2011

4-Bit Adder Design and Simulation

Amin Atwa

Ahmed Samir

Fady Soliman

THE AMERICAN UNIVERSITY IN CAIRO

Department of Electronics Engineering

ENGR 318 – VLSI Design

Spring 2011

4-Bit Adder

Project Report

Instructor: **Dr. MohabAnis**

Assistant**: Mr. AmrAbdulZahir**

Submitted by:

Amin Atwa **900071590**

Fady Reefaat **900072491**

Ahmed Samir **900072617**

Contents

Executive Summary

The project is to design a 4-bit digital adder, while taking care of performance parameters: area, speed and power consumption, the team has chosen to design according to the cost function: Area*Delay*Power.

The project is implemented in three phases: research phase, simulation phase, and evaluation/re-evaluation phase.

The adder circuit implemented as Ripple-Carry Adder (RCA), the team added improvements to overcome the disadvantages of the RCA architecture, for instance the first 1-bit adder is a Half Adder, which is faster and more power-efficient, the team was also carefully choosing the gates to match the stated cost function. Gates are implemented using different logic families, according to each gate usage and functionality in the circuit in order to achieve the desired performance.

Transistor sizes are also selected based upon simulation and optimization, to reach the needed performance according to the specified cost function.

The team was able to reach a 4-bit ripple carry adder that has delay of 1.22 ns with 0.6 uW power consumption (measured at 10 MHz), with 109 transistors. In the re-evaluation phase, the team was able to further improve this to reach 0.99 ns delay with 0.25 uW power consumption (10 MHz) with 97 transistors only.

Introduction

The topic of the course project is to design a 4-bit adder in the standard 0.25 um CMOS Technology. The main objectives of the project is to minimize the total delay of the adder (i.e. the worst case delay of the circuit), the area used to implement the adder, and its average power consumption. That in mind, the team was able to split the project into 2 phases: the research phase and the simulation phase.

In the research phase, the team had to compare different adder architectures clearly defining the advantages and disadvantages of each one in terms of area and delay to be able to choose what could be the most efficient adder architecture for the design of a 4-bit adder. Another essential task in the research phase was to decide on the gate level implementation of the circuit, compare the different logic families'implementations for each gate, and finally decide on the proper logic family implementation for each gate in light of the project objectives stated beforehand.

Once the research phase was accomplished, the team had to move on to the simulation phase. In the simulation phase, the team had to design each gate separately and optimize it to achieve the optimum delay and powerconsumption,thensimulate a 1-bit full adder, and finally simulate the whole 4-bit adder. The simulation phase concludes the project by estimating the worst case delay of the 4-bit adder design and the average power consumption of the circuit.

Assumptions

Design Criteria

The group members are not designing this adder for a very specific application that dictates certain design criteria or puts different weights on timedelay or circuit area, that is why group members assumed it is better to implement a design that balances between time delay, power consumption and area used in the implementation of the 4-bit adder without giving different weights to any of the design criteria. Therefore, the design criteria will be [A*(T^2)*(P^2)] (T: time delay, A: area, P: power) not T^2*A or T*A^2.

Half Adder

As the project description is to design a 4 bit adder, group members assumed they have 8 inputs which are the 2 sets of 4 bits to be added, so in the design it is more efficient in terms of delay, area, and power to design a half bit adder for the first bit adder as there is no carry-in bit for the first adder. This will show great performance improvement because the C_{out} bit will be result of 2 gate delays instead of 3.

Research Phase

"Research is formalized curiosity." In this section, the team presents the results of the research phase which was an integral part of the project. Research phase was divided into 3 sub phases: adder architectures comparison, gate level implementation of the chosen architecture, and logic families' comparison for gates of the chosen gate level implementation. The results the team came up with from each sub phase is of paramount importance for the 4-bit adder design.

Adder Architectures Comparison

In this section, a short description of the adder architecture and the exact time delay (T) and area (A) complexity based on unit gate model is presented. In the unit gate model each gate has a gate-count of one and a gate-delay of one excluding XOR and XNOR gates having gate counts and gate delays of two, while the gates with more than 2 inputs, the gate-counts and gate-delays can be computed in terms of the ones given for the gates with two inputs; also, inverters and buffers are ignored.

Ripple Carry Adder (RCA)is the simplest carry-propagate adder.Its time delay and area complexity are as followsforan n-bit RCA adder:

$T = 2n$
$A = 7n + 2$

Carry Skip Adder (CSKA) is the concatenation scheme with a carry-skip scheme. Its time delay and area complexity are as follows for an n-bit CSKA adder:

$K= (n - 1)^{½}$
$T = 4k$
$A = 8n + 6k - 6$

Carry Select Adder (CSLA) is the concatenation scheme with a selection scheme. Its time delay and area complexity are as follows for an n-bit CSLA adder:

$K = 1/2*(8n - 7)^{½} - ½$
$T = 2k + 2$
$A = 14n - 5k - 5$

Carry Look Ahead Adder (CLA) uses direct parallel-prefix scheme for carry computation. Its time delay and area complexity are as follows for an n-bit CLA adder:

$T = 2 \log(n) + 4$
$A = 3/2*n*\log(n) + 4n + 5$

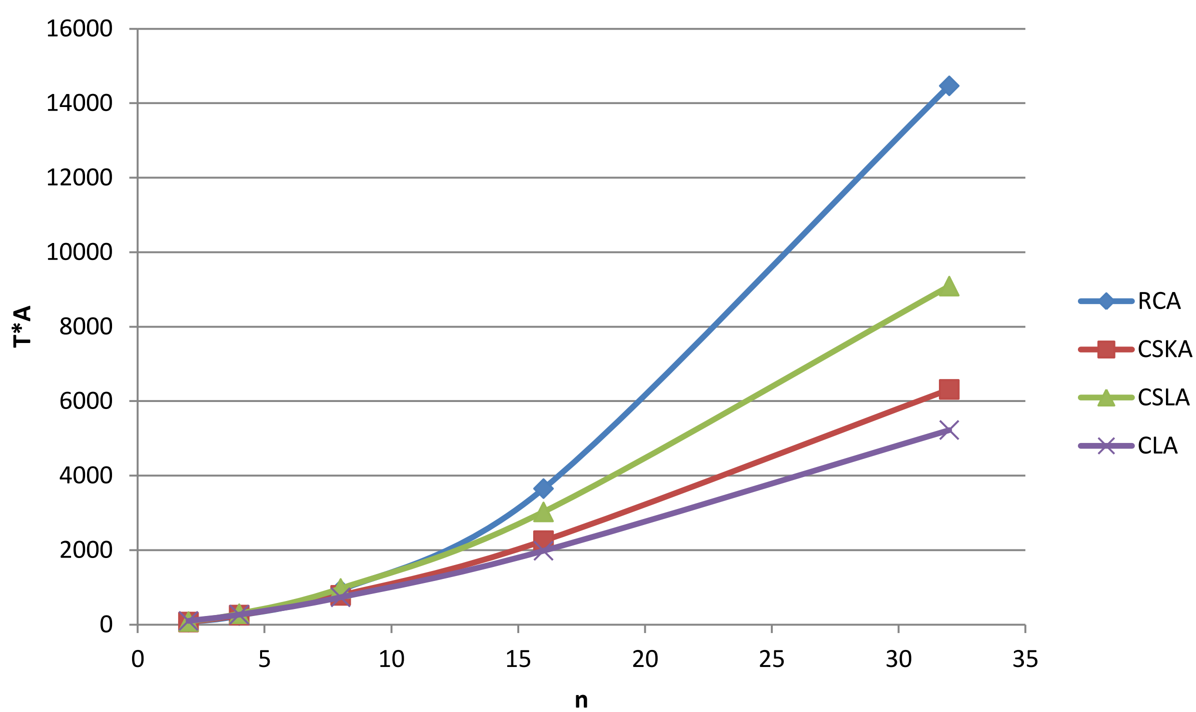
Results of the comparison can be clearly summarized in the following tables.

