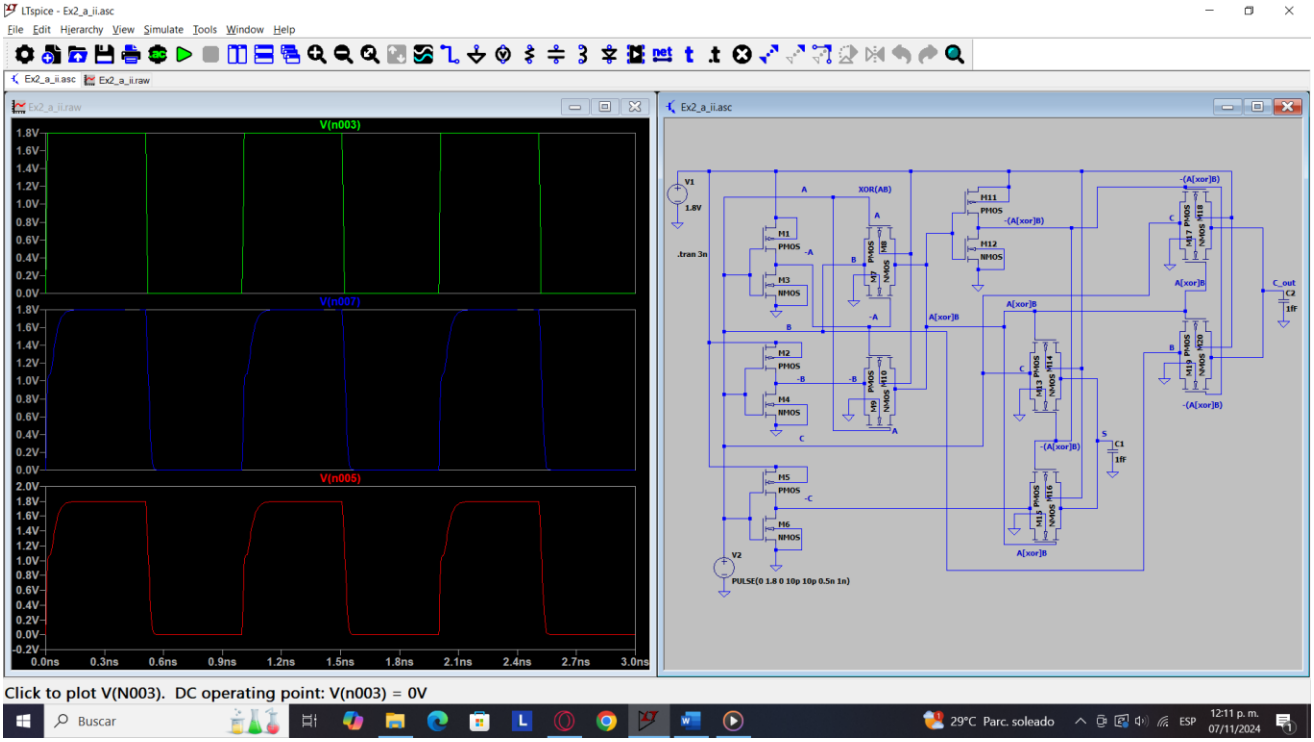


Figure 1Bii. Full adder with transmission gates with inputs A, B, and C switching between 1 and 0. Graph order goes as follows: Input (Top graph), Sum graph (Middle graph), and C_out graph (Bottom graph).

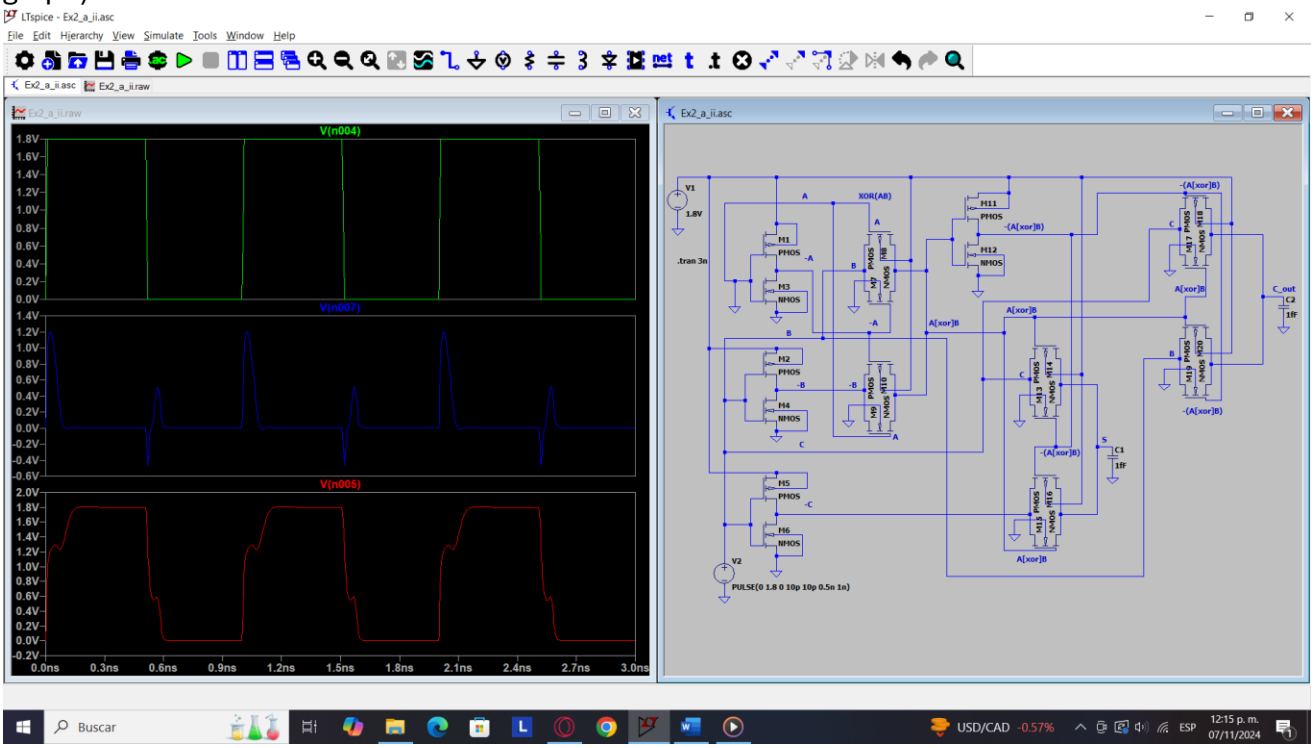


Figure 2Bii. Full adder with transmission gates with input A at 0, and inputs B and C switching between 1 and 0.

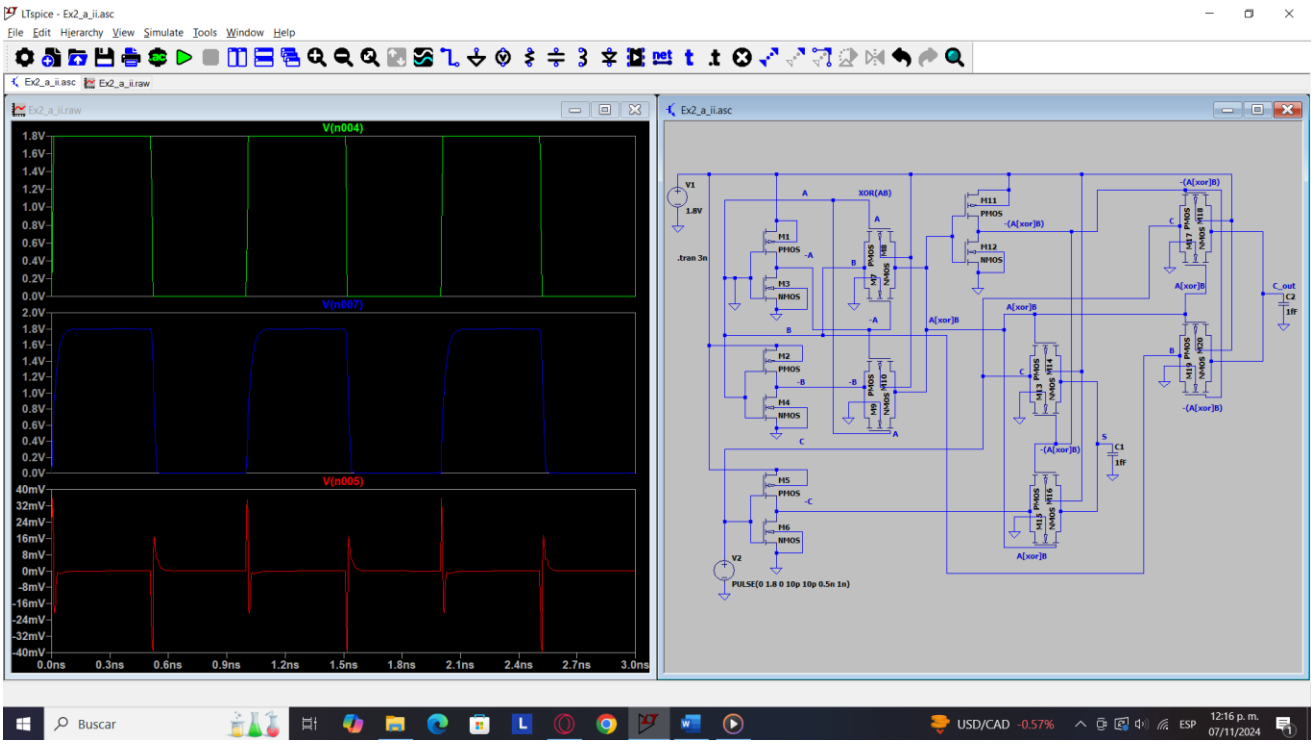


Figure 3Bii. Full adder with transmission gates with input A and B at 0, and C switching between 1 and 0.

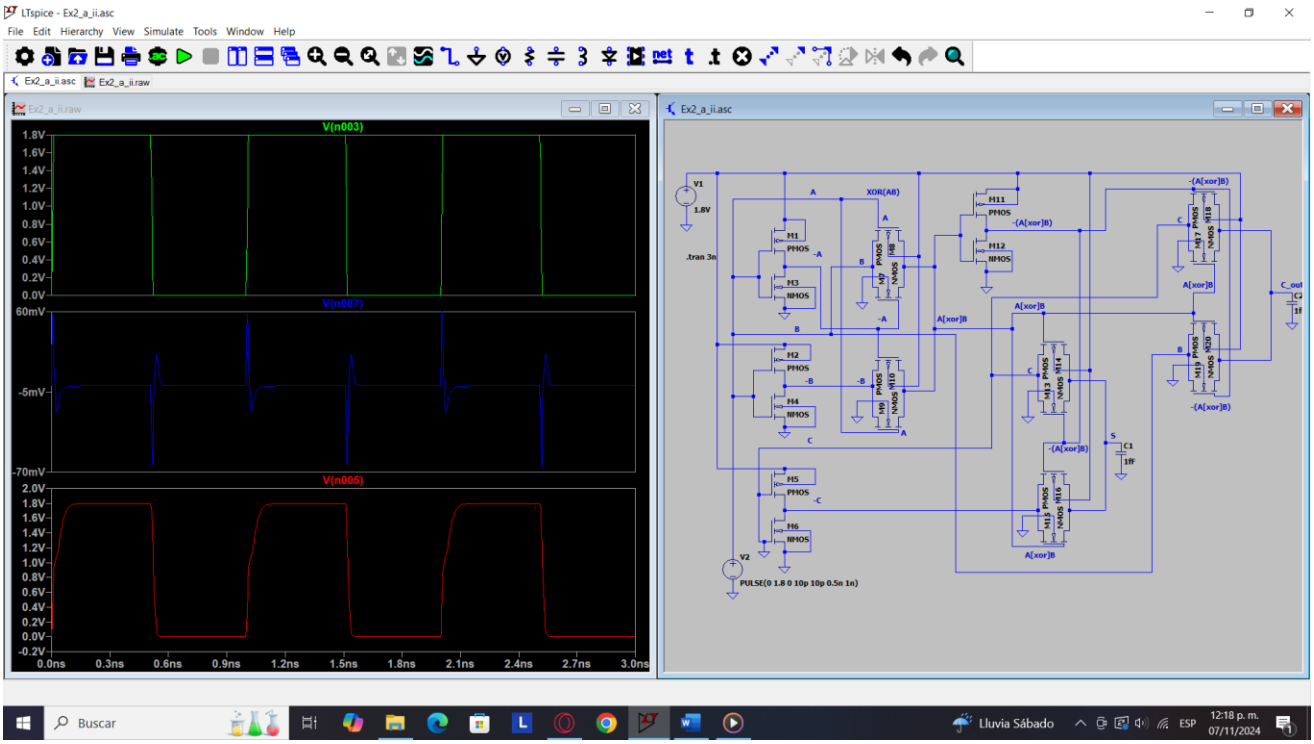


Figure 4Bii. Full adder with transmission gates with input C at 0, and inputs B and A switching between 1 and 0.

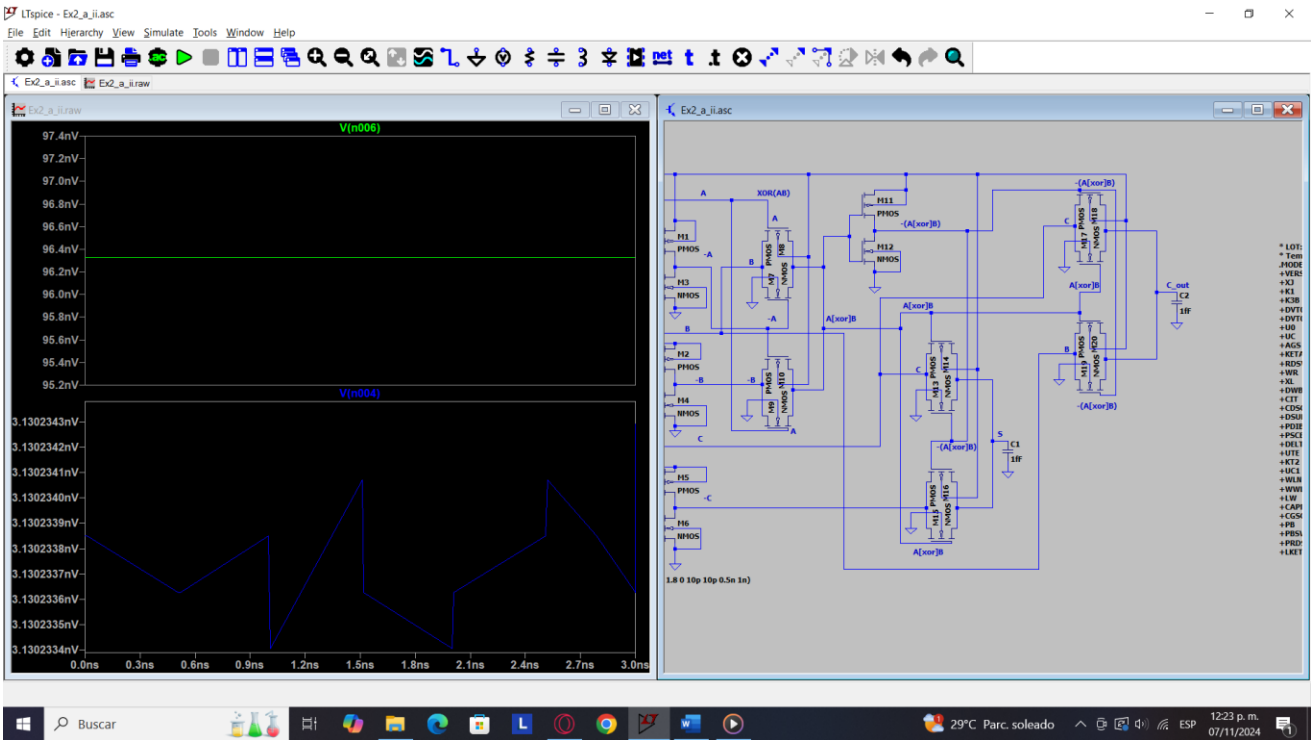


Figure 5Bii. Full adder with transmission gates with inputs A, B, and C at 0. Graph order: Sum (Top graph) and C_out (Bottom graph).

Figures 1Bii to 5Bii demonstrate the correct functioning of the full adder with transmission gates.

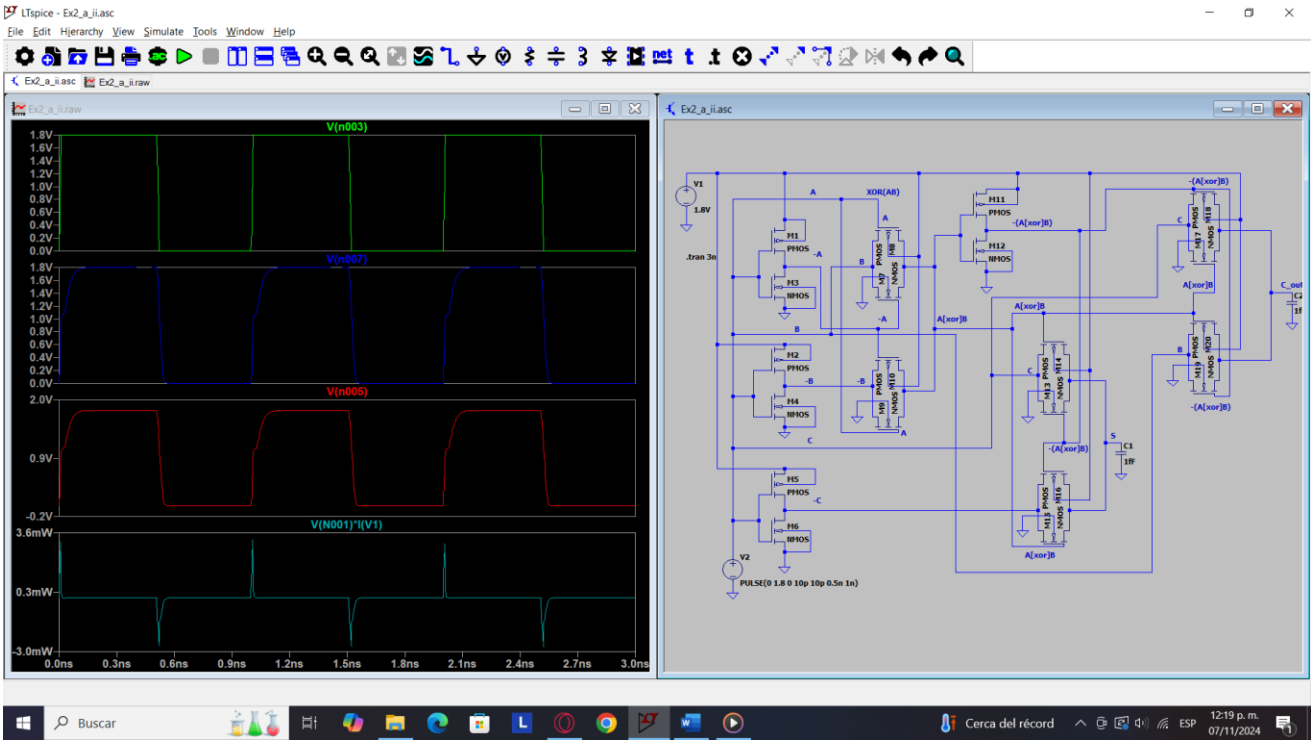


Figure 6Bii. Full adder with transmission gates with inputs A, B, and C switching between 1 and 0. Power consumption is the bottom graph (Worst case power consumption).

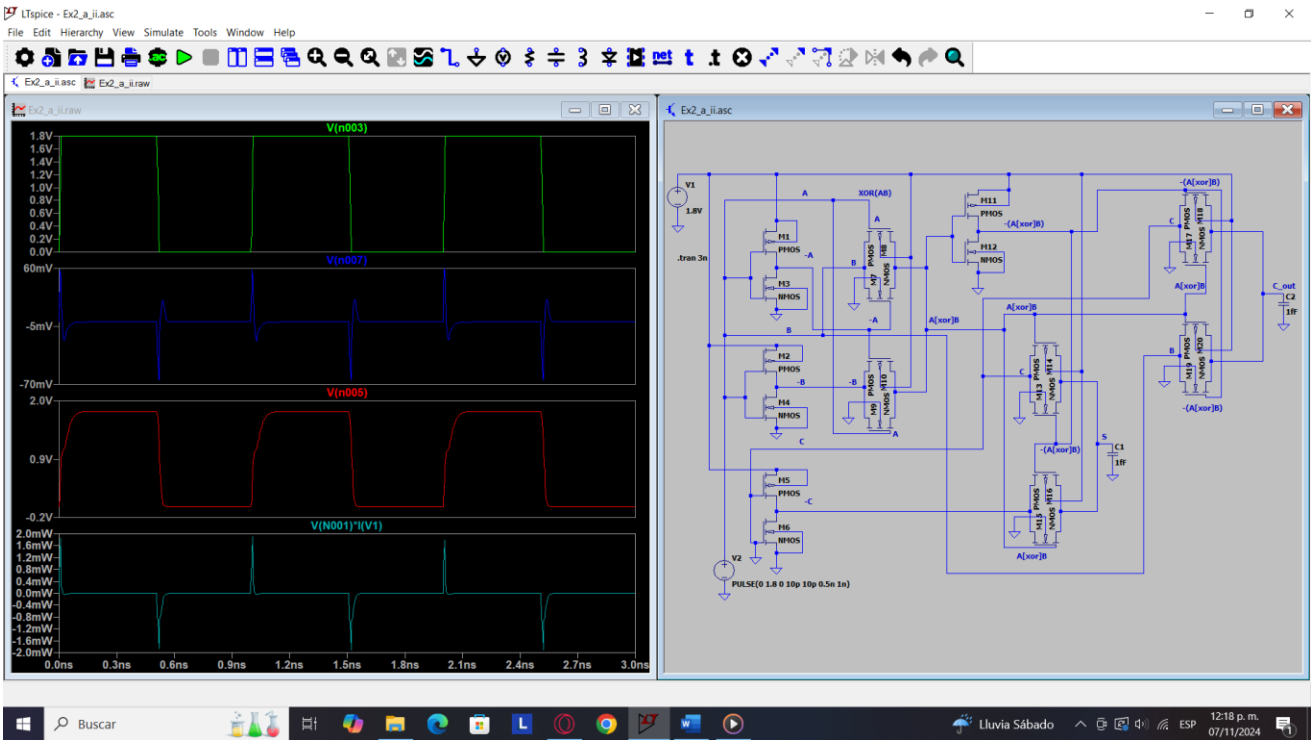


Figure 7Bii. Full adder with transmission gates with input C at 0, and inputs B and A switching between 1 and 0. Power consumption is the bottom graph.

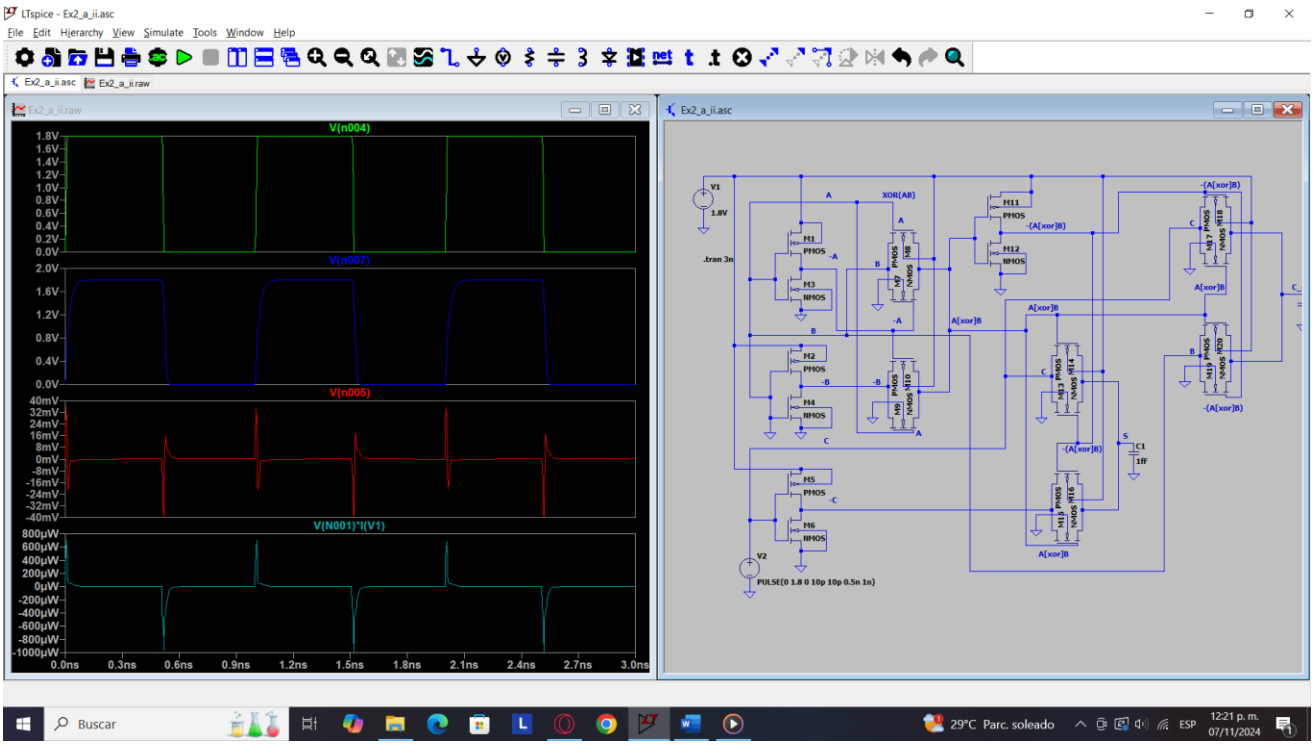


Figure 8Bii. Full adder with transmission gates with input A and B at 0, and C switching between 1 and 0. Power consumption is the bottom graph.

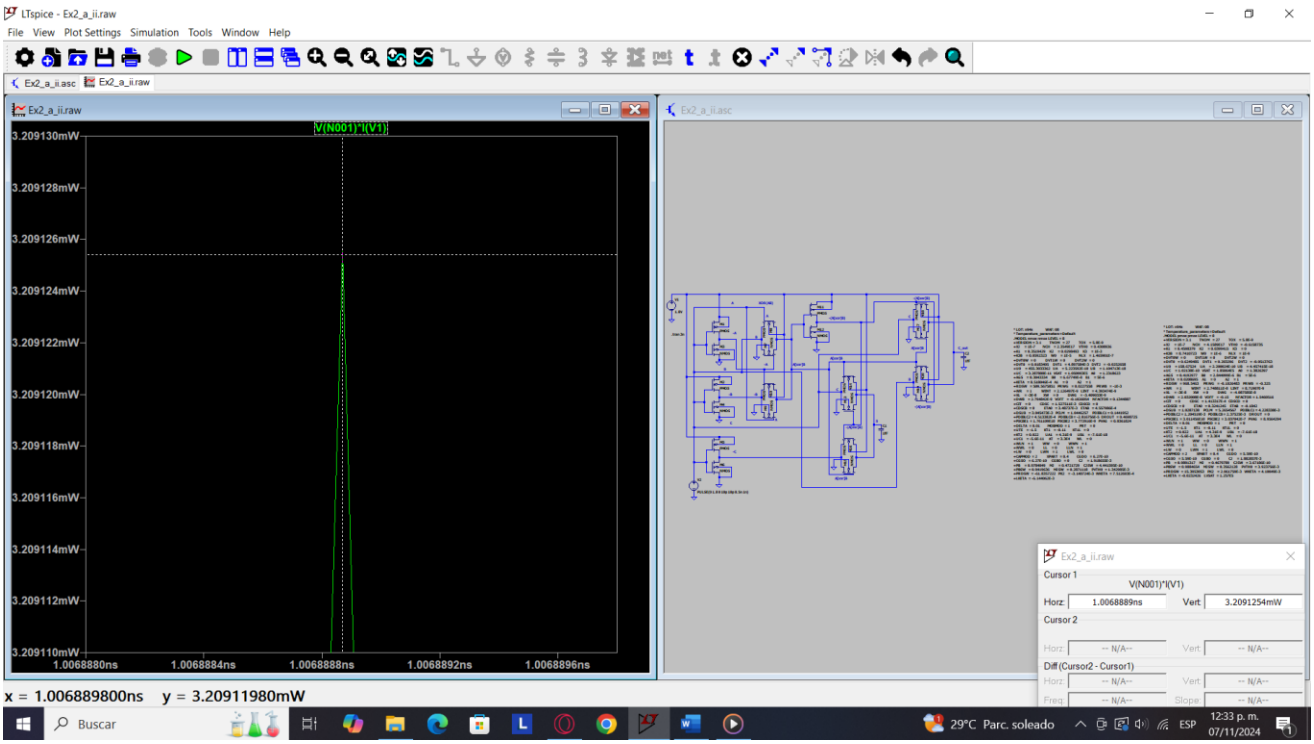


Figure 9Bii. Worst-case power consumption with inputs A, B, and C switching between 1 and 0.

