

## Mục lục

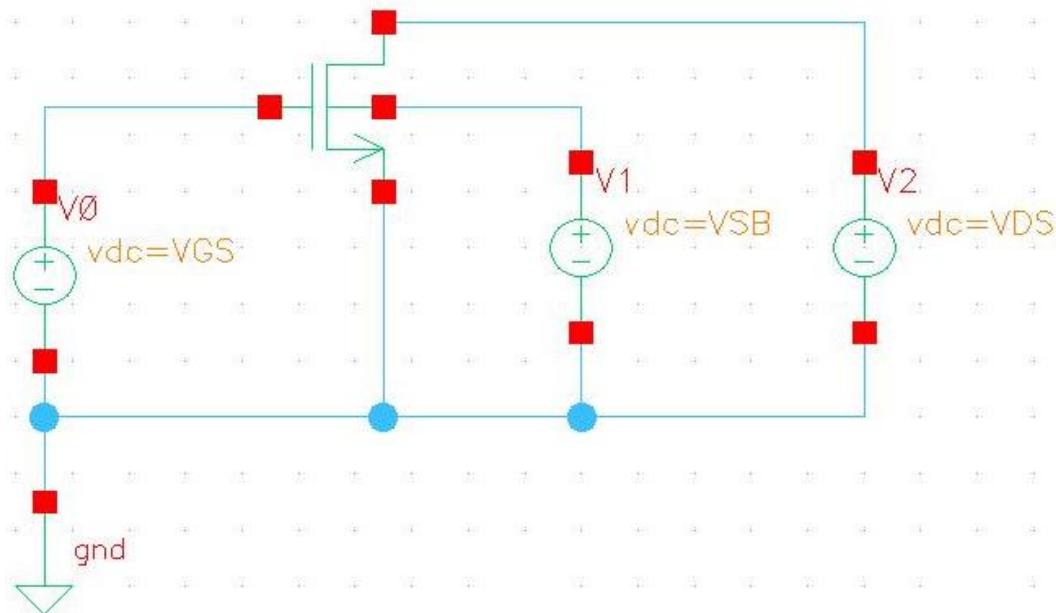
Chương 1: MOS TRANSISTOR CHARATERIZATION .....	3
1.1.    Experiment 1 .....	3
<b>1.2.    Experiment 2</b> .....	5
1.3.    Experiment 3 .....	7
1.4.    Experiment 4 .....	9
Chương 2: DIGITAL LOGIC CIRCUIT .....	13
<b>2.1.    Experiment 1</b> .....	13
2.1.1.    NAND .....	13
2.1.2.    NOR 2.....	16
2.1.3.    XNOR.....	20
<b>2.2.    Experiment 2</b> .....	24
2.3.    Experiment 3 .....	27
Chương 3: COMBINATIONAL AND SEQUENTIAL CIRCUIT .....	32
3.1.    Experiment 1 .....	32
3.2 Experiment2 .....	42
Chương 4: ARITHMETIC LOGIC UNIT & REGISTER FILE .....	47
4.1    Experiment1 .....	47
4.2.    Experiment 2 .....	69
Chương 5: INTRODUCTION TO MEMORY CIRCUIT DESIGN .....	84
5.1.    Experiment 1 .....	84
Chương 6: IC DESIGN FLOW .....	89
6.1 Giới thiệu chung.....	89
6.1.1 Định nghĩa về ASIC .....	89
6.1.2 Công nghệ CMOS .....	90
6.1.3 Tổng quan về ASIC flow .....	95
6.2 Front-end.....	97
6.2.1 System Specification.....	97
6.2.2 Design process.....	98
6.2.3 RTL Verification .....	102
6.2.4 Synthesis.....	109

6.3 Back-end .....	119
6.3.1 <b>Physical Design</b> .....	119
6.3.2 <b>Tóm tắt nội dung các bước trong PD flow</b> .....	121
6.4 Manufaturing.....	145
6.4.1 Các bước cơ bản trong Manufacturing.....	145

# Chương 1: MOS TRANSISTOR CHARACTERIZATION

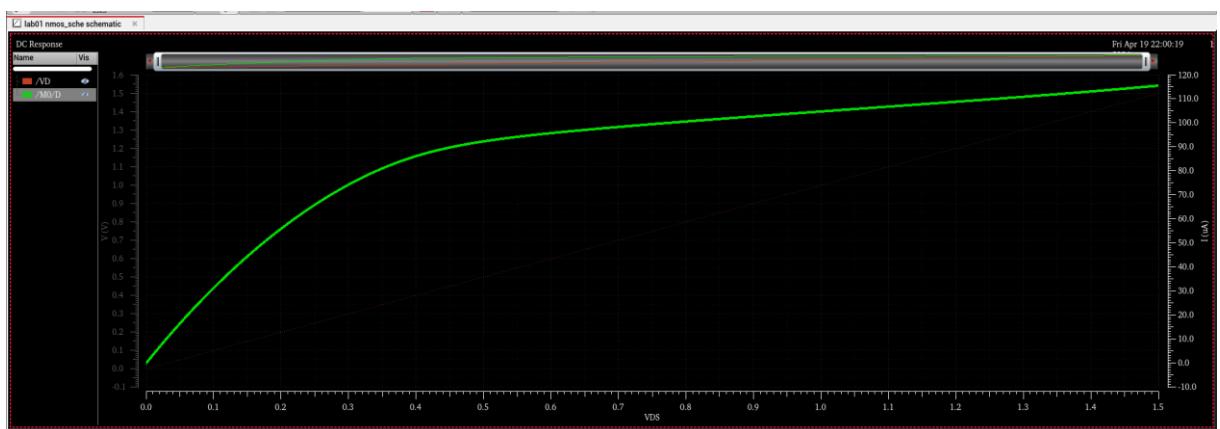
## 1.1. Experiment 1

NMOS



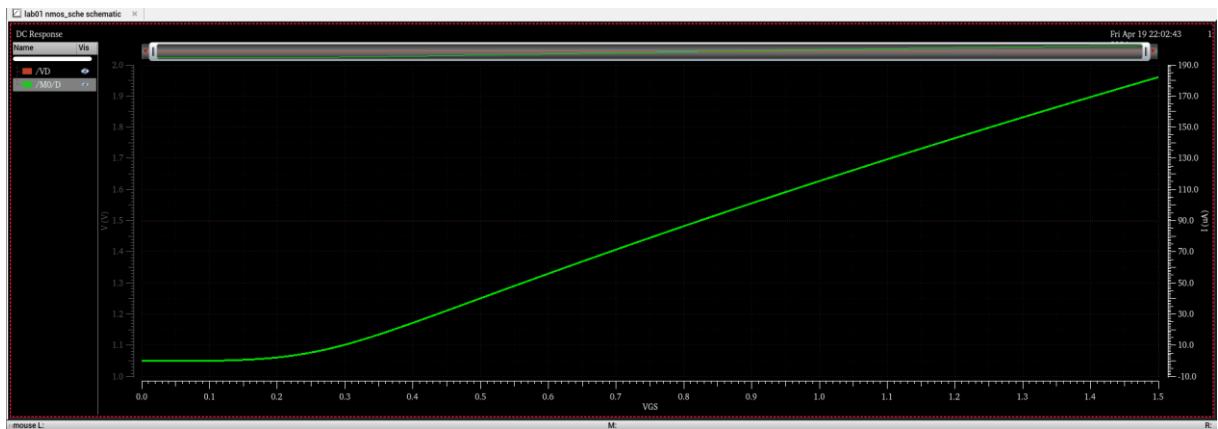
Hình 1-1 Test setup cho NMOS transistor.

Simulate **curves  $I_D$  vs  $V_{DS}$**  @  $V_{GS} = 1V$ , and sweeping variable  $V_{DS} = [0,1.5]V$  step 10mV.



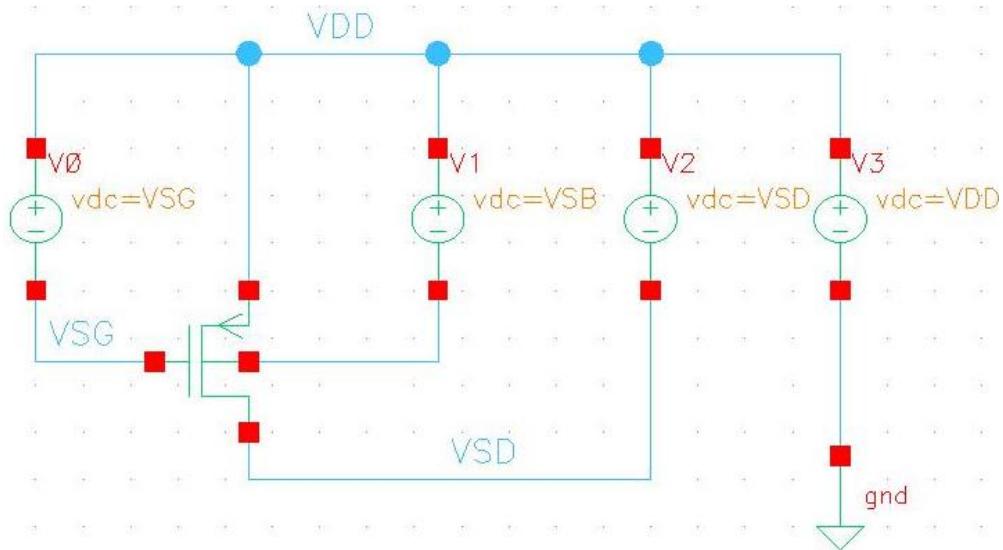
Hình 1-2  $I_D$  vs  $V_{DS}$  of NMOS @  $V_{GS} = 1V$ .

