

1 Introduction

An adder is a digital circuit that performs addition of numbers. In many computers and other kinds of processors adders are used in the arithmetic logic units or ALU. They are also used in other parts of the processor, where they are used to calculate addresses, table indices, increment and decrement operators and similar operations.

Although adders can be constructed for many number representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or ones' complement is being used to represent negative numbers, it is trivial to modify an adder into an adder-subtractor. Other signed number representations require more logic around the basic adder.

2 Lab Procedures

2.1 Design a Full Adder

- ❑ Choose two to three of full adder structures mentioned in class (28T, majority based adder, transmission gate adder, complementary pass transistor adder and hybrid adder) to simulate
- ❑ Verify your adder functionality with the full adder truth table
- ❑ Report delay, power, and area measurement of your full adder

A	B	CIN	COUT	SUM
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1

NOTE:

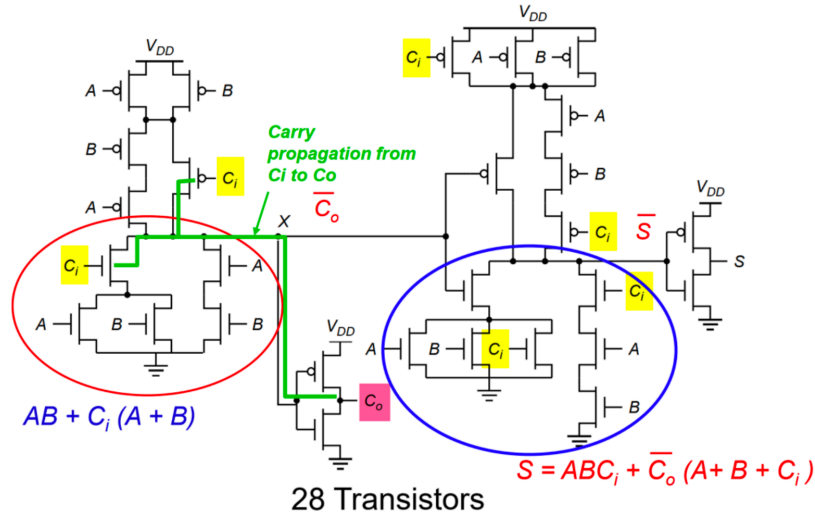
- ❑ Area: estimate the area of full-adder roughly by *accumulate the area of each single transistor*, i.e. $\sum Fin_number * [Width_fin * Lg]$

2.1.1 Conventional 28T circuits

The conventional 28T Adder is completely designed as CMOS logic. It's shown as below:

Conventional 28T Adder

- 28T加法器由两部分组成，分别产生进位信号和求和信号
- 进位信号的下拉网络满足逻辑表达式 $AB + C_i (A + B)$
- 求和信号下拉网络满足逻辑表达式 $ABC_i + \overline{C_o} (A + B + C_i)$



So, this task is going to connect the structure of the circuit.

I design 2 sub-circuits, 1 is for the **COUT**, another is for **SUM**. Then I use 2 inverters separately to transfer the output into the correct one.

The core code is shown as below:

```

1 .subckt cout28T a b ci cobar size=n
2 xcout_n_a1 cout_v1 a gnd gnd nfet l=length NFIN='2*size'
3 xcout_n_b1 cout_v1 b gnd gnd nfet l=length NFIN='2*size'
4 xcout_n_ci cobar ci cout_v1 gnd nfet l=length NFIN='2*size'
5 xcout_n_a2 cobar a cout_n_a_b gnd nfet l=length NFIN='2*size'
6 xcout_n_b2 cout_n_a_b b gnd gnd nfet l=length NFIN='2*size'
7 xcout_p_a1 cobar a cout_p_a_b vdd pfet l=length NFIN='4*size'
8 xcout_p_b1 cout_p_a_b b cout_p_b_a vdd pfet l=length NFIN='4*size'
9 xcout_p_a2 cout_p_b_a a vdd vdd pfet l=length NFIN='2*size'
10 xcout_p_b2 cout_p_ci_b b vdd vdd pfet l=length NFIN='2*size'
11 xcout_p_ci cobar ci cout_p_ci_b vdd pfet l=length NFIN='2*size'
12 .ends