	K	T	A
RCA		2*n	7*n+2
CSKA	(n – 1)^0.5	4*k	8*n+6*k-6
CSLA	0.5*(8*n – 7)^0.5-0.5	2*k+2	14n-5k-5
CLA		2*log(n)+4	1.5*n*log(n)+4*n+5

Equations for Time delay (T) and Area (A) complexity of each architecture.

n	**RCA**			**CSKA**				**CSLA**				**CLA**		
	T	A	**T*A**	K	T	A	**T*A**	K	T	A	**T*A**	T	A	**T*A**
2	4	16	64	1	4	16	64	1.621	5.243	14.89	78.08	6	16	96
4	**8**	**30**	**240**	**1.732**	**6.928**	**36.39**	**252.1**	**3.036**	**8.071**	**35.82**	**289.1**	**8**	**33**	**264**
8	16	58	928	2.646	10.58	73.87	781.8	4.839	11.68	82.81	966.9	10	73	730
16	32	114	3648	3.873	15.49	145.2	2250	7.278	16.56	182.6	3023	12	165	1980
32	64	226	14464	5.568	22.27	283.4	6312	10.66	23.32	389.7	9086	14	373	5222

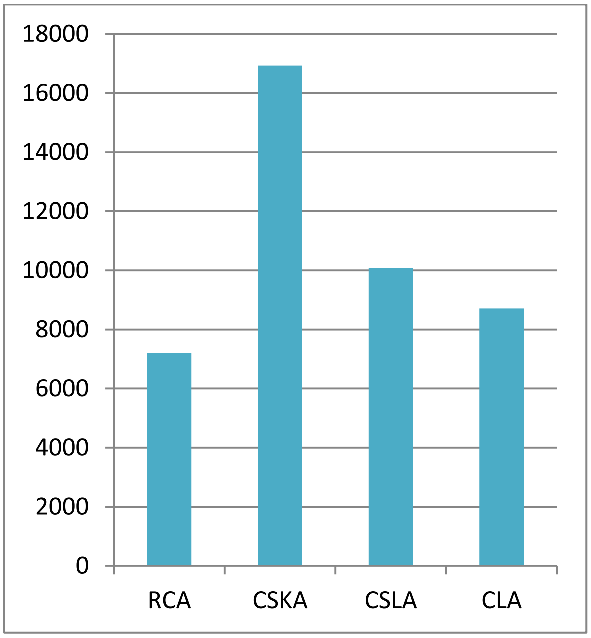
Different architectures delay (T) and area (A) for different number of bits (n).



Graphic representation of the results in previous table

	K	T	A	T*A	(T^2)*A	T*(A^2)
RCA		8	30	**240**	1920	**7200**
CSKA	2	8	46	368	2944	16928
CSLA	2	6	41	246	**1476**	10086
CLA		8	33	264	2112	8712

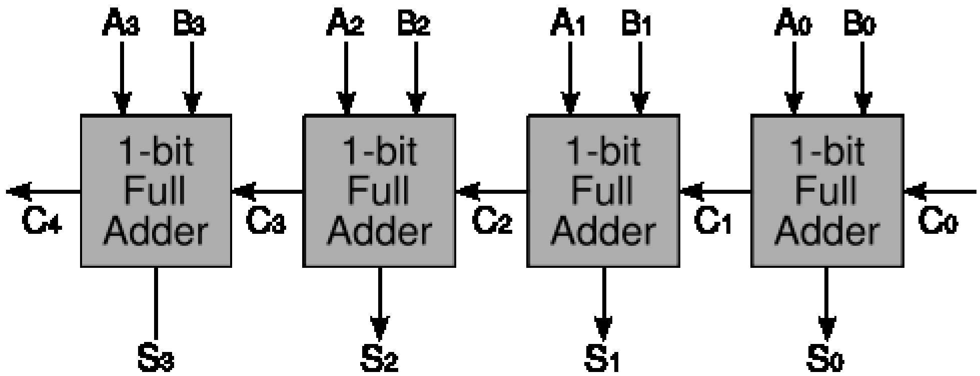
Equations of Time delay (T) and Area (A) complexity for each architecture when n=4 and rating their performance for different design criteria (T*A, T^2*A and T*A^2).



T*A for different adder architectures (4-bit)

A clear conclusion is that for small n-bit adders and design criteria balancing between area and time delay or giving more weight to area, the ripple carry is a better architecture, while for higher n-bit adders carry skip, carry select or carry look ahead might be a better choice for the designer.

Since in this project, the team is designing a 4-bit adder and assuming same weights for area and delay, the team concluded that the **ripple carry** could be the most efficient implementation for the 4-bit adder design.



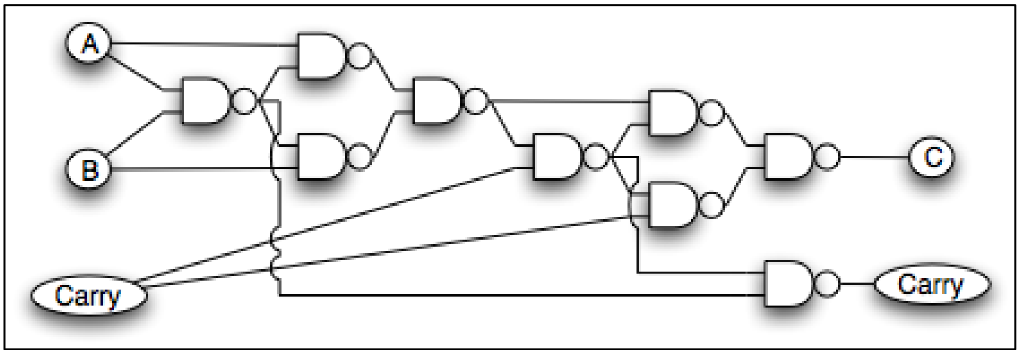
Schematic 1.1: Ripple Carry adder schematic adders level.

The ripple carry is probably the simplest architecture for an adder. In this architecture the delay simply propagates from one Full-adder to the next one, therefore the implementation of the full adder is all that matters in its design.

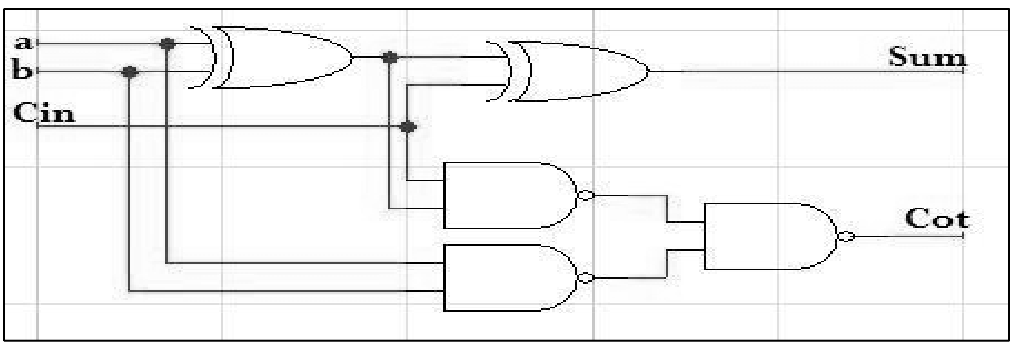
Gate Level Implementation of the Full Adder

In this section, a description of the gate level implementation of the 4-bit ripple carry adder is presented. After the group agreed on implementing ripple carry adder, it was crucial to research what available gate level implementation are there for the full adder, and mainly 3 implementations were compared.

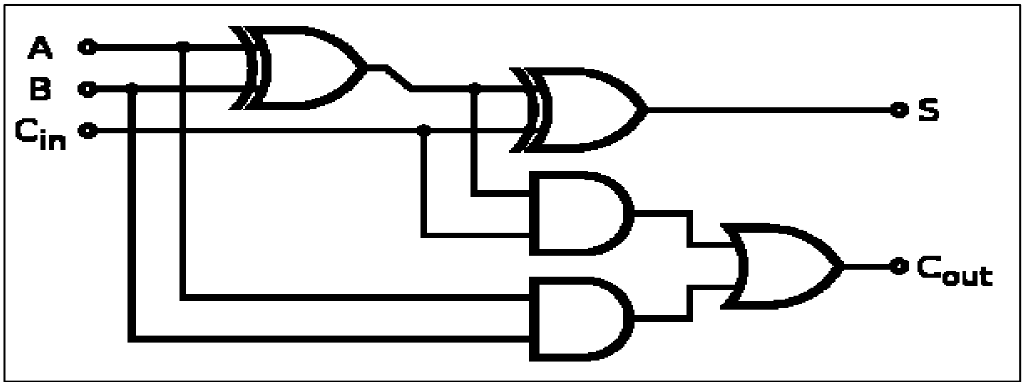
Implementation 1 uses only NAND gates to implement the logic of the full adder.
Implementation 2 uses 2 XOR gates and 3 NAND to implement the logic.
Implementation 3 uses 2 XOR, 2 AND and 1 OR to implement the logic.



Schematic 1.2.1: Gate level implementation 1 of the full adder

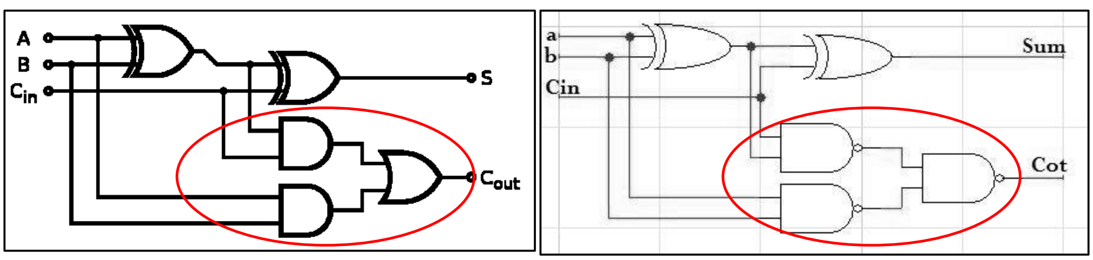


Schematic 1.2.2: Gate level implementation 2 of the full adder



Schematic 1.2.3: Gate level implementation 3 of the full adder

Comparing these different implementations in terms of area and delay, it was clear that implementation 1 will be too slow and takes too much area, while the other 2 implementations do not differ too much. However, as NAND gates can be implemented using CMOS logic family without the need for an inverter at the output, while AND and OR cannot, the team decided to choose **implementation 2** to have the option of using CMOS logic whenever it is needed without the need to use inverters.