Simulate **curves  $I_D$  vs  $V_{GS}$**  @  $V_{DS} = 1.5V$ , and sweeping variable  $V_{GS} = [0,1.5]V$  step 10mV.



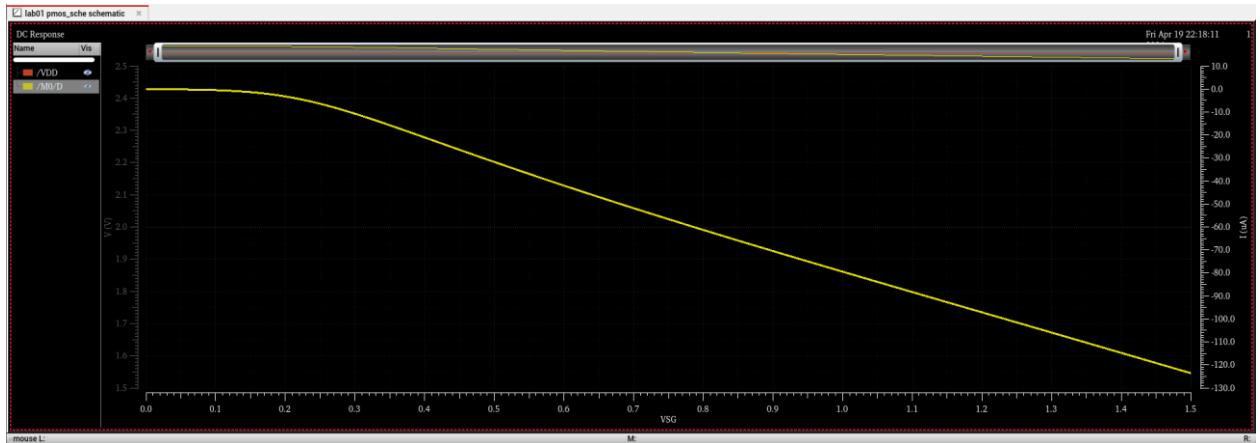
Hình 1-3  $I_D$  vs  $V_{GS}$  of NMOS @  $V_{DS} = 1.5V$ .

### PMOS



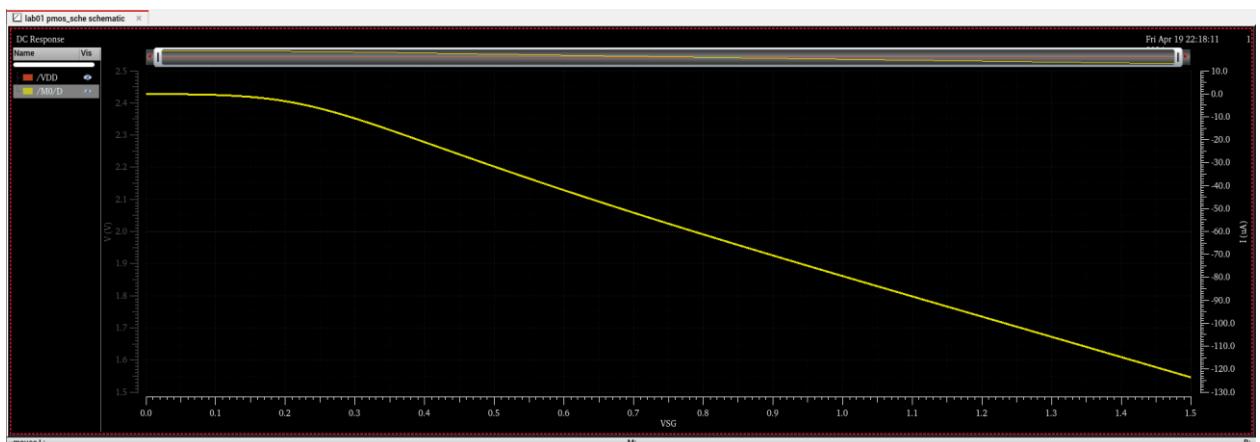
Hình 1-4 Test setup for the PMOS transistor.

Simulate **curves  $I_D$  vs  $V_{SD}$**  @  $V_{SG} = 1V$ , and sweeping variable  $V_{SD} = [0,1.5]V$  step 10mV.



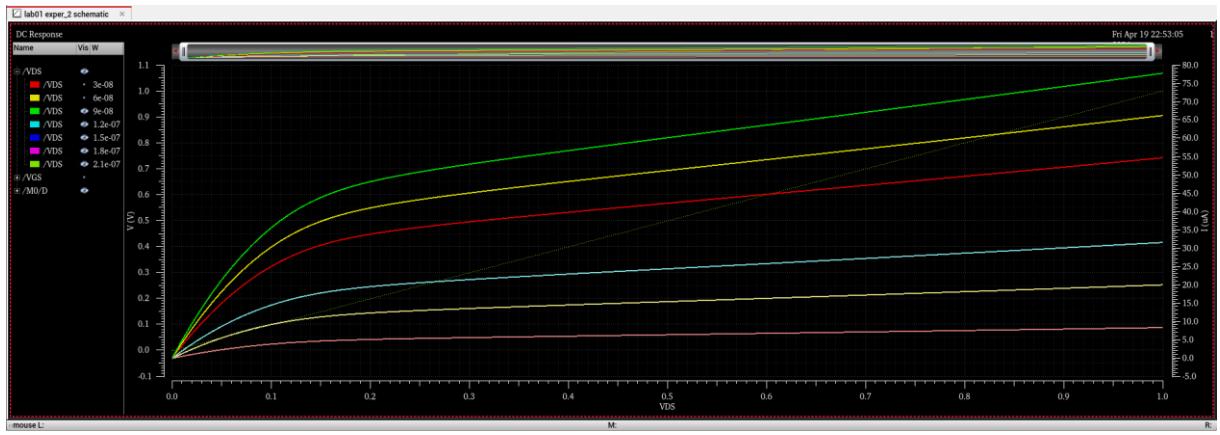
Hình 1-5  $I_D$  vs  $V_{SD}$  of NMOS @  $V_{GS} = 1V$ .

Simulate **curves  $I_D$  vs  $V_{SG}$**  @  $V_{SD} = 1.5V$ , and sweeping variable  $V_{SG} = [0,1.5]V$  step 10mV.

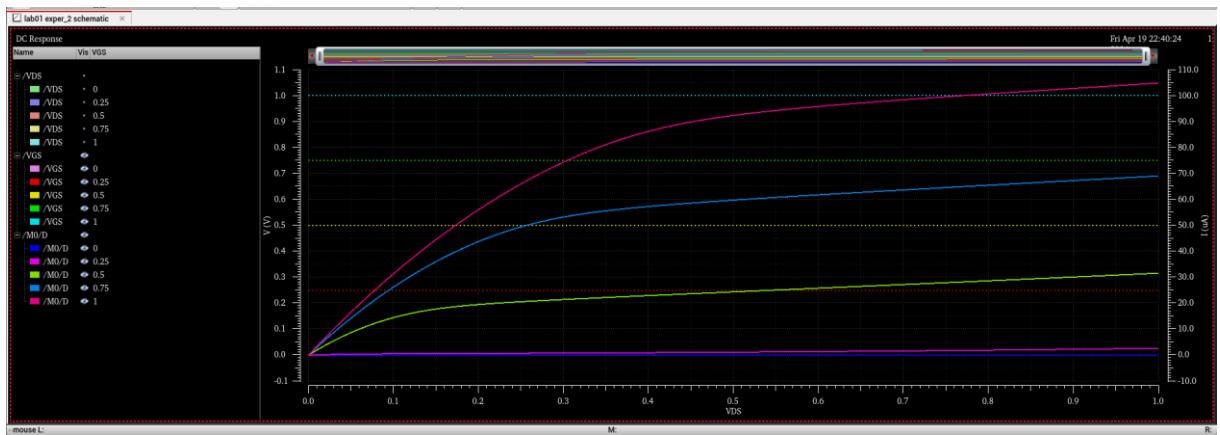


Hình 1-6  $I_D$  vs  $V_{GS}$  of NMOS @  $V_{SD} = 1.5V$ .