```

cout bar.sp

This code receives 3 input: **a**, **b**, **cin** and return the inversion of **cout**.

```

1 .subckt sum28T a b ci cobar sbar size=n
2 xsum_n_co sbar cobar co_n gnd nfet l=length NFIN='2*size'
3 xsum_n_a1 co_n a gnd gnd nfet l=length NFIN='2*size'
4 xsum_n_b1 co_n b gnd gnd nfet l=length NFIN='2*size'
5 xsum_n_ci1 co_n ci gnd gnd nfet l=length NFIN='2*size'
6 xsum_n_ci2 sbar ci sum_n_a_ci gnd nfet l=length NFIN='3*size'

```

```

7 xsum_n_a2 sum_n_a_ci a sum_n_a_b gnd nfet l=length NFIN='3*size'
8 xsum_n_b2 sum_n_a_b b gnd gnd nfet l=length NFIN='3*size'
9
10 xsum_p_co sbar cobar sum_v1 vdd pfet l=length NFIN='2*size'
11 xsum_p_ci2 sbar ci sum_p_ci_b vdd pfet l=length NFIN='6*size'
12 xsum_p_b2 sum_p_ci_b b sum_p_b_a vdd pfet l=length NFIN='6*size'
13 xsum_p_a2 sum_p_b_a a sum_v1 vdd pfet l=length NFIN='6*size'
14 xsum_p_a1 sum_v1 a vdd vdd pfet l=length NFIN='2*size'
15 xsum_p_b1 sum_v1 b vdd vdd pfet l=length NFIN='2*size'
16 xsum_p_ci1 sum_v1 ci vdd vdd pfet l=length NFIN='2*size'
17 .ends

```

sum bar.sp

This code receives 3 input: **a**, **b**, **cin** and return the inversion of **sum**.

Then, we just need to connect each module with teh sub-circuit:

```

1 xcout28T a b ci cobar cout28T
2 xsum28T a b ci cobar sbar sum28T
3 xcout cobar cout inv
4 xsum sbar sum inv
5 \\the input and output buffers codings are below:
6 xa ain a inv
7 xb bin b inv
8 xc cinin ci inv
9 xco cout co inv num=5
10 xsumo sum sumo inv num=5

```

t1 28T.sp

We must notice that the **SIZE** of each transistor is designed as by the **Logic Effort**. The specific produce's quite simple so I'm not about to show this here.(if you want to have a chat with me, just contact me)

The measure method is separated into 2 parts: **1.power 2.delay**:

Power estimation just can be done by this:

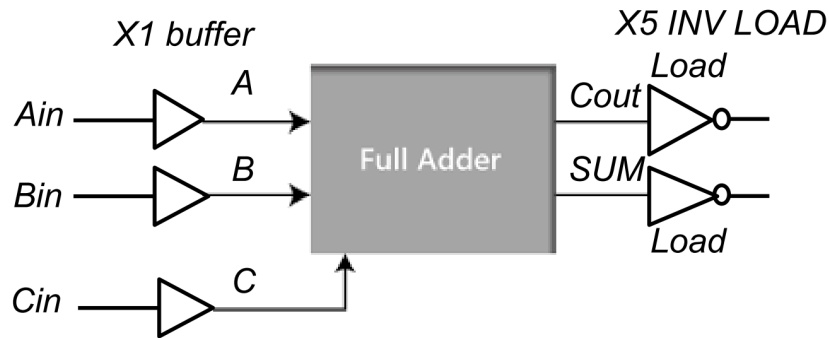
```

1 VA A GND PULSE (0 0.65 2N 50p 50p 4n 8n)
2 VB B GND PULSE (0 0.65 2N 50p 50p 2n 4n)
3 VC Ci GND PULSE (0 0.65 2N 50p 50p 1n 2n)
4 vdd vdd gnd supply
5 .tran 0.1n 20n
6 .measure tran Power AVG "abs(I(VDD)*V(VDD))" FROM=1ns TO=9ns

```

power measure.sp

And for the delay estimation, the buffers are necessary! So just add on input buffer with minimum size and an output buffer with 5 times size is ok.