Schematic 1.2.4: Comparing implementation 2 and 3 of the full adder

Therefore, the final implementation of the full adder in this project is as follows:

Sum = A XOR B XOR C
Carry out = (A NAND B) NAND [(A XOR B) NAND Cin]

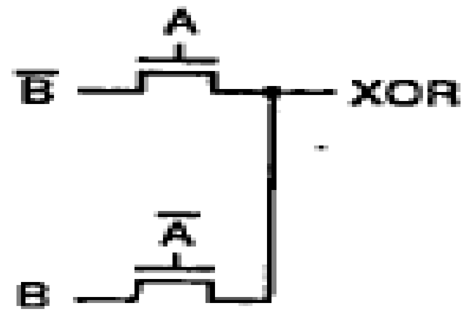
The next step will be to decide the logic family implementation for each gate.

Logic Families Comparison for XOR and NAND of full adder

In this section, a description for the different logic families to implement XOR and NAND gates of the full adder gate level implementation that was agreed upon in the previous section.

XOR gate has mainly 3 implementations:

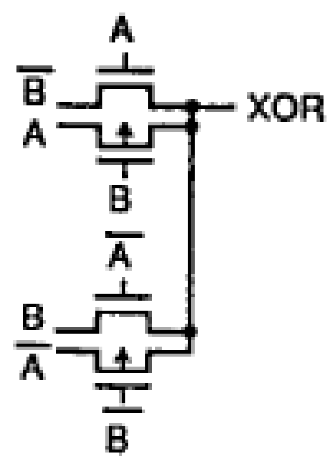
Complementary Pass-transistor Logic XOR (CPL)
The main advantage of the CPL logic family is that it uses few numbers of transistors so in terms of area it has an edge over other implementations. However, CPL has a reduced swing so it cannot be used as the output of any adder since according to project description; reduced swing at the output is unacceptable.



Schematic: XOR CPL implementation

Double Pass-transistor LogicXOR(DPL)

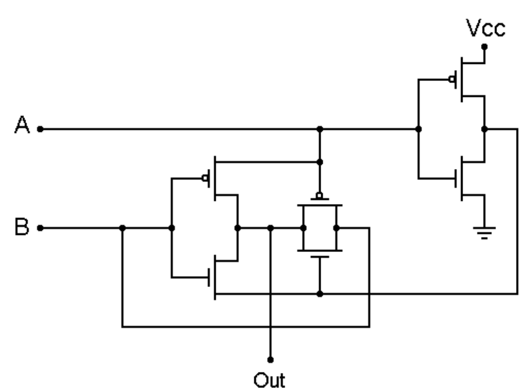
The main advantages of the DPL logic family is that its delay is low since always 2 transistors are ON in any charging or discharging input combinations.Also it has an advantage over the CPL that it has a full swing at the output and uses a reasonable number of gates.



Schematic: XOR DPL implementation

Transmission gateXOR(TG)

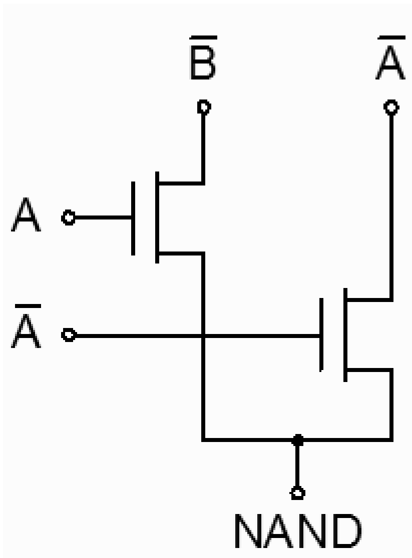
Transmission gate is another implementation for the XOR function. However, its worst case delay is probably higher that the DPL since when A is HIGH only 1 transistor is charging or discharging the output compared to two in the DPL implementation. So in terms of delay DPL has an edge over transmission gate. However, it uses less number of transistors than DPL.



Schematic: XOR Transmission gate implementation

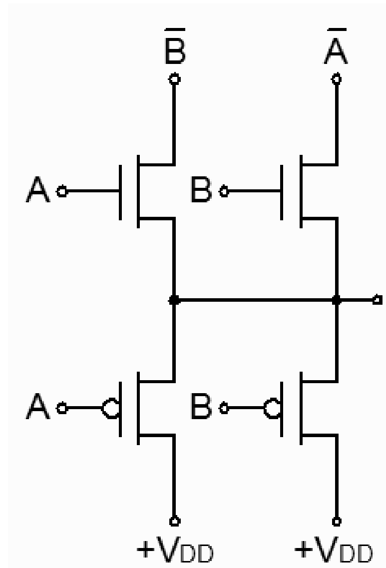
NAND gate has mainly 3 implementations:

CPL NAND



Schematic: NANDCPL implementation

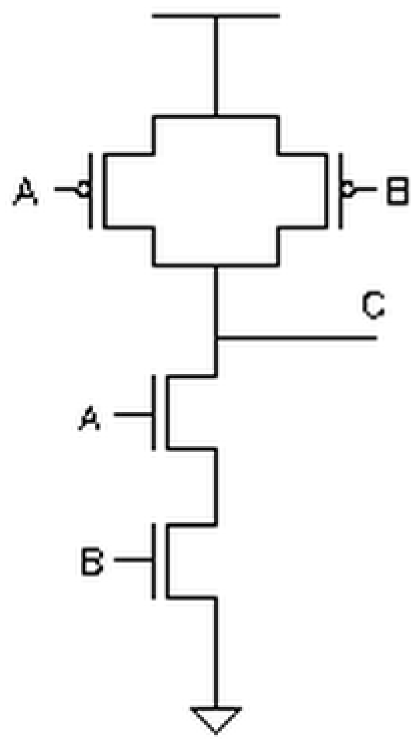
NAND DPL



Schematic: NANDDPL implementation

CMOS NAND

CMOS logic family has an advantage over DPL that it uses less number of transistors (no need for inverters), and has an edge over CPL that its output is full swing.

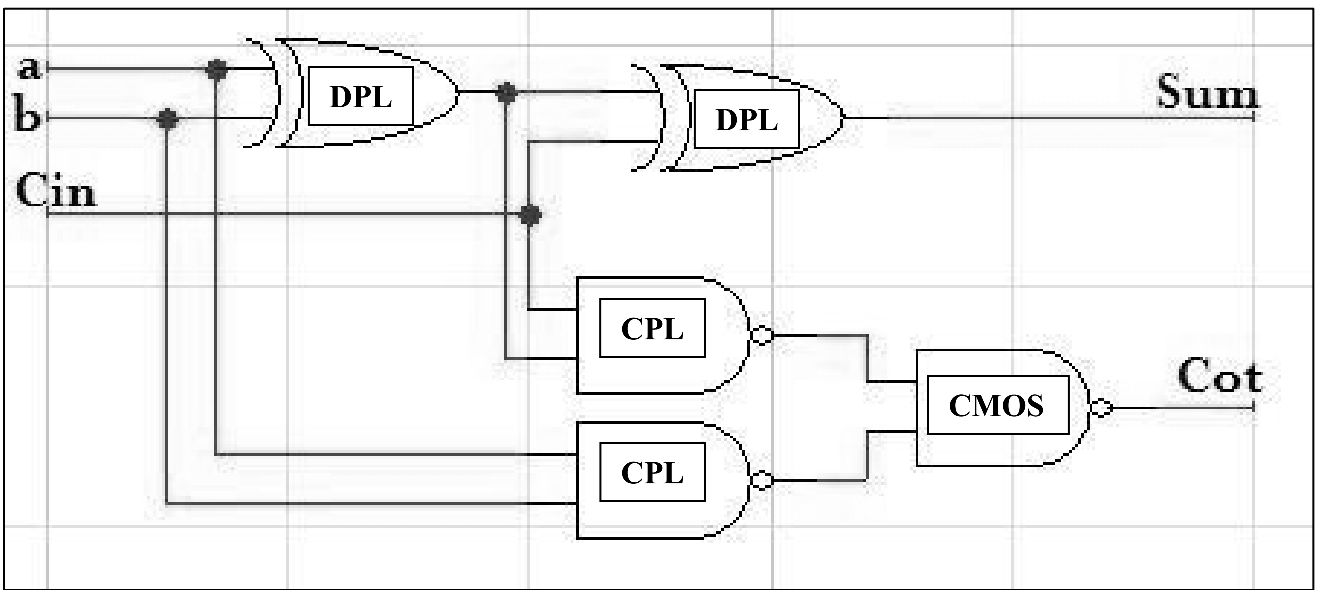


Schematic: NANDCMOS implementation

After comparing the different logic families in the different logic families in terms of swing, delay, and area, the team made some educated assumptions. First of all for the XOR, the CPL was excluded since the project description a full swing output. So, comparing DPL and transmission gate, the group assumed it is more efficient to use **DPLXOR**as XOR gatesmust be very fast since it is on the track of propagation of the delay.