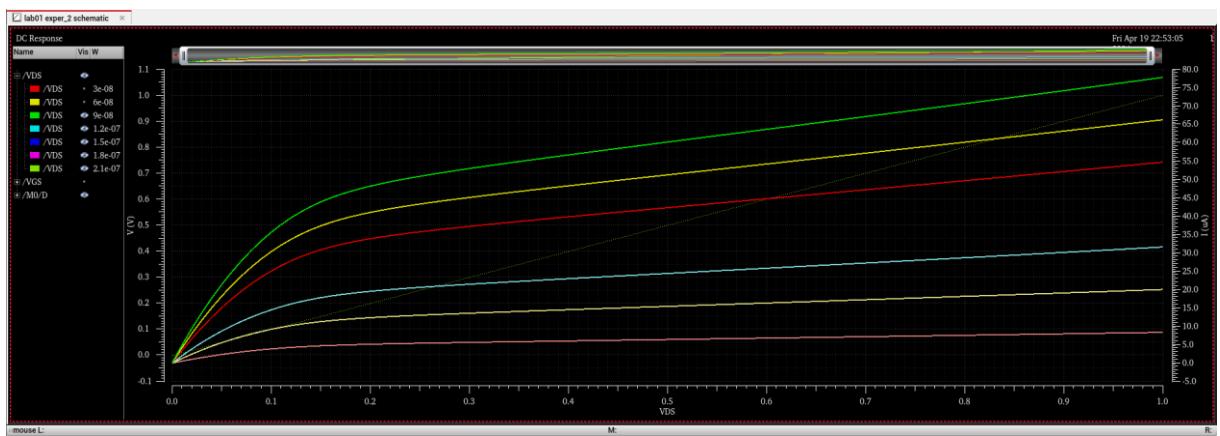
## 1.2. Experiment 2



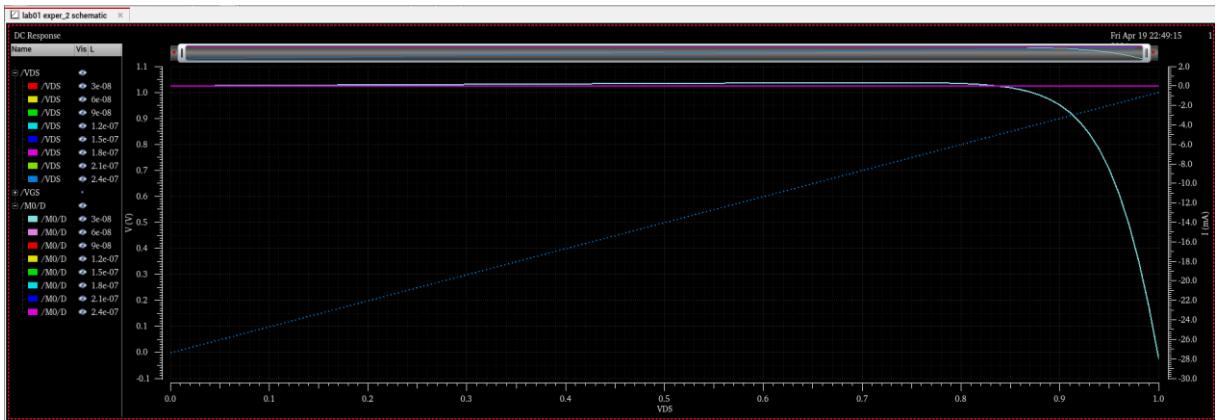
Hình 1-7  $I_D$  vs  $V_{DS}$  @  $V_{GS} = [0,1]$ V step 0.25V.



Hình 1-8  $I_D$  vs  $V_{GS}$  @  $V_{GS} = [0,1]$ V step 0.25V.



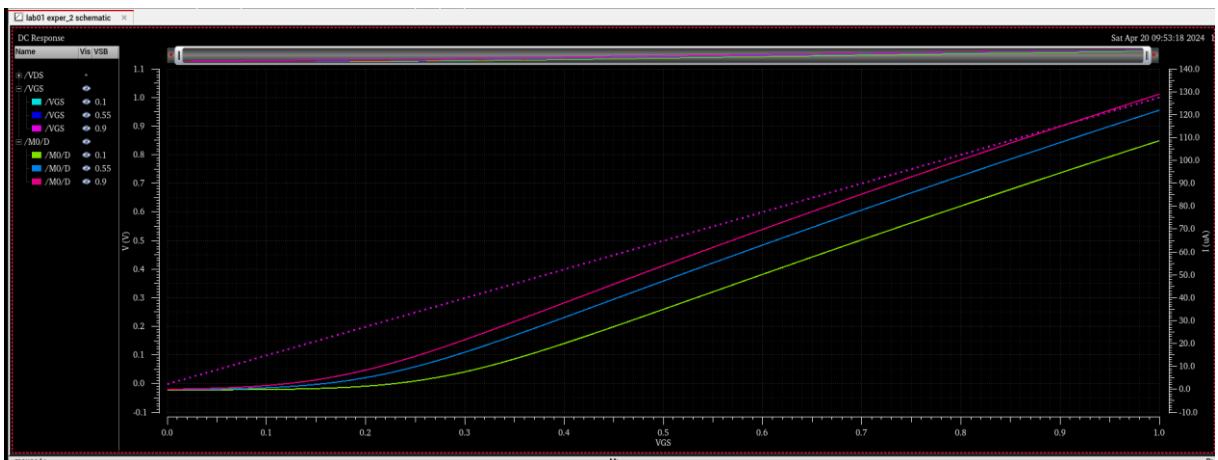
Hình 1-9  $I_D$  vs  $V_{DS}$  @  $W = [30,210]$ nm step 30nm.



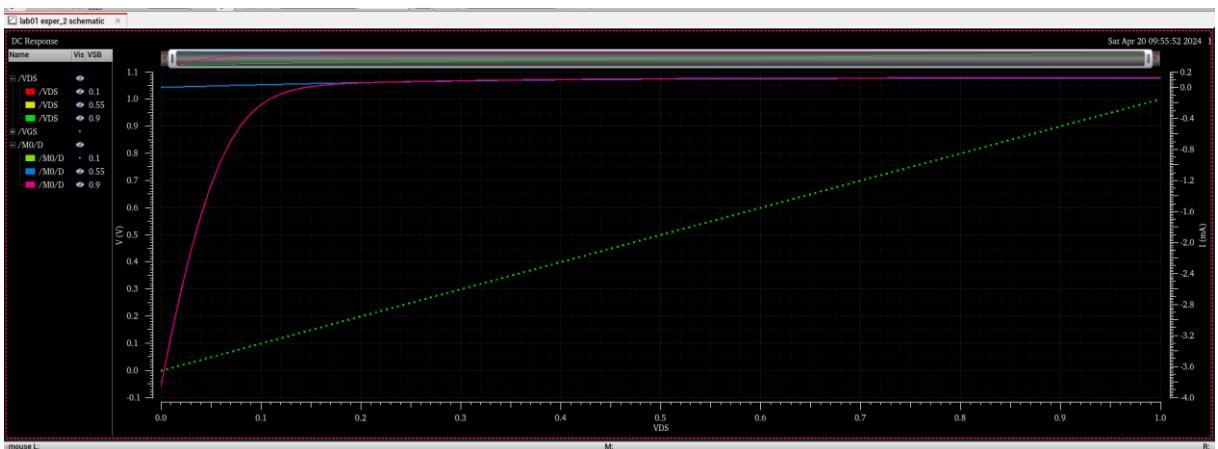
Hình 1-10  $I_D$  vs  $V_{DS}$  @  $L = [30,240]$ nm step 30nm.

### 1.3. Experiment 3

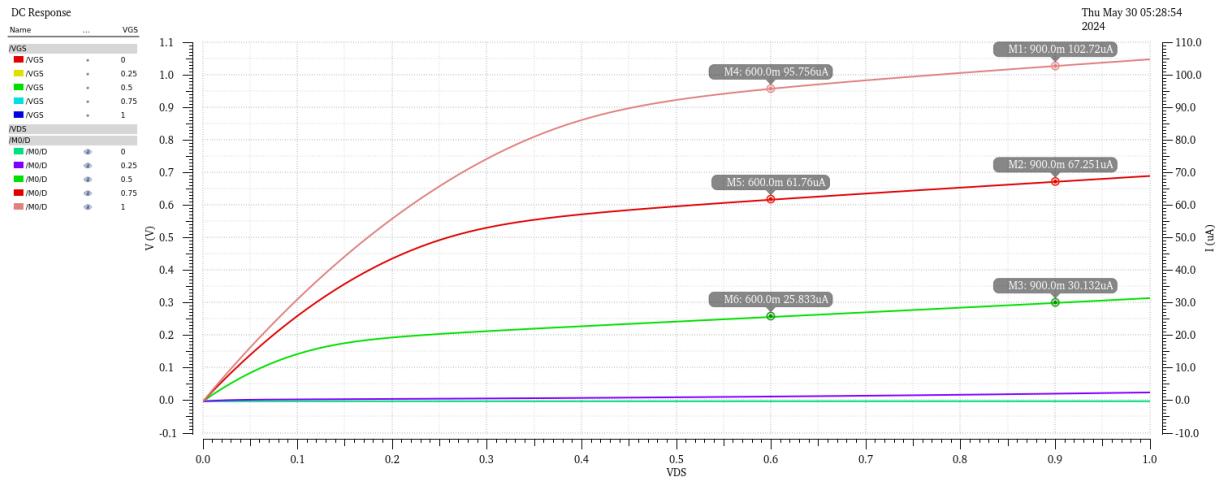
❖ NMOS transistor



Hình 1-11  $I_D$  vs  $V_{GS}$  @  $V_{DS} = 1$ V,  $V_{SB} = \{0.1, 0.55, 0.9\}$ V, and sweeping  $V_{GS} = [0,1]$ V with step 10mV.



Hình 1-12  $I_D$  vs  $V_{DS}$  @  $V_{GS} = 1V$ ,  $V_{SB} = \{0.1, 0.55, 0.9\}V$ , and sweeping  $V_{DS} = [0,1]V$  with step 10mV.



Hình 1-13 Tính toán NMOS transistor.

vth	vsb	vsat_marg	vgt	vgs
247.2m	-θ	959.5m	-247.2m	0
247.2m	-θ	936.2m	2.844m	250m
247.2m	-θ	816.8m	252.8m	500m
247.2m	-θ	688.8m	502.8m	750m
247.2m	-θ	565.3m	752.8m	1

Hình 1-14 Bảng thông số tính toán NMOS transistor.