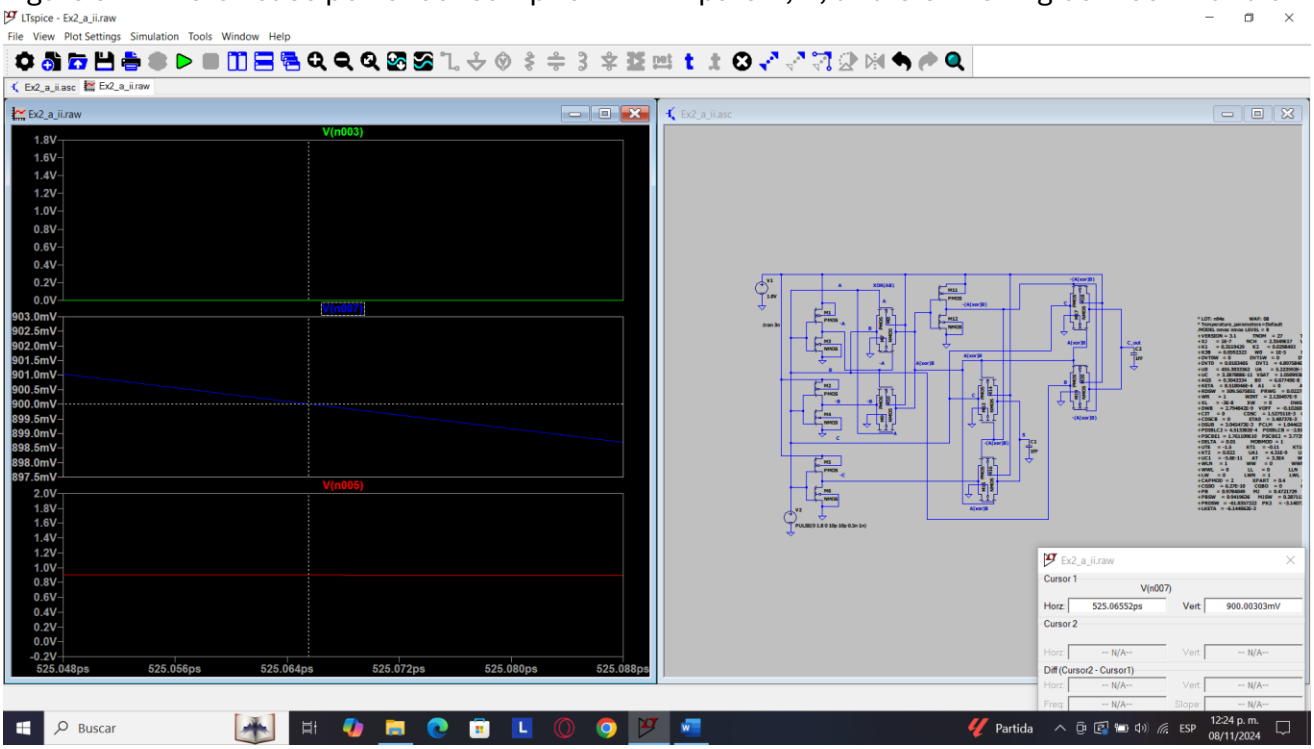


Figure 10Bii. Delay at S when A, B and C switch from 1 to 0. Full adder with transmission gates.

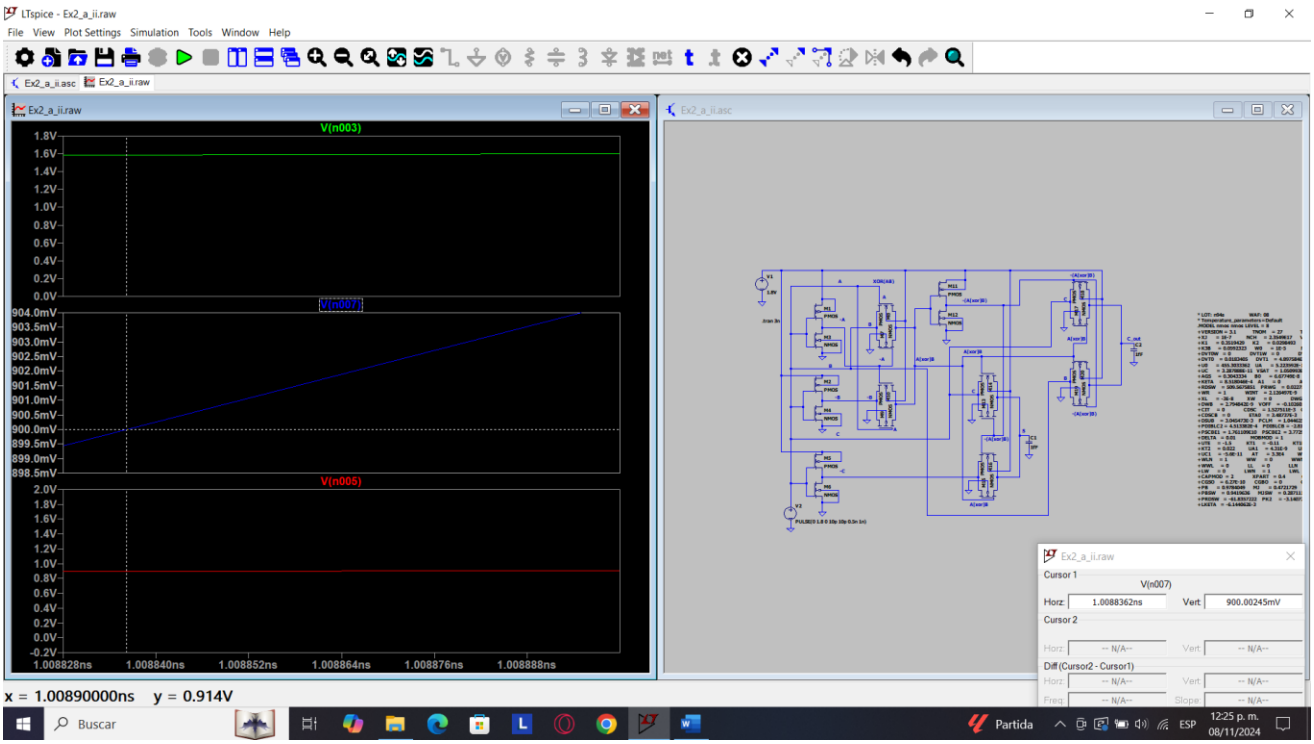


Figure 11Bii. Delay at S when A, B and C switch from 0 to 1. Full adder with transmission gates.

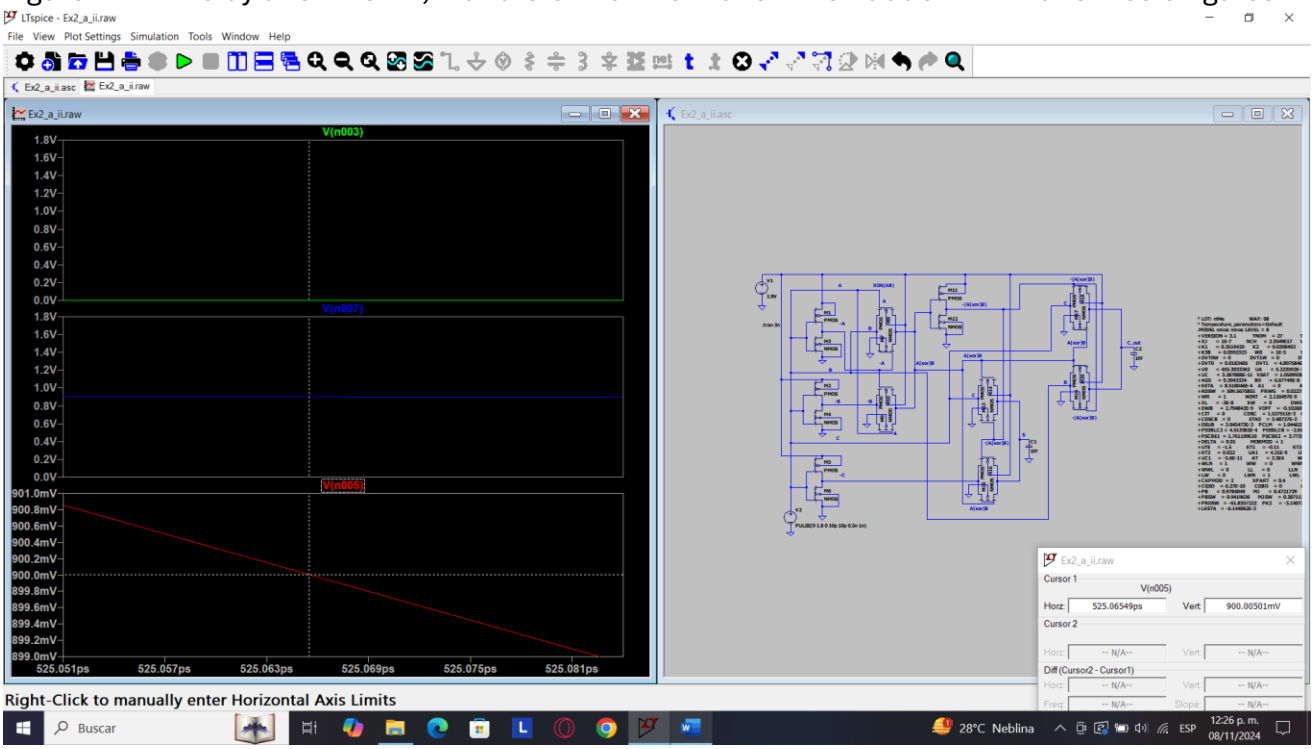


Figure 12Bii. Delay at C_out when A, B and C switch from 1 to 0. Full adder with transmission gates.

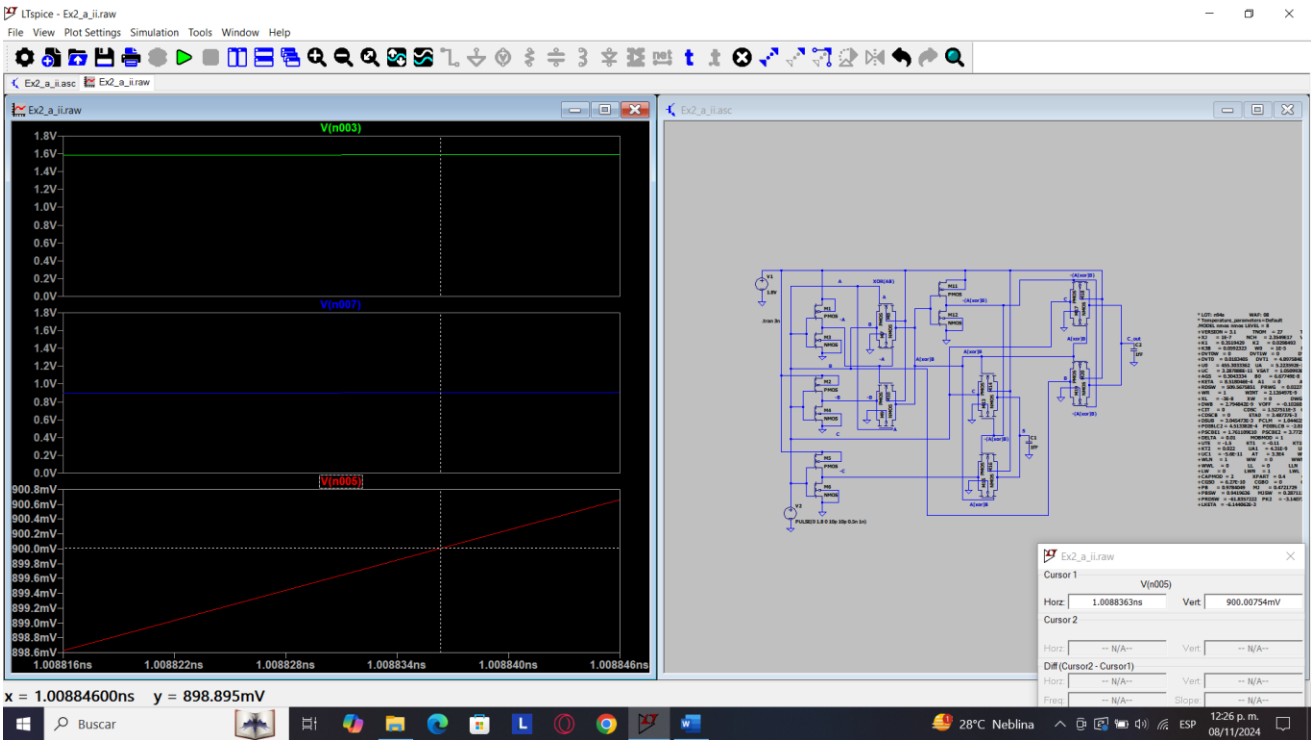


Figure 13Bii. Delay at C_out when A, B and C switch from 0 to 1. Full adder with transmission gates.

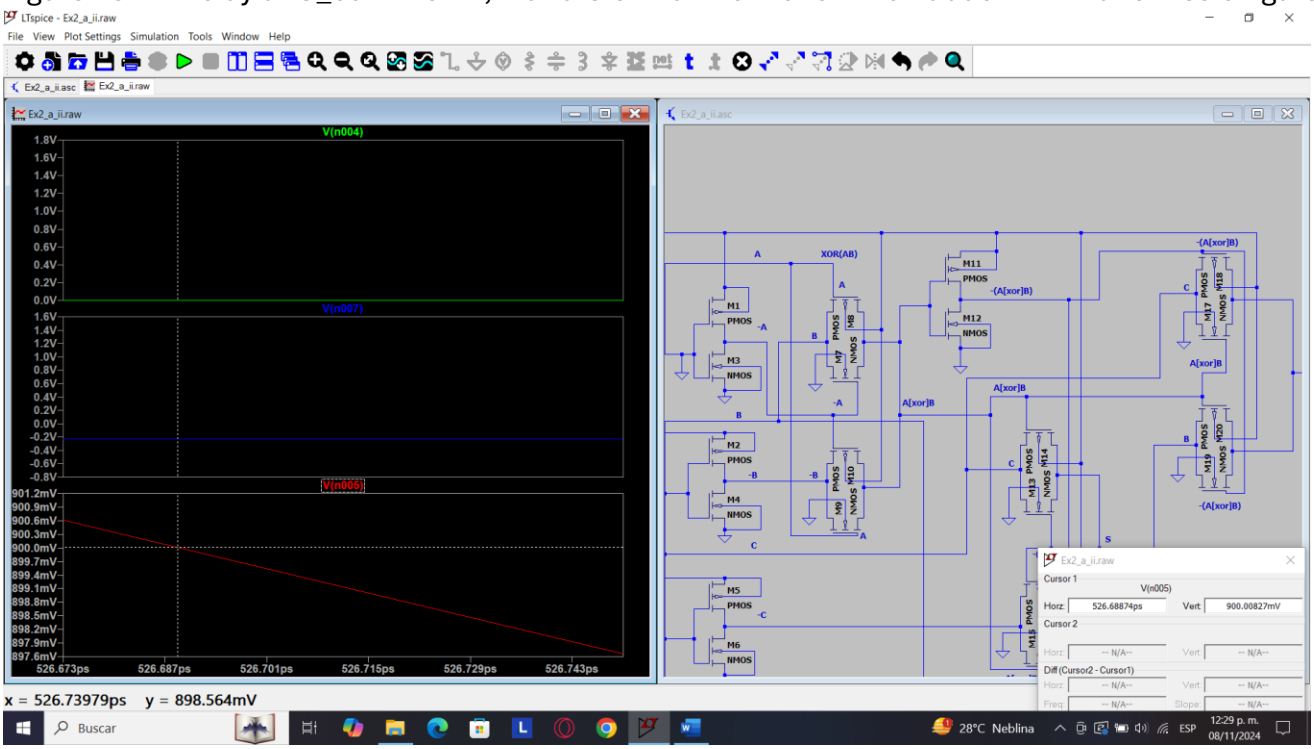


Figure 14Bii. Delay at C_out when A is at 0, and B and C switch from 1 to 0. Full adder with transmission gates.

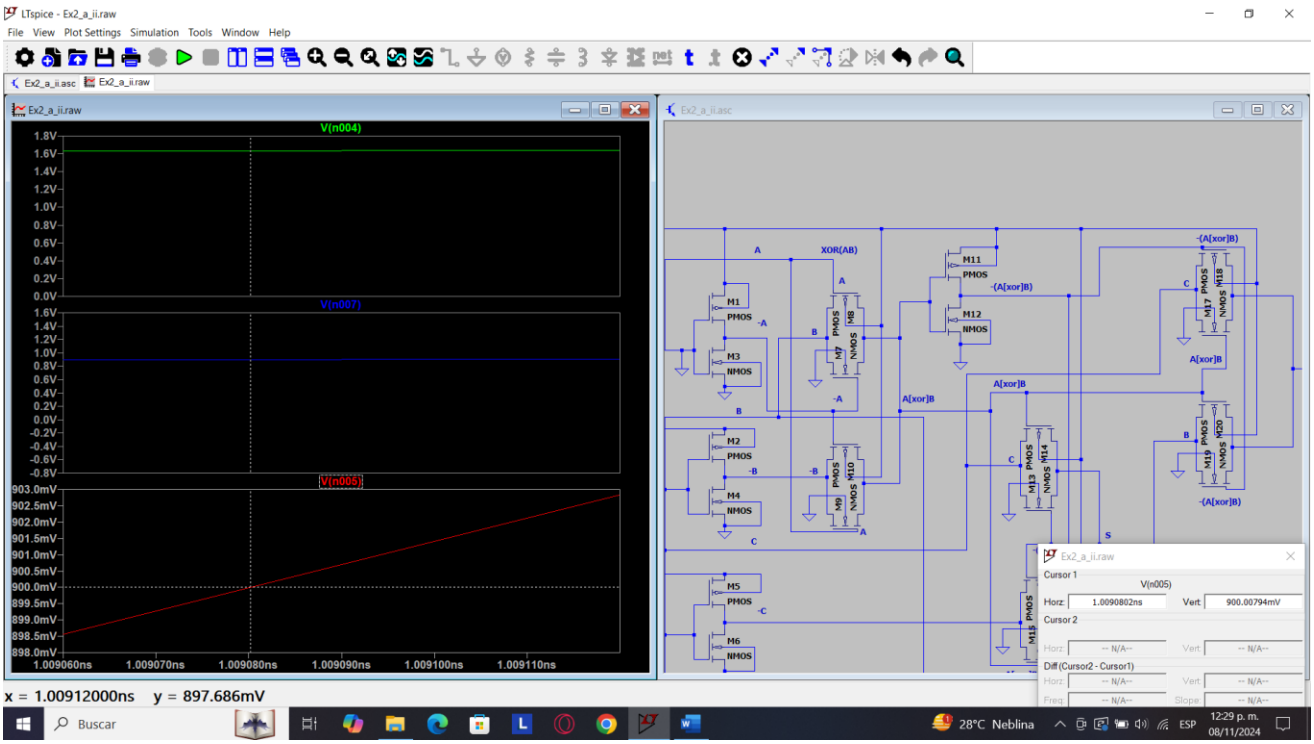


Figure 15Bii. Delay at C_out when A is at 0, and B and C switch from 0 to 1. Full adder with transmission gates.

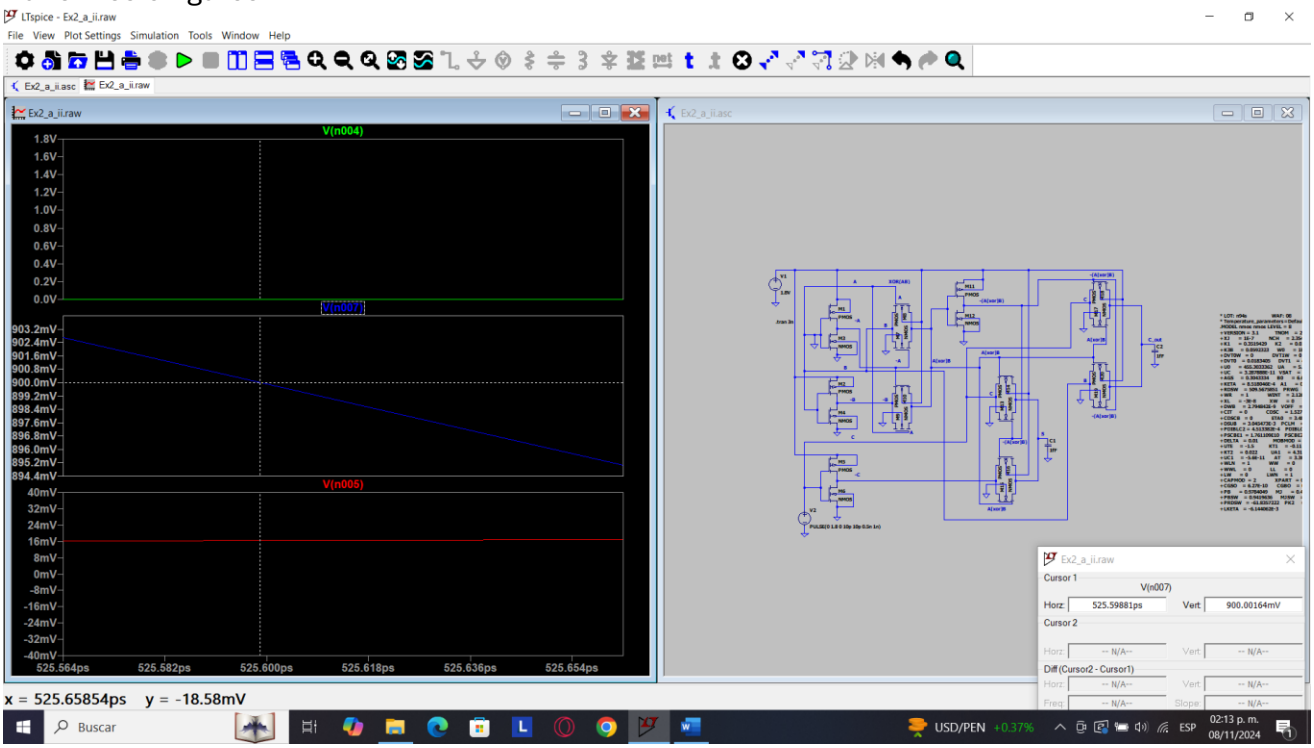


Figure 16Bii. Delay at S when A and B are at 0, and C switches from 1 to 0. Full adder with transmission gates.

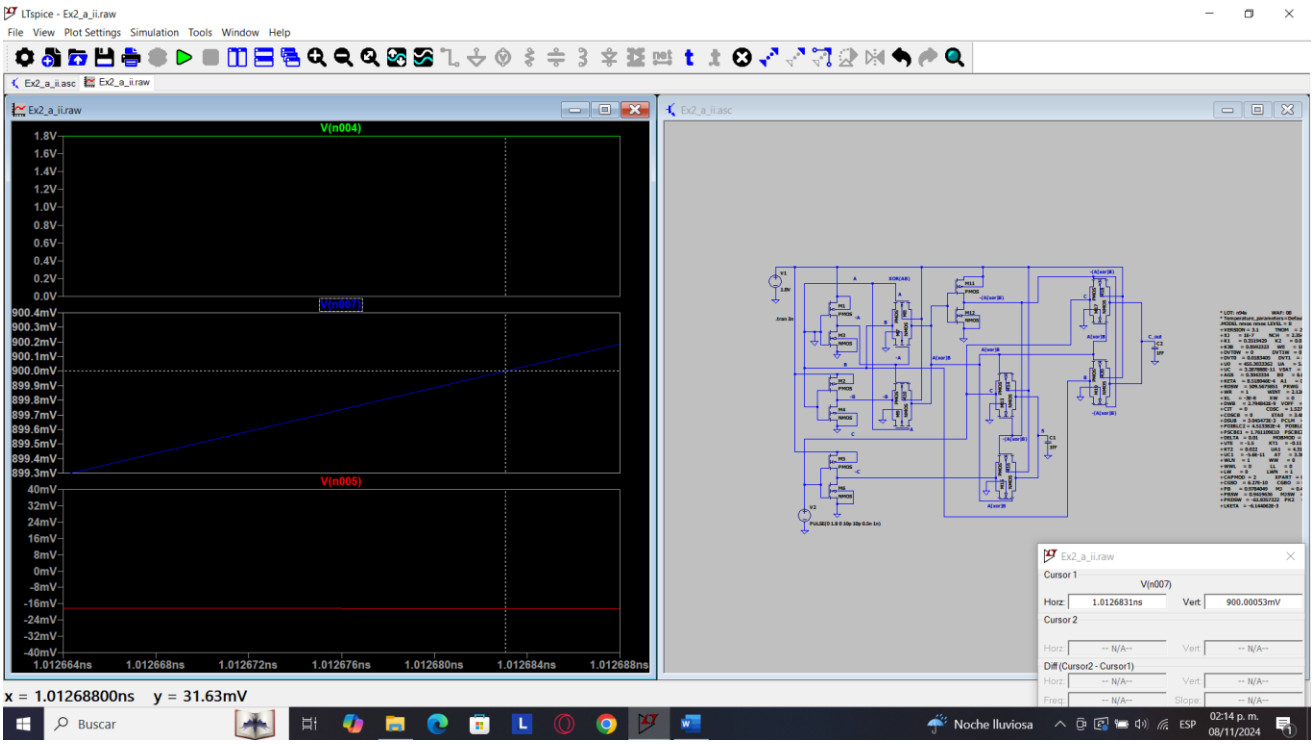


Figure 17Bii. Delay at S when A and B are at 0, and C switches from 0 to 1. Full adder with transmission gates.

$$t_0 = 0.515\ ns$$
$$t_1 = 0.52668874\ ns$$
$$t_2 = 1.005\ ns$$
$$t_3 = 1.0126831\ ns$$
$$Worst\ Case\ Propagation\ delay =$$
$$t_{phl} = t_1 - t_0 = 0.52668874\ ns - 0.515\ ns = 0.01168874\ ns$$
$$t_{plh} = t_3 - t_2 = 1.0126831\ ns - 1.005\ ns = 0.0076831\ ns$$
$$t_p = \frac{t_{phl} + t_{plh}}{2} = (0.01168874\ ns + 0.0076831\ ns)/2 = 0.00968592\ ns$$

The worst-case propagation delay for charging is when the inputs A, B and C go from (0,0,0) to (0,0,1) and for discharging the worst-case propagation delay is when A, B and C go from (0,1,1) to (0,0,0). This can be seen in Figures 14Bii and 17Bii.

$$Area =$$
$$A = A_{PMOS} + A_{NMOS} = 4 * 180 * 880 + 16 * 180 * 440$$
$$= 1{,}900{,}800\ nm^2$$
$$Worst\ Case\ Power = 3.2091\ mW$$

The worst power consumption scenario is when A, B, and C are both switching from 0 to 1 because it is the situation with most transistors with changing input, which leads to dynamic power consumption (Seen in Figures 6Bii and 9Bii).