As for the NAND gate, it is also important for it to be fast but still we need the output to be full swing, so the team assumed it is more efficient to implement **2 CPL NAND** gates which outputs are input to a **CMOS NAND** to ensure full swing at the output (thanks to its Pull-Up Network). Another reason for choosing CMOS NAND to calculate the carry-out is that it uses only 4 transistors compared to DPL that needs 8 because it requires inverters at the inputs.

The following schematic shows the logic family of each gate in the project.

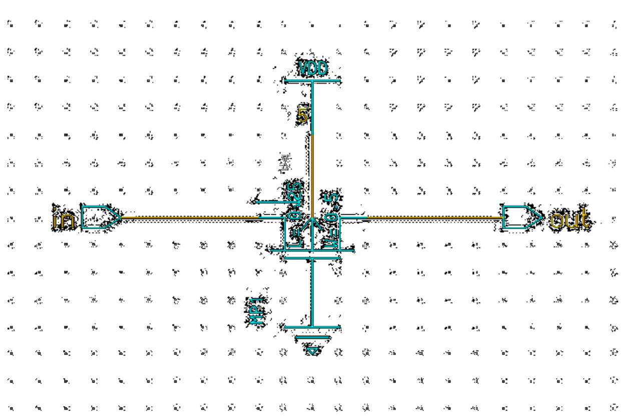


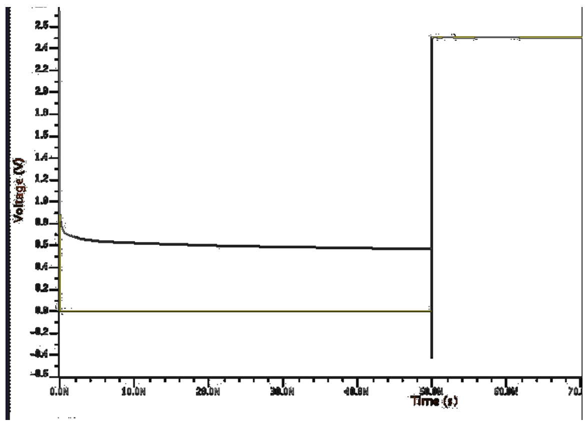
Schematic: Logic Family for each gate.

This schematic also ensures there will be no more than 2 transistors in series as CPL requires inverters so input is buffered, and CMOS NAND acts as a buffer since its inputs are to the gate of the transistors. We took the advantage of the CMOS NAND following this reduced swing CPL NAND to return output to full swing. But we had to take care of the Short Circuit Power dissipation!

However a threat was that CPL could cause static power due to its reduced swing and since itis driving a CMOS NAND gate. So, the group needed to prove during simulation that this reduced swing will not cause a static power dissipation phenomenon, which is due to the situation that the reduced swing may lead to the PMOS devices in the CMOS NAND to be ON, while NMOS devices are ON as well, so a short circuit current can find its path from Vdd to ground causing power dissipation.

Testing actually showed that that the value of the output swing of the CPL ranges from 0 to 2.1 V (ieVthn is around 0.4V), and then Vthp was found to be around 0.6V, therefore no static power will take place which ensures that there will be no short circuit power dissipation as the PMOS device will not be on by the reduced input swing output (Vdd-Vthn) of the CPL.

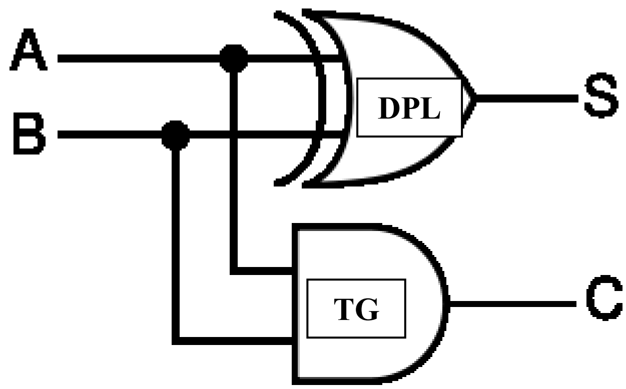




Schematic:PMOS device tested to measure Vth_p output signal low=Vth_p

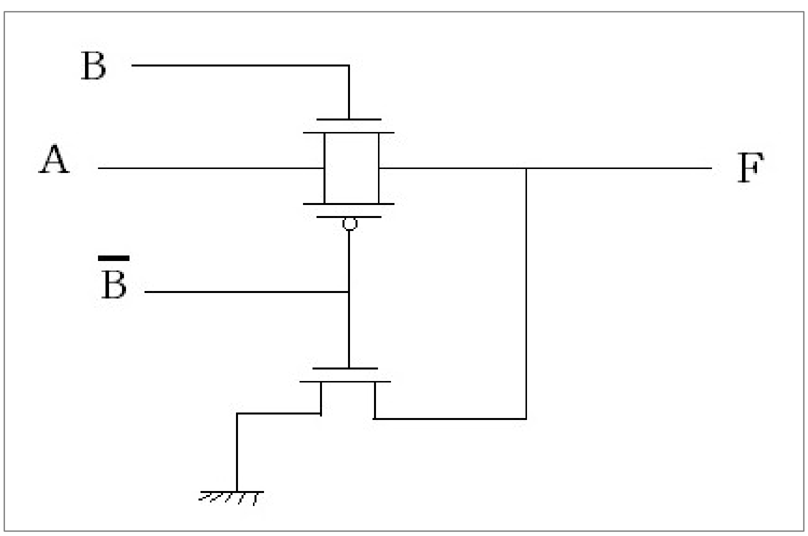
Gates Implementation of Half Adder

As previously stated, we assume inputs are 2 sets of 4 bits, so it is more efficient to implement a half adder for the first bit as there is no input carry.