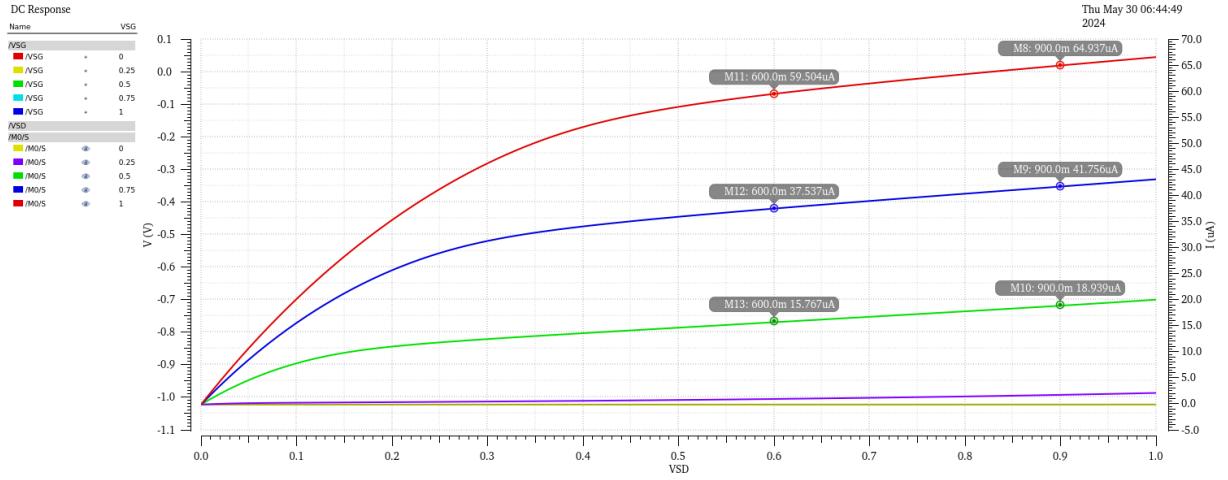
$$\lambda = \frac{I_{D2} - I_{D1}}{I_{D1}V_{DS2} - I_{D2}V_{DS1}} = \frac{102.72 - 95.756}{95.756*0.9 - 102.72*0.6} \approx 0.28$$

$$V_{Th0} = \frac{V_{gs1} - V_{gs2} \sqrt{\frac{I_{ds1}}{I_{ds2}}}}{1 - \sqrt{\frac{I_{ds1}}{I_{ds2}}}} = \frac{1 - 0.75 \sqrt{\frac{102.72}{67.251}}}{1 - \sqrt{\frac{102.72}{67.251}}} \approx 0.06$$

$$k_p = \frac{2I_D}{\frac{W}{L}(V_{GS} - V_{Th0})^2(1 + \lambda V_{DS})} = \frac{2*102.72}{\frac{90}{50}(1 - 0.06)^2(1 + 0.28*0.9)} \approx 103.169$$

$$\gamma = \frac{V_{Th} - V_{Th0}}{\sqrt{|2\emptyset_F| + |V_{SB}|} - \sqrt{|2\emptyset_F|}} \approx 3.24$$

❖ PMOS transistor



Hình 1-15 Tính toán PMOS transistor.

vth	vsb	vsat_marg	vgt	vgs
-212m	-0	958.5m	212m	0
-212m	-0	924.2m	-38.04m	-250m
-212m	-0	809.8m	-288m	-500m
-212m	-0	699.5m	-538m	-750m
-212m	-0	592.1m	-788m	-1

Hình 1-16 Bảng thông số tính toán PMOS transistor.

$$\lambda = \frac{I_{D2} - I_{D1}}{I_{D1}V_{DS2} - I_{D2}V_{DS1}} = \frac{64.937 - 59.504}{59.504*0.9 - 64.937*0.6} \approx 0.37$$

$$V_{Th0} = \frac{V_{gs1} - V_{gs2} \sqrt{\frac{I_{ds1}}{I_{ds2}}}}{1 - \sqrt{\frac{I_{ds1}}{I_{ds2}}}} = \frac{1 - 0.75 \sqrt{\frac{64.937}{41.756}}}{1 - \sqrt{\frac{64.937}{41.756}}} \approx -0.26$$

$$k_p = \frac{2I_D}{\frac{W}{L}(V_{GS} - V_{Th0})^2(1 + \lambda V_{DS})} = \frac{2*64.937}{\frac{90}{50}(1 + 0.26)^2(1 + 0.37*0.9)} \approx 34.09$$

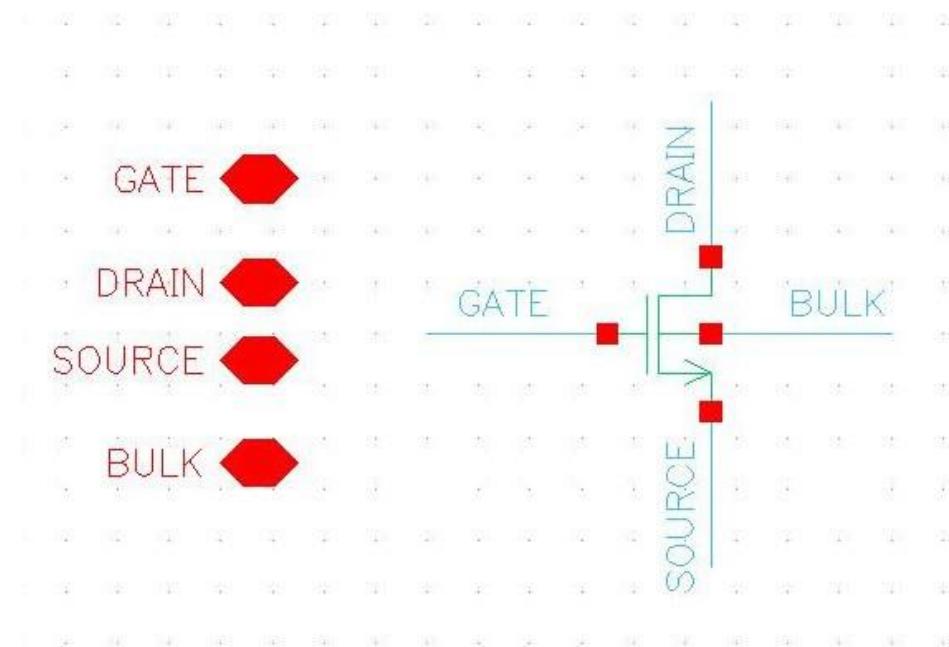
$$\gamma = \frac{V_{Th} - V_{Th0}}{\sqrt{|2\emptyset_F| + |V_{SB}|} - \sqrt{|2\emptyset_F|}} \approx 0.83$$

#### 1.4. Experiment 4

Thiết kế NMOS

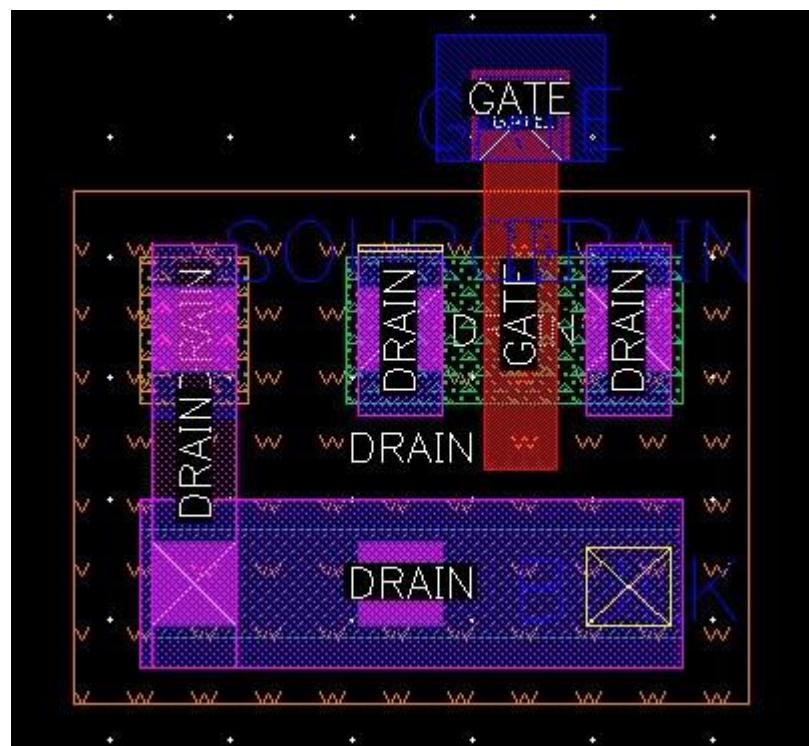
120n/60n NMOS\_VTL

Schematic



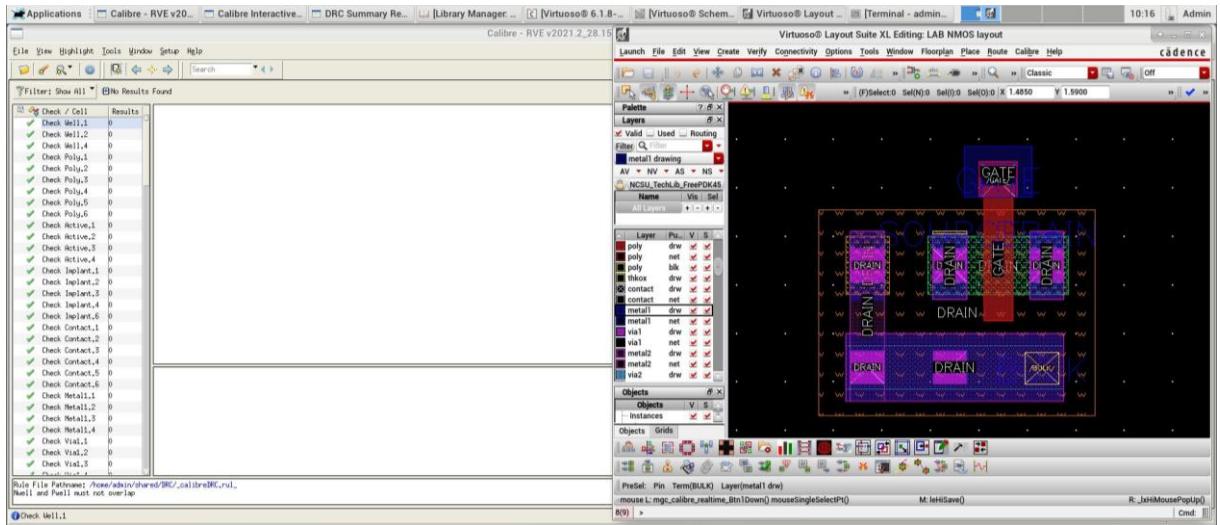
Hình 1-17 Schematic thiết kế NMOS.