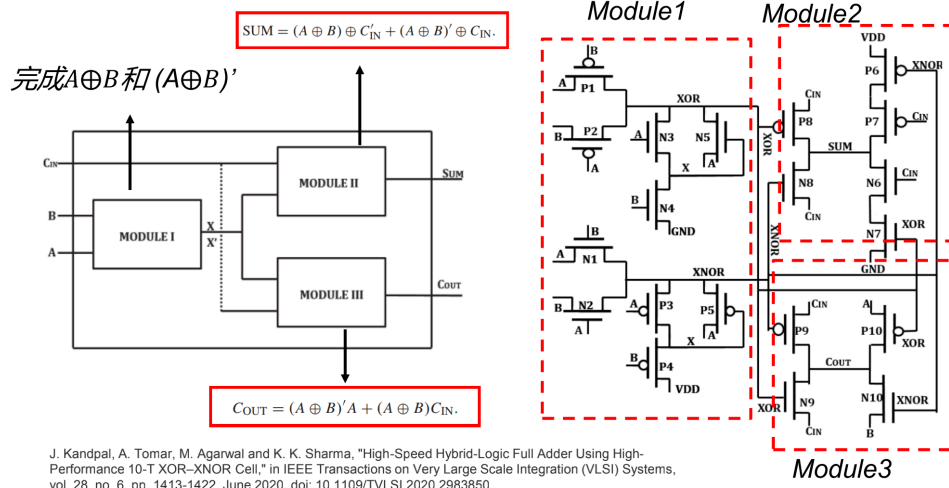


2.1.2 hybrid adder

Recently, in IEEE conference, there's a paper about adder design: Hybrid logic style is widely used to implement full adder (FA) circuits. Performance of hybrid FA in terms of delay, power, and driving capability is largely dependent on the performance of XOR-XNOR circuit. In this article, a high-speed, low-power 10-T XOR-XNOR circuit is proposed, which provides full swing outputs simultaneously with improved delay performance. The performance of the proposed circuit is measured by simulating it in cadence virtuoso environment using 90-nm CMOS technology. The proposed circuit reduces the power delay product (PDP) at least by 7.5% than that of the available XOR-XNOR modules. Four different designs of FAs are also proposed in this article utilizing the proposed XOR-XNOR circuit and available sum and carry modules. The proposed FAs provide 2%-28.13% improvement in terms of PDP than that of other architectures. To measure the driving capabilities, the proposed FAs are embedded in 2-, 4-, and 8-bit cascaded full adder (CFA) structures. Results show that two of the proposed FAs provide the best performance for a higher number of bits among all the FAs. [1]

So in this part, I'm gonna simulate it. It's easy to understand:

□ 下图是第二种混合型全加器：由互补传输管逻辑，传输门逻辑，传输管逻辑和CMOS逻辑组成



I designed 3 modules for this:

```

1 .subckt module1 a b xor xnor
2 xp1 xor b a vdd pfet l=length NFIN=1
3 xp2 xor a b vdd pfet l=length NFIN=1
4 xn1 a b xnor gnd nfet l=length NFIN=1
5 xn2 b a xnor gnd nfet l=length NFIN=1
6 xn3 xor a x gnd nfet l=length NFIN=1
7 xn4 x b gnd gnd nfet l=length NFIN=1
8 xn5 xor x a gnd nfet l=length NFIN=1
9 xp3 x a xnor vdd pfet l=length NFIN=1
10 xp4 vdd b x vdd pfet l=length NFIN=1
11 xp5 a x xnor vdd pfet l=length NFIN=1
12 .ends

```

module1.sp

This module implement the logic of **XOR**, **XNOR** by pass transistor gate.

```

1 .subckt module2 xor xnor cin sum
2 xp8 sum xor cin vdd pfet l=length NFIN=1
3 xn8 sum xnor cin gnd nfet l=length NFIN=1
4 xp6 d1 xnor vdd vdd pfet l=length NFIN=1
5 xp7 sum cin d1 vdd pfet l=length NFIN=1
6 xn6 sum cin d2 gnd nfet l=length NFIN=1
7 xn7 d2 xor gnd gnd nfet l=length NFIN=1
8 .ends

```

module2.sp

This module implement the logic of **SUM** by pass transistor gate.

```

1 .subckt module3 xor xnor cin cout

```

```

2 xp9 cout xnor cin vdd pfet l=length NFIN=1
3 xn9 cout xor cin gnd nfet l=length NFIN=1
4 xp10 cout xor a vdd pfet l=length NFIN=1
5 xn10 cout xnor b gnd nfet l=length NFIN=1
6 .ends

```

module3.sp

This module implement the logic of **COUT** by pass transistor gate.