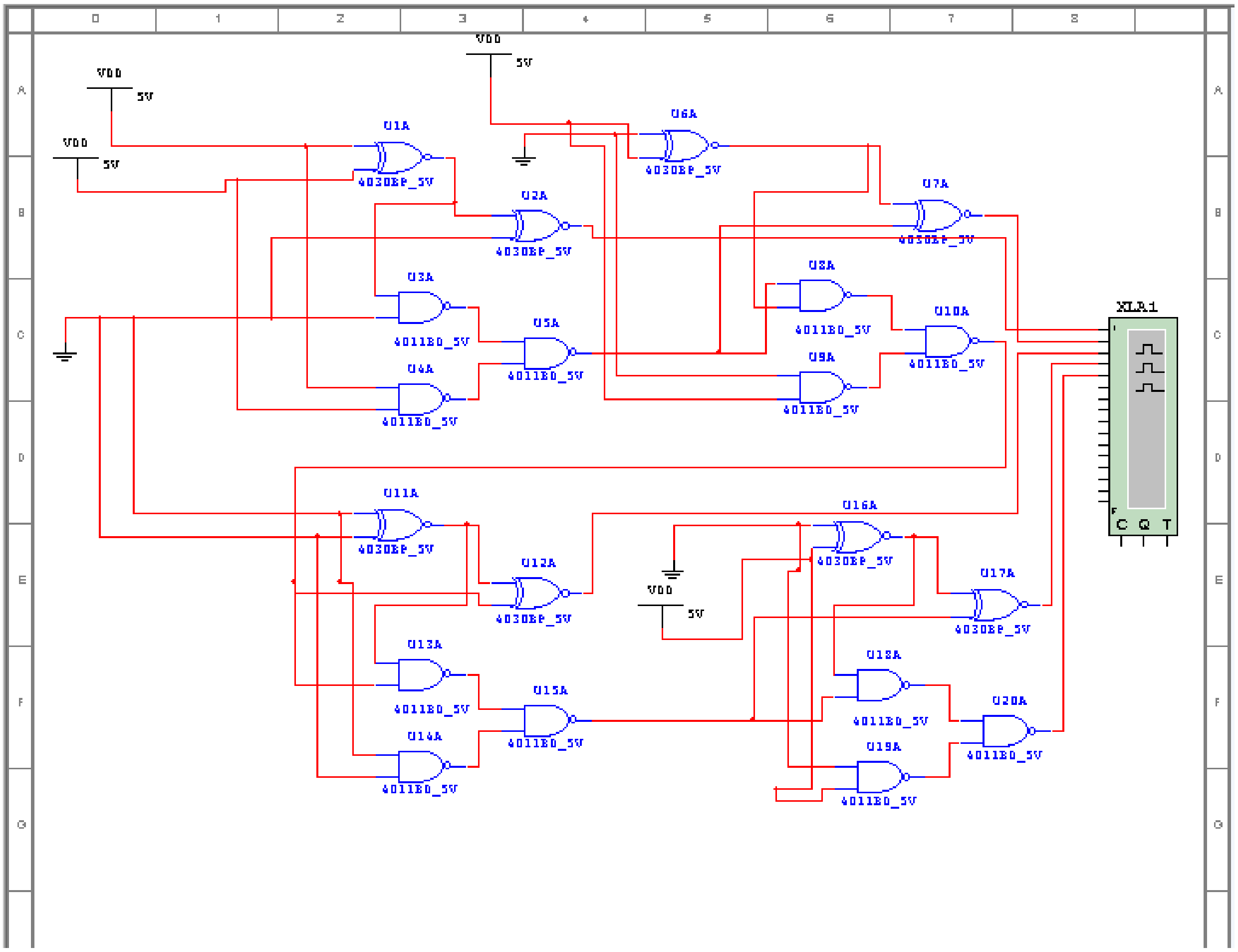


Schematic of Half Adder

The half adder consists of a XOR and an AND gate. So based on previous analysis, the team agreed to use **DPL family for XOR**, and also use **transmission gate(TG) for the AND** gate since the AND is on the carry propagation path and transmission is probably the fastest logic implementation for the AND gate and uses only 3 transistors and 1 inverter, and outputs a full swing.



SchematicTG-AND implementation



Multisim Scheme for the circuit before substituting the first full adder with a half adder

Simulation Phase

“Simplicity is the ultimate sophistication”. In this section, the team presents results of the simulation for the 4-bit adder. Since optimization is a very complex task as delay, area, and power are all affected whenever size of transistors are changed, the team decided to simply design each gate separately first to ensure the logic is correct, then optimize it to find the size that gives the lowest worst case delay and lowest power consumption for each gate. Then, concatenate gates together to form the 1-bit full adder and 1-bit half adder, before actually implementing the whole 4-bit adder and estimating the worst case delay and the average power consumption of the adder.

As the project objective is to balance area, power and delay, and since the group has chosen Ripple Carry Adder Architecture that has an edge over other architectures that it requires less area, the group decided that during optimization they will give higher weight to time delay and power over area to ensure this balance(because it is know that RCA is disadvantageous when it comes to speed).

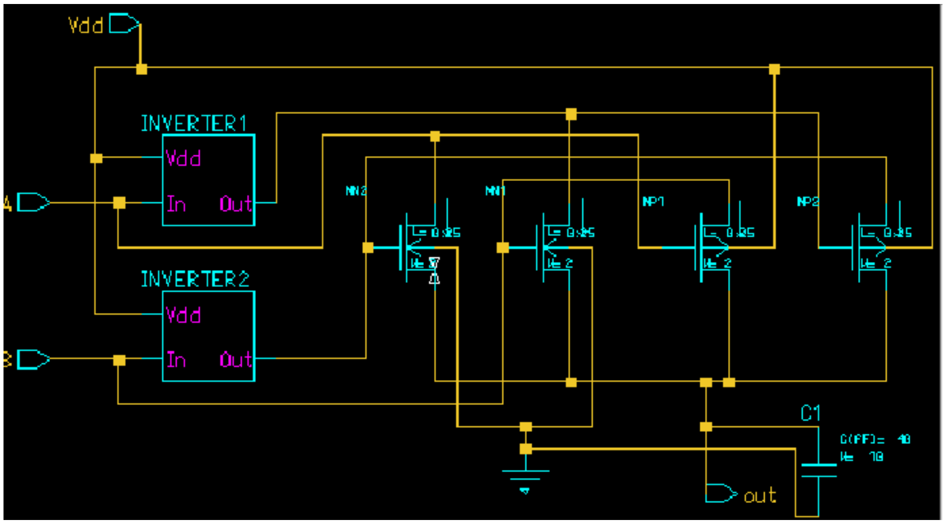
Optimization of the logic gates

Optimization is finding optimum values of transistor W/L that would achieve the desired performance balance between area, power and delay.

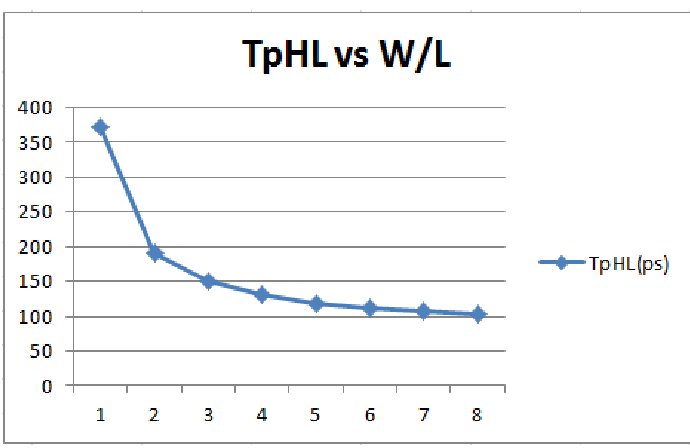
Note: during optimization and simulation of individual gates, a **40fF** capacitor was put at the output terminal (C_L) and frequency pulse used was **10 MHz**. Also, higher weights was given to power and delay as the team’s decision to use ripple carry adder gives the adder an edge in terms of area, so it can give away part of this advantage to ensure low delay and power consumption. However, in the overall balance is the ultimate goal of the design.

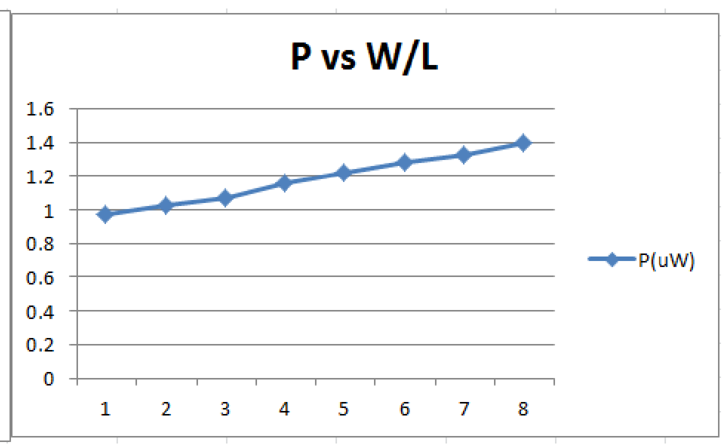
DPL XOR gate

The following graphs show the design and the optimization of the DPL XOR gate. In DPL, there is no worst delay as always an NMOS and a PMOS are ON.



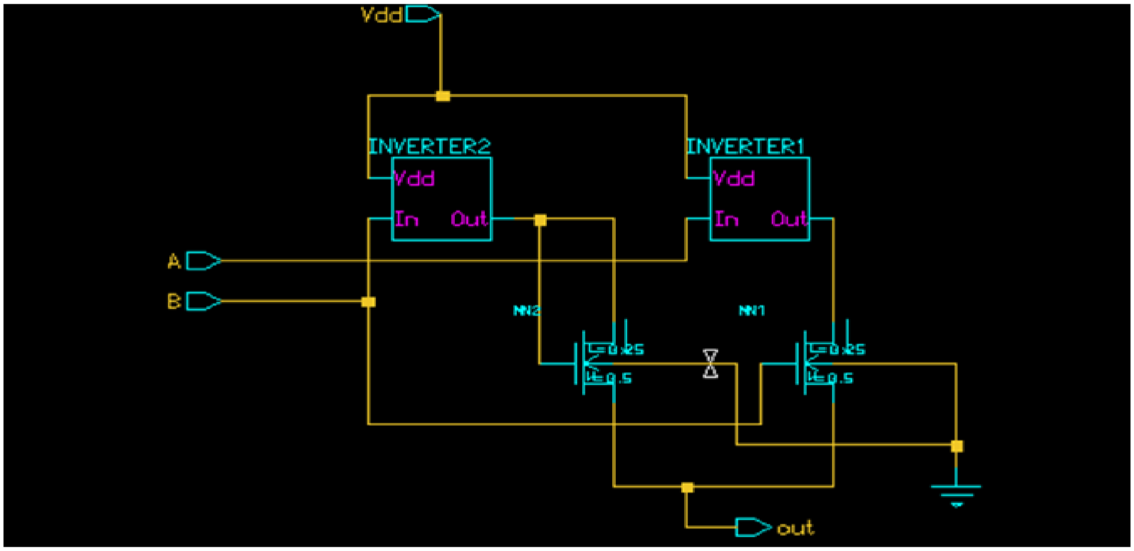
DPL XOR				
	A(W/L)n	TpHL(ps)	P(uW)	A*(T^2)*(P^2)
1	0.25/0.25	373	0.975	132259.5056
2	**0.5/0.25**	**190**	**1.03**	**76596.98**
3	0.75/0.25	150	1.075	78004.6875
4	1/0.25	132	1.16	93782.9376
5	1.25/0.25	119	1.22	105386.162
6	1.5/0.25	112	1.28	123312.5376
7	1.75/0.25	107	1.33	141764.9527
8	2/0.25	104	1.4	169594.88