Half adder using static CMOS

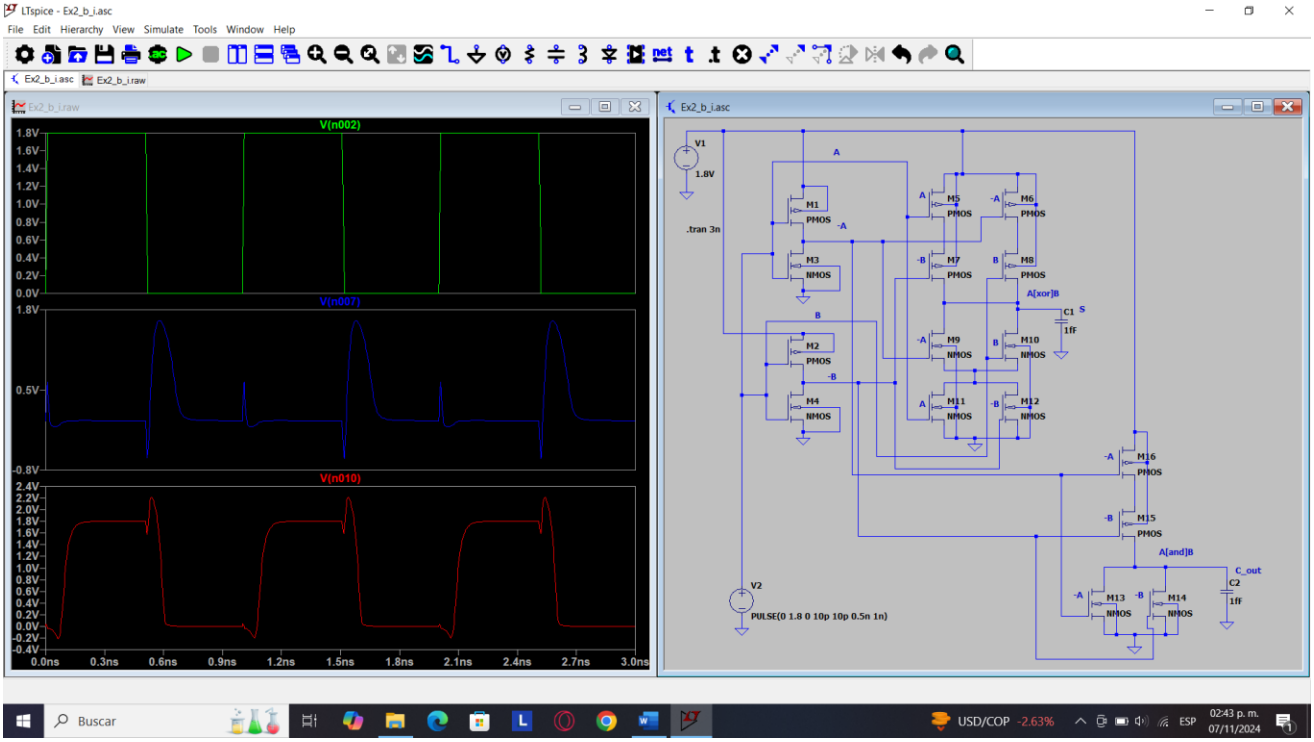


Figure 1Ci. Half adder with static CMOS with inputs A and B switching between 1 and 0.

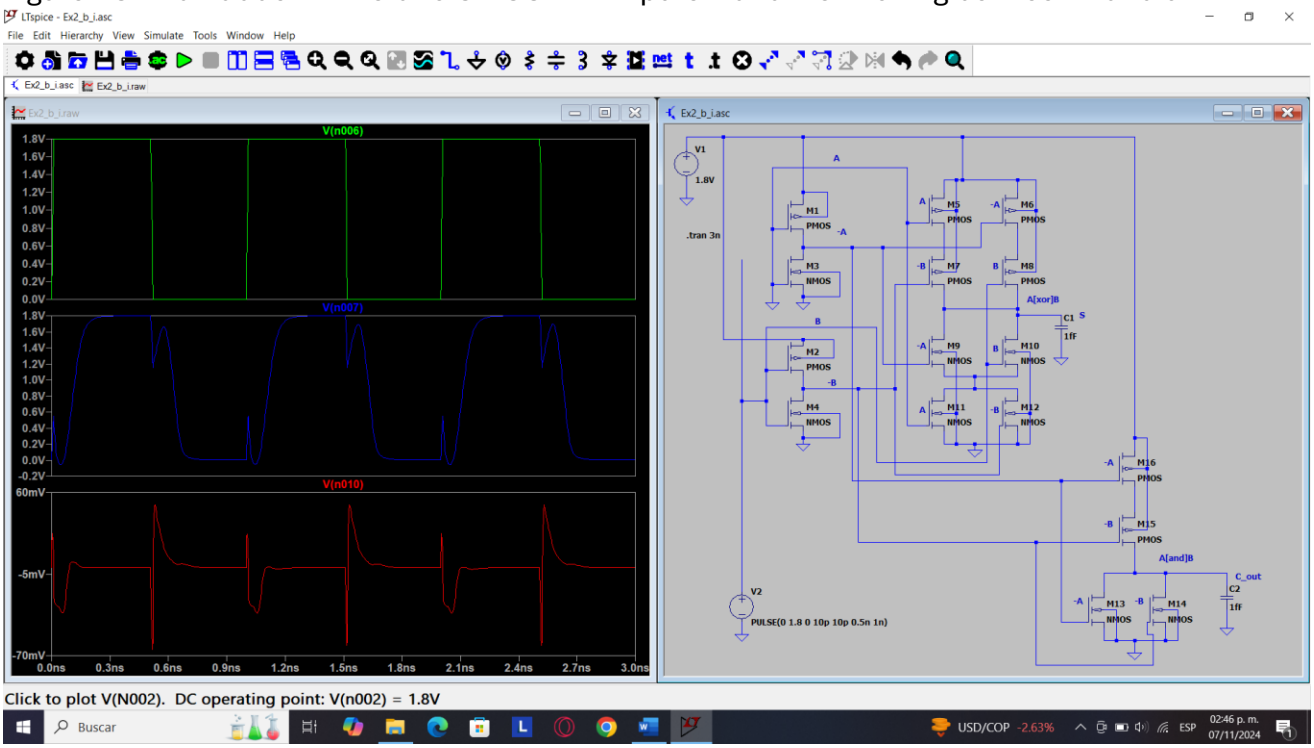


Figure 2Ci. Half adder with static CMOS with input A at 0 and B switching between 1 and 0.

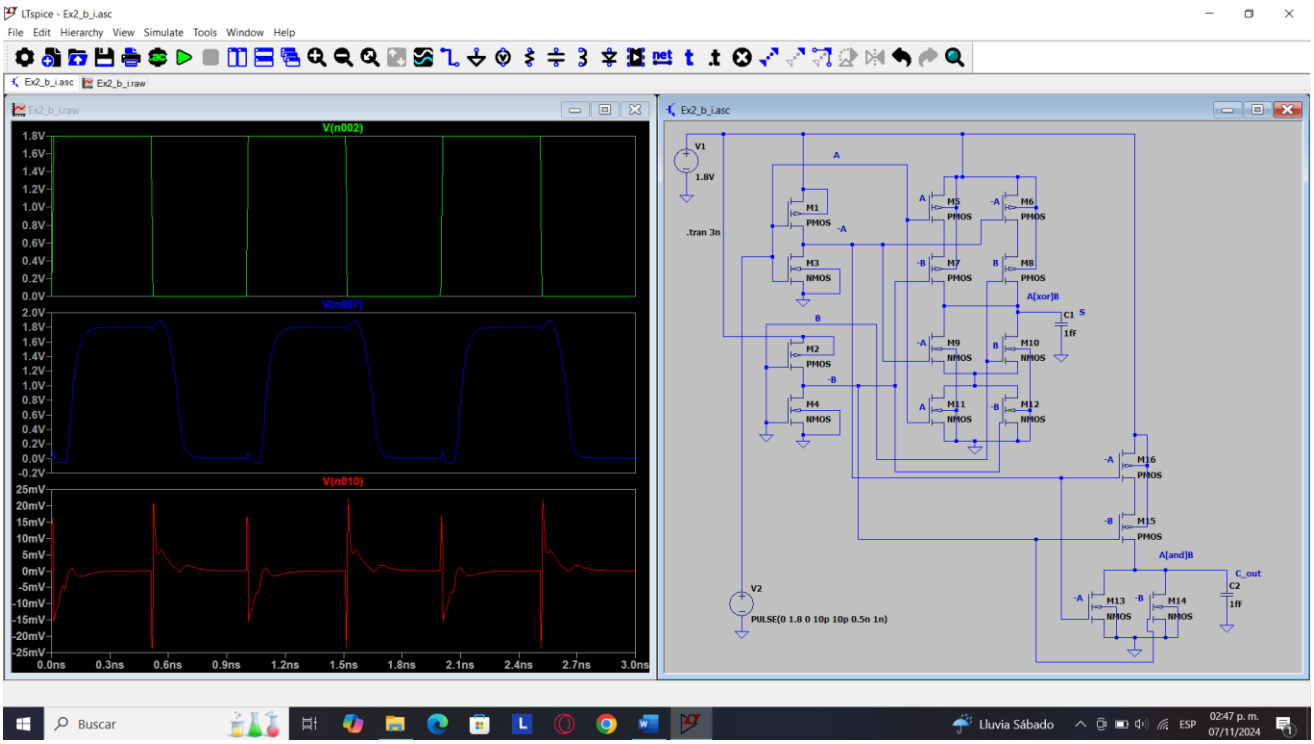


Figure 3Ci. Half adder with static CMOS with input B at 0 and A switching between 1 and 0.

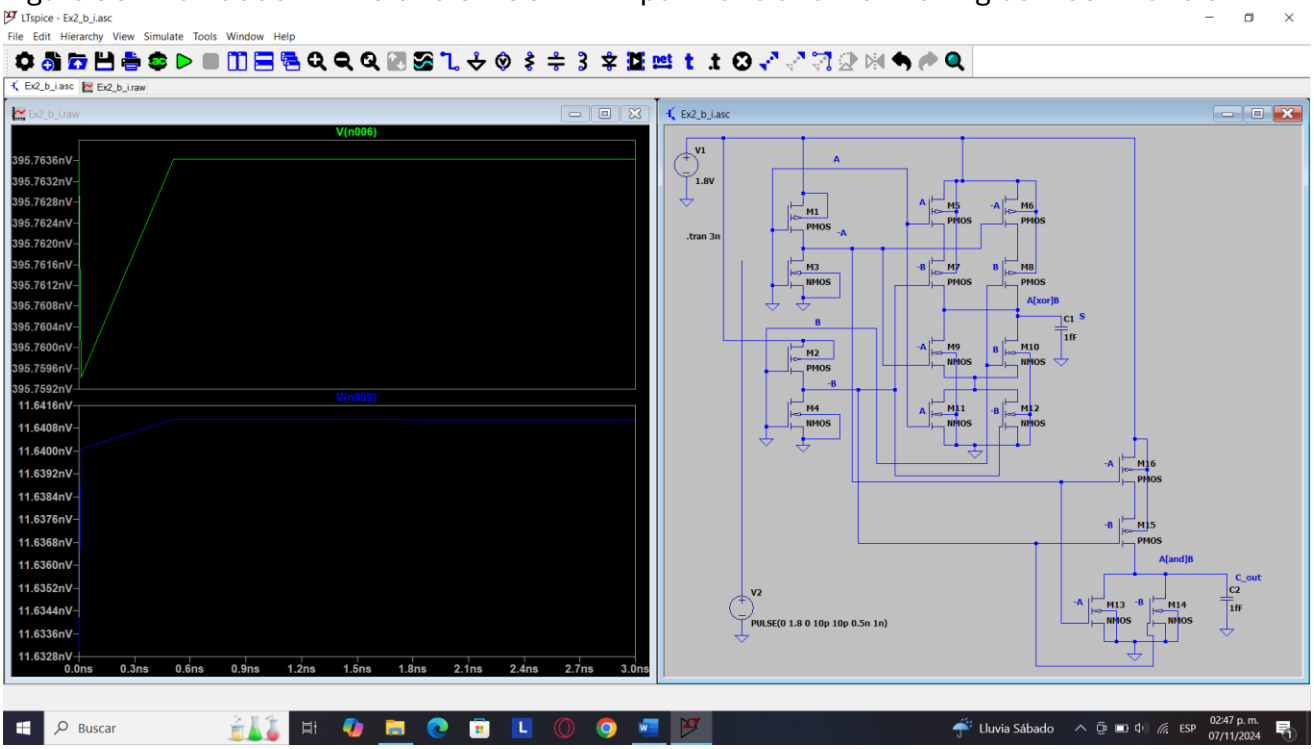


Figure 4Ci. Half adder with static CMOS with inputs A and B at 0. Graph order is S (Top graph), and C_out (Bottom graph).

Figures 1Ci to 4Ci demonstrate the correct functioning of the half adder with static CMOS.

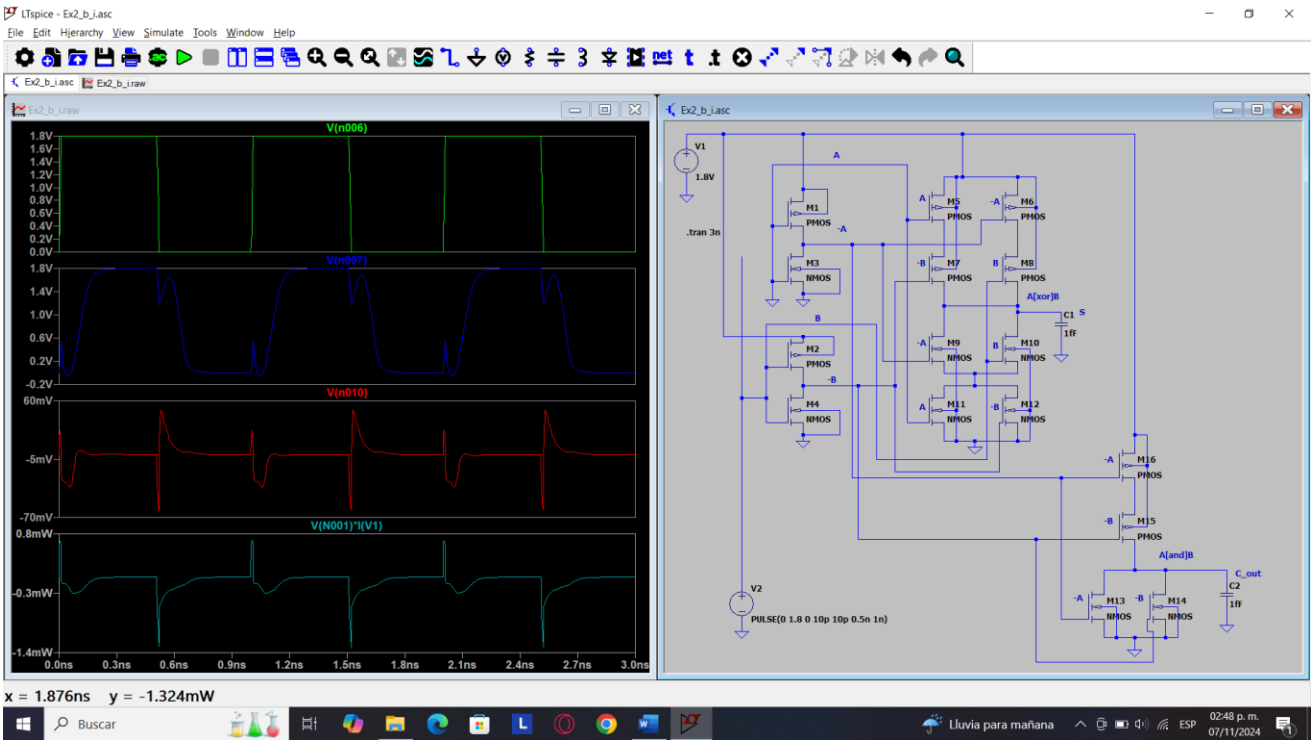


Figure 5Ci. Half adder with static CMOS with input A at 0 and B switching between 1 and 0 power consumption (Power consumption on bottom graph).

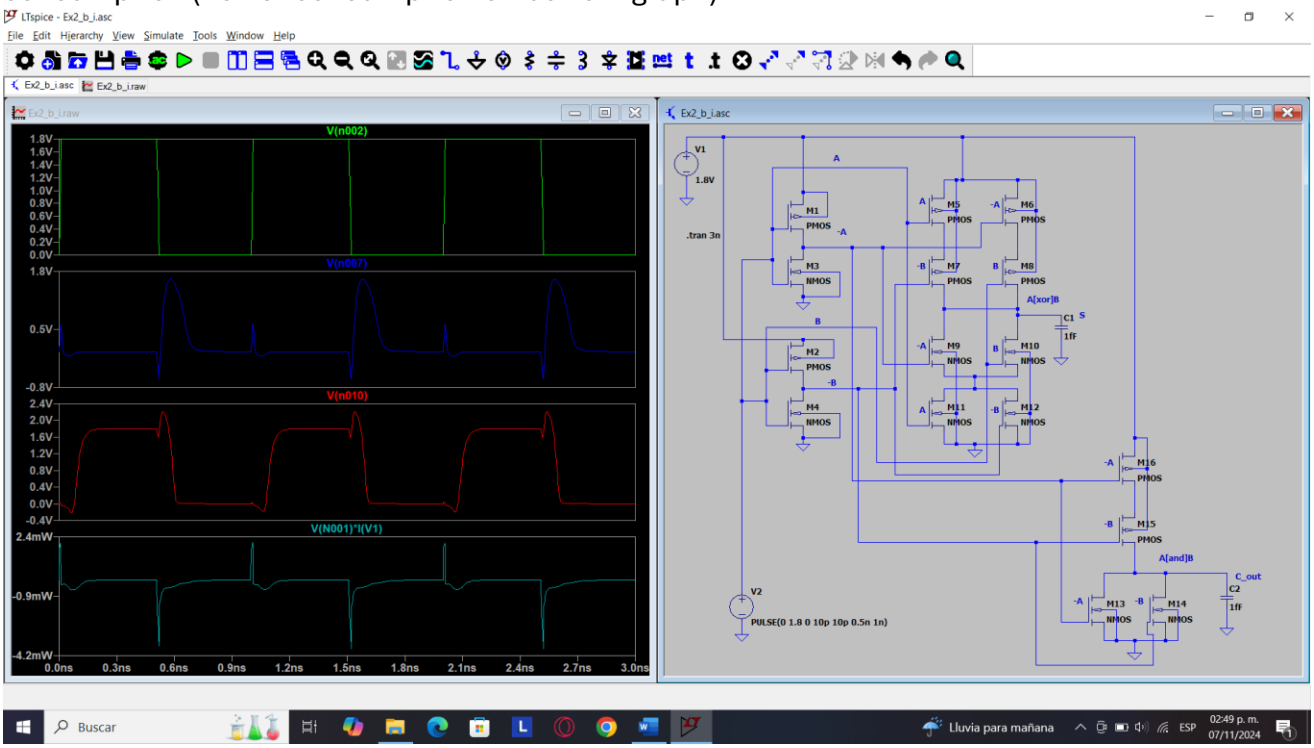


Figure 6Ci. Half adder with static CMOS with inputs A and B switching between 1 and 0 power consumption.

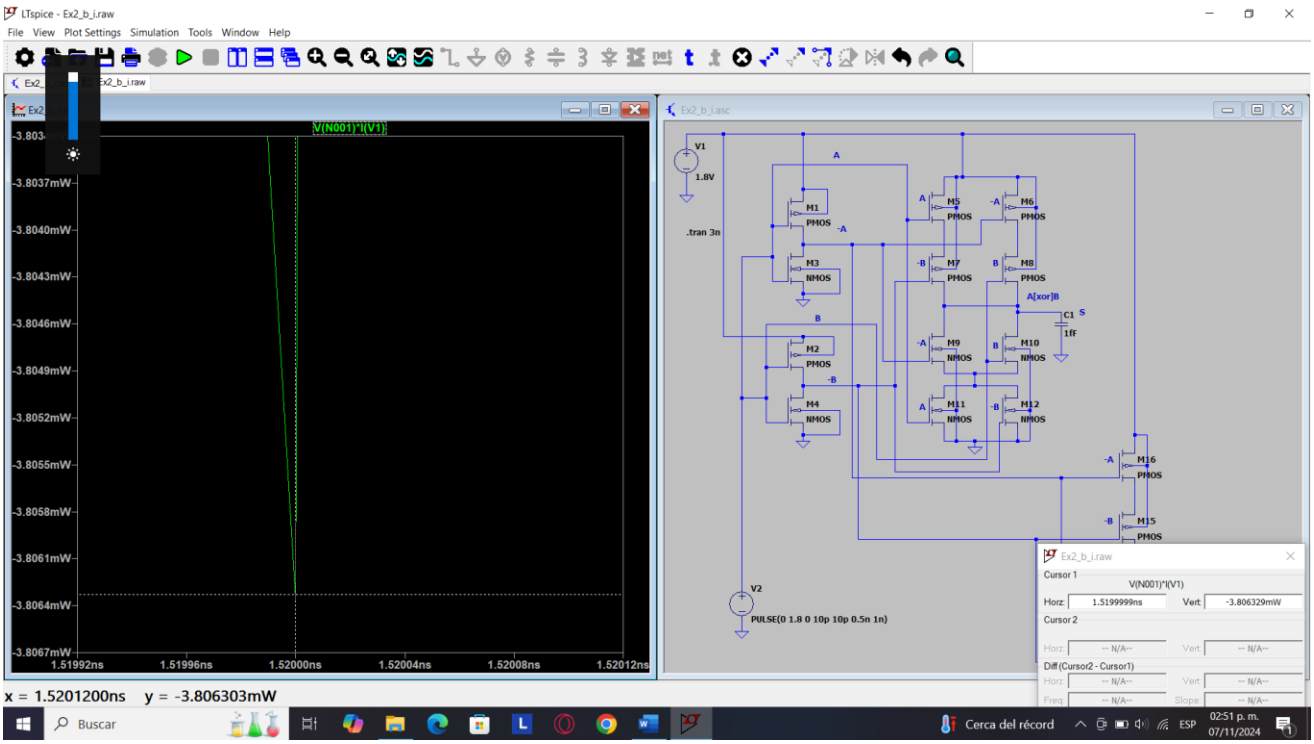


Figure 7Ci. Worst-case power consumption with inputs A and B switching between 1 and 0.

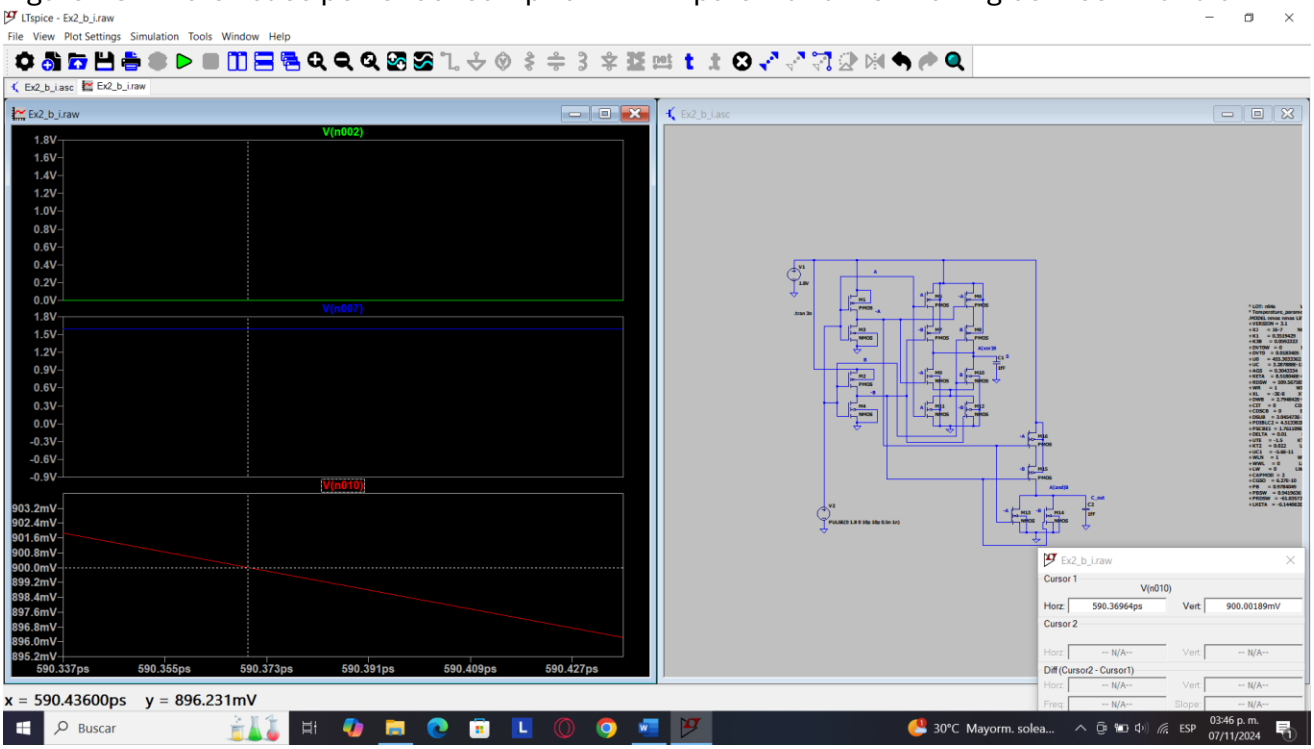


Figure 8Ci. Delay at C_out when inputs A and B switch from 1 to 0.