### Layout



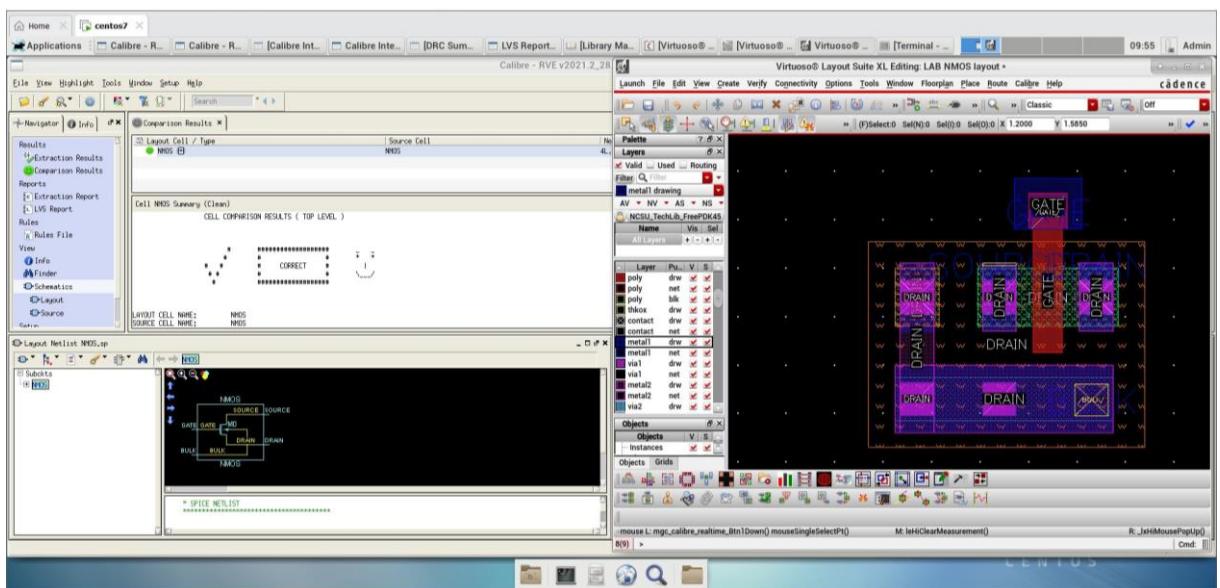
Hình 1-18 Layout 120n/60n NMOS\_VTL.

### Check DRC



Hình 1-19 Kiểm tra DRC 120n/60n NMOS\_VTL thành công.

## Check LVS

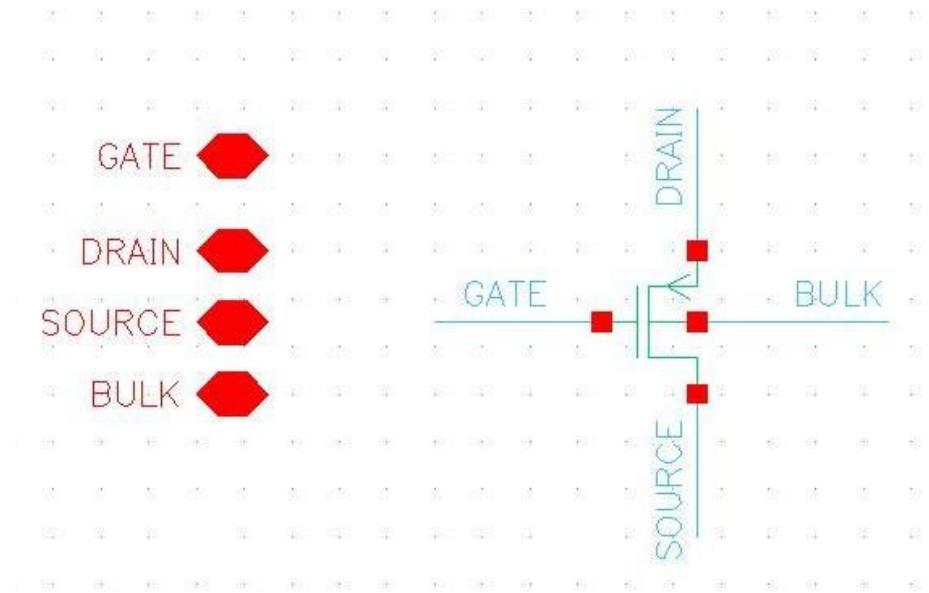


Hình 1-20 Kiểm tra LVS 120n/60n NMOS\_VTL thành công.

## Thiết kế PMOS

50n/40n NMOS\_VTL

Schematic



*Hình 1-21 Schematic thiết kế PMOS.*

Layout

*Hình 1-22 Layout 50n/40n PMOS\_VTL.*

**Check DRC**

*Hình 1-23 Kiểm tra DRC 50n/40n PMOS\_VTL thành công.*

**Check LVS**

*Hình 1-24 Kiểm tra LVS 50n/40n PMOS\_VTL thành công.*

## Chuong 2: DIGITAL LOGIC CIRCUIT

### 2.1. Experiment 1

Implement logic gate using CMOS technology.

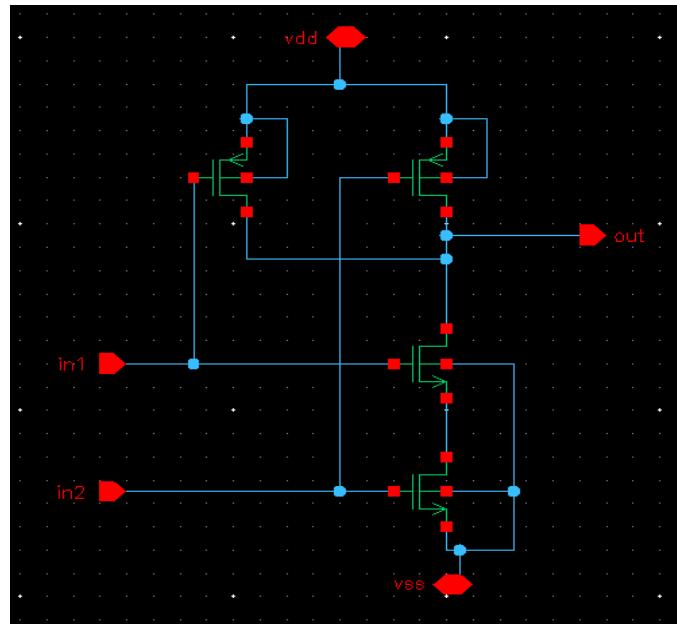
#### 2.1.1. NAND

##### a. Schematic

Truth table:

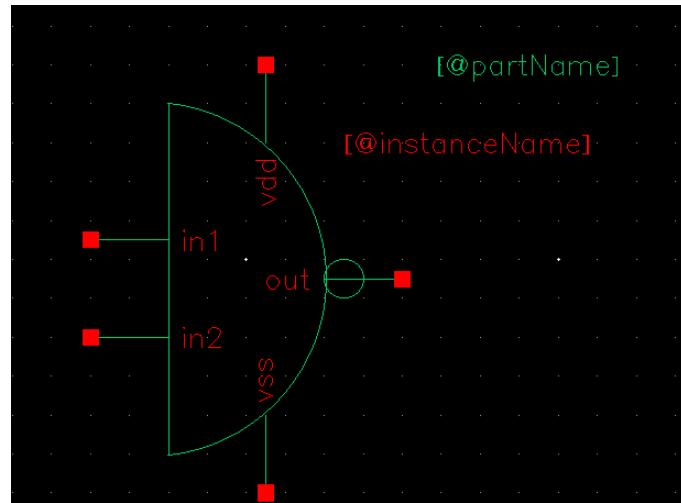
In1	In2	Out
0	0	1
0	1	1
1	0	1
1	1	0

Schematic of NAND gate

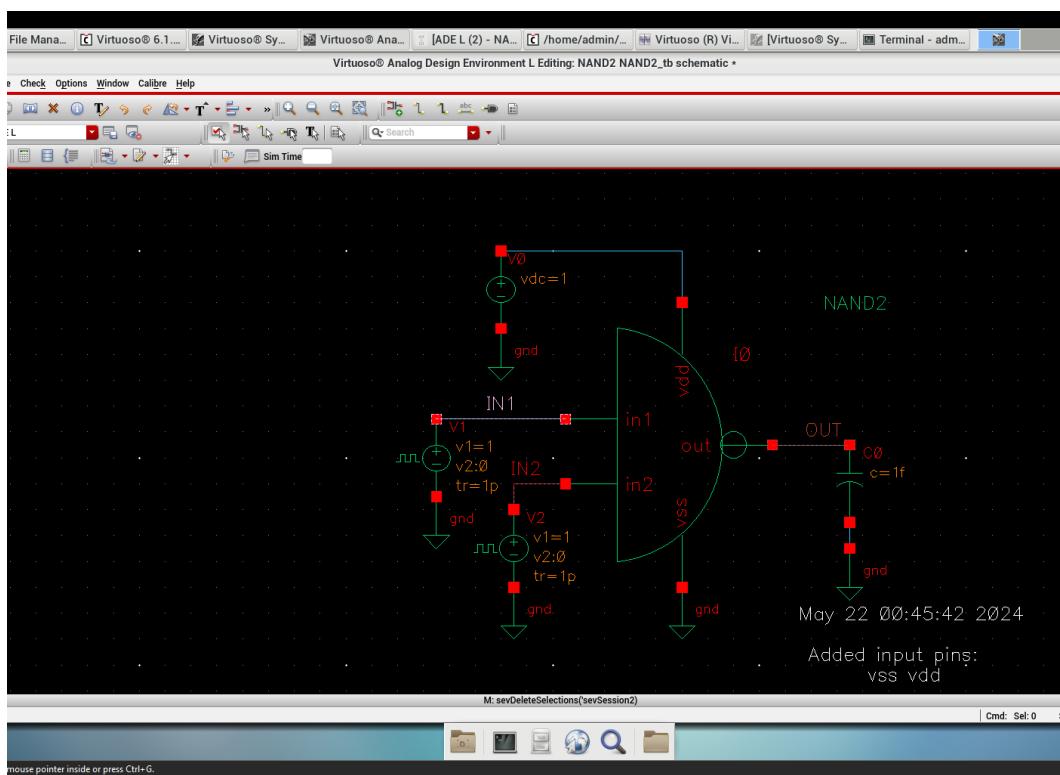


Hinh 2-1 NAND2 Schematic.

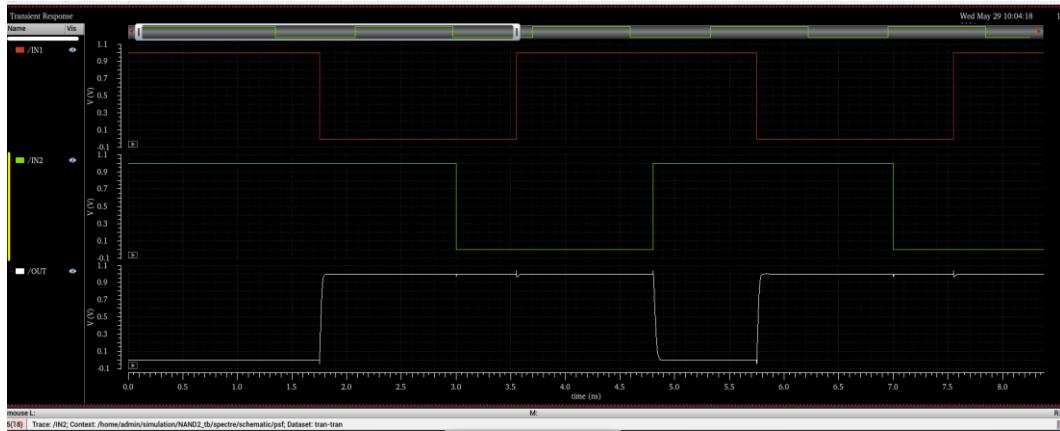
Symbol of NAND gate



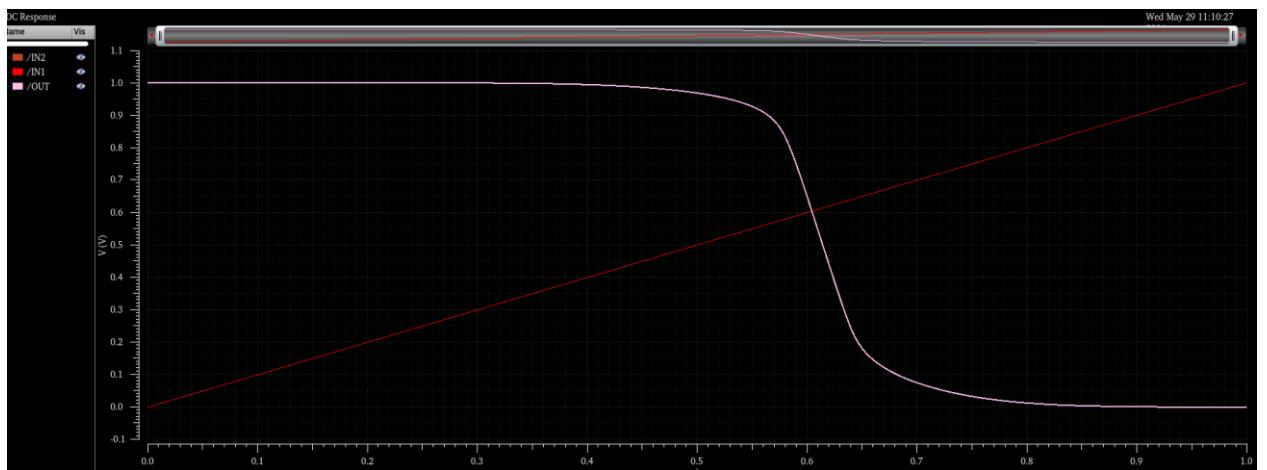
Hình 2-2 NAND2 Symbol



Hình 2-3 NAND2 test band setup



Hinh 2-4 NAND2 Transient simulation



Hinh 2-5 NAND2 DC Analysis

Timing measurements:

Parameters	Results
$t_{rise}$	0.0693(ns)
$t_{fall}$	0.044(ns)
$t_{pdr}$	0.0396(ns)
$t_{pdf}$	0.0253(ns)
$t_{pd}$	0.03245(ns)
Dynamic power	12.249pW
Static power	27.834pW

Output voltage value at various input values:

V <sub>in1</sub> (V)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
V <sub>in2</sub> (V)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
V <sub>out</sub> (mV)	1000	999.46	997	978.1	918	260.8	60	11.35	1.21	0

Nhận xét: về dạng sóng  $V_{out}$  khi hoạt động ché độ DC

- Khi  $V_{in}$  chạy từ 0V-0.5V (2 ngõ vào đang ở mức 0), qua công NAND2 thì ngõ ra ở mức 1 tương đương  $V_{out}$  ngược pha có giá trị xấp xỉ 1V.
- Khi  $V_{in}$  chạy từ 0.5V-1V (2 ngõ vào tiệm cận mức 1) qua công NAND2 thì ngõ ra tiến tới ở mức 0.
- Tại  $V_{in1} = V_{in2} = 1V$  thì ngõ ra  $V_{out}=0V$ .

→ Hoạt động của mạch: Khi 2 ngõ vào có giá trị điện áp thấp (mức logic 0) thì ngõ ra sau khi qua công NAND2 sẽ có giá trị điện áp cao (mức logic 1). Càng tăng giá trị điện áp 2 ngõ vào thì ngõ ra sẽ tiến dần tới điện áp thấp (mức logic 0). Khi 2 giá trị ngõ vào đồng thời ở mức logic 1 thì ngõ ra sẽ ở mức logic 0.

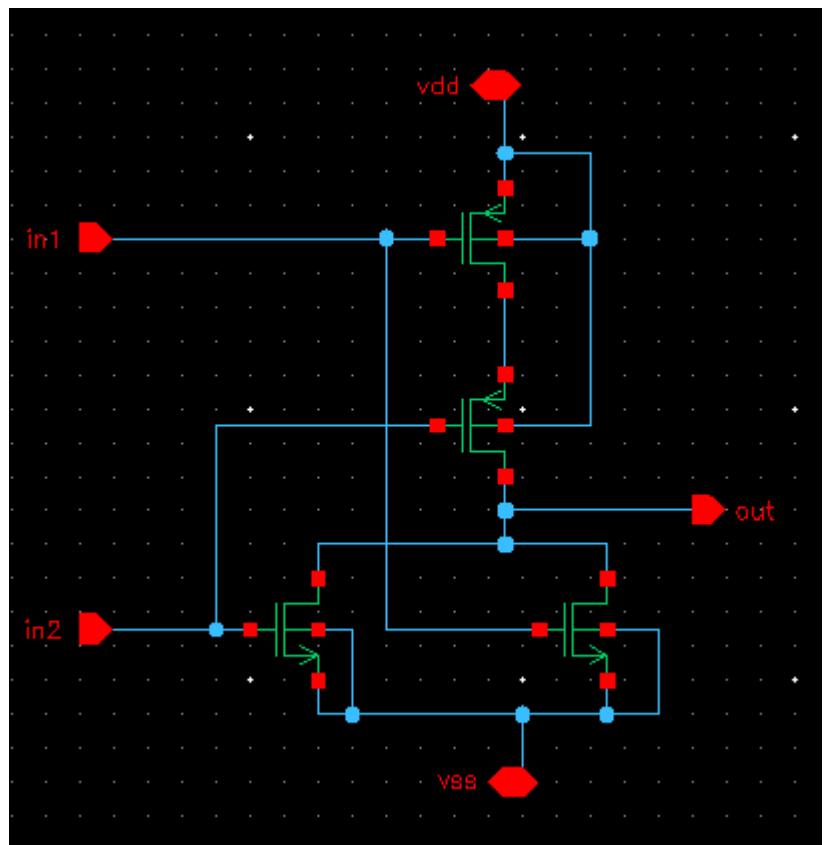
→ Thực hiện mô phỏng đáp ứng DC giống với lý thuyết hoạt động.

