Two-bits Binary Parallel Ripple Carry Adder Using CMOS NAND Gates

Ahamd Hamad - 1180060 Baraa Atta - 1180445 Tarek Ziedan - 1180404

Faculty of Engineering and Technology, Department of Computer and Electrical Engineering at Birzeit University
Ramallah, Palestine.

1180060@student.birzeit.edu 1180445@student.birzeit.edu 1180404@student.birzeit.edu

Abstract

The main aim of this project was to implement our understanding of building logic gates and use these gates in building higher components, using transistor level. So, CMOS transistors which we took in Integrated Circuits course this semester was implemented in our project which is an area efficient, low power, reduced delay two-bits binary parallel ripple carry adder using NAND gates, using both DSCH tool for basic schematic design and Micro Wind tool which allows us to work in very precise way at the transistor level to use all transistor components individually. NAND gates must be reduced since they are the source of transistors that we try to reduce them the most on the chip, so we can have smaller area, more efficient design with low power and cost.

Introduction

First, we tried to implement a full adder using multiple logic gates based on our analyzation of its circuit and K-Maps, so, we built the design using DSCH with only P and N transistors, and we ran the simulation which was good enough, so we went to the Micro Wind tool to apply our schematic design on, but before using the tool, we draw a stick diagram to help us in implementing the circuit in layout, furthermore, we built our own transistor in the tool, which we will show later, it was special transistor for our task, with our defined dimensions and components, finally, we used it to bult our layout. And after building the circuit we ran the file, it gave acceptable logic results as it was on the schematic earlier, but unfortunately there appeared few glitches in the sum output, which we tried hard to remove and finally could. Then, we decided to build a new design, which will also be better than the given and the previous, with reduced power, area number of transistors and delay. This new design was implemented based on NAND gates only, which were 9 gates. This means that we have to build 36 transistors to implement this circuit for one-bit full adder, which is 2 transistors less than the earlier.

So, we built the schematic design using DSCH tool again, and ran the file so we can make sure that the circuit is logically true, and it was. So, we went to the Micro Wind again to apply our understanding, so we used our P and N transistors, and implemented the schematic in layout form, finally, we ran the file to see the results, and they were convincing and acceptable.

Problem Statement

In this report, we will introduce our two designs for our two-bits adder, and compare them with all requirements, and finally, determine which of them is better.

Methodology and Solution

A	B	Carry in	Sum	Carry out
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

The table shown above, represents a full adder truth table, it has three inputs, and two outputs, one is the sum and the other is the carry. Now, we will create K-Maps to implement our design in Boolean algebra.

0	1	0	1
1	0	1	0

the table shown above, represents the sum equation, which is Sum=a XOR b XOR carry in.

0	0	1	0
0	1	1	1

the table shown above, represents the carry equation, which is ~Carry out= A.B + Carryin(A+B)

now, we will use the pull-up and pull-down networks to implement the equations shown earlier in a CMOS transistor level.

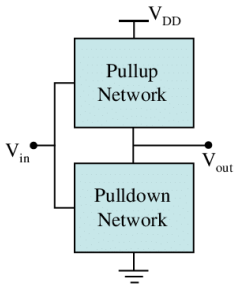

Figure 1: General Pull Up & Down Network

The figure shown above, represents Pull-up and Pull-down networks design.

Now, we implemented the carry out equation using DSCH tool as the following figure shows:

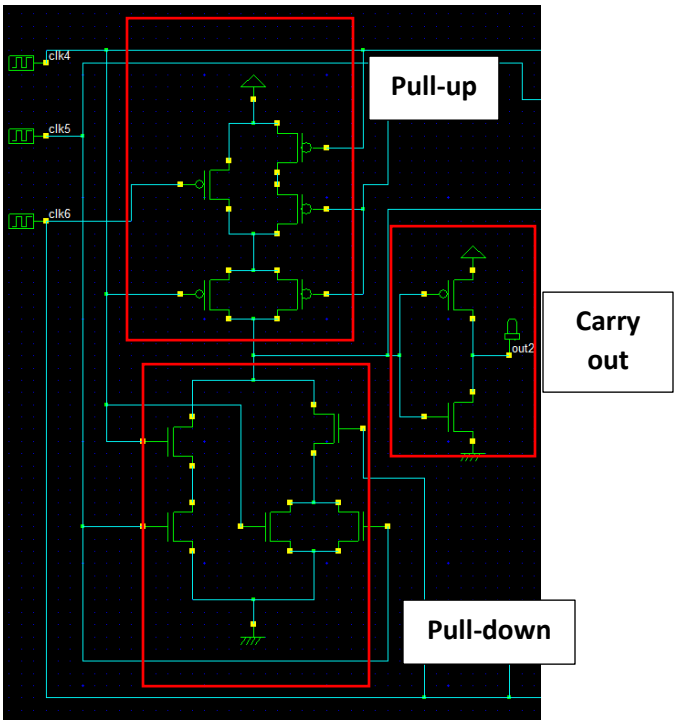

Figure 2: Carry Schematic

The design shown above consists of both pull up circuit and pull-down circuit, we can clearly see the intersection between them, and if we trace the transistors, we clearly see the pull up network =(A+B) . Carryin+(A.B) which is the inverted equation, and for the Pull-down network

= A.B + Carryin(A+B), finally, we see the output entered to an inverter since it is negated.

Now, since the XOR gate is very consuming to implement using transistors, and it is a main logical gate for the sum equation, we will try to figure a new equation to use, that does not contain XOR in it.

A	B	~(Carry in)	Sum	Carry out	When the SUM is 1?
0	0	0	0	0	
0	0	1	1	0	Cin. ~Cout
0	1	0	1	0	B. ~Cout
0	1	1	0	1	
1	0	0	1	0	A. ~Cout
1	0	1	0	1	
1	1	0	0	1	
1	1	1	1	1	A.B.Cin

Now, we have a new equation that we can use to find the sum, which is the sum of previous red columns were sum is equal to one.

Sum = (Cin. ~Cout) + (A. ~Cout) + (B. ~Cout) + (A.B. Cin) which we can also reduce using Boolean algebra to:

~Sum = (A+B+Cin). Cout + A.B. Cin

Now, we will use this equation to implement using DSCH to get the following:

Note that we will use inverter to the output of Pull-up and Pull-down circuits since the output is inverted originally so we can get a true value of the sum.

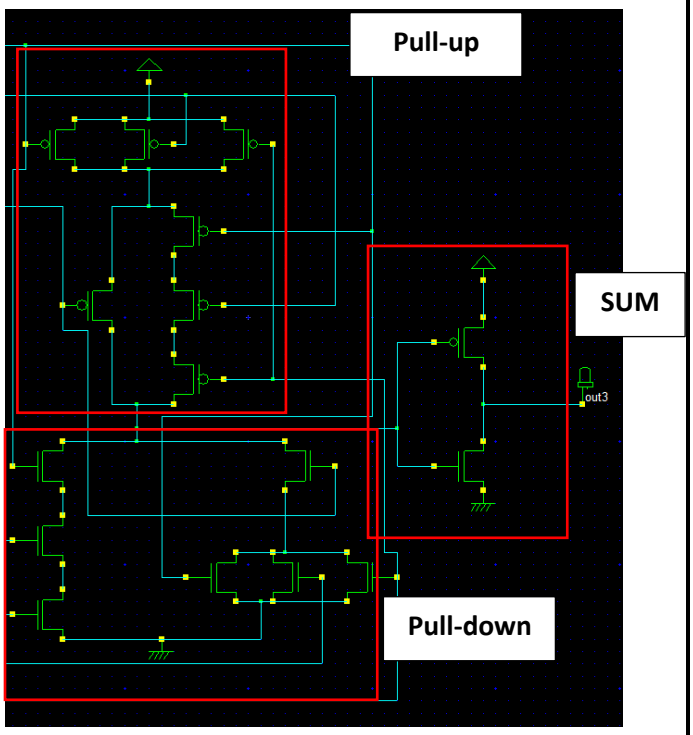

Figure 3: Sum Schematic

And if we trace the design shown in the figure above, starting from the pull-down network and from the left: Sum = (A.B. Cin) + ((~Cout) + (A+B+Cin)) and from the Pull up circuit is: Sum = (A+B+Cin) + ((~Cout) + (A.B. Cin))

Note that the Pull-down network always represents the ~OUTPUT that we aim to have, where the pull-up network is the exact logical opposite of the pull down.