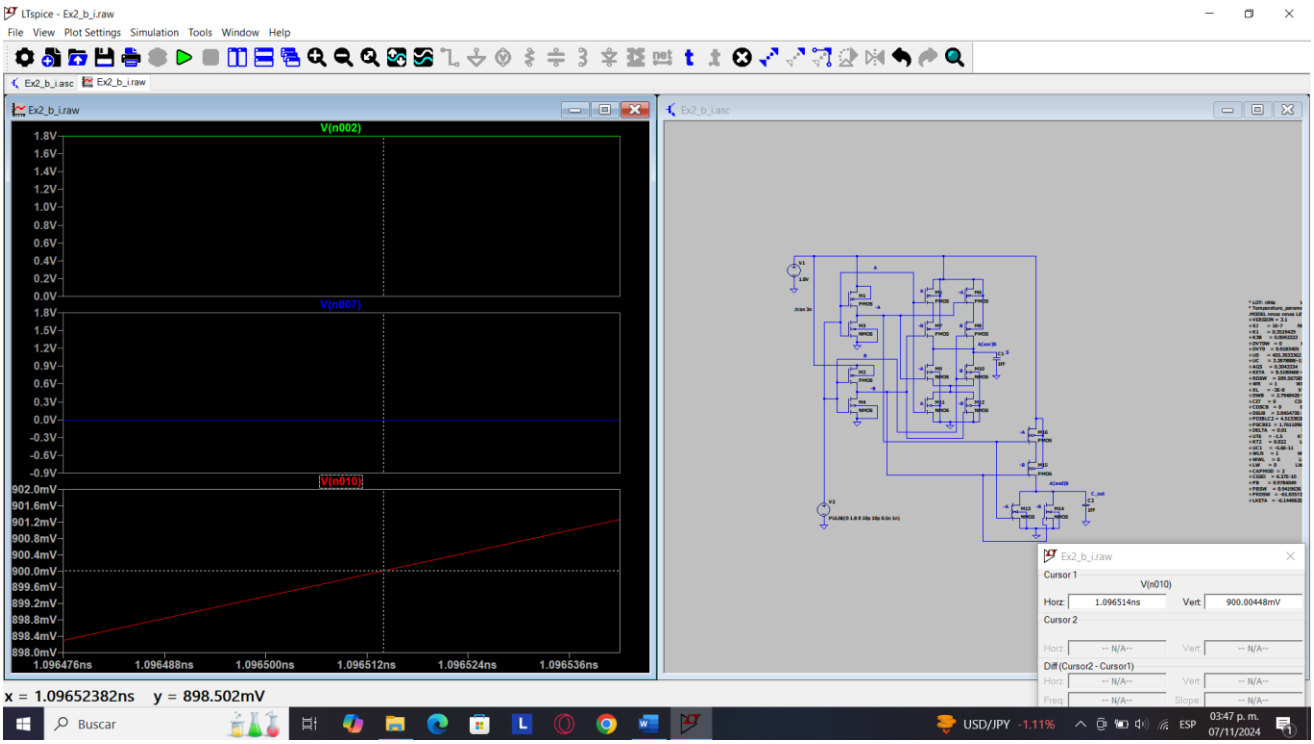


Figure 9Ci. Delay at C_out when inputs A and B switch from 0 to 1.

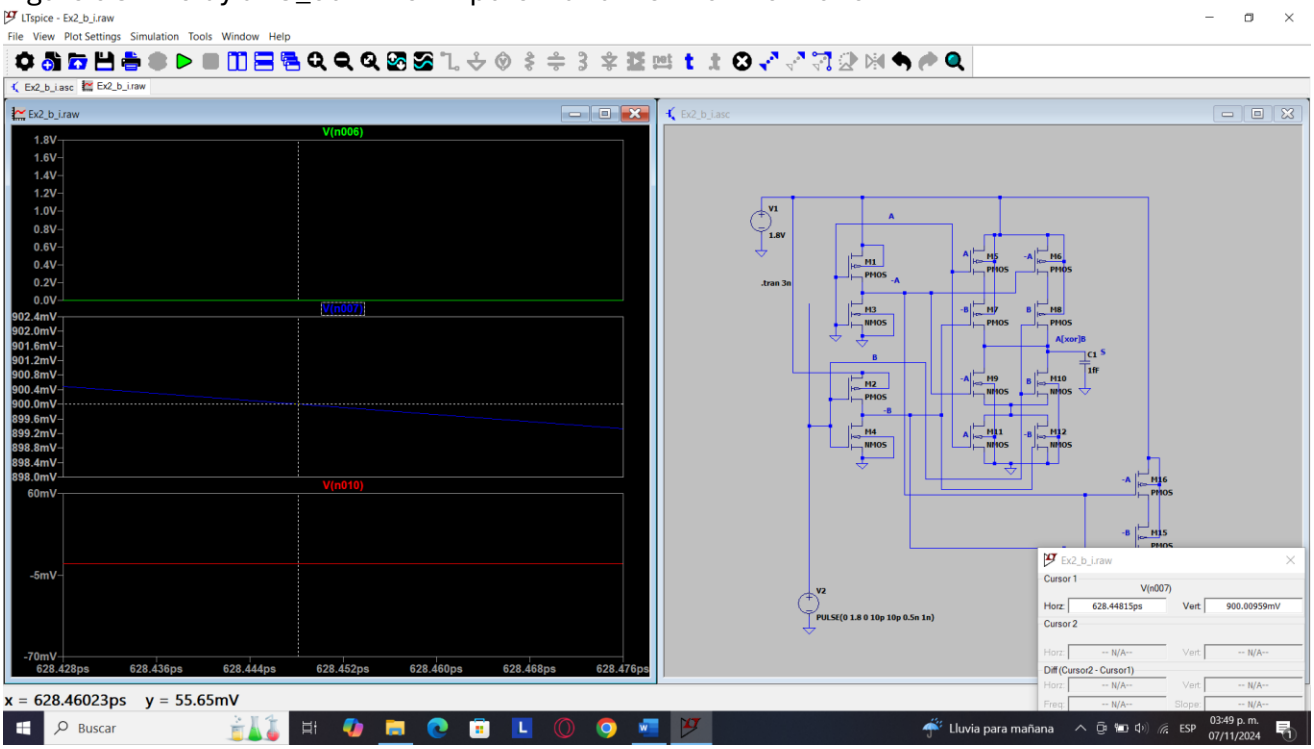


Figure 10Ci. Delay at S when input A stays at 0 and B switches from 1 to 0.

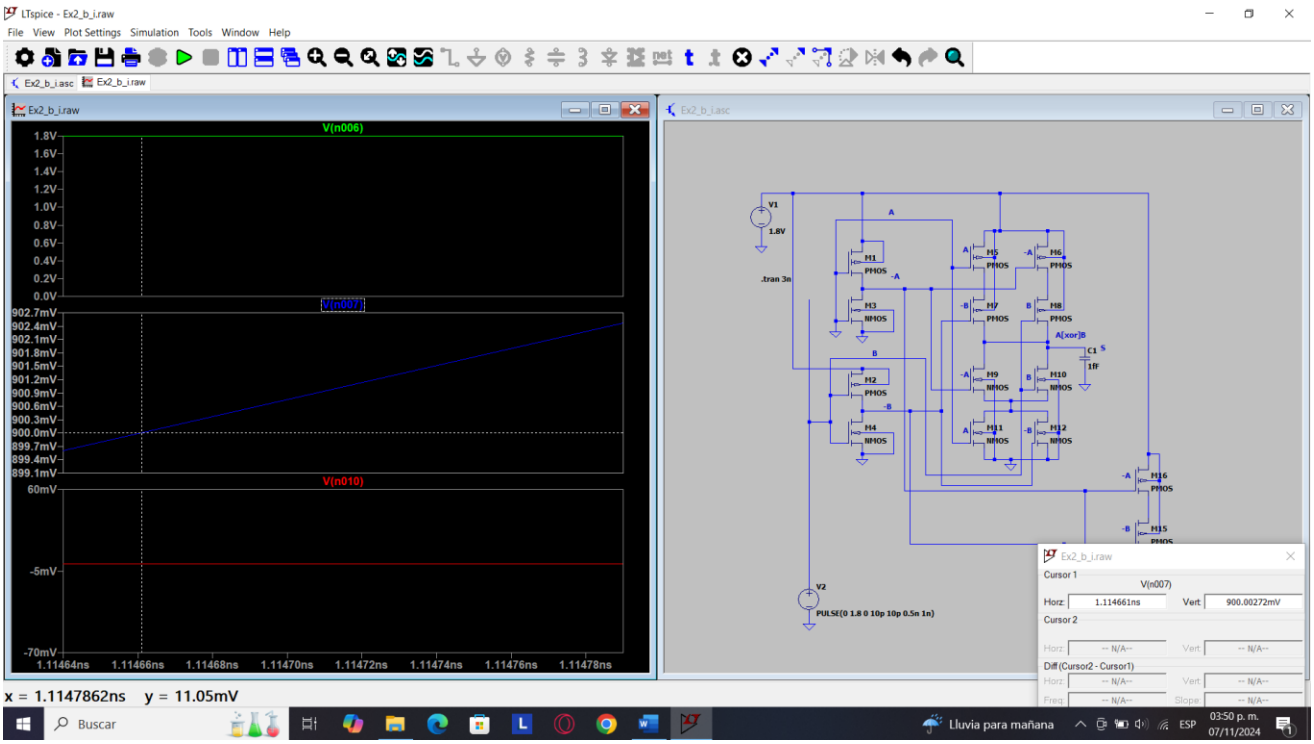


Figure 11Ci. Delay at S when input A stays at 0 and B switches from 0 to 1.

$$t_0 = 0.515\ ns$$
$$t_1 = 0.59036964\ ns$$
$$t_2 = 1.005\ ns$$
$$t_3 = 1.114661\ ns$$
$$Worst\ Case\ Propagation\ delay =$$
$$t_{phl} = t_1 - t_0 = 0.59036964\ ns - 0.515\ ns = 0.07536964\ ns$$
$$t_{plh} = t_3 - t_2 = 1.114661\ ns - 1.005\ ns = 0.109661\ ns$$
$$t_p = \frac{t_{phl} + t_{plh}}{2} = (0.07536964\ ns + 0.109661\ ns)/2 = 0.09251532\ ns$$

The worst-case propagation delay for charging is when inputs A and B are changing from (0,0) to (0,1) and worst-case propagation delay for discharging is when inputs A and B change from (0,1) to (0,0). This can be seen in Figures 10Ci and 11Ci.

$$Area =$$
$$A = A_{PMOS} + A_{NMOS} = 6 * 180 * 880 + 4 * 180 * 440 + 6 * 1760 * 180$$
$$= 3{,}168{,}000\ nm^2$$
$$Worst\ Case\ Power = 3.8063\ mW$$

The worst power consumption scenario is when A and B are both switching from 0 to 1 because it is the situation with most transistors with changing input, which leads to dynamic power consumption (Seen in Figures 6Ci and 7Ci).

Full adder with static CMOS (Mirror full adder):

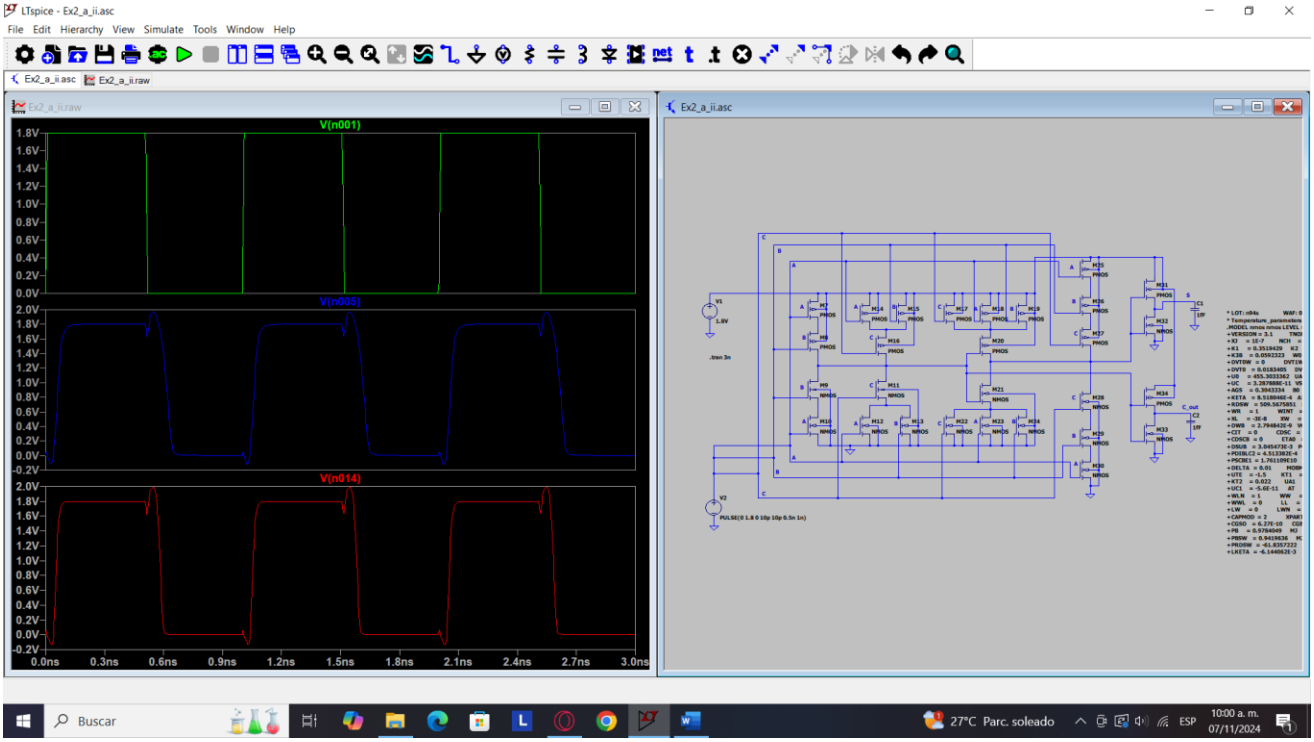


Figure 1Cii. Full adder with static CMOS with inputs A, B, and C switching between 1 and 0. Graph order goes as follows: Input (Top graph), Sum graph (Middle graph), and C_out graph (Bottom graph).

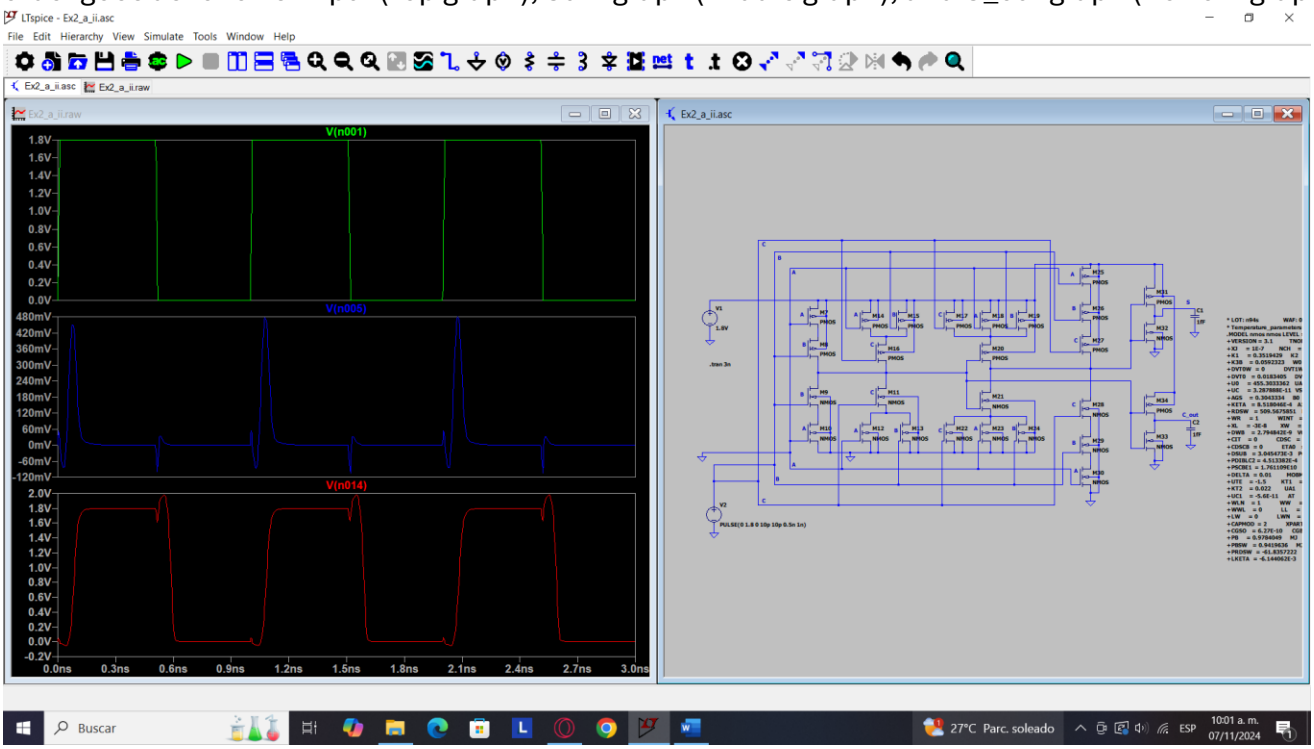


Figure 2Cii. Full adder with static CMOS with input A at 0, and inputs B and C switching between 1 and 0.

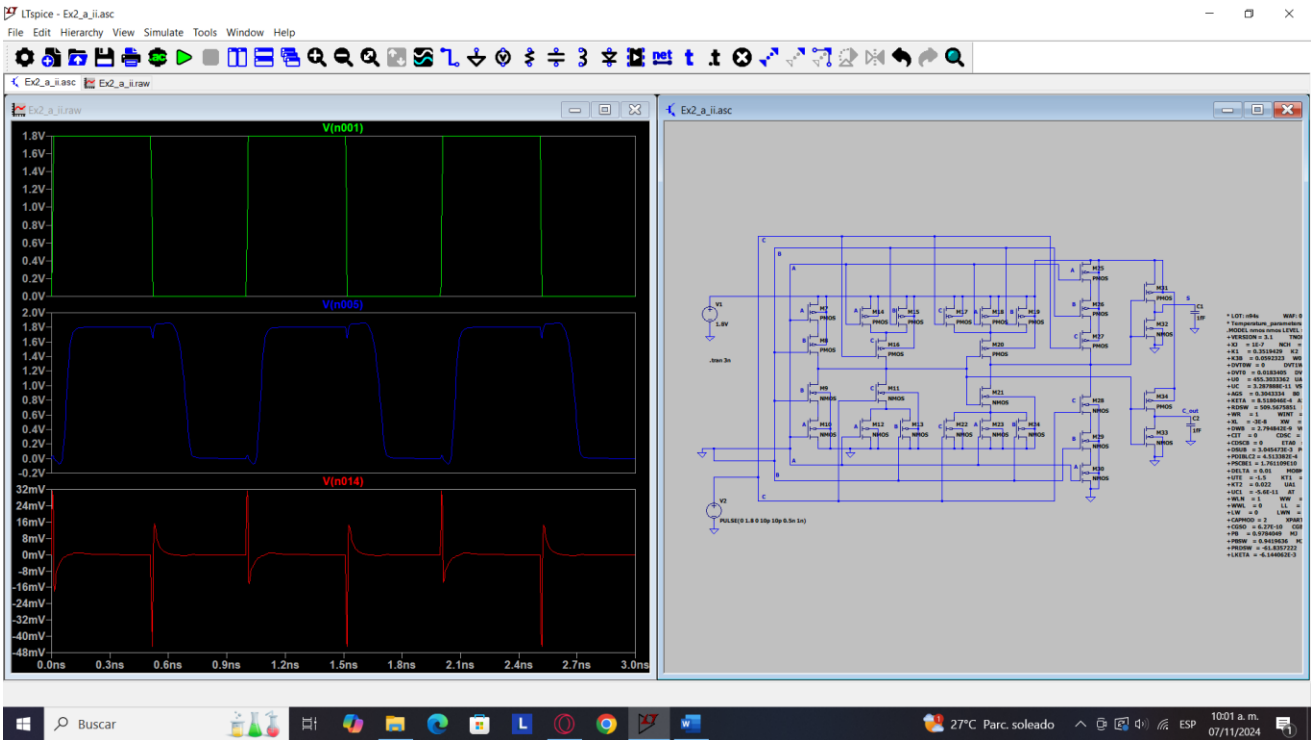


Figure 3Cii. Full adder with static CMOS with input A and B at 0, and C switching between 1 and 0.

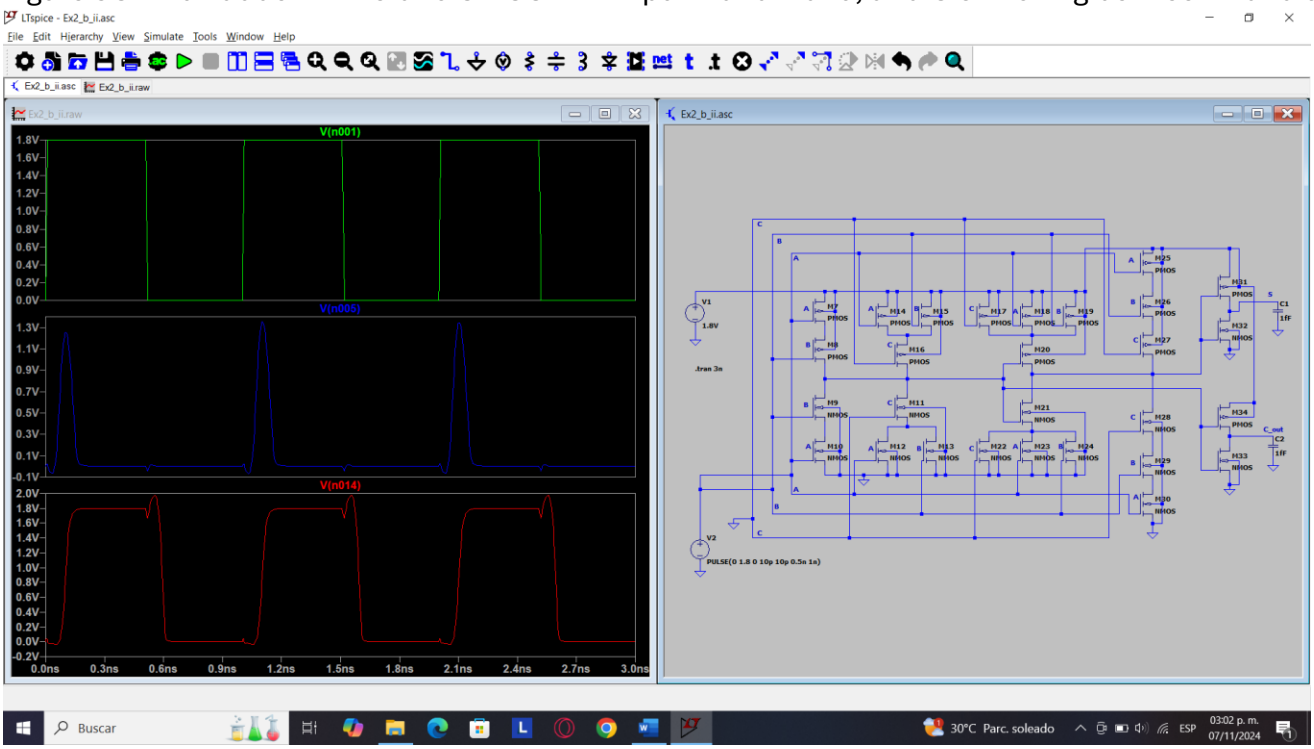


Figure 4Cii. Full adder with static CMOS with input C at 0, and inputs B and A switching between 1 and 0.

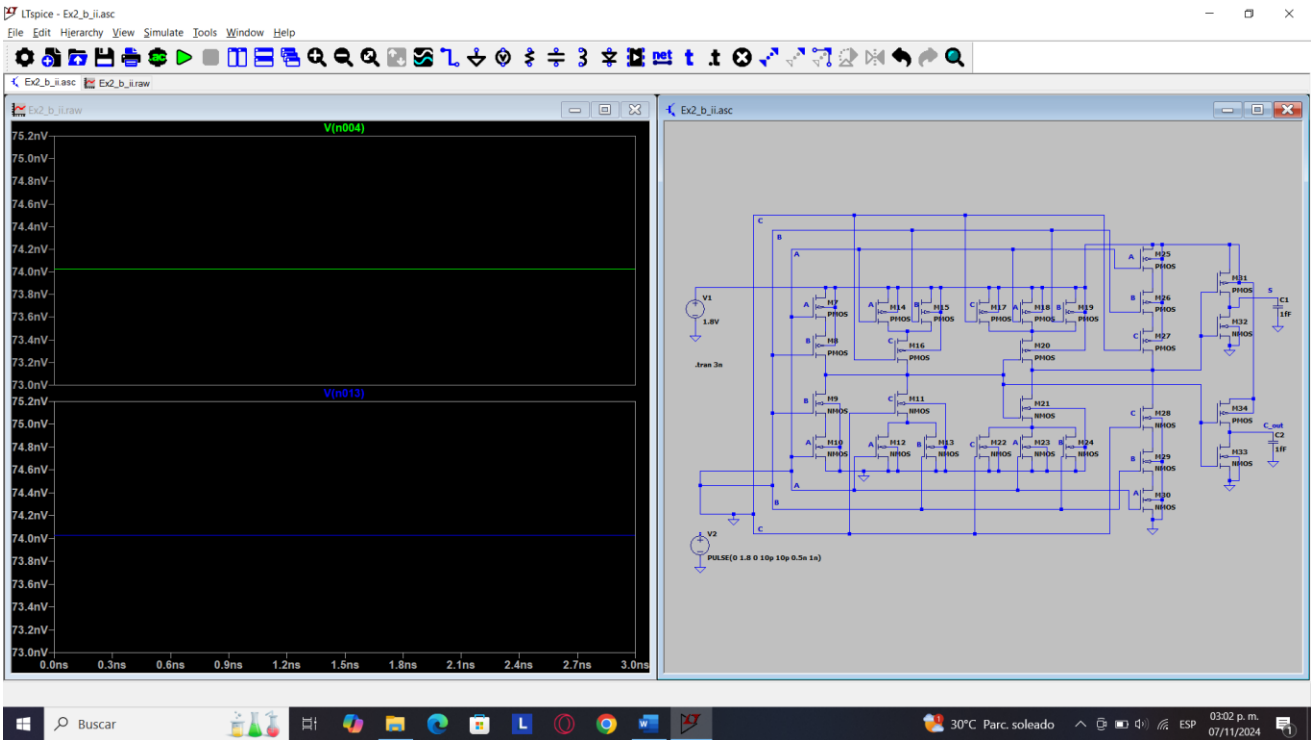


Figure 5Cii. Full adder with static CMOS with inputs A, B, and C at 0. Graph order: Sum (Top graph) and C_out (Bottom graph).

Figures 1Cii to 5Cii demonstrate the correct functioning of the full adder with static CMOS (In this case a mirror full adder).

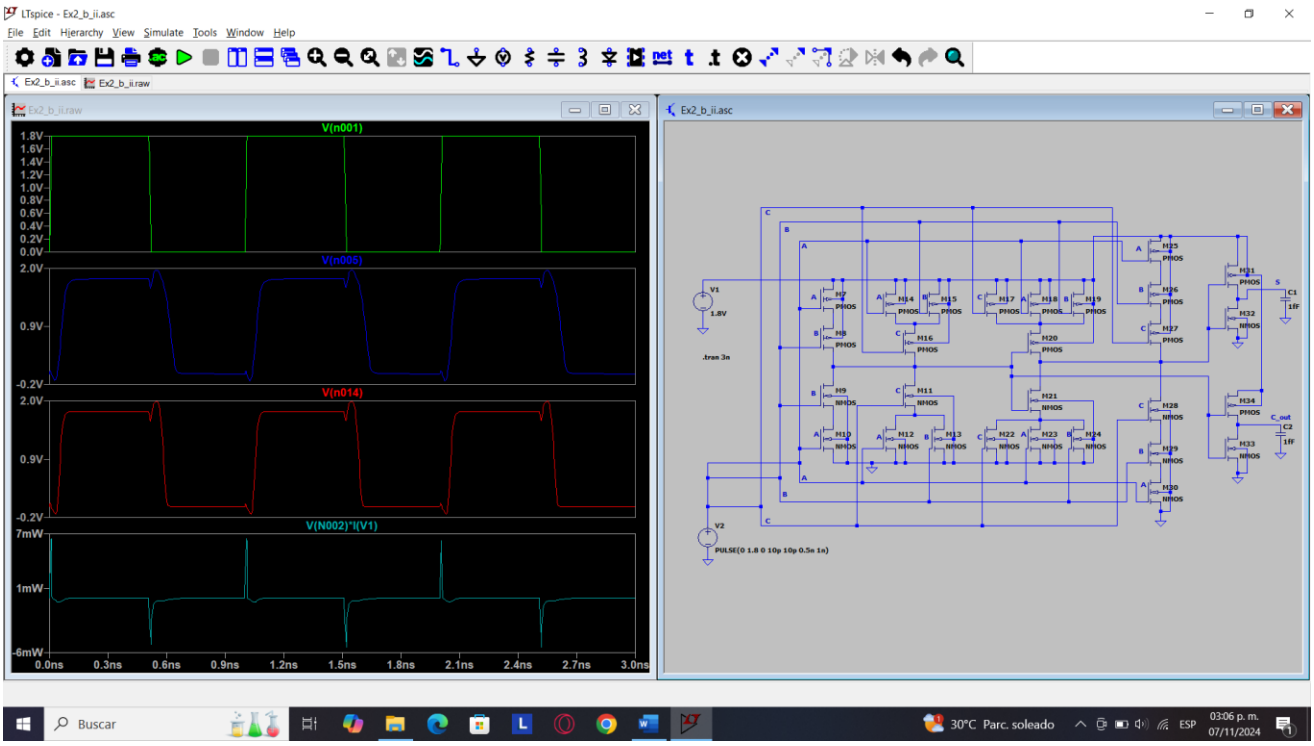


Figure 6Cii. Full adder with static CMOS with inputs A, B, and C switching between 1 and 0. Power consumption is the bottom graph (Worst case power consumption).