### 2.1.2. NOR 2

Truth table:

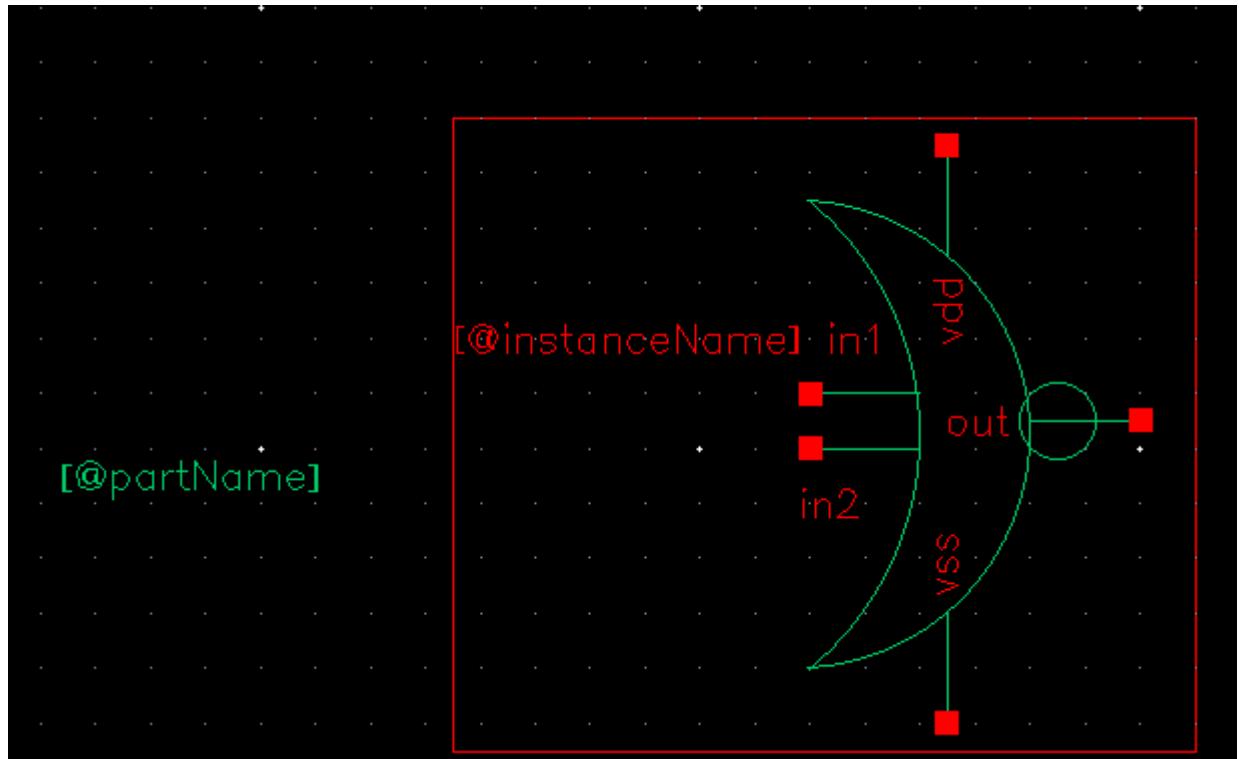
In1	In2	Out
0	0	1
0	1	0
1	0	0
1	1	0

Schematic of NOR gate

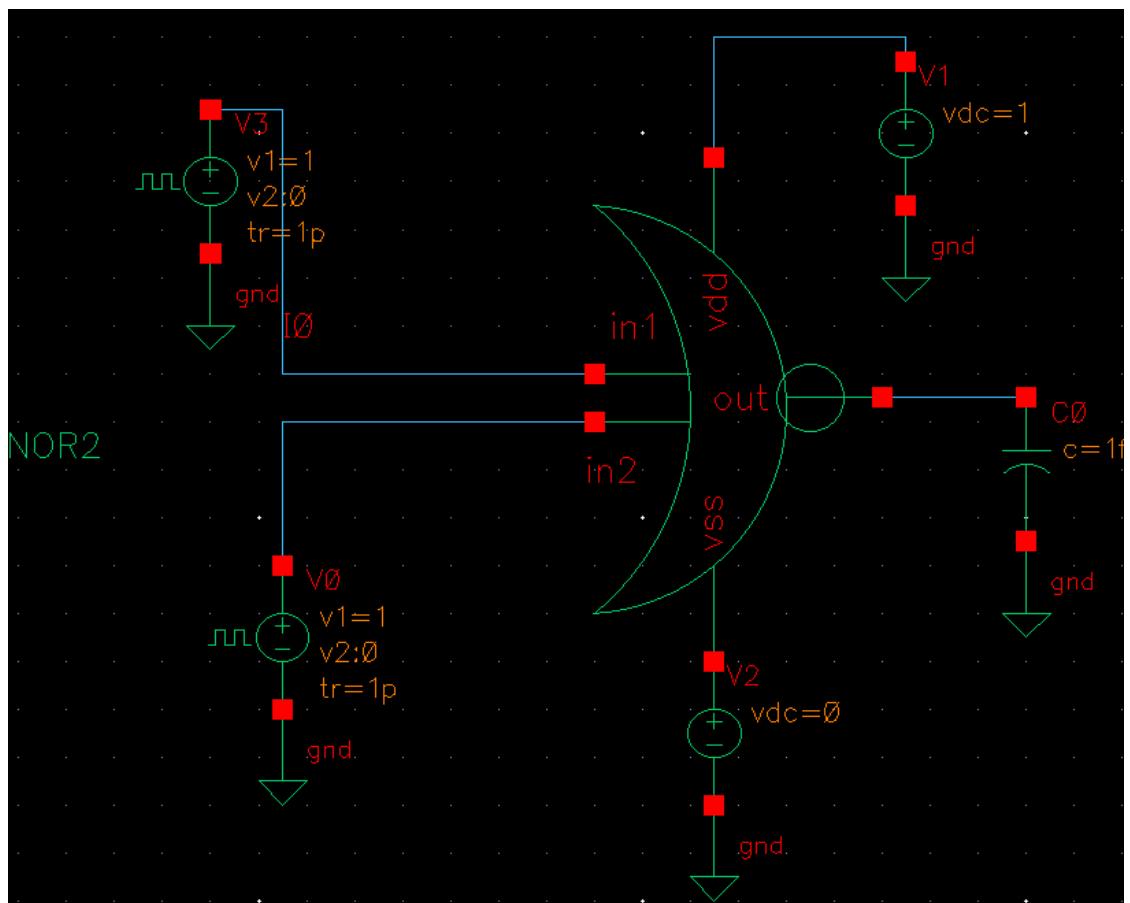


Hinh 2-6 NOR2 Schematic.

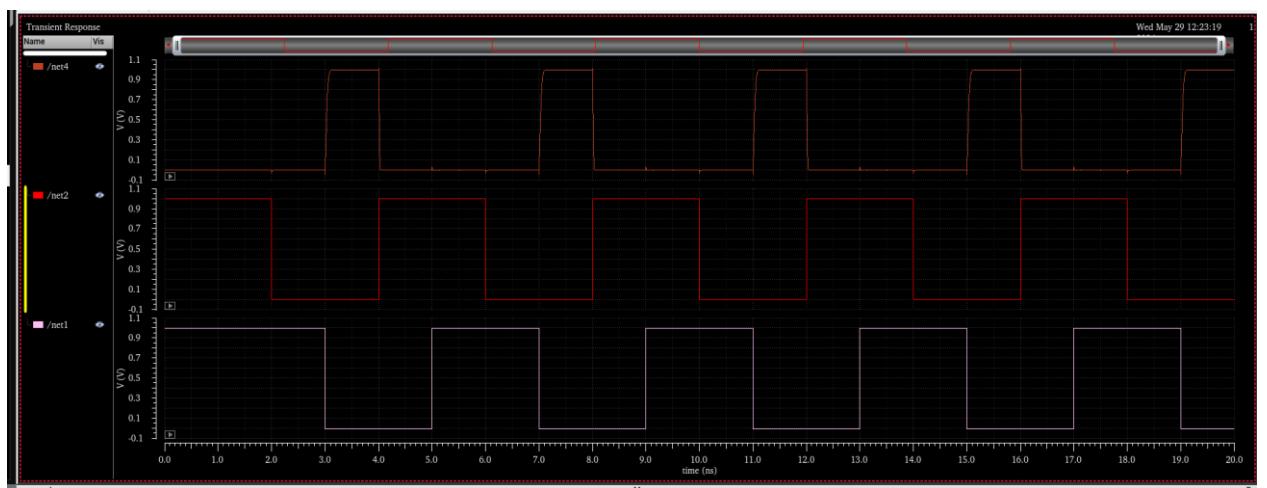
Symbol of NOR gate



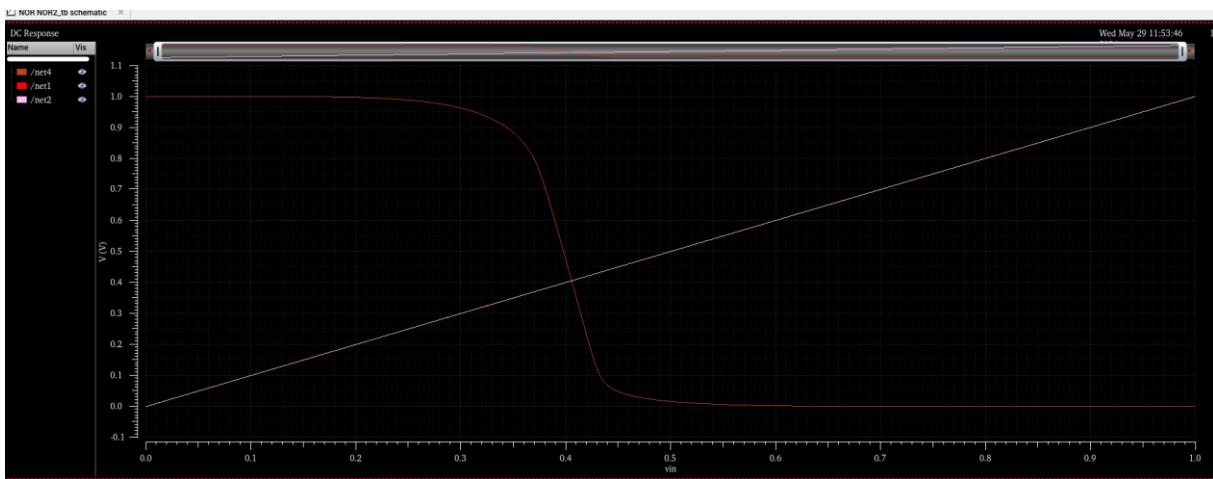
Hình 2-7 NOR2 Symbol.