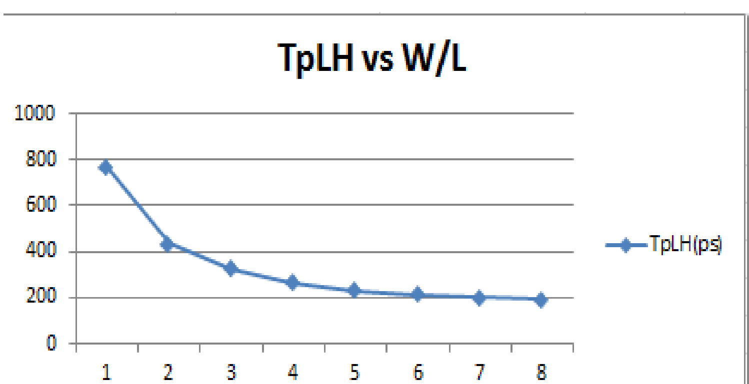


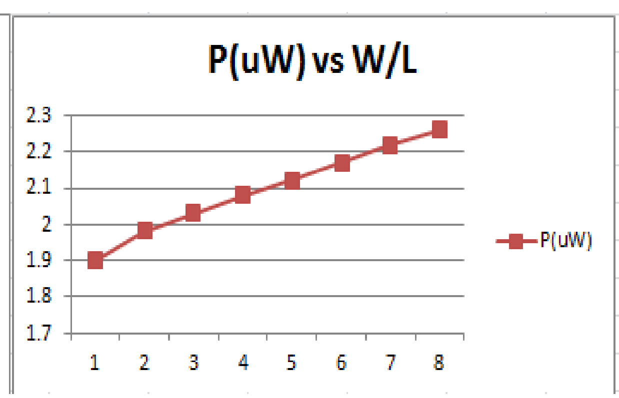
CPL NAND Gate

The CPL NAND gate worst case delay is TpLH since it has a reduced swing from 0 to Vdd-Vth. The following table and graphs present the design and results of the optimization.



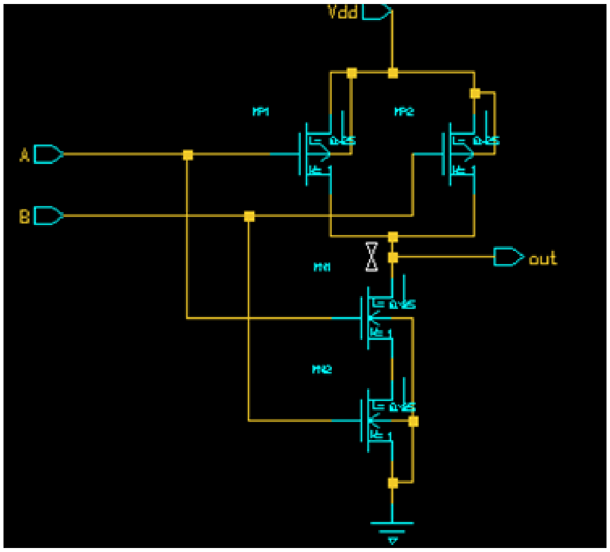
CPL NAND				
	A(W/L)n	TpLH(ps)	P(uW)	A*(T^2)*(P^2)
1	0.25/0.25	771	1.9	8583728.04
2	0.5/0.25	434	1.98	5907446.899
3	0.75/0.25	327	2.03	5287724.593
4	1/0.25	268	2.08	4971829.658
5	**1.25/0.25**	**231**	**2.12**	**4796513.568**
6	1.5/0.25	214	2.17	5175570.826
7	1.75/0.25	202	2.22	5630756.141
8	2/0.25	192	2.26	6025170.125



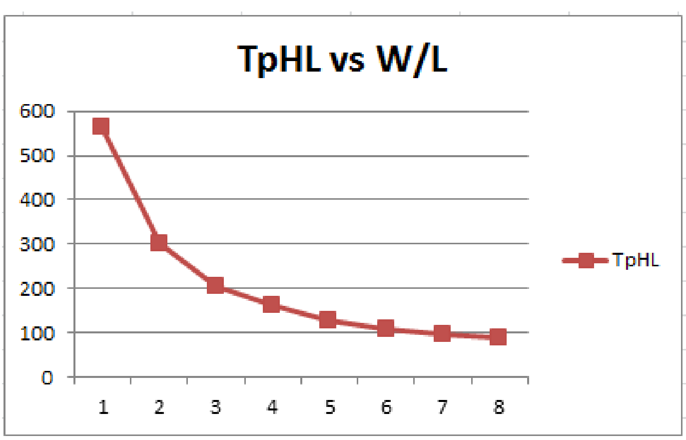


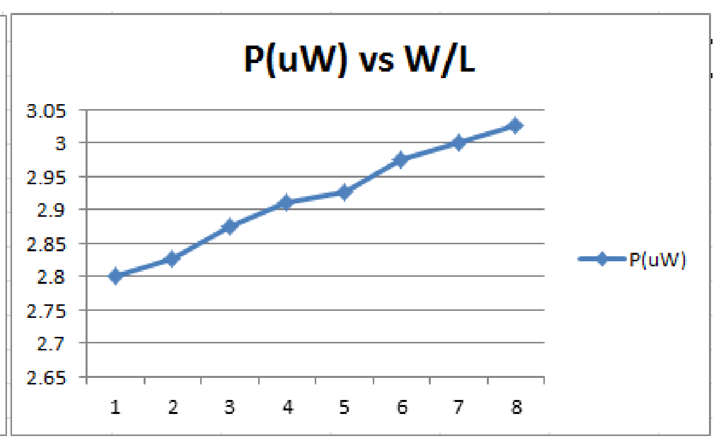
CMOS NAND Gate

The worst case delay of the CMOS is either when only one input is low or both inputs are high. . The following table and graphs present the design and results of the optimization.



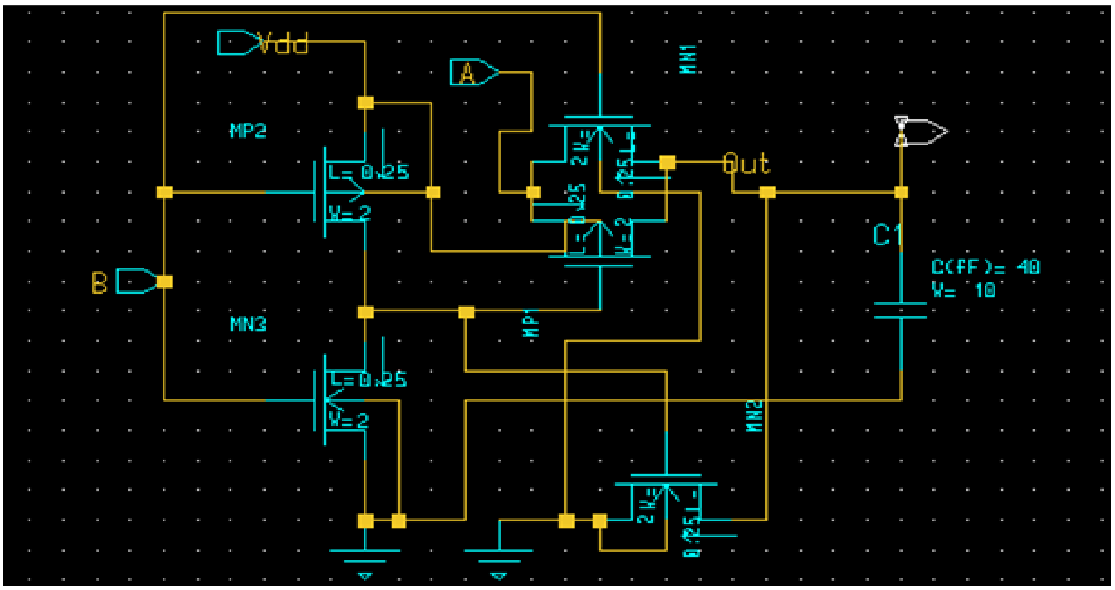
CMOS NAND				
	A(W/L)n	TpHL(ps)	P(uW)	A*(T^2)*(P^2)
1	0.25/0.25	565	2.8	2502724
2	0.5/0.25	300	2.825	1436512.5
3	0.75/0.25	205	2.875	1042088.672
4	1/0.25	162	2.91	888947.2656
5	1.25/0.25	128	2.925	700876.8
6	1.5/0.25	110	2.975	642555.375
7	1.75/0.25	97	3	592767
8	**2/0.25**	**88**	**3.025**	**566899.52**