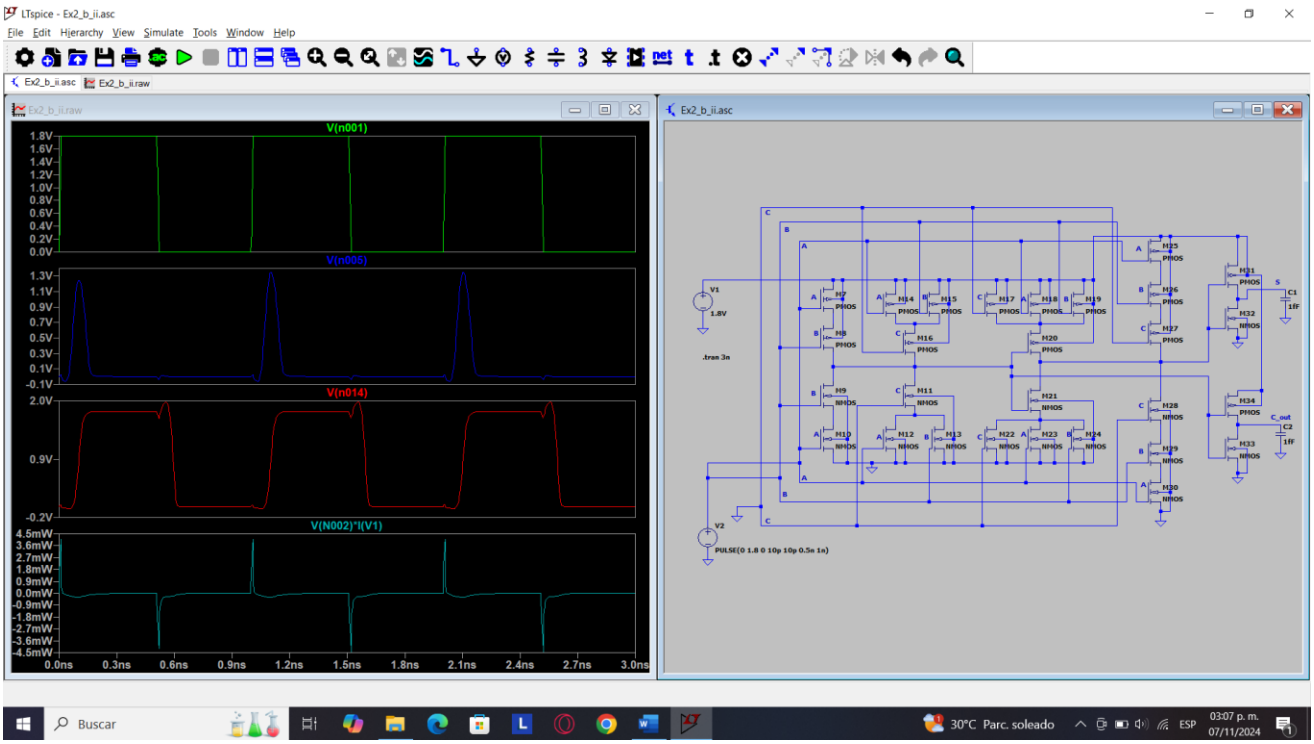


Figure 7Cii. Full adder with static CMOS with input C at 0, and inputs B and A switching between 1 and 0. Power consumption is the bottom graph.

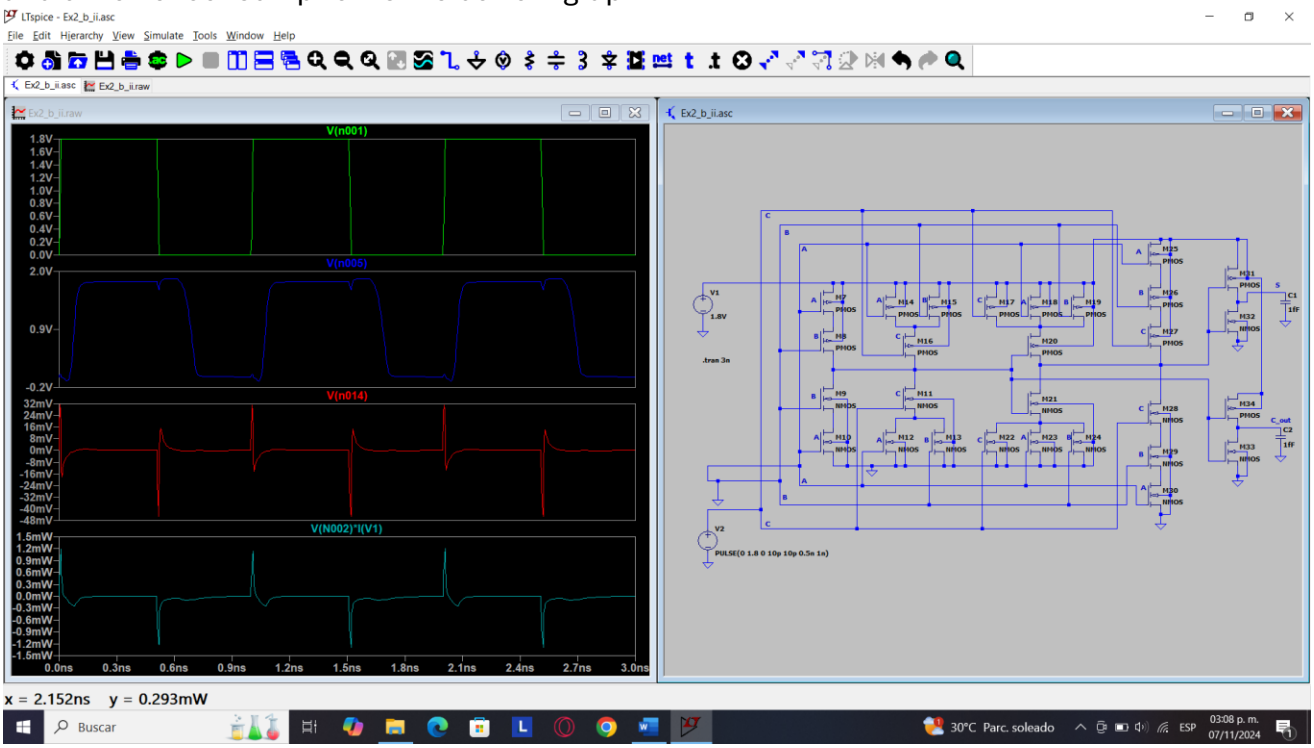


Figure 8Cii. Full adder with static CMOS with input A and B at 0, and C switching between 1 and 0. Power consumption is the bottom graph.

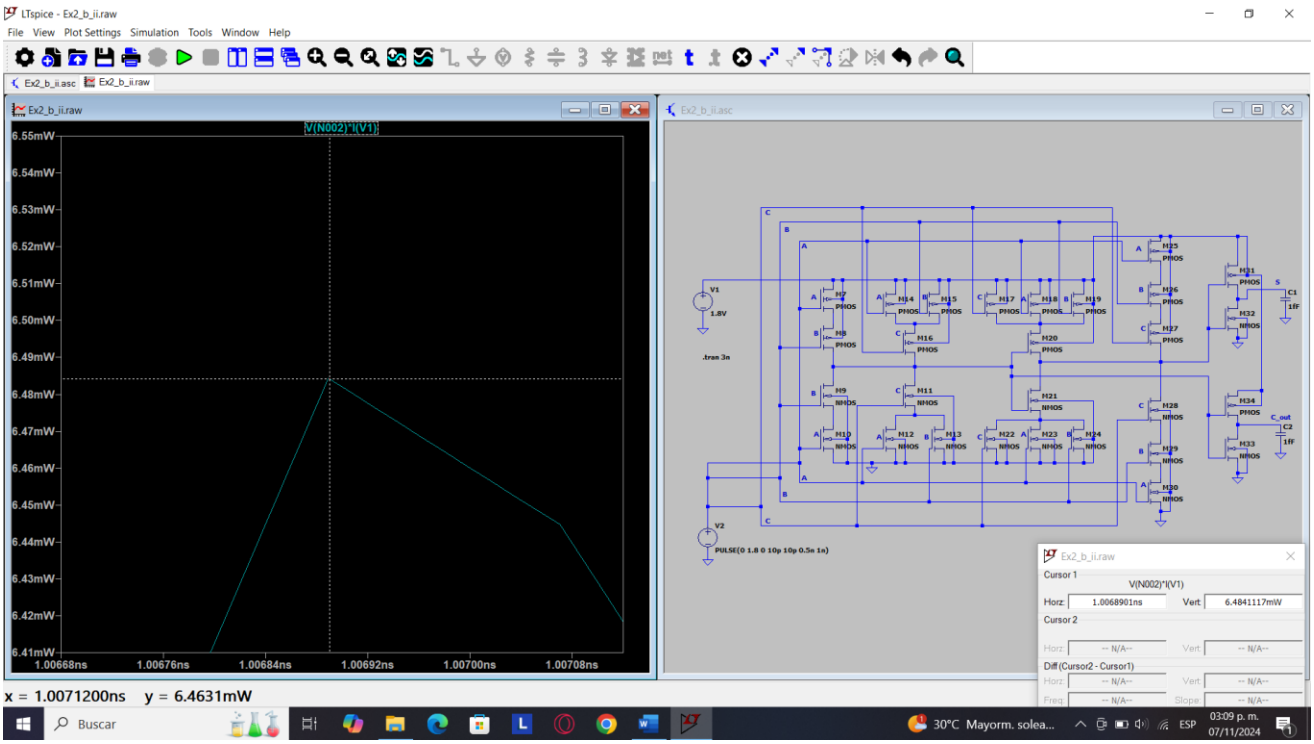


Figure 9Cii. Worst-case power consumption with inputs A, B, and C switching between 1 and 0.

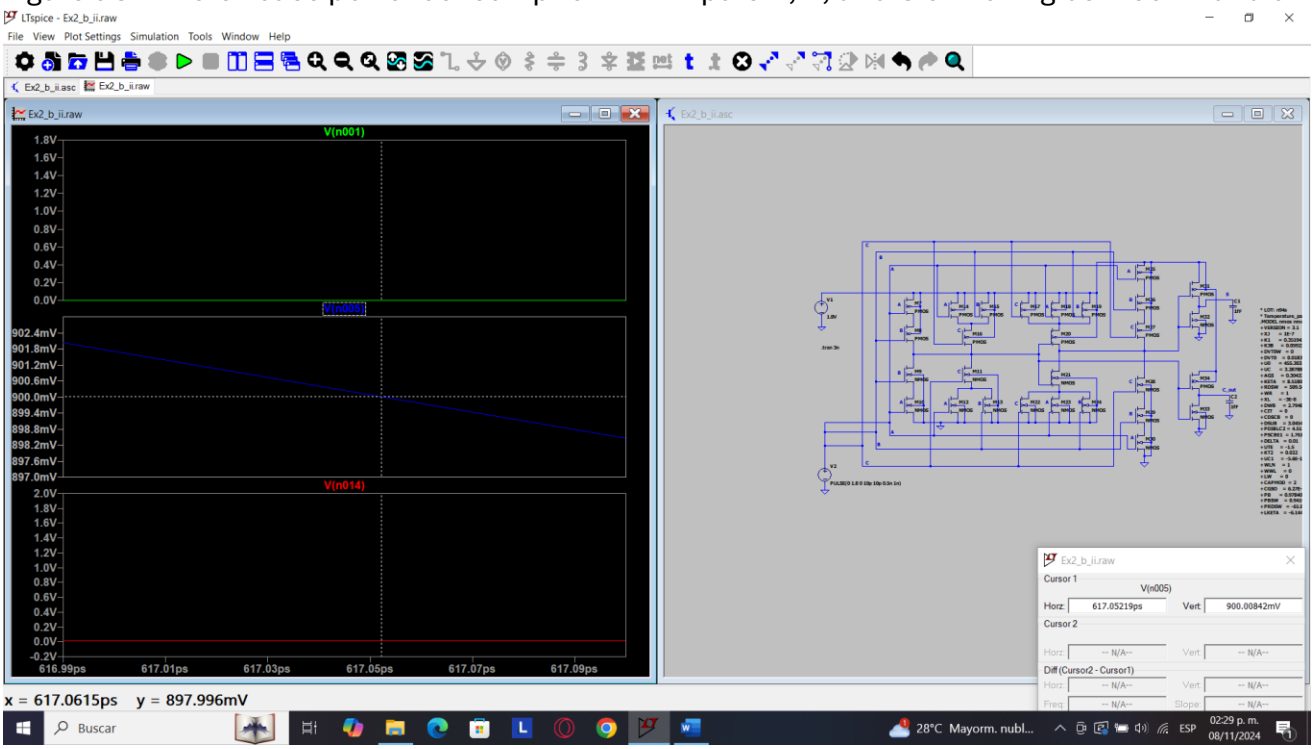


Figure 10Cii. Delay at S when A, B and C switch from 1 to 0. Full adder static CMOS.

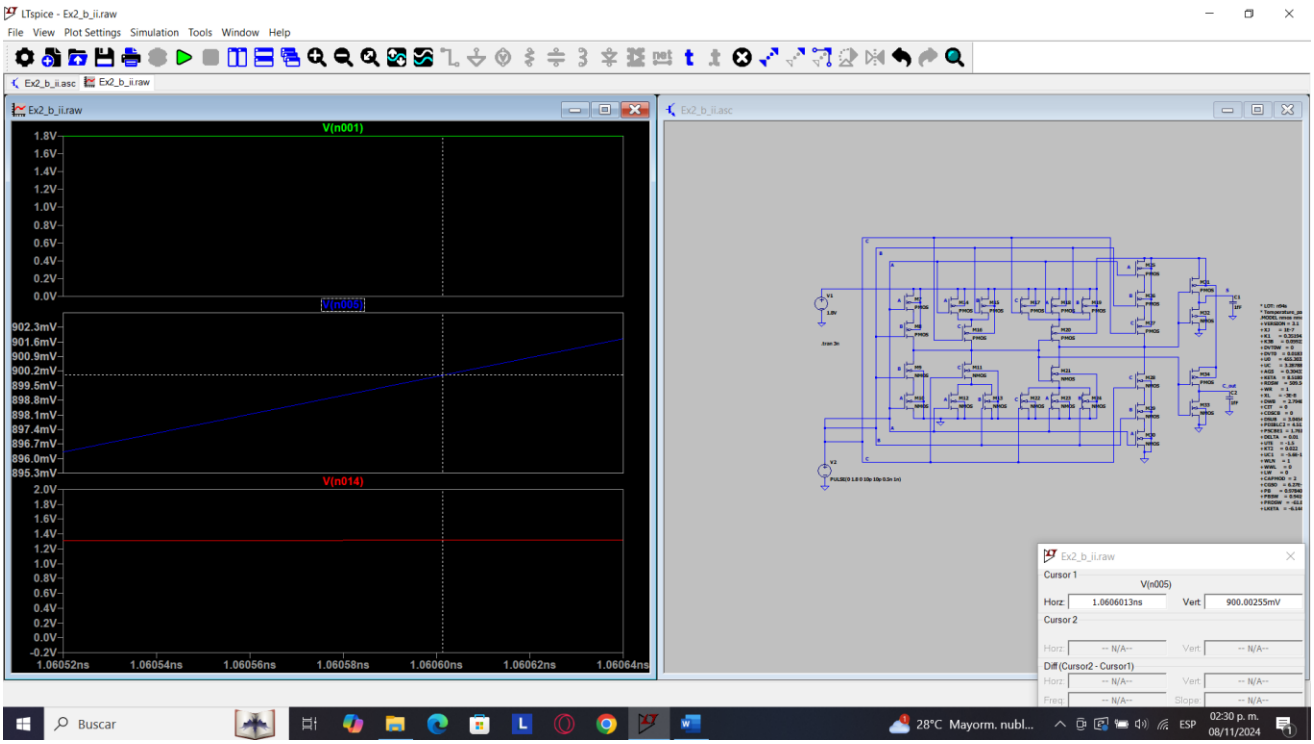


Figure 11Cii. Delay at S when A, B and C switch from 0 to 1. Full adder static CMOS.

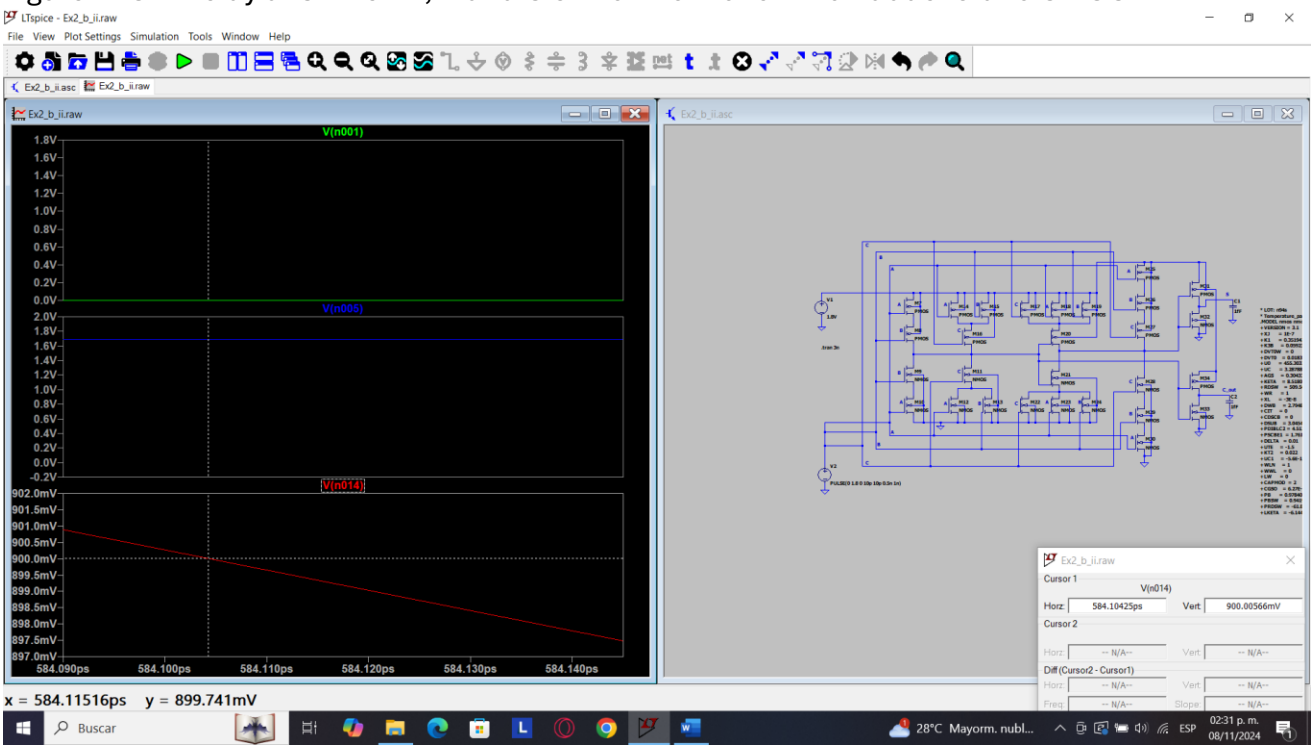


Figure 12Cii. Delay at C_out when A, B and C switch from 1 to 0. Full adder static CMOS.

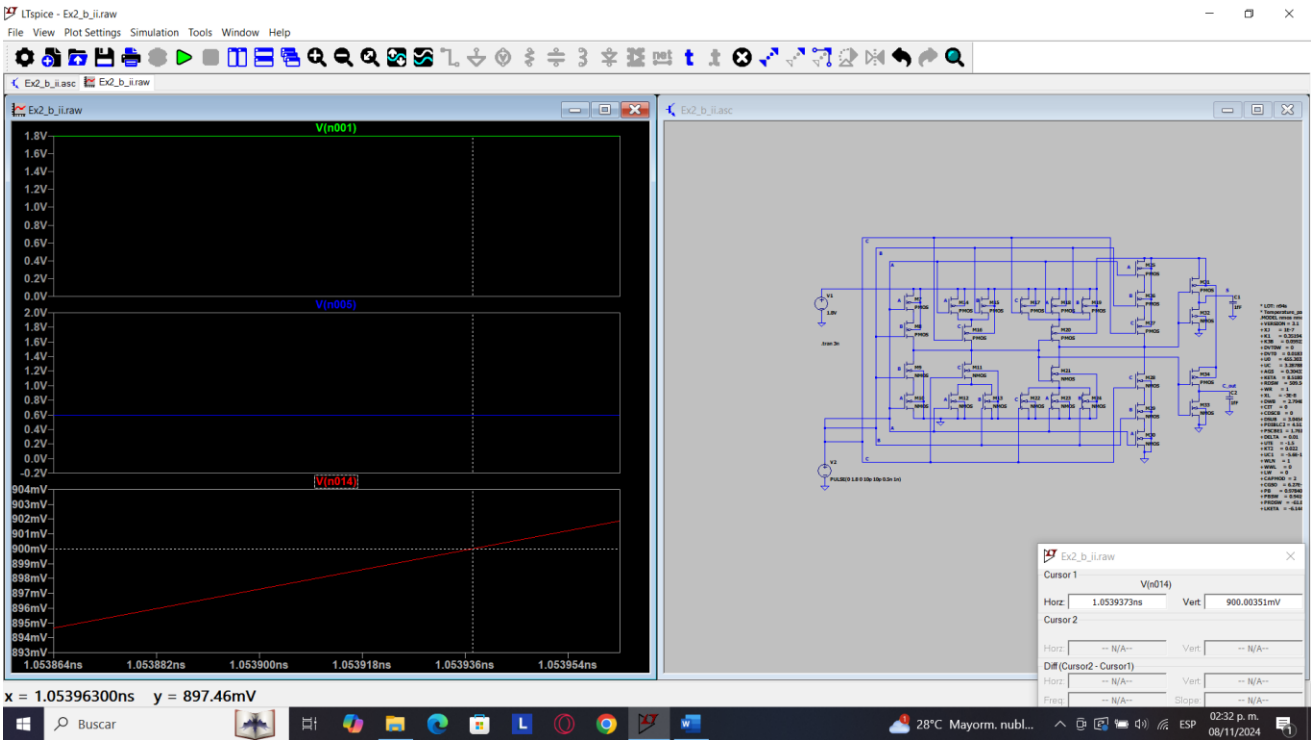


Figure 13Cii. Delay at C_out when A, B and C switch from 0 to 1. Full adder static CMOS.

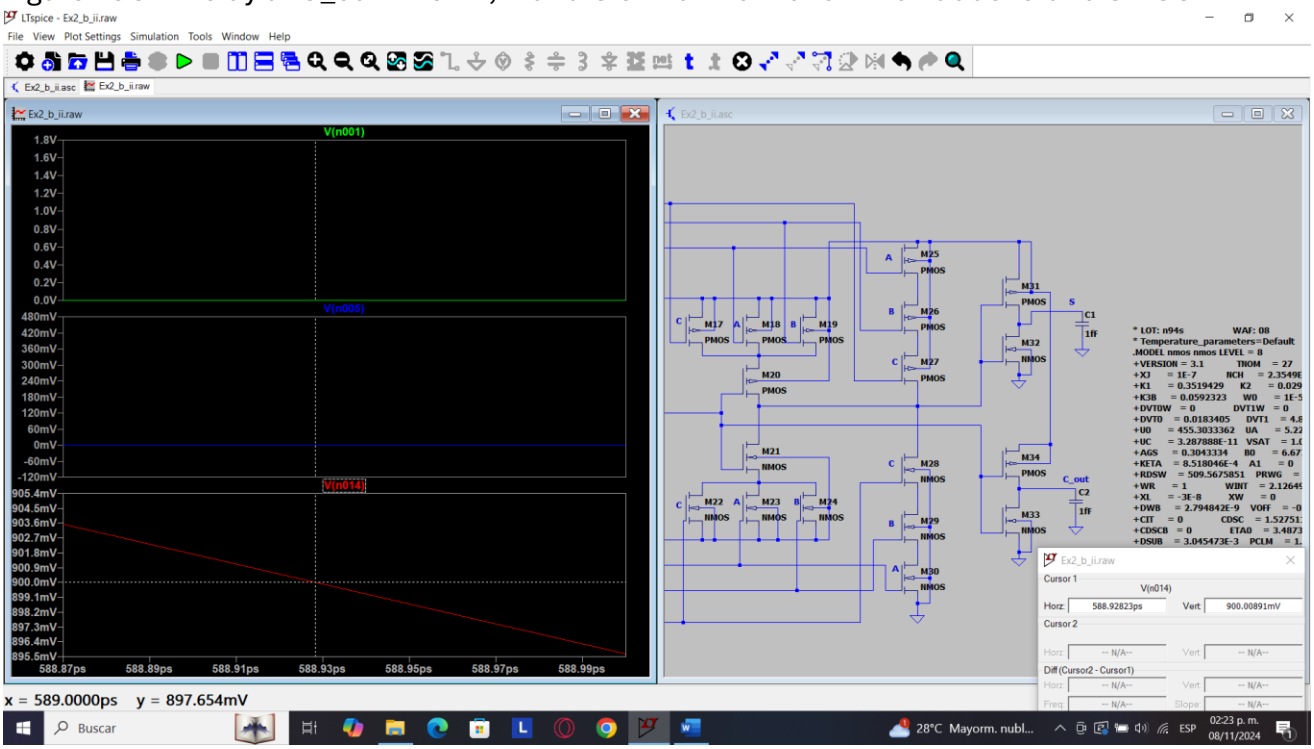


Figure 14Cii. Delay at C_out when A is at 0, and B and C switch from 1 to 0. Full adder static CMOS.