Then the only thing we should do is connecting the 3 sub-circuits together:

```

1 x1 a b xor xnor module1
2 x2 xor xnor cin sum module2
3 x3 xor xnor cin cout module3
4
5 xa ain a inv
6 xb bin b inv
7 xc cinin cin inv
8 xcout cout co inv num=5
9 xsum sum sumo inv num=5

```

t1 hybrid.sp

The power estimation and delay estimation module is teh same as 28T.

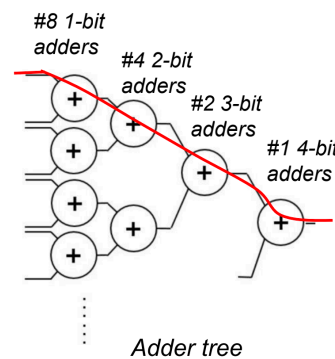
2.2 Design an Adder Tree

In this task, we're going to finish an adder tree. The demands are like these:

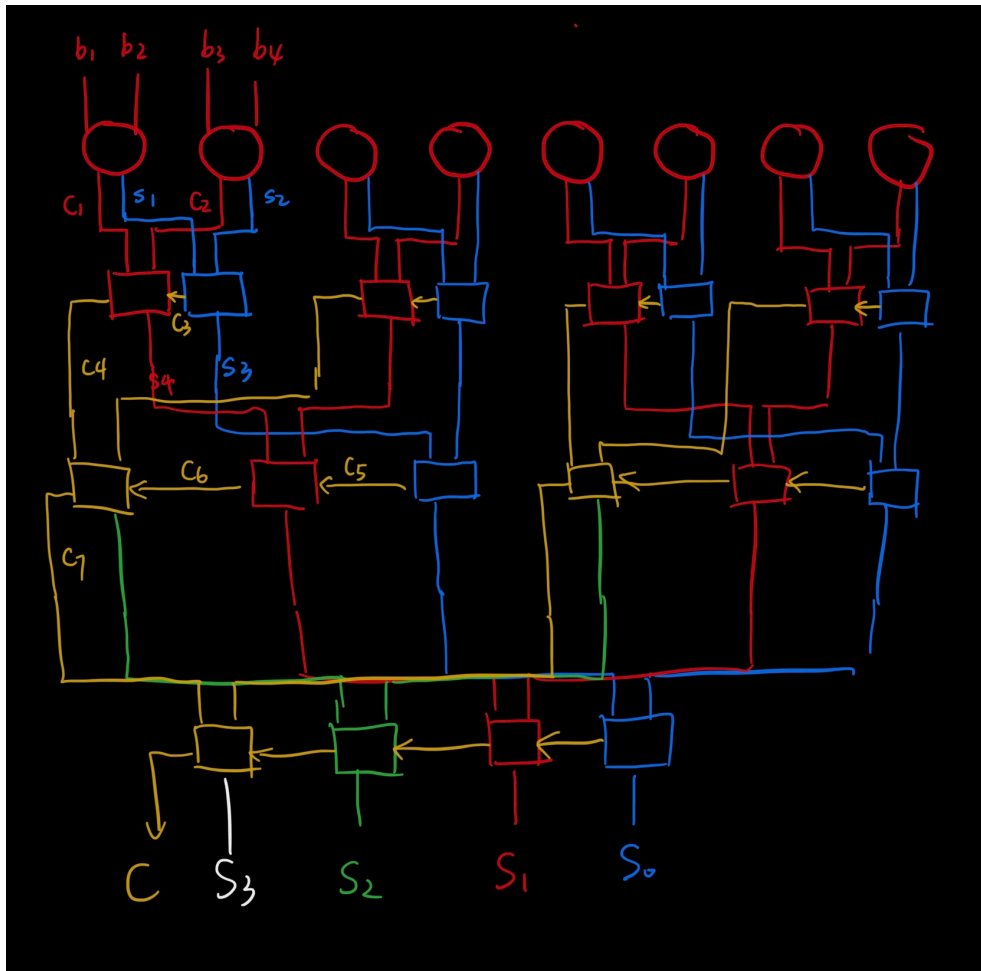
Task2(optional): Design an Adder Tree

- ▣ Construct an adder tree to accumulate 16*1b unsigned input to 1*4b unsigned output
- ▣ You can choose different full adder structures
- ▣ Report latency of critical path (from data input to the 4 bits adder final carry out)