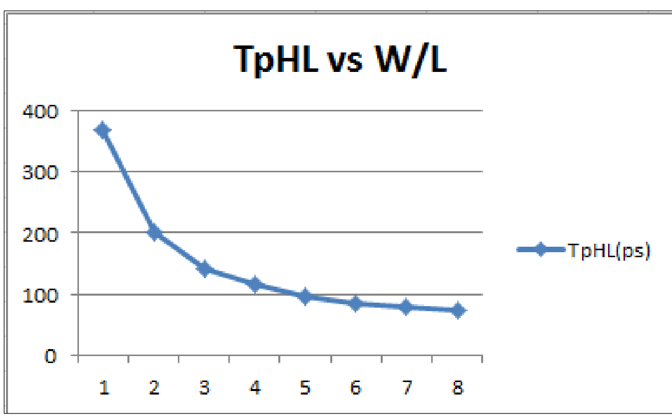


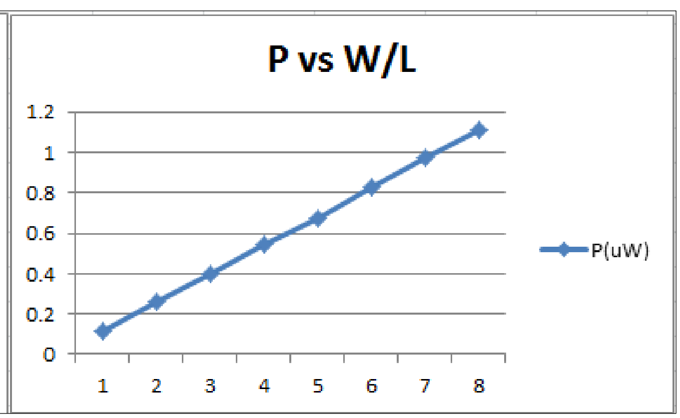
TG AND

The following table and graphs present the design and results of the optimization.



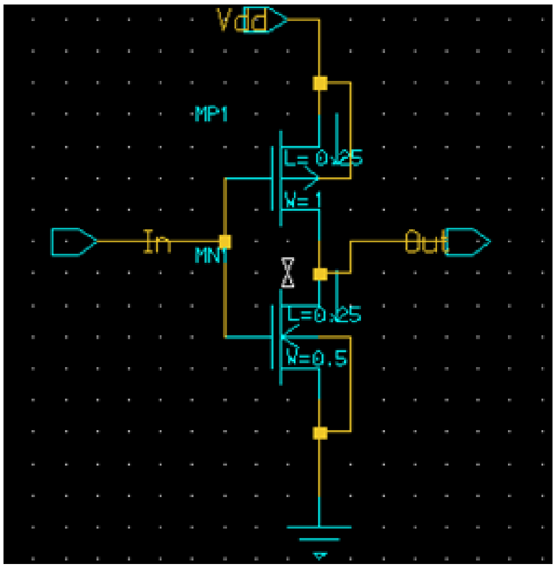
TG AND				
	A(W/L)n	TpHL(ps)	P(uW)	A*(T^2)*(P^2)
1	**0.25/0.25**	**368**	**0.117**	**1853.819136**
2	0.5/0.25	201	0.257	5336.891298
3	0.75/0.25	143	0.4	9815.52
4	1/0.25	116	0.54	15695.0784
5	1.25/0.25	97	0.675	21434.87813
6	1.5/0.25	86	0.825	30203.415
7	1.75/0.25	79	0.97	41105.0983
8	2/0.25	73	1.11	52526.8872



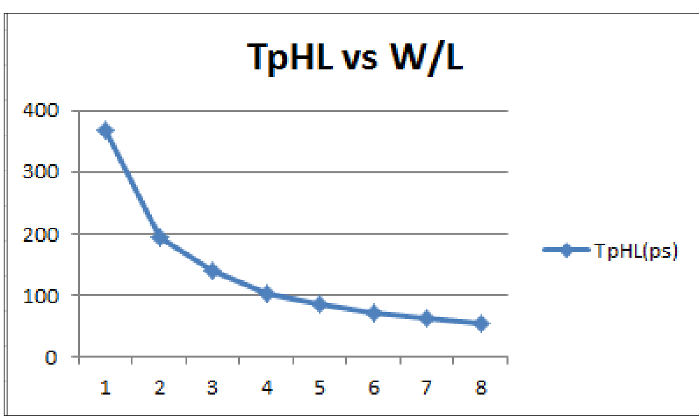


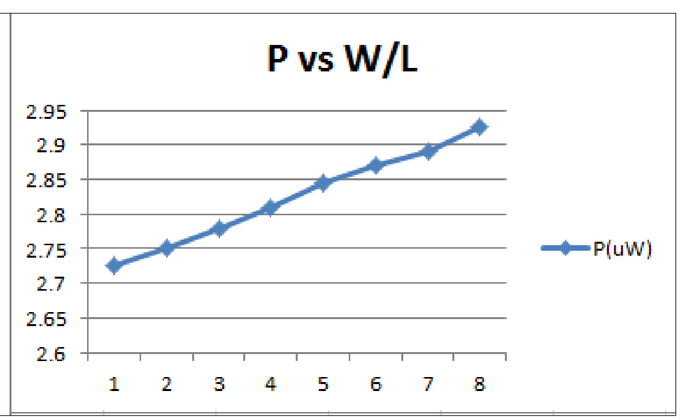
Inverter Gate

Note:(PUN W/L)$_p$=2 (PDN W/L)$_n$to ensureV_m sets to $V_{dd}/2$. The following table and graphs present the design and results of the optimization.



Inverter				
	A(W/L)n	TpHL(ps)	P(uW)	A*(T^2)*(P^2)
1	0.25/0.25	367	2.725	1000150.006
2	0.5/0.25	193	2.75	563391.125
3	0.75/0.25	140	2.78	454429.92
4	1/0.25	103	2.81	335078.8996
5	1.25/0.25	85	2.845	292396.6531
6	1.5/0.25	72	2.87	256200.5376
7	1.75/0.25	62	2.89	224738.3068
8	**2/0.25**	**55**	**2.925**	**207046.125**





Testing Circuit Logic Output

To prove that the adder is working and producing correct logic, we did some waveform tests:

We put these input combinations, and monitor the output form:

0101+ xy01where x is a pulse from 0 to 1

For instance when x=0 & y=1, output should be **10**10

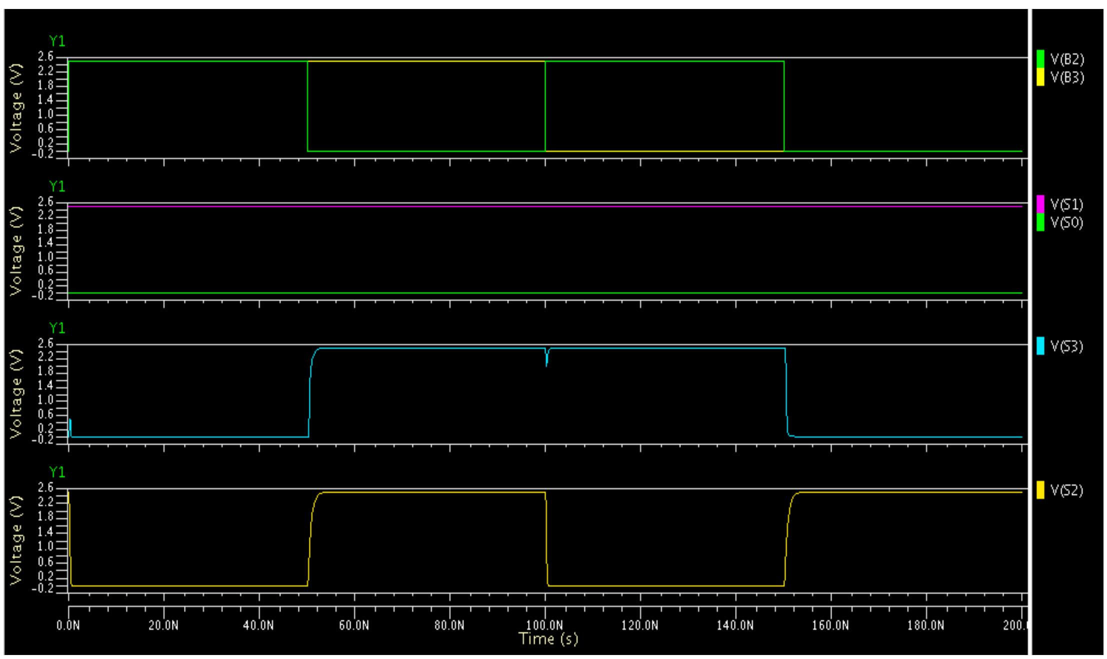
When x=0 & y=0, output should be **01**10

When x=1 & y=0, output should be **11**10

And the output waveform figure below proves that the adder produces the expected output

Note: x resembles B3, y resembles B2, MSB for sum=S3, then S2

The following graph shows the output of the adder.