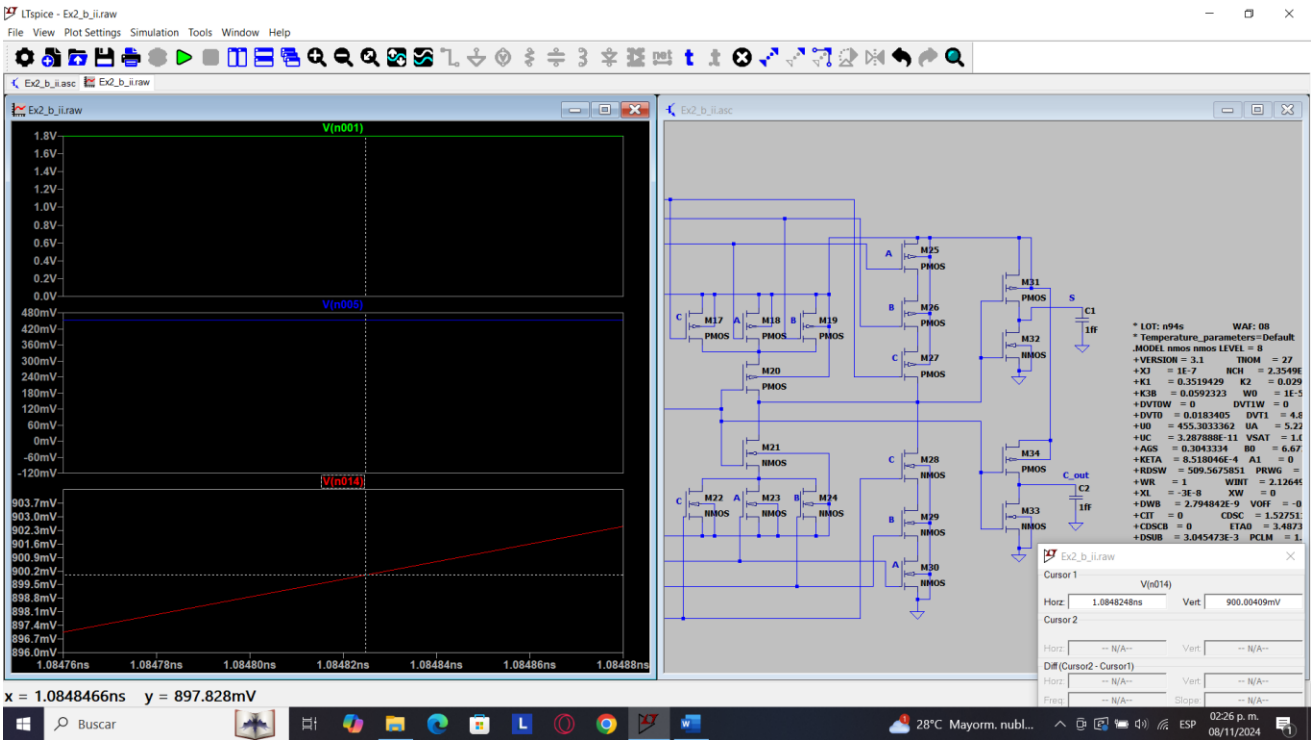


Figure 15Cii. Delay at C_out when A is at 0, and B and C switch from 0 to 1. Full adder static CMOS.

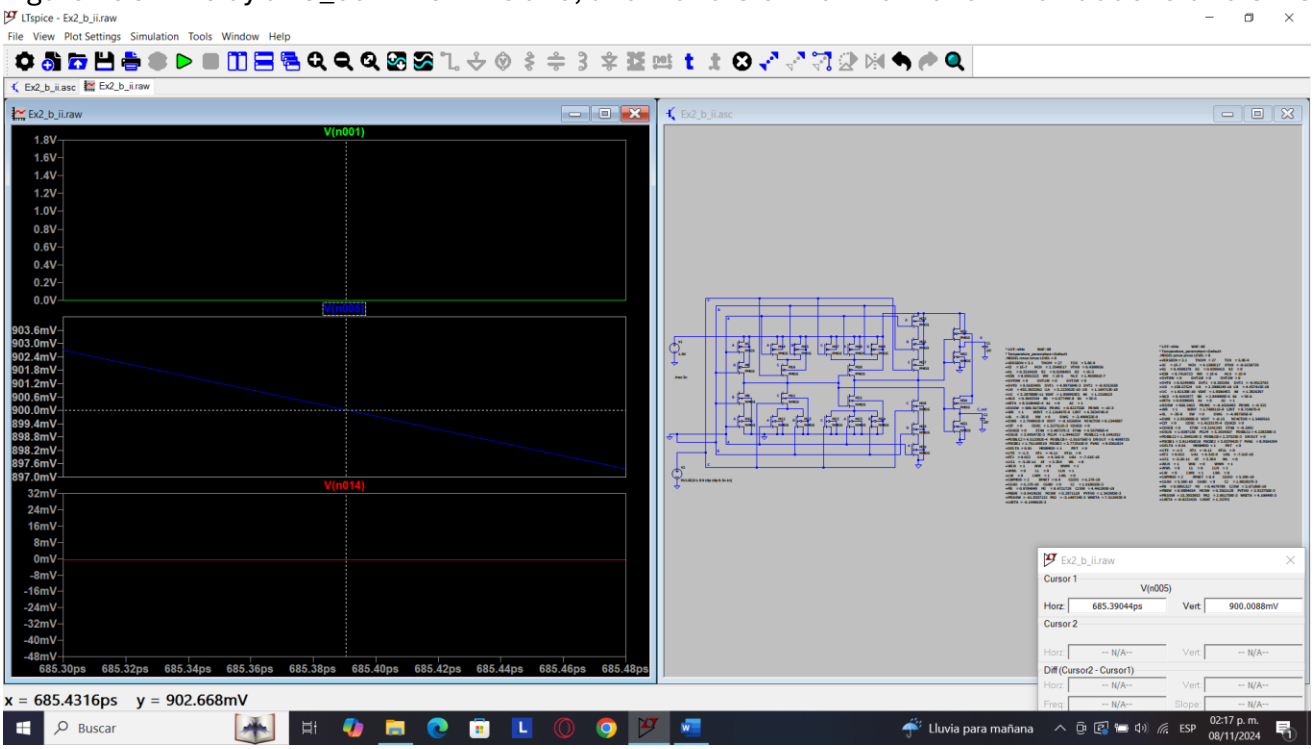


Figure 16Cii. Delay at S when A and B are at 0, and C switches from 1 to 0. Full adder static CMOS.

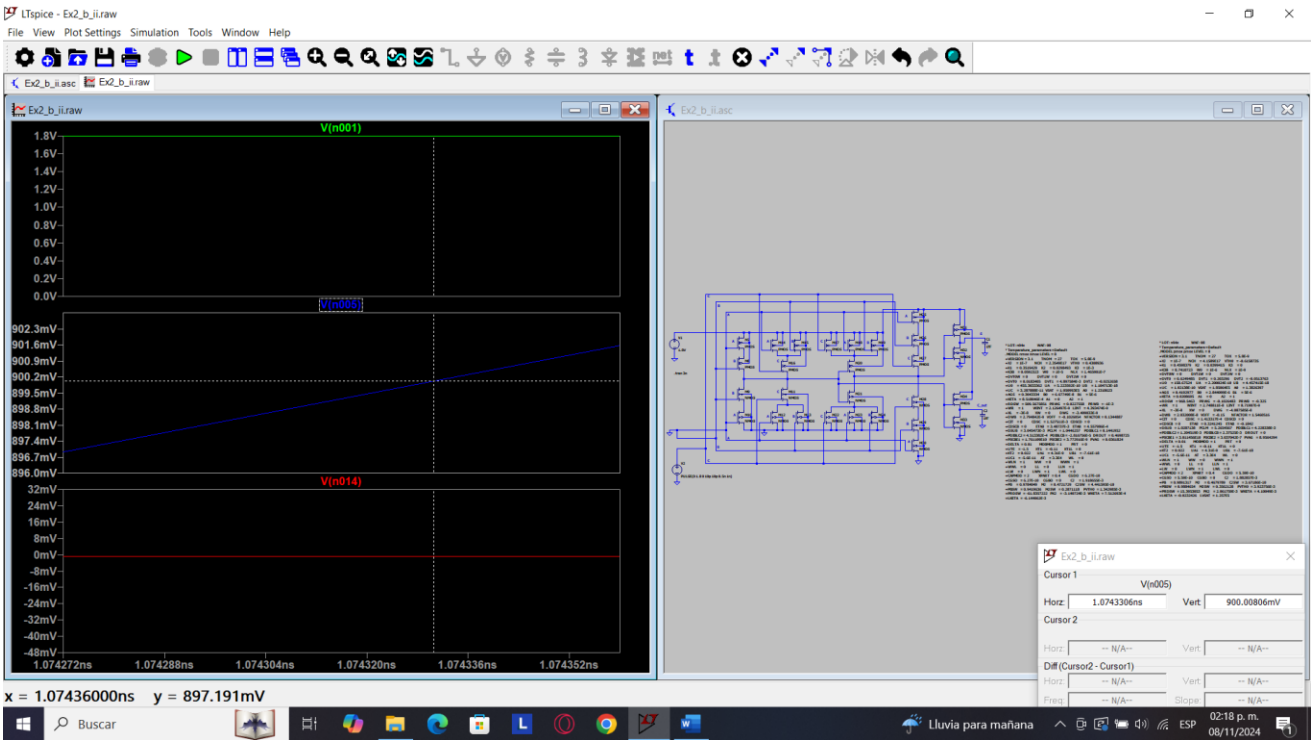


Figure 17Cii. Delay at S when A and B are at 0, and C switches from 0 to 1. Full adder static CMOS.

$$t_0 = 0.515\ ns$$
$$t_1 = 0.68539044\ ns$$
$$t_2 = 1.005\ ns$$
$$t_3 = 1.0743306\ ns$$
$$Worst\ Case\ Propagation\ delay =$$
$$t_{phl} = t_1 - t_0 = 0.68539044\ ns - 0.515\ ns = 0.17039044\ ns$$
$$t_{plh} = t_3 - t_2 = 0.17039044\ ns - 1.005\ ns = 0.0693306\ ns$$
$$t_p = \frac{t_{phl} + t_{plh}}{2} = (0.01168874\ ns + 0.0693306\ ns)/2 = 0.04050967\ ns$$

The worst-case propagation delay for charging is when inputs A, B and C change from (0,0,0) to (0,1,1) and the worst-case propagation delay for discharging is when inputs A, B and C change from (0,0,1) to (0,0,0). This can be seen in Figures 15Cii and 16Cii.

$$Area =$$
$$A = A_{PMOS} + A_{NMOS} = 11 * 180 * 880 + 2 * 180 * 440 + 9 * 180 * 1760$$
$$+3 * 180 * 1320 + 3 * 180 * 2640$$
$$= 6{,}890{,}400\ nm^2$$
$$Worst\ Case\ Power = 6.4841\ mW$$

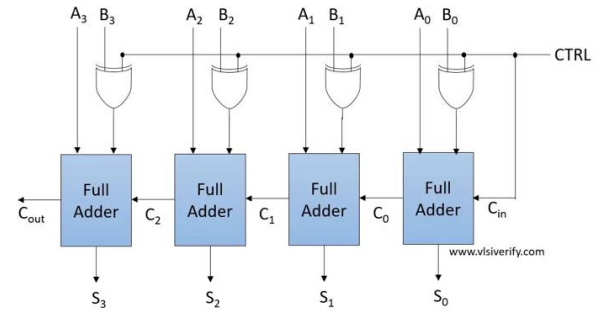
The worst power consumption scenario is when A, B, and C are both switching from 0 to 1 because it is the situation with most transistors with changing input, which leads to dynamic power consumption (Seen in Figures 6Cii and 9Cii).

To summarize, for the half-adder designs with transmission gates and static CMOS, the transmission gate design had a better propagation delay (Worst-case) of 0.08 ns than the 0.09 ns with the static CMOS design. The area is also better for the transmission gate design with an area of 1,346,400 nm2, compared to the static CMOS design of 3,168,000 nm2. Finally, the power

consumption is also better from the transmission gate design with a worst-case power consumption of 2.0542 mW compared to the worst-case power consumption of the static CMOS design of 3.8063 mW. So, the transmission gate design is better than the static CMOS design for the half-adder.

For the full adder design with transmission gates and static CMOS (Mirror full adder design), the transmission gate design had a better propagation delay (Worst-case) of 0.009 ns than the 0.04 ns with the static mirror full adder design. The area is also better for the transmission gate design with an area of 1,900,800 nm2, compared to the mirror full adder design of 6,890,400 nm2. Finally, the power consumption is also better from the transmission gate design with a worst-case power consumption of 3.2091 mW compared to the worst-case power consumption of the mirror full adder design of 6.4841 mW. So, the transmission gate design is better than the static CMOS design for the full adder.

3. Using the better design (from Q2), make a 4-bit adder and subtractor. Your design(s) should be able to perform $Y = A + B$ and $Y = A - B$, where both A and B are 4-bit integer inputs. You can use any design method for the other gates that are not part of the full/half adders (e.g., XOR gates in below picture). Verify that your circuit works as intended via at least five test input combinations.



4-Bit Adder Subtractor

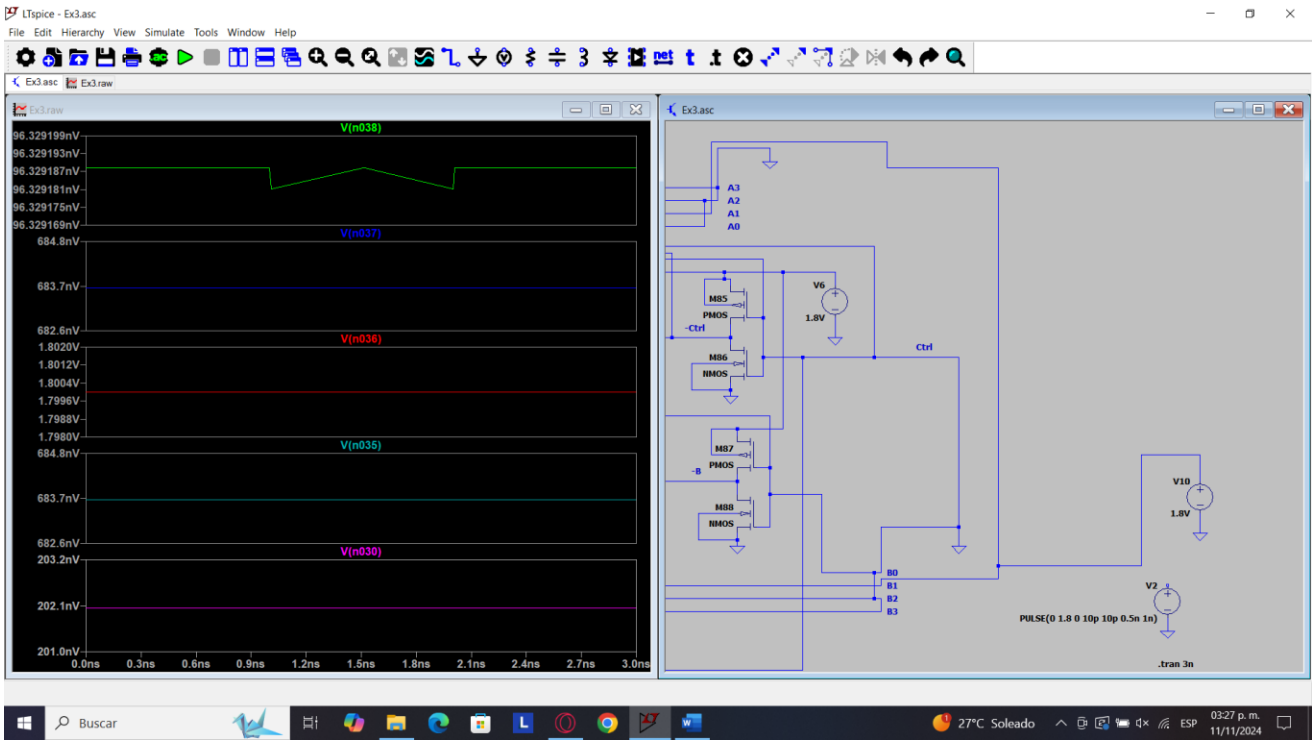


Figure 1D. Inputs A (0010), B (0010), and Ctrl equals 0. The order of graphs is S0 (Top graph), S1 (Middle top graph), S2 (Middle graph), S3 (Middle bottom graph), and Carry (Bottom graph).

For Figure 1D as expected the output of the sum of 0010 and 0010 equals 00100 (2 plus 2 equals 4).

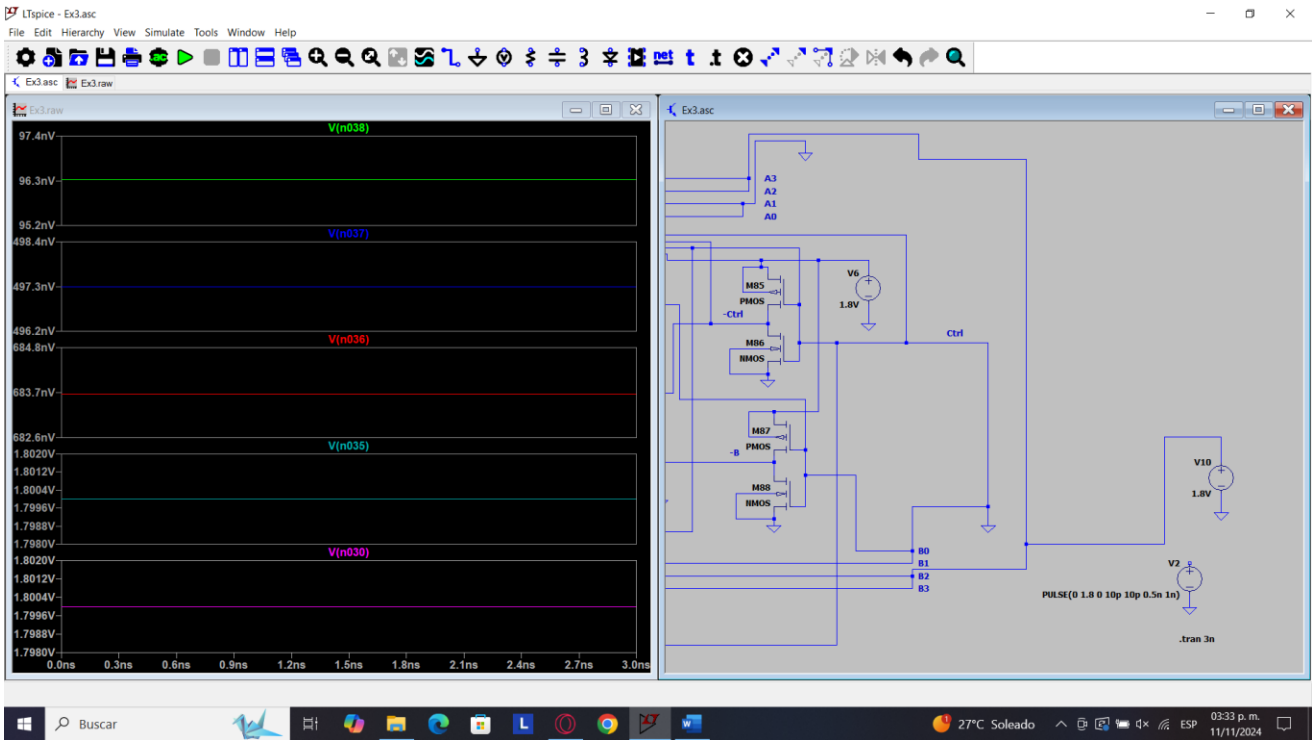


Figure 2D. Inputs A (1100), B (1100), and Ctrl equals 0. The order of graphs is S0 (Top graph), S1 (Middle top graph), S2 (Middle graph), S3 (Middle bottom graph), and Carry (Bottom graph).
For Figure 2D as expected the output of the sum of 1100 and 1100 equals 11000 (Sum of 12 plus 12 equals 24).

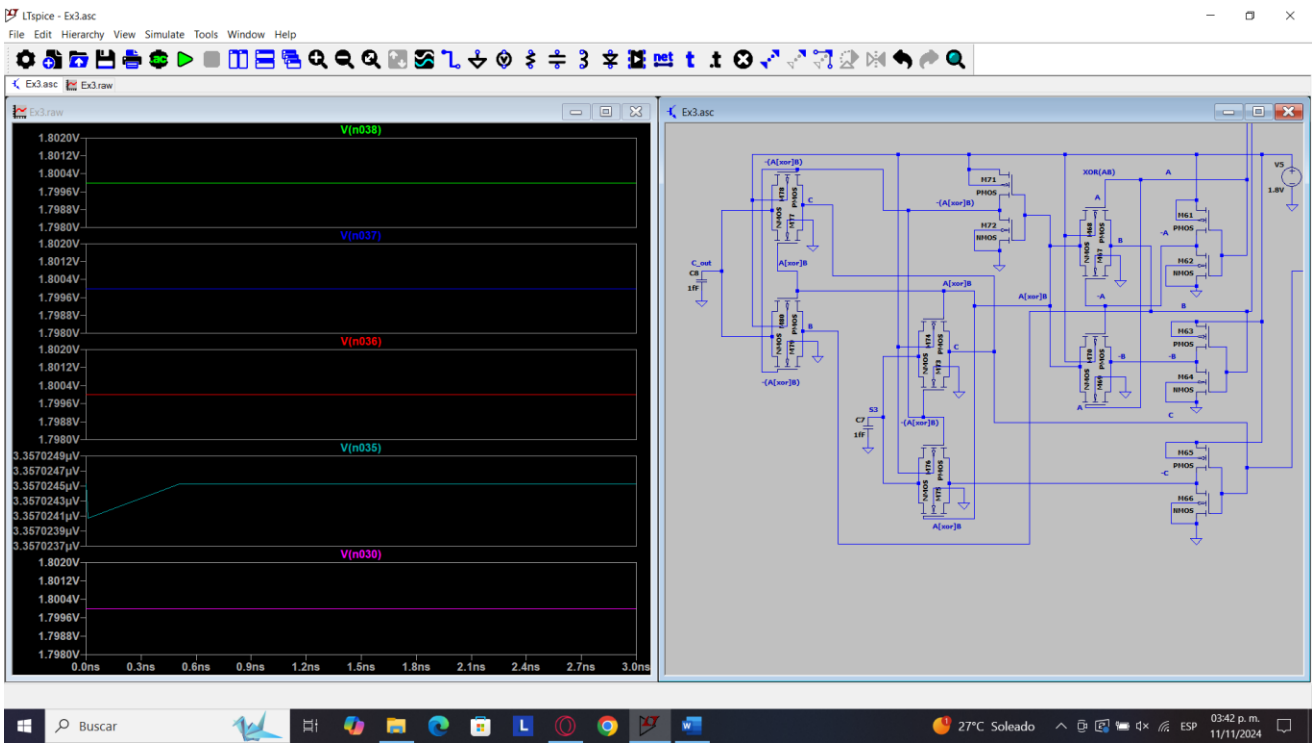


Figure 3D. Inputs A (1000), B (0001), and Ctrl equals 1. The order of graphs is S0 (Top graph), S1 (Middle top graph), S2 (Middle graph), S3 (Middle bottom graph), and Carry (Bottom graph).
For Figure 3D as expected the output of the subtraction of 1000 minus 0001 equals 0111 (Subtraction of 8 minus 1 equal 7). And the carry being 1.8 V is a sign of a subtraction.

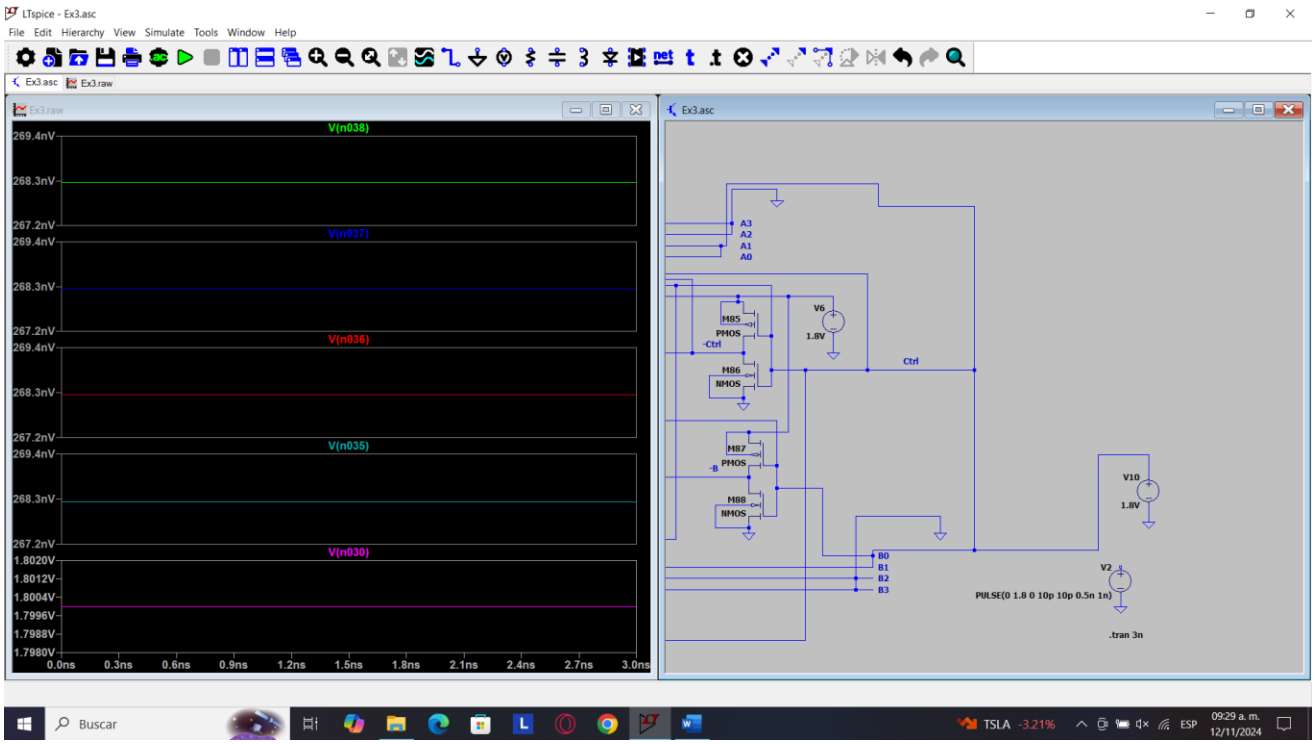


Figure 4D. Inputs A (0011), B (0011), and Ctrl equals 1. The order of graphs is S0 (Top graph), S1 (Middle top graph), S2 (Middle graph), S3 (Middle bottom graph), and Carry (Bottom graph).

For Figure 4D as expected the output of the subtraction of 0011 minus 0011 equals 0000 (Subtraction of 3 minus 3 equal 0). And the carry being 1.8 V is a sign of a subtraction.

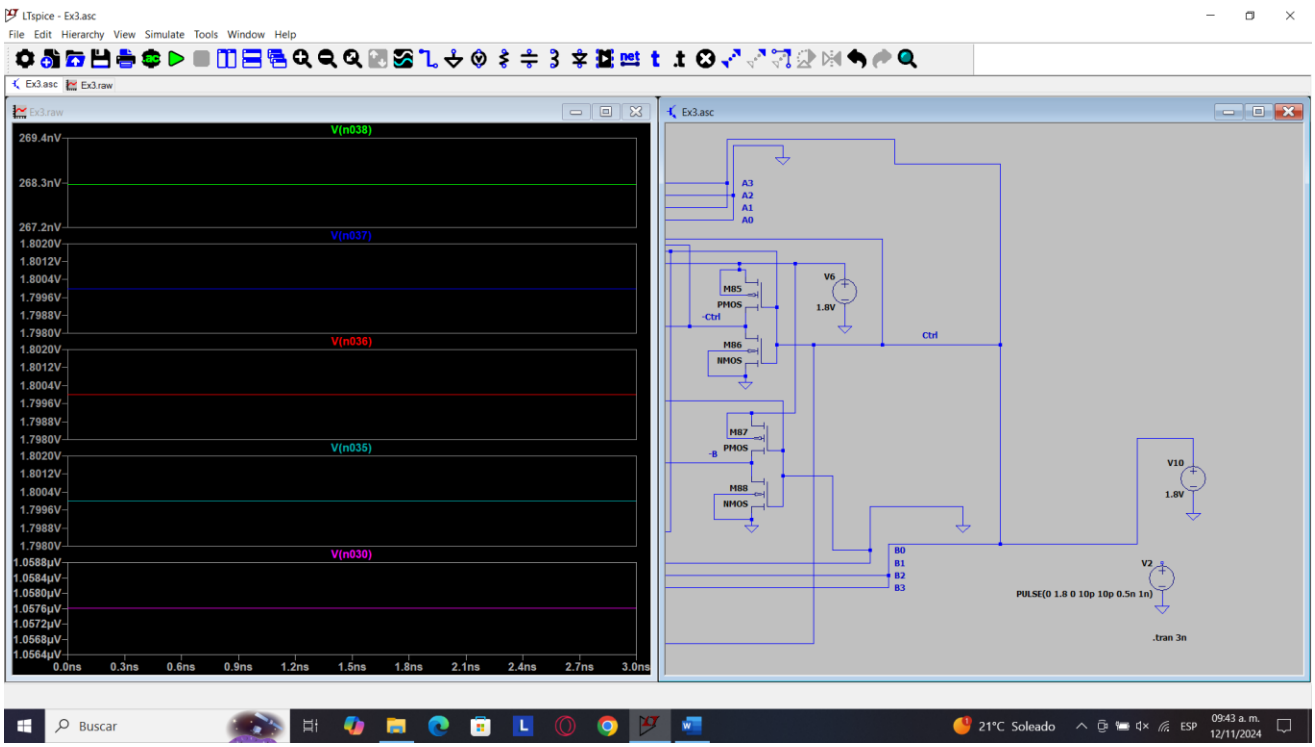


Figure 5D. Inputs A (1010), B (1100), and Ctrl equals 1. The order of graphs is S0 (Top graph), S1 (Middle top graph), S2 (Middle graph), S3 (Middle bottom graph), and Carry (Bottom graph).