Adder Type	Number
1 bits adder	8
2 bits adder	4
3 bits adder	2
4 bits adder	1



Due to the structure, I design a **ripple** adders. The main structure is like this:



Just like the diagram above, each circle is a basic 1-bit adder designed in task1. I choose **hybrid** adder to inform this due to its fast efficiency and less power cost.

```

1
2 .subckt adder a b cin sum cout
3 x_module1 a b xor xnor module1
4 x_module2 xor xnor cin sum module2
5 x_module3 xor xnor cin cout module3
6 .ends \\ definition of a single bit adder
7
8 x_adder1 b11 b12 0 s1 c1 adder
9 x_adder2 b21 b22 0 s2 c2 adder
10 x_adder3 b31 b32 0 s3 c3 adder
11 x_adder4 b41 b42 0 s4 c4 adder
12 x_adder5 b51 b52 0 s5 c5 adder
13 x_adder6 b61 b62 0 s6 c6 adder
14 x_adder7 b71 b72 0 s7 c7 adder
15 x_adder8 b81 b82 0 s8 c8 adder
16
17 x_adder2_1 c1 c2 c2_2 s2_1 c2_1 adder
18 x_adder2_2 s1 s2 0 s2_2 c2_2 adder
19 x_adder2_3 c3 c4 c2_4 s2_3 c2_3 adder

```

```

20 x_adder2_4 s3 s4 0 s2_4 c2_4 adder
21 x_adder2_5 c5 c6 c2_6 s2_5 c2_5 adder
22 x_adder2_6 s5 s6 0 s2_6 c2_6 adder
23 x_adder2_7 c7 c8 c2_8 s2_7 c2_7 adder
24 x_adder2_8 s7 s8 0 s2_8 c2_8 adder
25
26 x_adder3_1 c2_1 c2_3 c3_2 s3_1 c3_1 adder
27 x_adder3_2 s2_1 s2_3 c3_3 s3_2 c3_2 adder
28 x_adder3_3 s2_2 s2_4 0 s3_3 c3_3 adder
29 x_adder3_4 c2_5 c2_7 c3_5 s3_4 c3_4 adder
30 x_adder3_5 s2_5 s2_7 c3_6 s3_5 c3_5 adder
31 x_adder3_6 s2_6 s2_8 0 s3_6 c3_6 adder
32
33 x_adder4_1 c3_1 c3_4 c4_2 s4_1 c4_1 adder
34 x_adder4_2 s3_1 s3_4 c4_3 s4_2 s4_2 adder
35 x_adder4_3 s3_2 s3_5 c4_4 s4_3 c4_3 adder
36 x_adder4_3 s3_3 s3_6 0 s4_4 c4_4 adder // the structure description based on the diagram above

```

t2.sp

Simply measure the delay from the input to the output. (I select from the b16 to the co, you can see them in my source code).