Finally, by combining both circuits, and taking in consideration that the output of the left one is an output for the right one which is the inverted Cout, we have a complete design of both sum and carry which represents one bit adder using pull up and pull-down circuits which is shown in the figure below in schematic design.

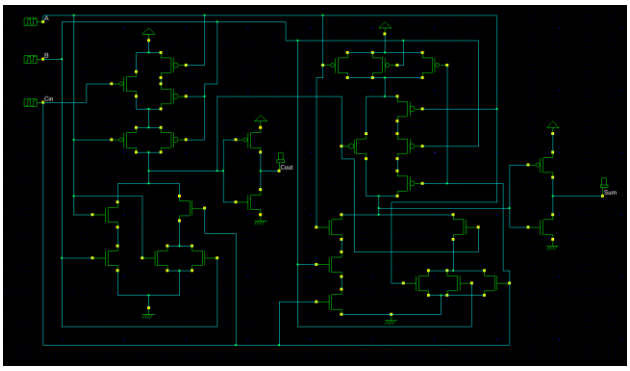

Figure 4: 1 – Bit Full Adder Schematic

We tested the circuit shown above and it was working verry well, so we moved on to the layout where we created our transistor and used it in the design to achieve the following one-bit full adder:

For carry design:

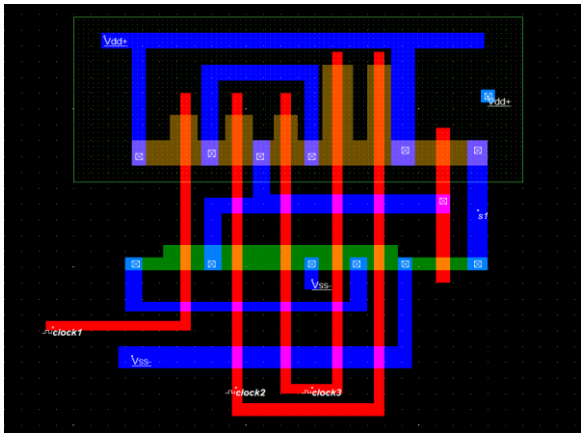

Figure 5: Carry Layout

Full design:

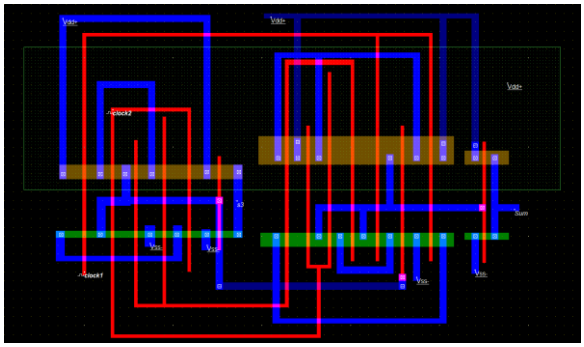

Figure 6: 1 - Bit Full Adder Layout

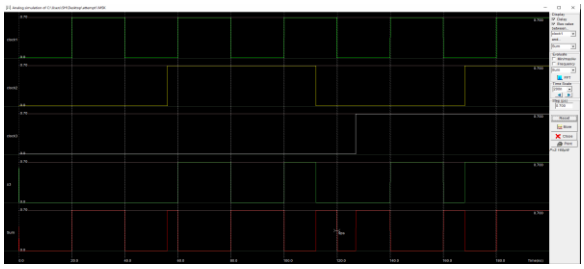

Figure 7: Simulation of 1 - Bit FA

The figure above shows the output of one-bit full adder which is smooth, fast, low power and no glitches occurred.