For Figure 5D as expected the output of the subtraction of 1010 minus 1100 equals 1110 (2s complement of -2) (Subtraction of 10 minus 12 equal -2). The carry being 0 V even though Ctrl equals 1 means the result (1110) is in 2s complement form.

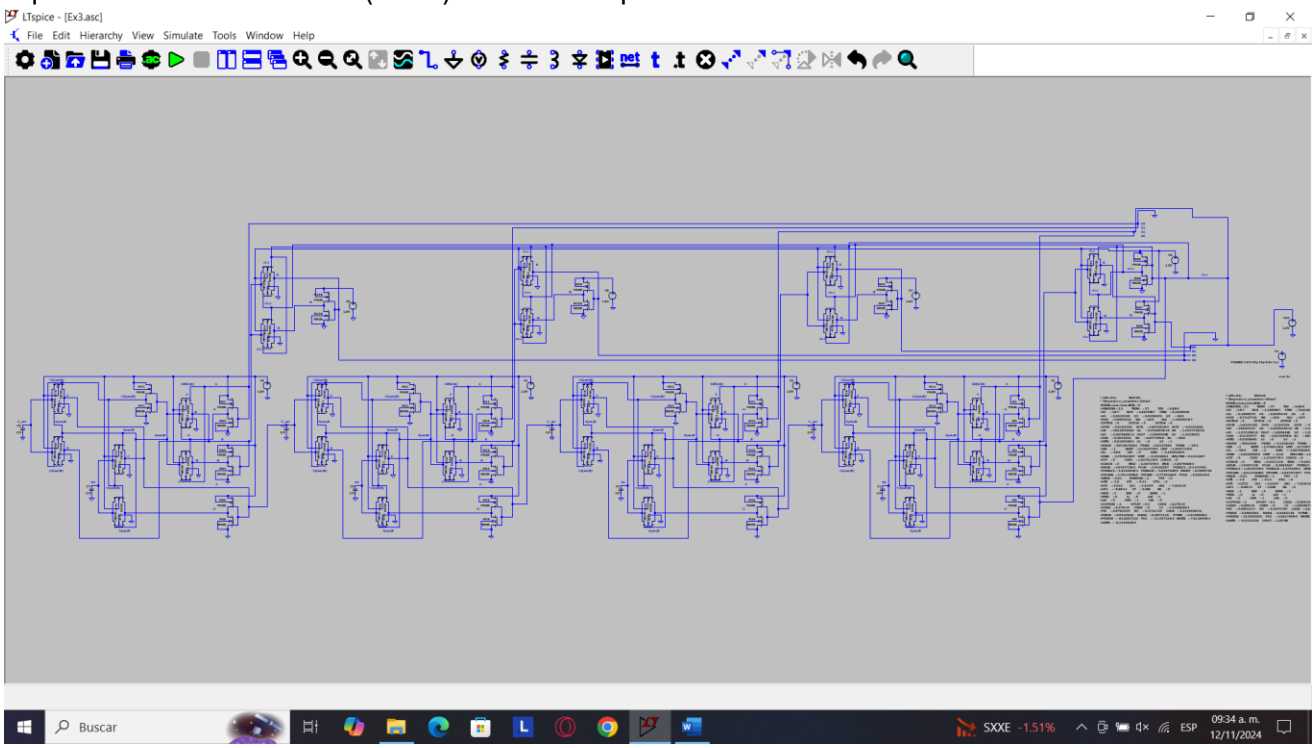
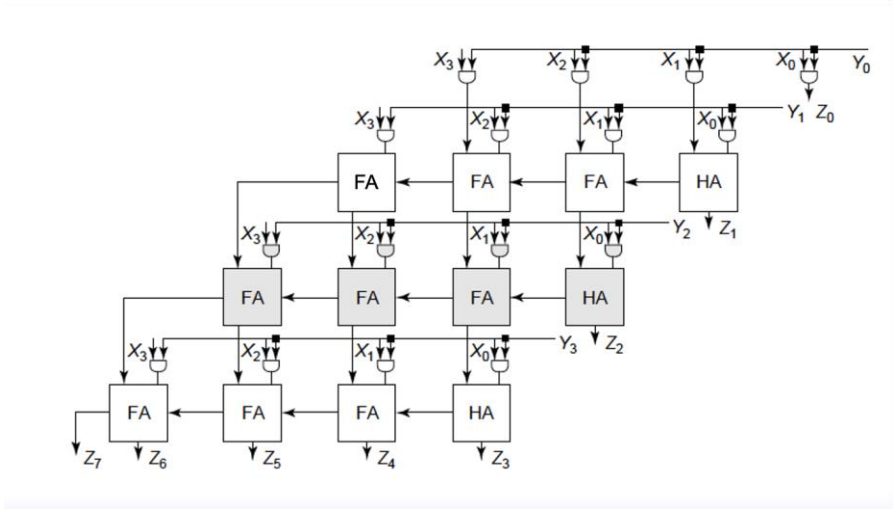


Figure 6D. Whole 4-bit adder circuit from a holistic point of view

4. Next, make a 4-bit integer multiplier. Your design should be able to perform $Y = A \times B$, where both A and B are 4-bit integers. Using the better design (from Q2) for the half and full adders. For other gates besides the adders, you can use any design style.



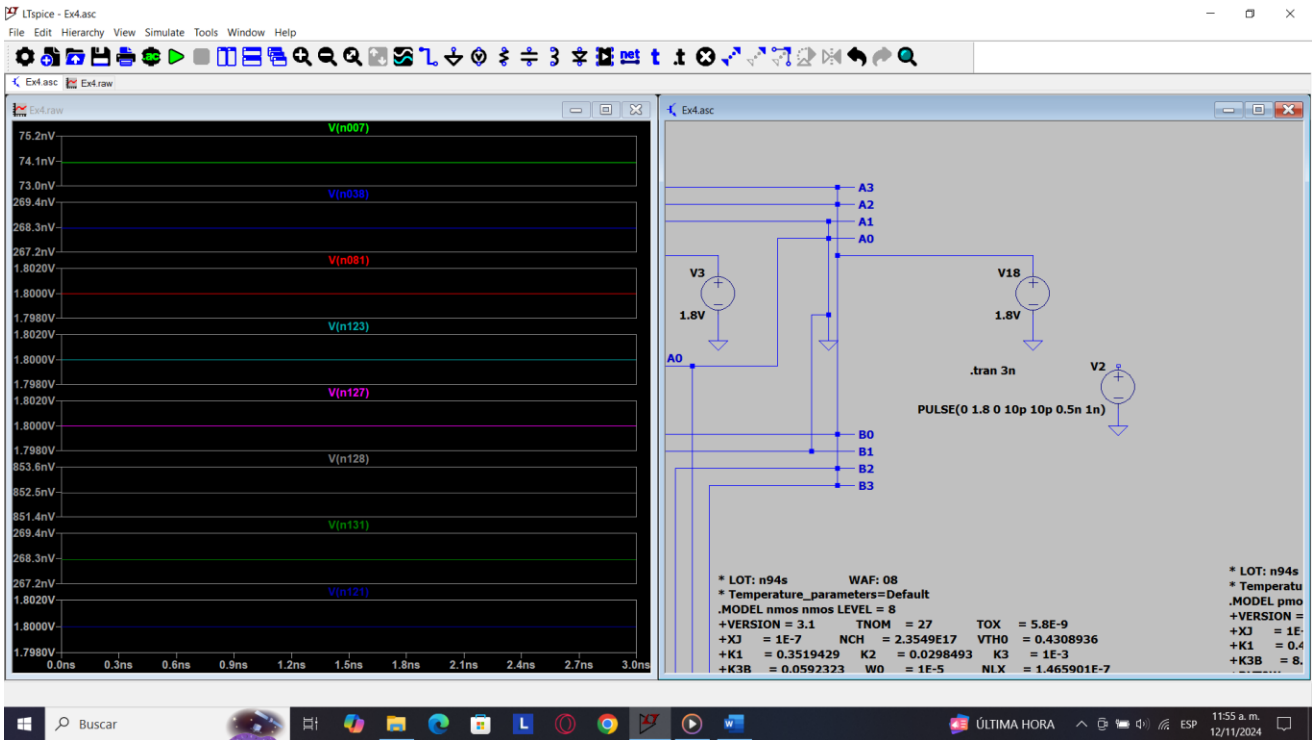


Figure 1E. Inputs A (1100), B (1101). The order of graphs is P0 (Top graph), P1 (Second graph from the top), P2 (Third graph), P3 (Fourth graph), P4 (Fifth graph), P5 (Sixth graph), P6 (Seventh graph), and P7 (Bottom graph).

As can be seen from Figure 1E, the 4-bit integer multiplier works correctly. In Figure 1E, it can be seen that when multiplying integer 12 (1100) times 13 (1101) equals 156 (10011100).

P0 refers to the resulting least important bit for the multiplication result, and P7 the most important.

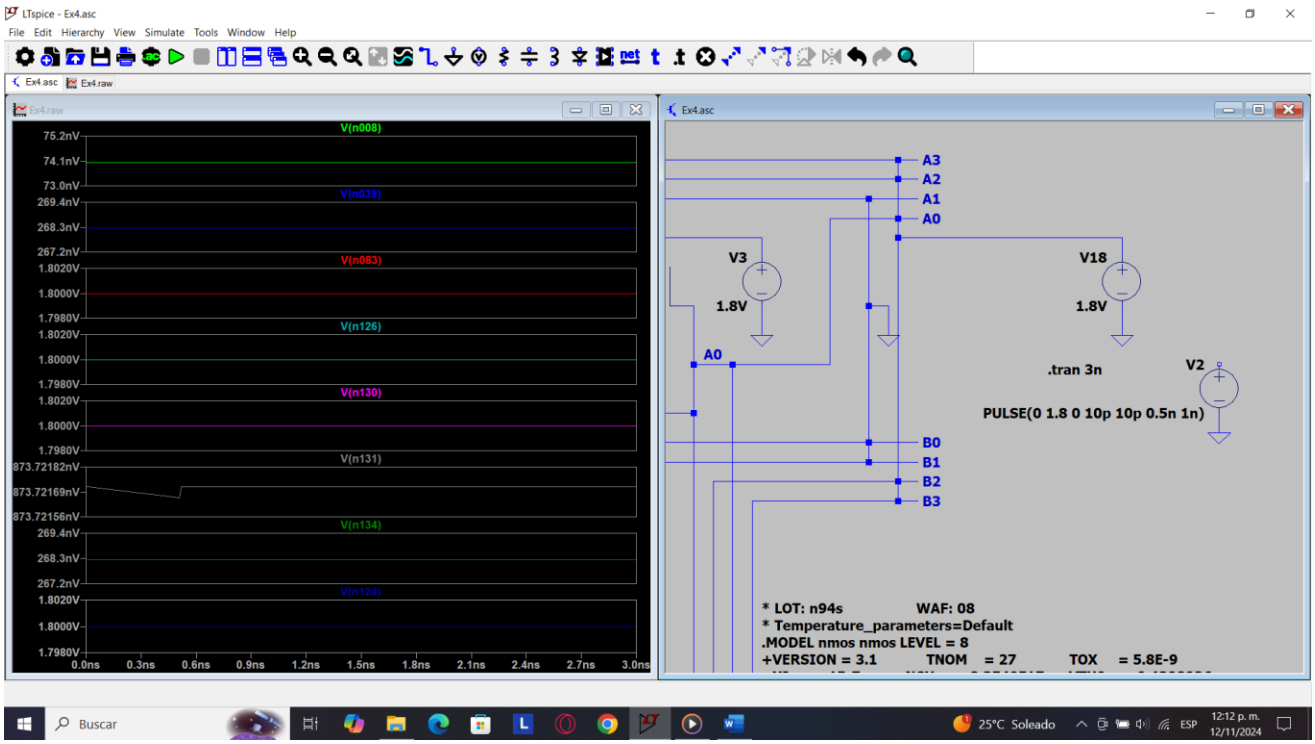


Figure 2E. Inputs A (1101), B (1100). The order of graphs is P0 (Top graph), P1 (Second graph from the top), P2 (Third graph), P3 (Fourth graph), P4 (Fifth graph), P5 (Sixth graph), P6 (Seventh graph), and P7 (Bottom graph).

In Figure 2E, it can be seen that when multiplying integer 13 (1101) times 12 (1100) equals 156 (10011100). And this shows that even if the integers are flipped (A being B, and B being A), it won't affect the result.

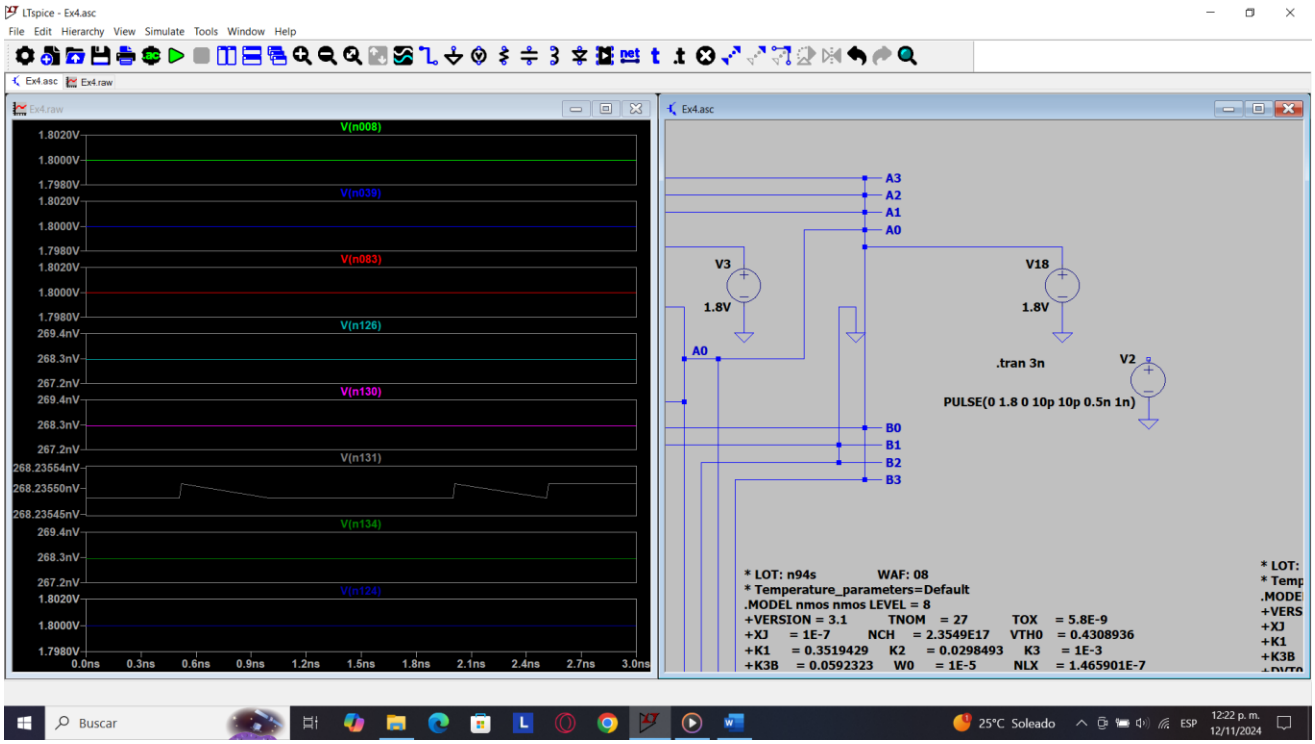


Figure 3E. Inputs A (1111), B (1001). The order of graphs is P0 (Top graph), P1 (Second graph from the top), P2 (Third graph), P3 (Fourth graph), P4 (Fifth graph), P5 (Sixth graph), P6 (Seventh graph), and P7 (Bottom graph).

In Figure 3E, it can be seen that when multiplying integer 15 (1111) times 9 (1001) equals 135 (10000111). The circuit is working properly.

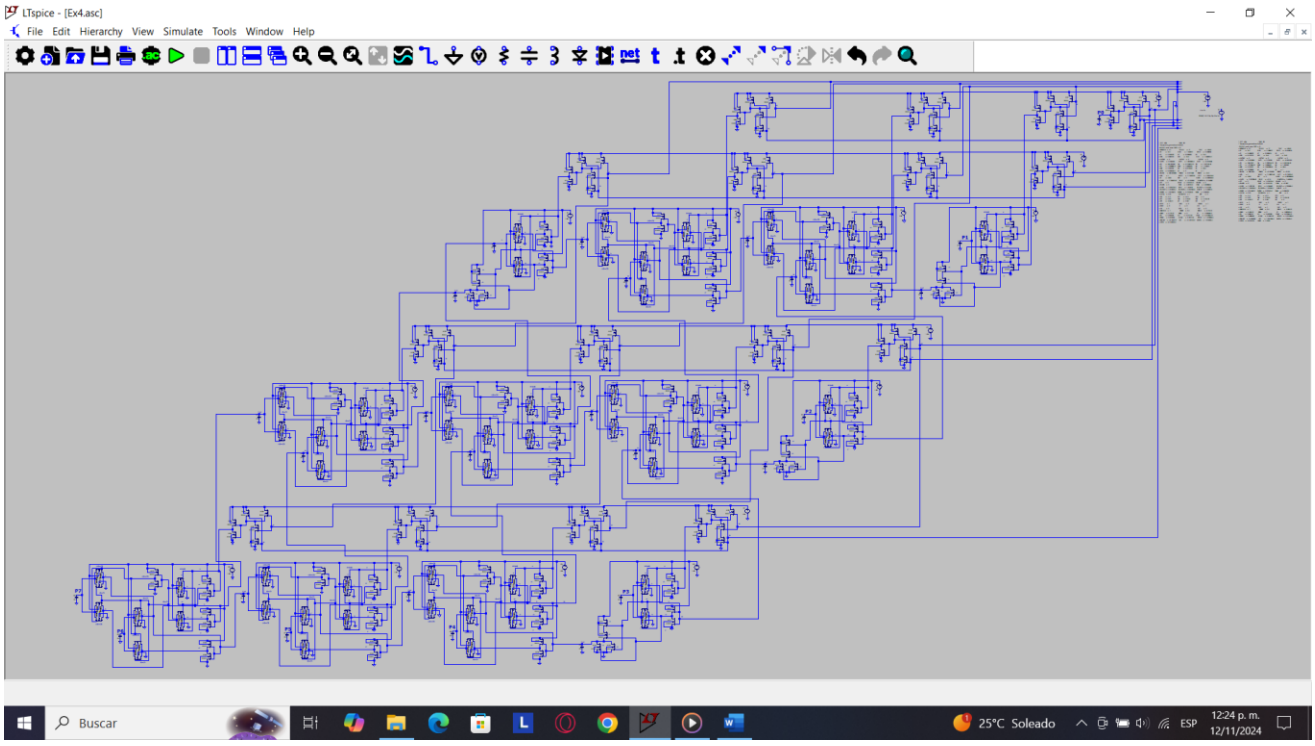
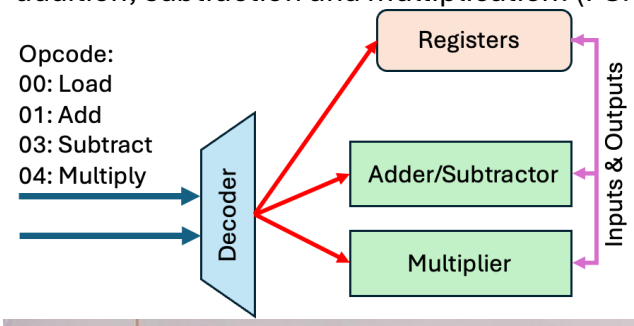


Figure 4E. Whole 4-bit multiplier circuit from a holistic point of view

From this point onwards (Q5 & Q6), it is optional for 5000 level students. If you attempt successfully, you will get bonus points.

5. Finally, let's put everything together to make our hypothetical ALU that can only do load, addition, subtraction and multiplication. (PS: Real ALUs can do lot more)



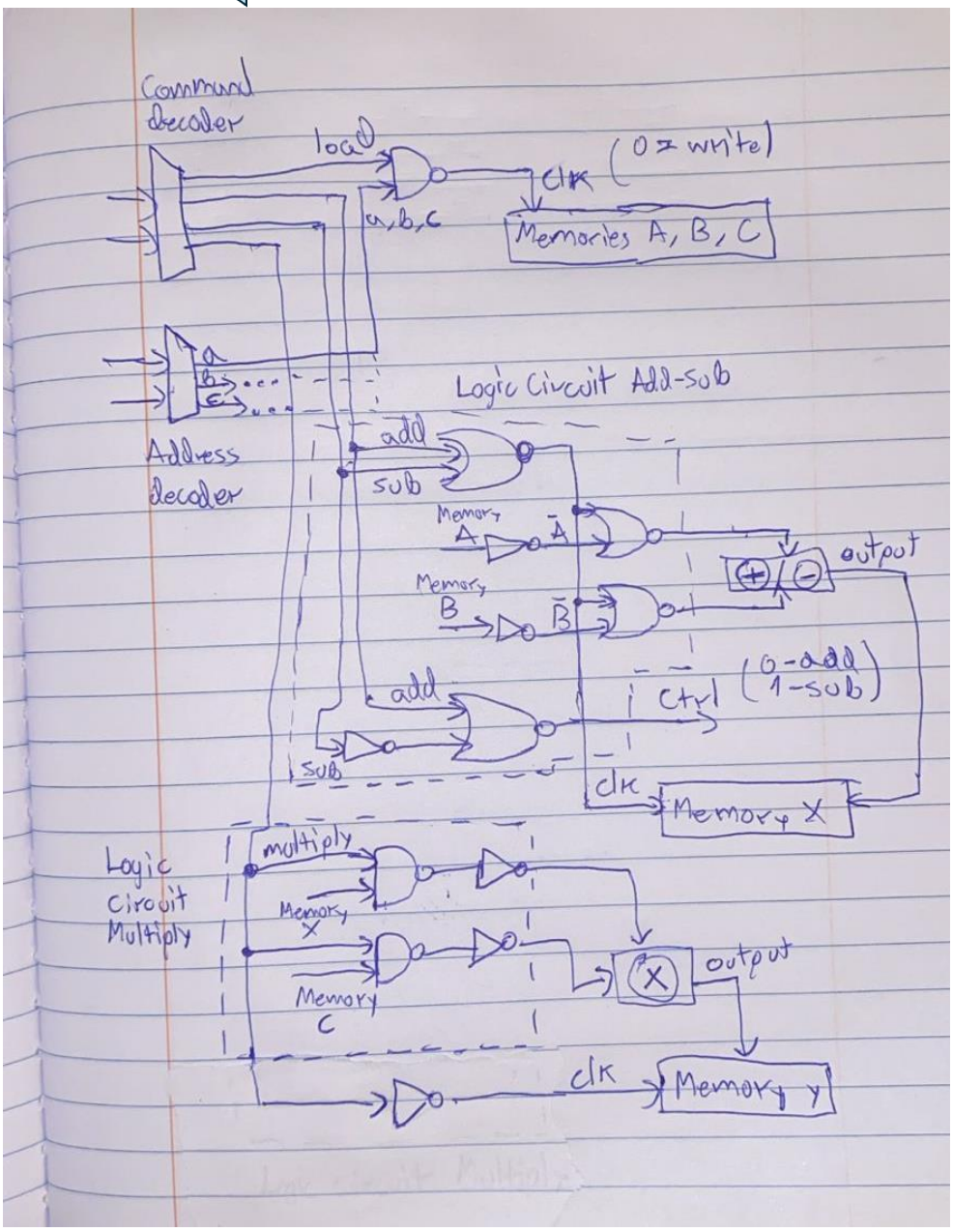


Figure 1F. ALU circuit view at the block and gate level

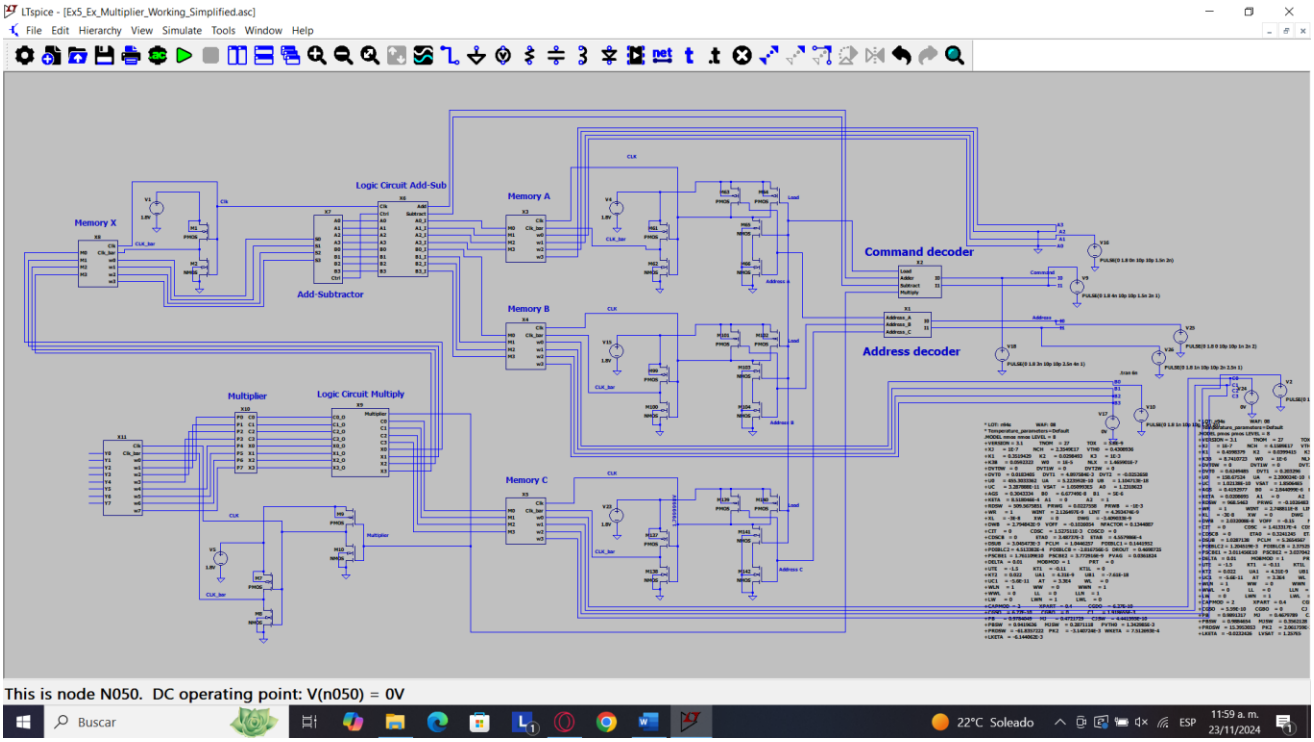


Figure 2F. ALU circuit with elements summarized in blocks

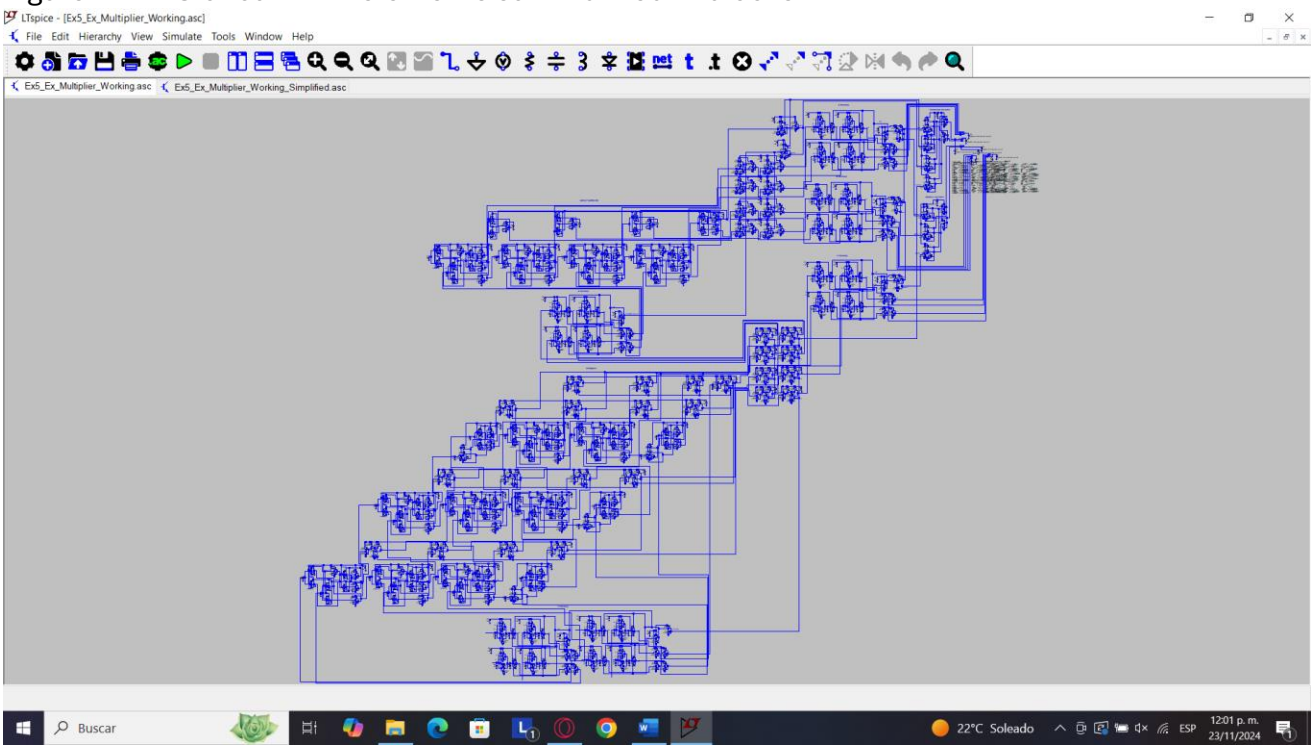


Figure 3F. ALU circuit with elements un-summarized

The reason for the inclusion of Figure 1F is to understand the gates involved in the processing of bits in between ALU elements. Figure 2F was included as the circuit that is being utilized for tests; this circuit uses blocks instead of having only transistors, so it is much more readable. And Figure 3F was included, in the case that it is desired to see the view at the transistor level. For revision on the proper functioning of the circuit see the next section.