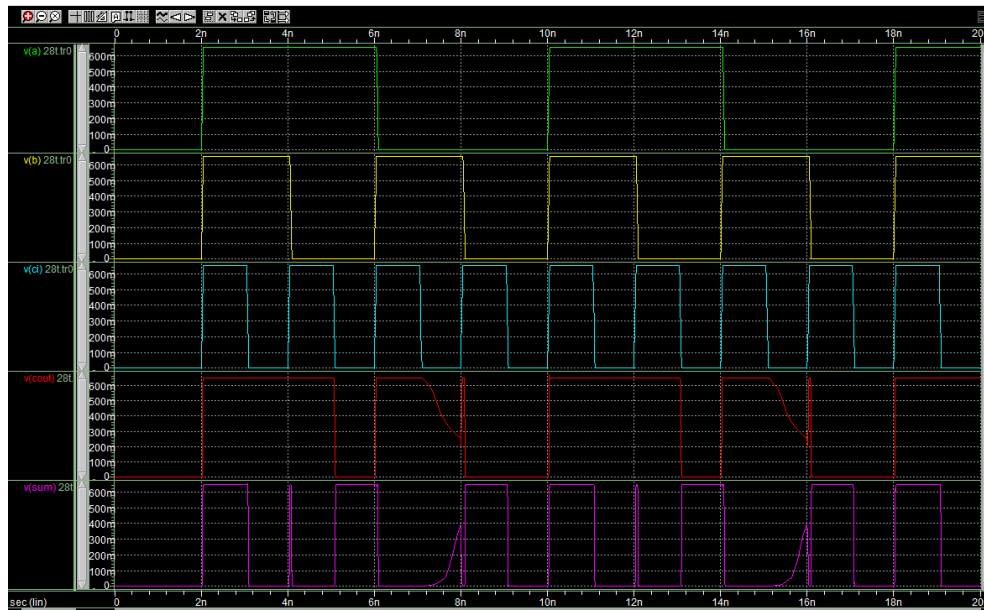
3 Lab Results

3.1 Design a Full Addder

3.1.1 Conventional 28T circuits

Function First, we must verify the function of the adder is correct. It's based on the truth table shown before.

The following patterns can overlap all the possible input patterns in the truth table. Here are the results:



We can conclude that the logic function of our adder is correct.

DelayIn the task, we must check **5 patterns**:

Delay path	Data pattern	Delay path	Data pattern
① A to SUM	000->100	④ C to Cout	100->101
② B to SUM	000->010		010->011
③ C to SUM	000->001		


pattern1:

28t - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'
.TITLE '.title 28t'
tphl      tphl      tp      temper
alter#
4.232e-11  4.277e-11  4.255e-11  25.0000
1
```

pattern2:

 28t - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'


tplh	tplh	tp	temper
------	------	----	--------

alter#

3.913e-11	4.009e-11	3.961e-11	25.0000
-----------	-----------	-----------	---------

1

pattern3:

 28t - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'


tplh	tplh	tp	temper
------	------	----	--------

alter#

3.756e-11	3.993e-11	3.874e-11	25.0000
-----------	-----------	-----------	---------

1

pattern4:

 28t - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'


tplh	tplh	tp	temper
------	------	----	--------

alter#

7.020e-10	2.253e-11	3.623e-10	25.0000
-----------	-----------	-----------	---------

1

pattern5:

 28t - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'


tplh	tplh	tp	temper
------	------	----	--------

alter#

3.601e-11	3.606e-11	3.603e-11	25.0000
-----------	-----------	-----------	---------

1

Power The power of 28T adder is:

 28t - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'

power	temper	alter#
-------	--------	--------

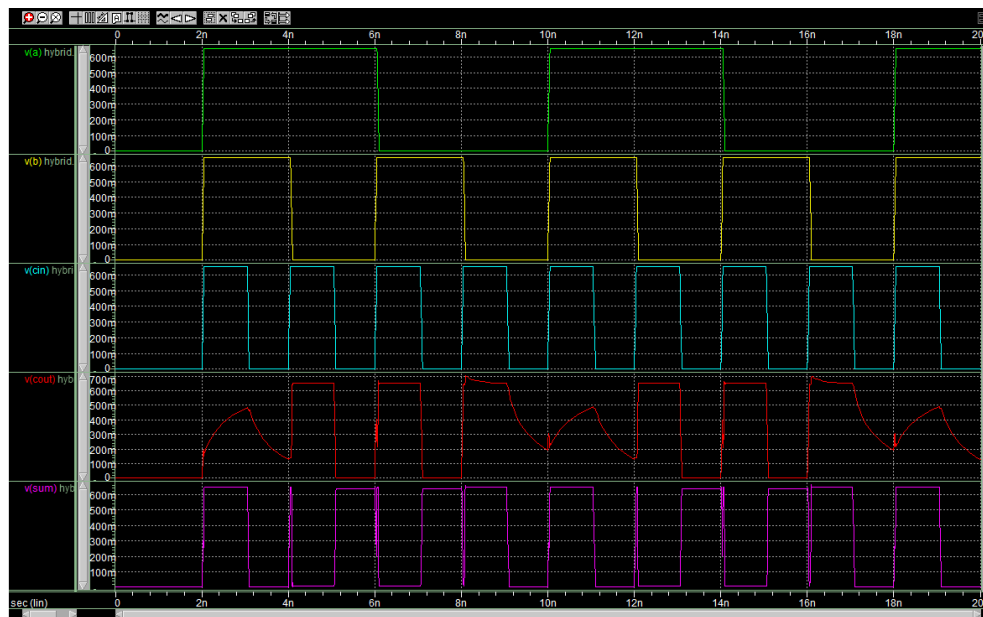
5.160e-06	25.0000	1
-----------	---------	---

Area I estimate the area by counting the number of the transistors the adder uses. It's **28**

3.1.2 hybrid adder

First, we must verify the function of the adder is correct. It's based on the truth table shown before.

The following patterns can overlap all the possible input patterns in the truth table. Here are the results:



We can conclude that the logic function of our adder is correct.

DelayLike before:

pattern1:

hybrid - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'

tphl	tphl	tp	temper
------	------	----	--------

alter#

3.435e-11	3.110e-11	3.272e-11	25.0000
-----------	-----------	-----------	---------

1

pattern2:

hybrid - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'

tphl	tphl	tp	temper
------	------	----	--------

alter#

3.038e-11	3.293e-11	3.165e-11	25.0000
-----------	-----------	-----------	---------

1

pattern3:

hybrid - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'
.TITLE '.title 28t'
tphl      tphl      tp      temper
alter#
1.961e-11  2.064e-11  2.012e-11  25.0000
1
```