In the 256 outputs logic diagram of the 4-bit adder, some spices appear due to errors in synchronization between the inputs of some gates. To get rid of that a simple solution can be to add buffers to the faster signal; however these spices do not affect the logic of the design.

Evaluation Phase

This section presents the methodology used to test and measure performance parameters.

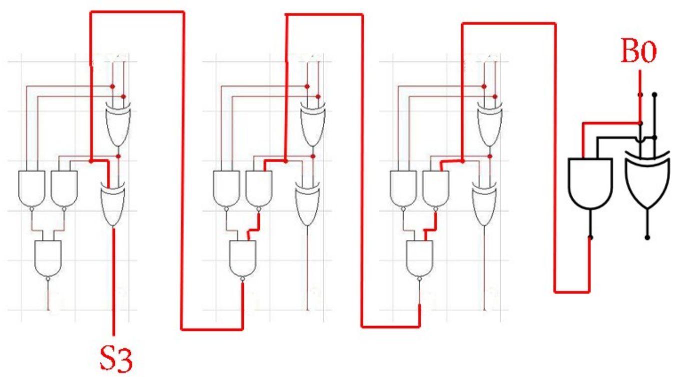
Measuring worst case delay

Based on critical path analysis, we applied an input such that the LSB of one of the inputs triggers the MSB of the sum (last bit in the sum), so that the signal can ripple through the critical path of the Ripple-Carry Adder, going from C_{out} of a 1-bit adder to the C_{in} of the other.

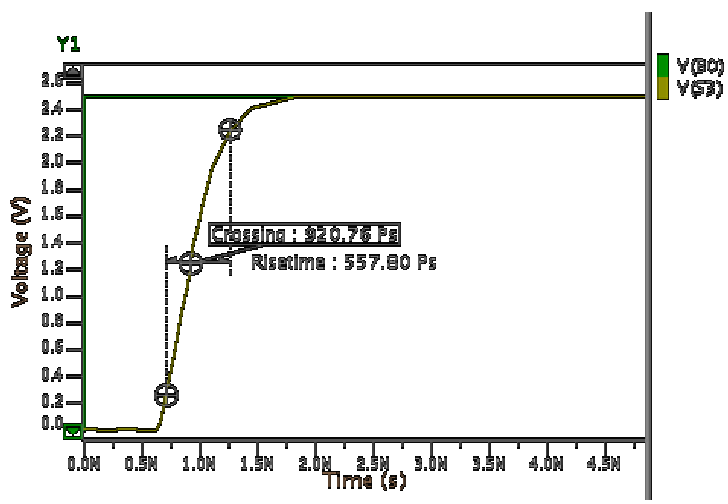
Inputs illustrated in this diagram:

Initial condition	Final condition
A: 0111	A: 0111
+ B: 000**0**	+B: 000**1**
---------------	-------------
S: **0**111	S: **1**000

Where the MSB of the sum will change from low to high, in response to the LSB of the B signal transiting from low to high. Showing in the following schematic the critical path for this adder:



Measured quantities:
TpLH: 0.92 ns
Rise time: 557 ps



Calculating number of transistors

Gate/Unit	Composed of	#transistors
DPL-XOR	4+ 2 inv.	8
TG- AND	5(3+ 1 inv.)	5
CMOS-NAND	4	4
CPL-NAND	2+ 2 inv.	6
Inverter	2	2
Half-Adder	1 DPL- XOR + 1 TG-AND = 8 + 5	13
Full-Adder	2 DPL-XOR + 2 CPL-NAND+ 1 CMOS-NAND=16+12+4	32

Four-bit adder = 1 HA + 3 FA= 13 + (3x32)=**109 transistors**

Measuring power consumption

Power consumption was measured by measuring average current supplied by V_{dd} and multiplying it by V_{dd}

Frequency used: 10 MHz

$I(V_{dd})$=0.28uA

Power= 0.24x2.5 = 0.7uW

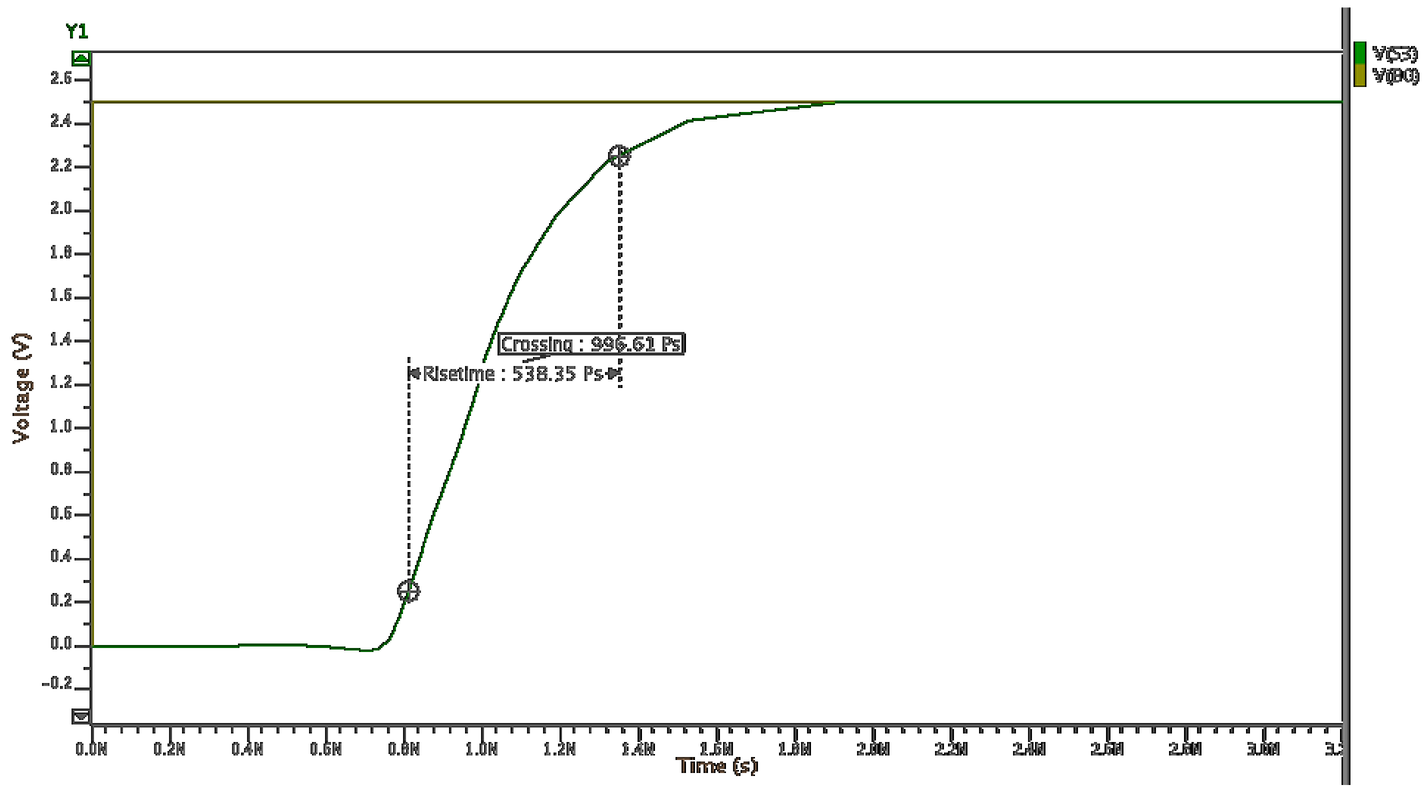
Re-evaluation Phase:

We decided to do some other testing, we realized that we can replace the two CPL NAND of the Full Adder with CMOS NAND, based on simulation results, the CPL is better in terms of power, while CMOS is better in terms of delay.

After we did the simulation to the 4-bit adder, one time using the CPL NAND, while the other time replacing the two CPL NANDs with CMOS NANDs, in the Full Adder Circuit above, we got these results:

	4-bit RCA using CPL NAND in FA	**4-bit RCA using CMOS NAND in FA**
# transistors	109	97
Power	0.7uW	0.25uW
Area	815	707
Delay (tpLH)	0.92 ns	0.99 ns

Operating frequency=**10 MHz**



T_{rise}&TpLH for the improved/re-evaluated adder

This proves that using the CMOS NAND instead of the CPL NAND is actually a better option. Measuring our cost function**A*T*P**for:

4-bit RCA using CPL NAND: cost function= 524.86

4-bit RCA using CMOS NAND: cost function= 174.98

It is obvious that using CMOS NAND is a great and major improvement in the performance of the adder, and is surely considered instead of the CPL NAND.