6. You are given the following software program. Execute it on your ALU and verify that you are getting correct Y. Test with another input (A,B,C) combination to verify further.

Load 0010→A
Load 0001→B
A+B →X
Load 0100 →C
X*C → Y

I will show that each part of the Assembly code is working well, though I will do this with multiple Figures because otherwise if I show it all in one Figure the graphing area will be too crowded with graphs.

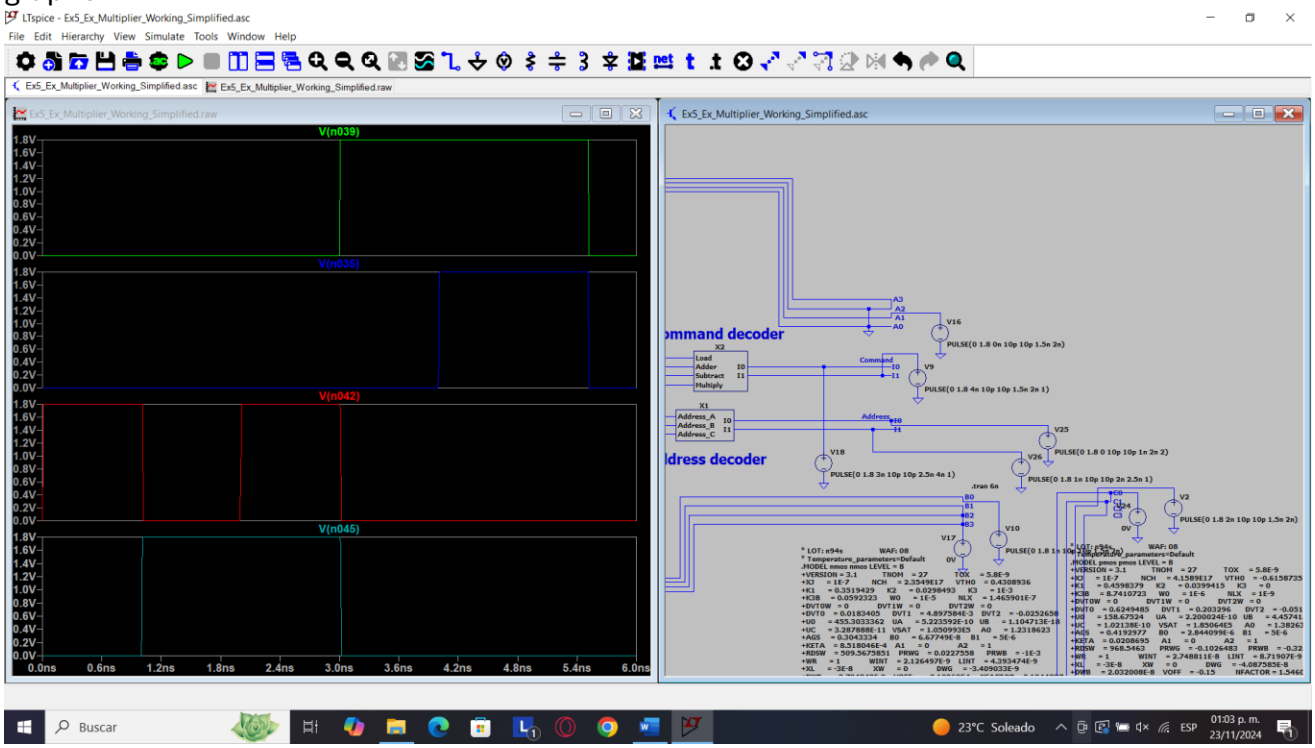


Figure 1G. Inputs of the system – Command decoder and Address decoder. I0 (Top graph), I1 (Top middle graph), I0 (Bottom middle graph), and I1 (Bottom graph).

As can be seen in the top two graphs, for the first 3 ns the Command decoder transforms the input of 00 into the loading command (1 ns per memory). And on the bottom two graphs are the inputs of the Address decoder, going from 01 (Activating A memory), to 10 (Activating B memory), to 11 (Activating C memory), and finally to 00 deactivating memory write.
Then, at 3 ns the command in the Command decoder changes from 00 to 01 which marks addition, and then at 4 ns the command changes to 11 signifying multiplication. Then, at 5.5 ns, the signals to the Command decoder are turned off marking the end of the use of the ALU.

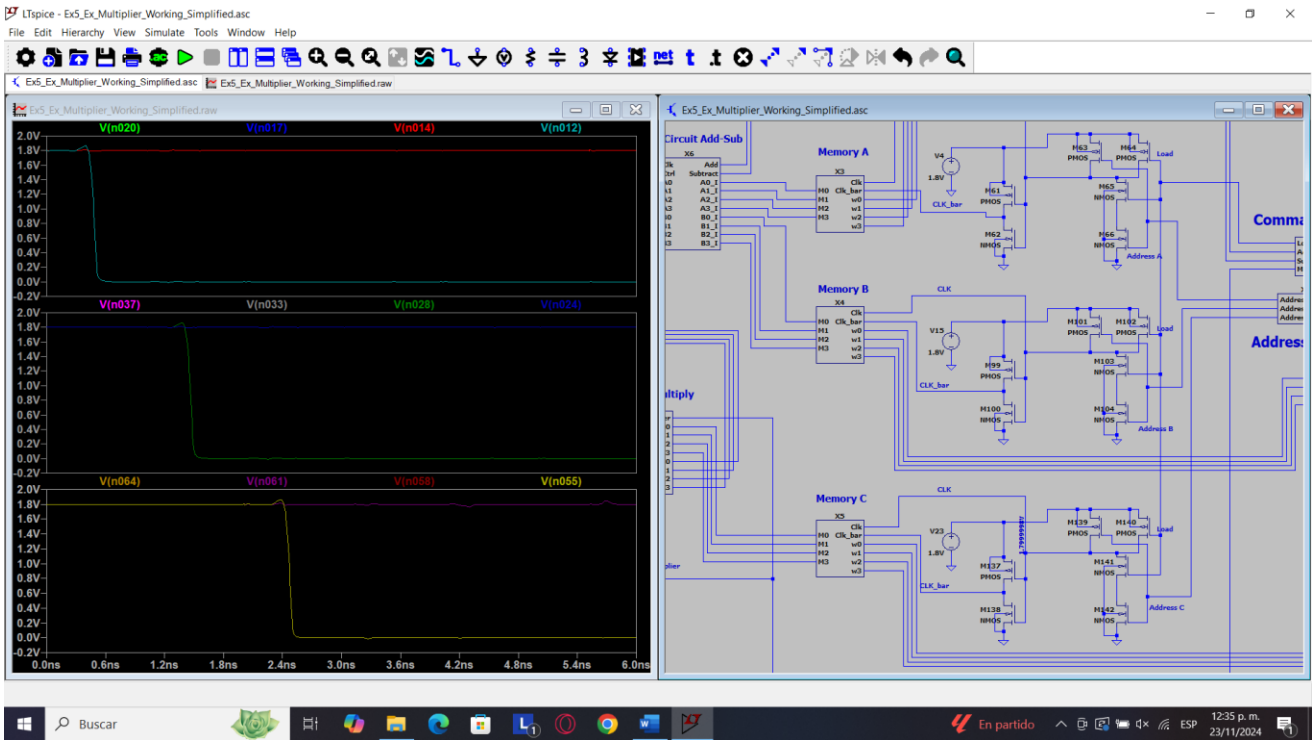


Figure 2G. Memory write on bits of Memory A, B, and C. Memory A (Top graph), memory B (Middle graph), memory C (Bottom graph).

Each graph of Figure 2G has four bits corresponding to the four bits of each memory; with [V(n#), V(n#), V(n#), V(n#)] each V(n#) having its own identifying color; the left most V(n#) corresponds to the most significant bit and the right most V(n#) corresponds to the least significant bit. And as can be seen A has saved correctly bits 0010, B saving 0001, and C saving 0100.

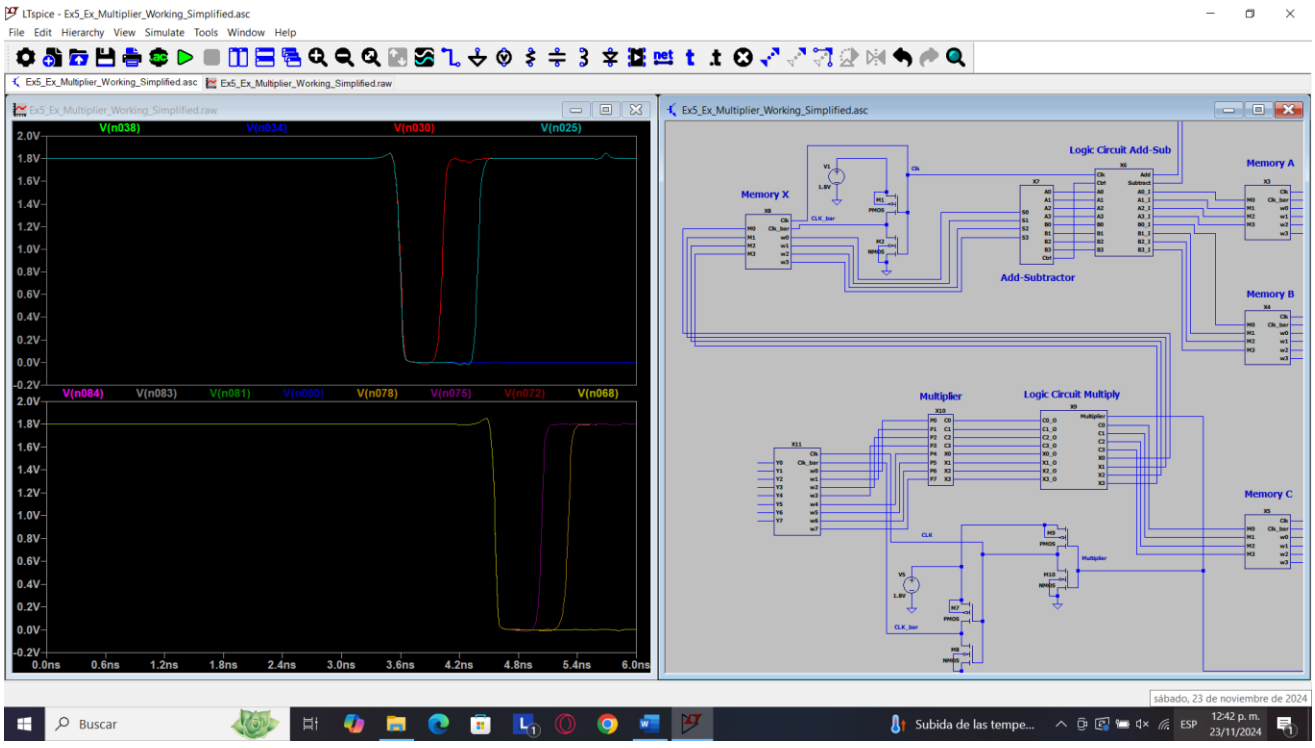


Figure 3G. Memory write on bits of Memory X (Top graph) and Memory Y (Bottom graph).

The logic of Figure 3G is the same as in Figure 2G, the most significant bit is the left most V(n#) and the least significant is the right most V(n#). And as can be seen, the circuit has saved correctly the sum of A (0010) and B (0001) which is saved on X with a value 0011. Additionally, Y has also been saved correctly, with the multiplication of X (0011) and C (0100) which equals Y (00001100).

The Assembly code for the checking of the circuit with other inputs is the following:

Load 0110 → A
Load 0100 → B
Load 0100 → C
A-B → X
X*C → Y

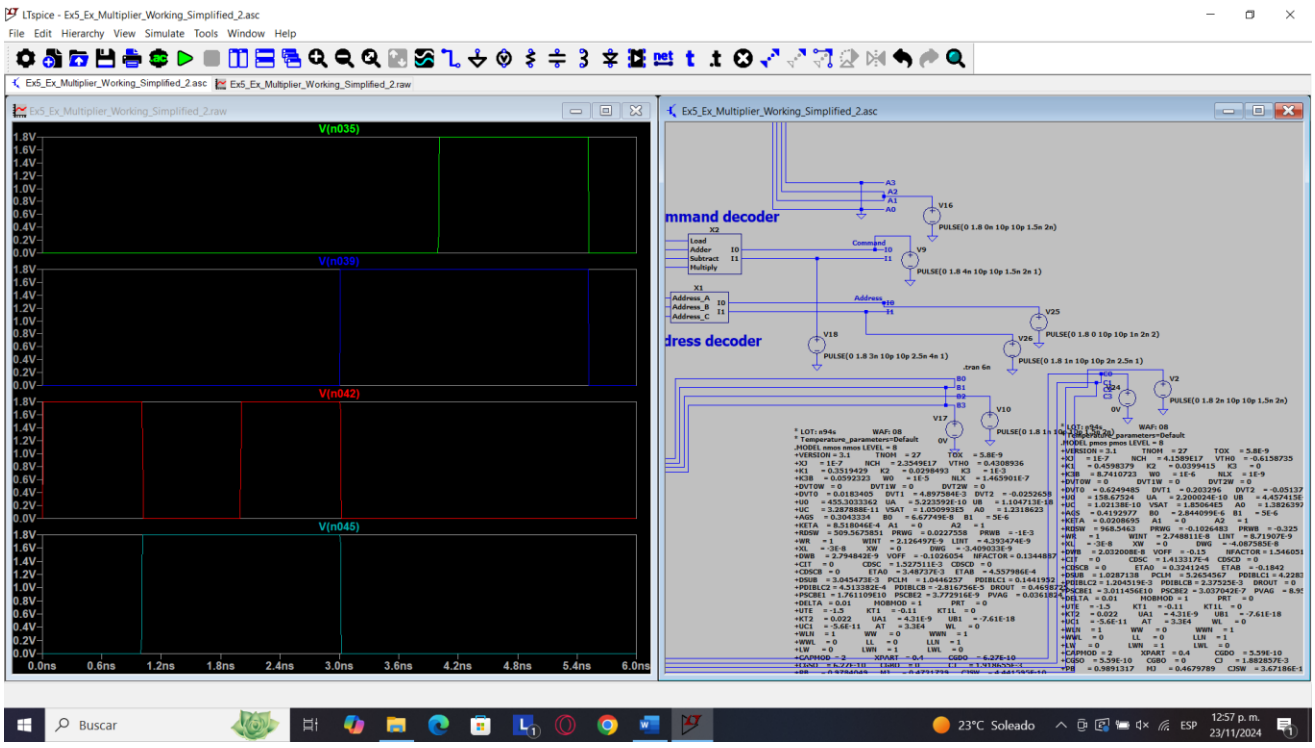


Figure 4G. Inputs of the system with new Assembly code – Command decoder and Address decoder. I0 (Top graph), I1 (Top middle graph), I0 (Bottom middle graph), and I1 (Bottom graph).

As can be seen in the top two graphs, for the first 3 ns the Command decoder transforms the input of 00 into the loading command (1 ns per memory). And on the bottom two graphs are the inputs of the Address decoder, going from 01 (Activating A memory), to 10 (Activating B memory), to 11 (Activating C memory), and finally to 00 deactivating memory write.
Then, at 3 ns the command in the Command decoder changes from 00 to 10 which marks subtraction, and then at 4 ns the command changes to 11 signifying multiplication. Then, at 5.5 ns, the signals to the Command decoder are turned off marking the end of the use of the ALU.

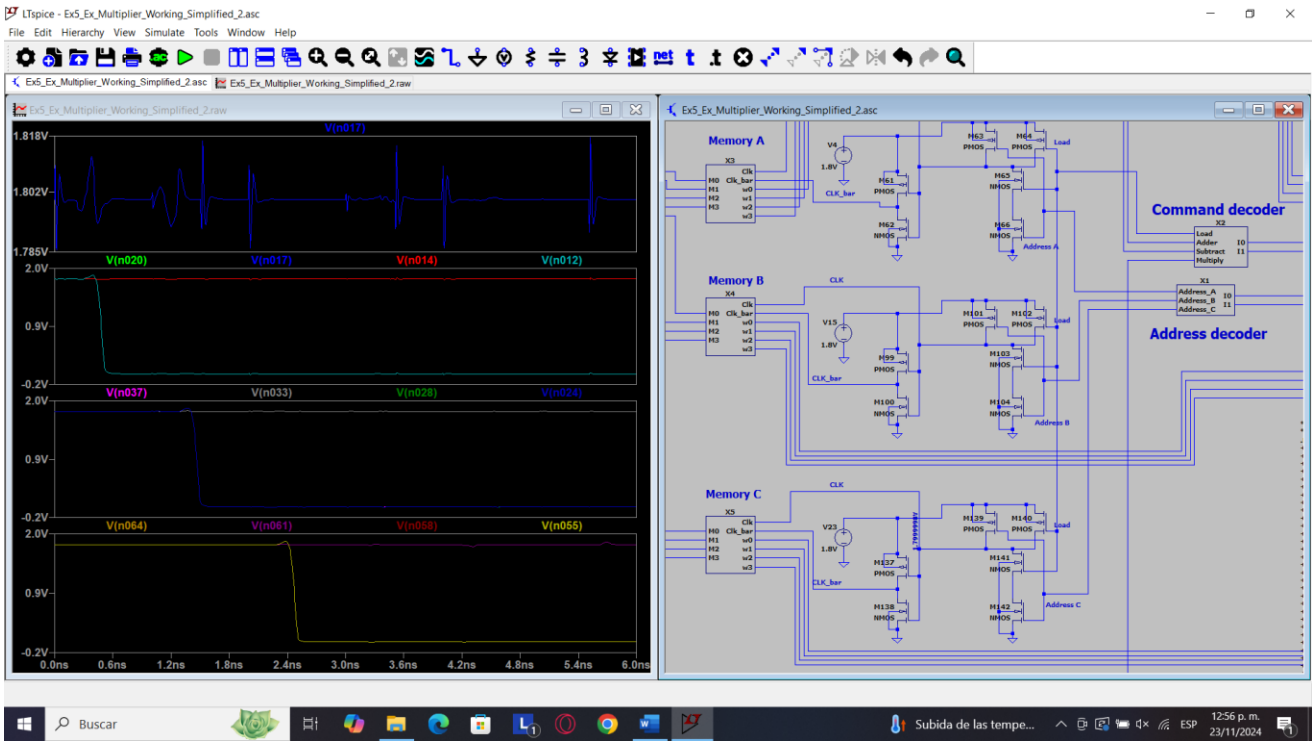


Figure 5G. Memory write on bits of Memory A, B, and C with new Assembly code. Second most significant bit of memory A (Top graph), Memory A (Top middle graph), memory B (Bottom middle graph), memory C (Bottom graph).

The logic of Figure 5G is the same as in Figure 2G, the most significant bit is the left most V(n#) and the least significant is the right most V(n#). And as can be seen, A has saved correctly bits 0110, B saving 0100, and C saving 0100. The reason for the inclusion of the top graph in Figure 5G is because the second most significant of Memory A can't be seen in the graph of Memory A, so it is included on the top graph.

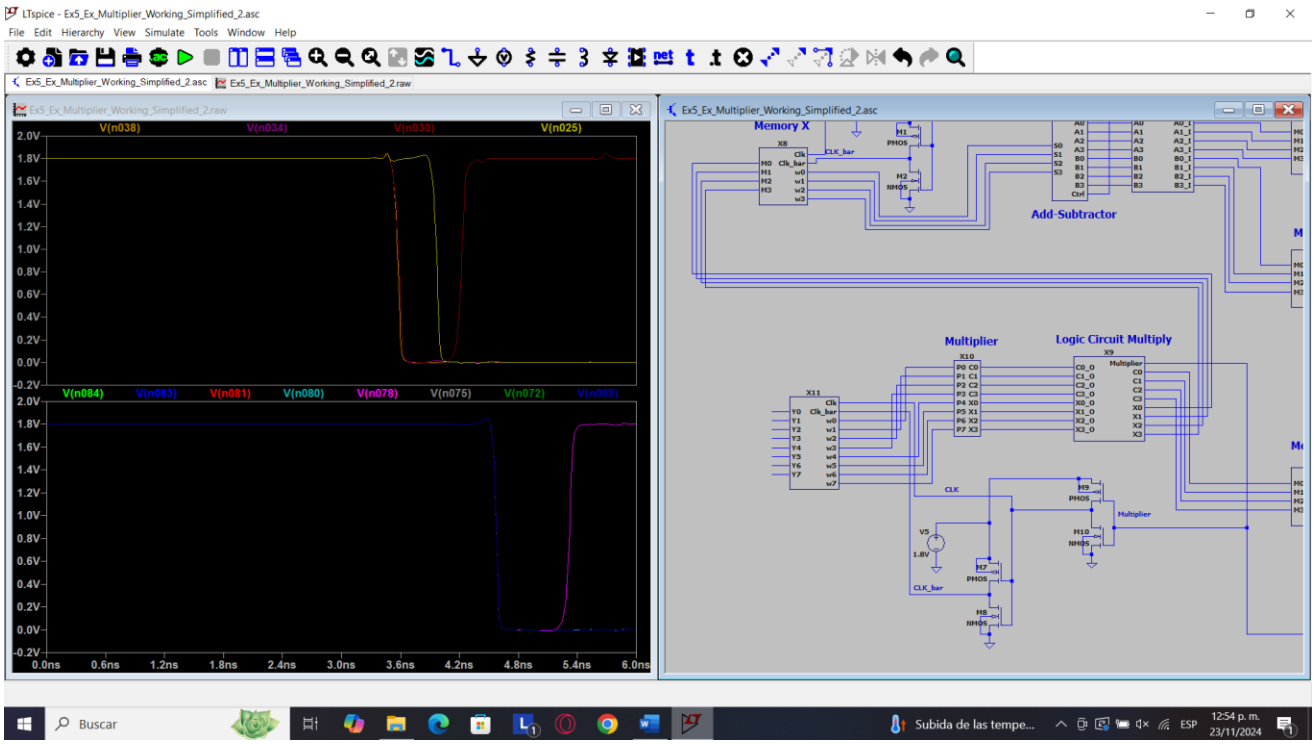


Figure 6G. Memory write on bits of Memory X (Top graph) and Memory Y (Bottom graph) for new Assembly code.

The logic of Figure 6G is the same as in Figure 2G, the most significant bit is the left most V(n#) and the least significant is the right most V(n#). And as can be seen, the circuit has saved correctly the subtraction of A (0110) minus B (0100) which is saved on X with a value 0010. Additionally, Y has also been saved correctly, with the multiplication of X (0010) and C (0100) which equals Y (00001000).

With this, it can be seen that the circuit is functioning properly.

Project is due on Dec 1st, 11:59:59 PM. For all experiments, use LTSpice (recommended) or Cadence with 180nm. There will not be any extensions due to tool related problems. So start early to avoid last minute problems. Please submit the screenshots of your outputs (entire screen **including date/time when screenshot is taken**). If you use any online resources, proper citation is required. No copying or sharing circuits/screenshots/answers. For any other questions, please email me. I might request live demo if needed later on.

References:

Full mirror adder reference:
Narasimhamurthy K. C. [VU3DWF]. (2020, 3 diciembre). *Carry look ahead and Mirror adder* [Vídeo]. YouTube. https://www.youtube.com/watch?v=j51ONH7h_Sg

4-bit adder reference:
ALL ABOUT ELECTRONICS. (2022, 19 febrero). *4-bit Adder and Subtractor Circuit Explained* [Vídeo]. YouTube. https://www.youtube.com/watch?v=J7gPUP0aRug

4-bit multiplier reference:
ALL ABOUT ELECTRONICS. (2023, 30 abril). *How to Design Binary Multiplier Circuit | 2-bit, 3-bit, and 4-bit Binary Multiplier Explained* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=O34KquoMpT0