pattern4:

hybrid - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'
.TITLE '.title 28t'
tphl      tphl      tp      temper
alter#
1.864e-11  2.248e-11  2.056e-11  25.0000
1
```

pattern5:

hybrid - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BI'
.TITLE '.title 28t'
tphl      tphl      tp      temper
alter#
1.855e-11  2.208e-11  2.031e-11  25.0000
1
```

Power The power of hybrid adder is:

hybrid - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'

power	temper	alter#
9.784e-07	25.0000	1

Area It's only 20 transistors

3.2 Design an Adder Tree

The measure pattern is this:

□ 延迟测量数据模式:

初始数据 16 bit 输入 (1111 1111 1111 1110) → 16 bit 输入 (1111 1111 1111 1111)

□ 加法树输入(所有1 bit adder) 均加经过1X buffer, 输出(4 bit adder) 均加上5X INV_load, 延迟测量为buffer输出到最终进位输出

□ 输入激励的特性按照左侧表格给定

Metrics	Value
V1	0V
V2	0.65v
Delay Time	1ns
Raise Time	50ps
Fall Time	50ps

Set the pattern and simulate it. The result is like below(more specific measure patterns, you can see them in my source code):

tree - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

\$DATA1 SOURCE='HSPICE' VERSION='J-2014.09-2 64-BIT'

.TITLE '.title 28t'

tph	tphl	tp	temper
alter#			
1.110e-09	-1.000e-09	5.483e-11	25.0000
1			

4 Technical Analysis of the Simulation Results

4.1 task1

1. We can clearly conclude that the hybrid adder is faster and costs less power than the traditional 28T adders
2. It's common that we can get some **glitch** in the simulation because of current feed-through effect. Due to this effect when the gate is turned off, charge under the gate moves toward either drain or source sides and distorted output finds out. However, these glitches do not affect the xor-xnor outputs.
3. The delay is calculated from the 50% voltage level at the input to 50% voltage level at the output for all the rise and fall transition, and the worst case delay is selected.

4.2 task2

1. When comes to an adder tree, the delay is mainly depends on the tree structure, but not the single adder performance. The ripple structure I design is so bad, even the simulation process lasts for a long time.

5 Thanks

This is the final lab in this semester, thanks a lot to all of the faculties, the Prof. Fu, Teacher Sun, also very appreciated to TAs, Dengfeng Wang and Mengyuan Guan. I actually learned a lot even though there're unavoidable problems in labs and course, you have solved many of my confusion! Thanks a lot again and see you...

参考文献

- [1] Jyoti Kandpal, Abhishek Tomar, Mayur Agarwal, and KK Sharma. High-speed hybrid-logic full adder using high-performance 10-t xor-xnor cell. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(6):1413–1422, 2020.