Finally, after everything went well and as expected, we used 2 one-bit adders and re organized the inputs and outputs to achieve the following schematic & layout which represents our aimed two-bits ripple adder:

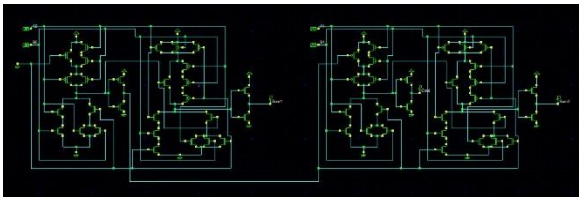

Figure 8: 2 - Bit FA Schematic

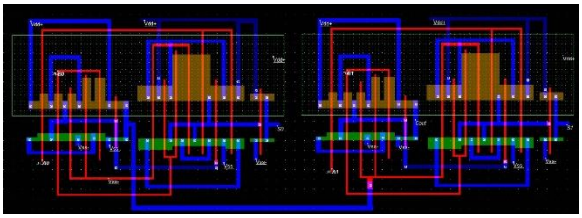

Figure 9: 2 - Bit Full Adder Layout

Furthermore, the following figure shows the simulation of the previous two-bit full adder layout, which behaved as expected.

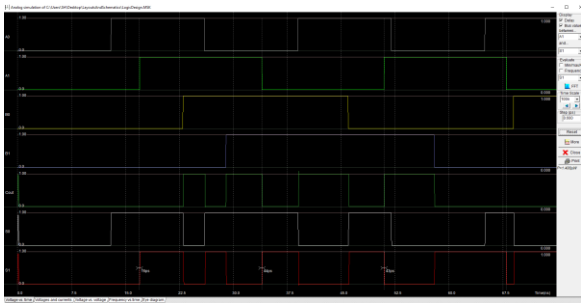

Figure 10: Simulation of 2 - Bit FA

Finally, the following figure shows the area of our design:

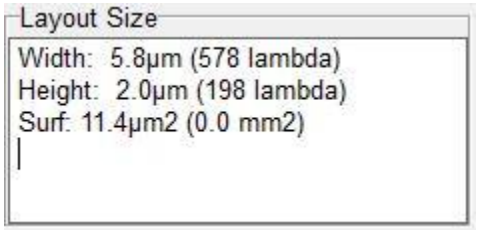

Figure 11: 2 - Bit FA Layout Size

Now, for the second design, we built a new only NAND gates full adder, which has less power than the earlier, smoother and faster. This design was implemented using 9 NAND gates. So first, lets get introduced to the NAND gate in transistor level, which consists 4 transistors with both Pull-up and Pull-down networks, 2 parallel transistors at the Pull-up network and 2 series transistors at the Pull-down.

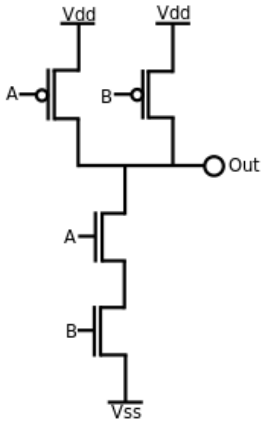

Figure 12: NAND Gate Schematic

So, we created the NAND gate using DSCH tool as the following figure shows:

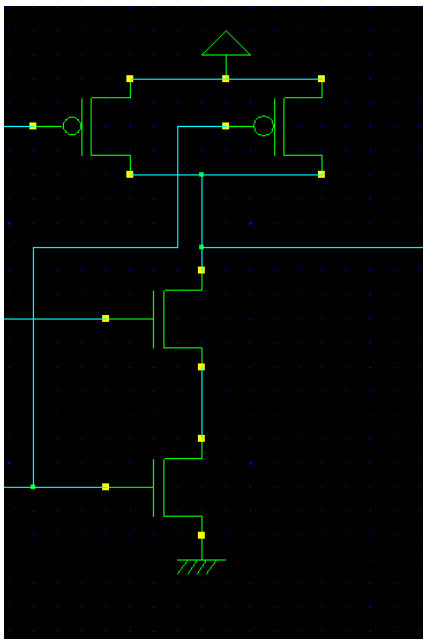

Figure 13: NAND Gate Schematic in DSCH

Then, we created the entire following design using DSCH:

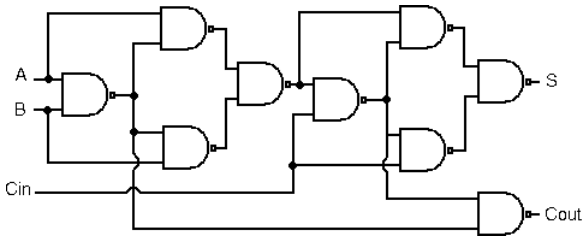

Figure 14: 1 - Bit FA Using NAND Gates

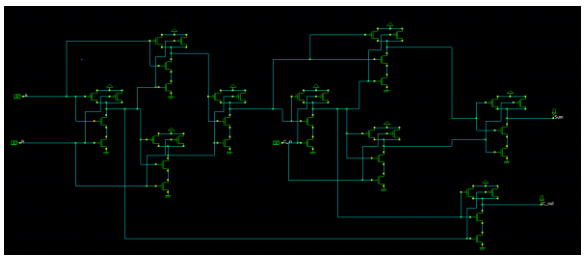

Figure 15: Schematic of 1 - Bit FA Using NAND Gates

Finally, we ran the file and observed that all cases were correct and as expected.

Now, after making sure that our design is good enough and make sure that it is better than earlier, we went to Micro Wind tool in order to build our design, so we built a NAND gate in layout level, determined its dimensions correctly, and attached all parts together to achieve a full design, then, we performed sizing on it as we learned earlier, set clocks correctly, and made sure that everything is connected well and achieves all design rules of 22n design.

The following figure shows the layout implementation of one-bit full adder that we built from CMOS level:

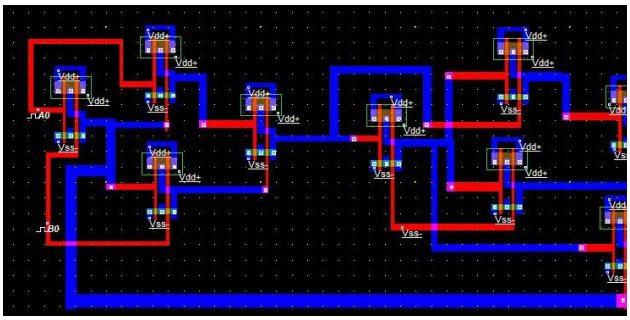

Figure 16: 1 - Bit FA Layout

Then we simulated the layout shown above, the following figure, shows the simulation of this one-bit adder:

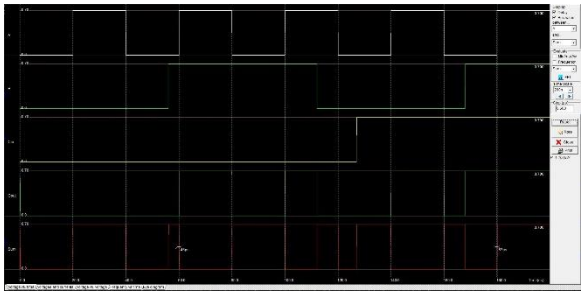

Figure 17: Simulation of 1 - Bit FA Using NAND Gates

Furthermore, we connected two one-bit adders together and reconnected inputs and outputs as it must be as the following figures show:

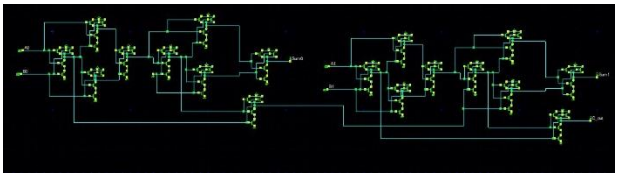

Figure 18: 2 - Bit FA Schematic Using NAND Gates

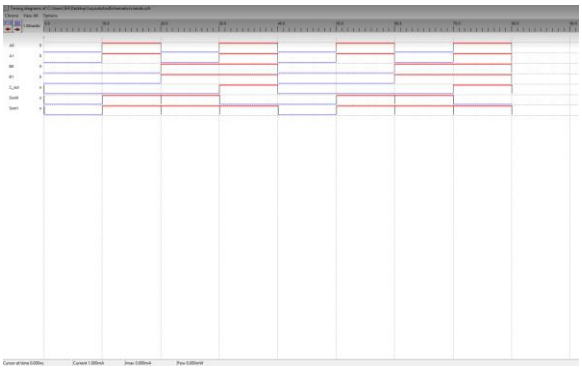

Figure 19: Schematic simulation

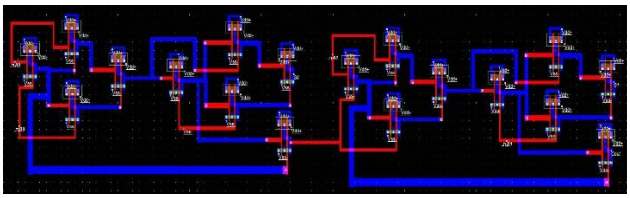

Figure 20: 2 - Bit FA Using NAND Gates Layout

Finally, we simulated the layout shown above of the two-bit binary ripple adder to get the following results:

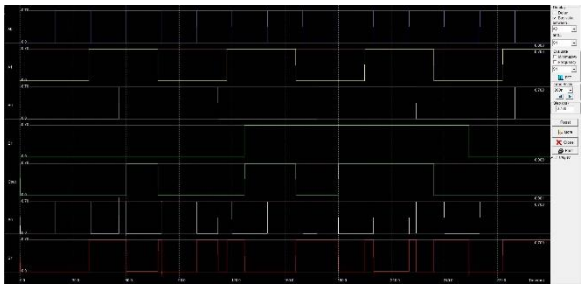

Figure 21: Simulation of 2 - Bit FA Using NAND Gates

Which were satisfying and much better than the 65n design, and better than our earlier design.

Finally, the following figure shows the area of our design:

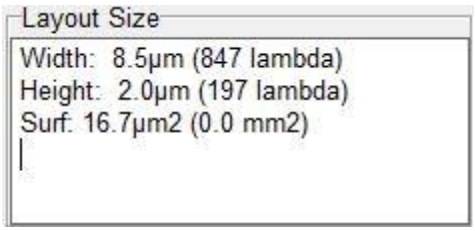

Figure 22: 2 - Bit FA Using NAND Gates Layout Size

Note that we will compare between all design after this page, so it can be easier for viewers to notice difference between the designs, and how the 22n designs are better than 65n, and how the two 22n design are different and which is better.

Discussion

Finally, to see differences between different designs, and how process affects on size, power and delay, we made the following table. But before, we must note that the following 65n design values are for only one-bit adder, and if it was built two-bits adder from it, these values will increase, which means our two-bits designs are even better than the one bit design at process 65n.

	65n	22n - LOGIC	22n - NAND
Area	**59.3µm^2**	**11.4µm^2**	**16.7µm^2**
Delay	**207ps**	**44ps**	**38ps**
Power	**25.797 µw**	**1.4µw**	**1.198µw**

We can clearly see the big difference in area when we use 22n process instead of 65n, it jumped down which is good factor for us, since it will take less size and less cost. Also, we can see how the power went down a lot, which is also a good factor since our designs are power efficient.

Conclusion:

In conclusion, we've seen how the power, area and delay were decreased using a lower process for each design and how different designs differs even whit using the same process for them just as the logic and NAND gates designs, both were done using 22 nm process, but the NAND design was better in terms of power and delay while the logical design has a better area. However, both designs were better than the 65 nm design as the full adder that is designed

using a 65 nm process had the largest area,
delay and power among them all.