Hình 2-8 NOR2 test bench.



Hình 2-9 NOR2 Transient simulation.



Hình 2-10 NOR2 Analysis.

Timing measurements:

Parameters	Results
$t_{rise}$	0.0703(ns)
$t_{fall}$	0.042(ns)
$t_{pdr}$	0.0316(ns)
$t_{pdf}$	0.0225(ns)
$t_{pd}$	0.05221(ns)
Dynamic power	13.232pW
Static power	22.115pW

Output voltage value at various input values:

$V_{in1}(V)$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$V_{in2}(V)$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$V_{out}$	997mV	975.3mV	855.2mV	204.1mV	21mV	8.723mV	1.701mV	121.5uV	13.1uV	1.2uV

-Khi  $V_{in}$  chạy từ 0V-0.3V (2 ngõ vào đang ở mức 0), qua công NOR2 thì ngõ ra ở mức 1 tương đương  $V_{out}$  ngược pha có giá trị xấp xỉ 1V.

-Khi  $V_{in}$  chạy từ 0.4V-1V (2 ngõ vào tiệm cận mức 1) qua công NOR2 thì ngõ ra tiến tới ở mức 0.

→ Hoạt động của mạch: Khi 2 ngõ vào có giá trị điện áp thấp (mức logic 0) thì ngõ ra sau khi qua công NOR2 sẽ có giá trị điện áp cao (mức logic 1). Càng tăng giá trị điện áp 2 ngõ vào thì ngõ ra sẽ tiến dần tới điện áp thấp (mức logic 0). Khi ngõ vào đồng thời ở mức logic 0 thì ngõ ra ở mức logic 1.

→ Thực hiện mô phỏng đáp ứng DC giống với lý thuyết hoạt động.

-Khi  $V_{in}$  chạy từ 0V-0.3V (2 ngõ vào đang ở mức 0), qua công NOR2 thì ngõ ra ở mức 1 tương đương  $V_{out}$  ngược pha có giá trị xấp xỉ 1V.

-Khi  $V_{in}$  chạy từ 0.4V-1V (2 ngõ vào tiệm cận mức 1) qua công NOR2 thì ngõ ra tiến tới ở mức 0.

→ Hoạt động của mạch: Khi 2 ngõ vào có giá trị điện áp thấp (mức logic 0) thì ngõ ra sau khi qua công NOR2 sẽ có giá trị điện áp cao (mức logic 1). Càng tăng giá trị điện áp 2 ngõ vào thì ngõ ra sẽ tiến dần tới điện áp thấp (mức logic 0). Khi ngõ vào đồng thời ở mức logic 0 thì ngõ ra ở mức logic 1.

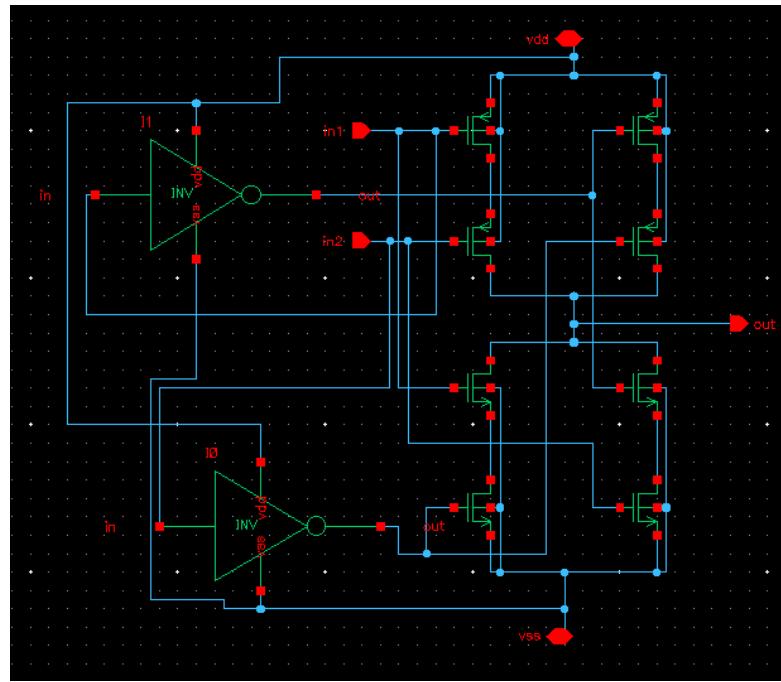
→ Thực hiện mô phỏng đáp ứng DC giống với lý thuyết hoạt động.

### 2.1.3. XNOR

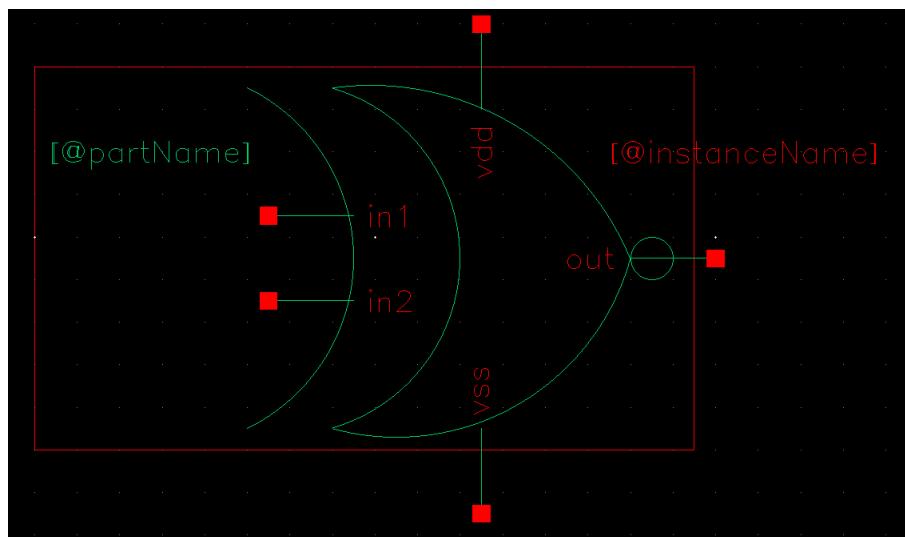
Truth table:

In1	In2	Out
0	0	1
0	1	0
1	0	0
1	1	1

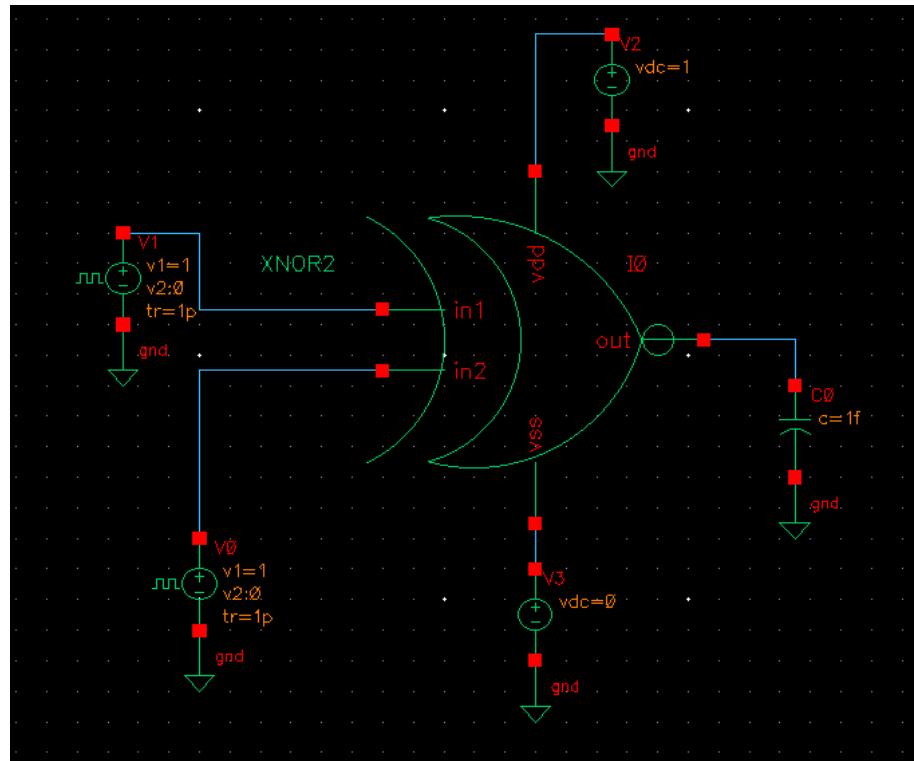
Schematic of XNOR gate



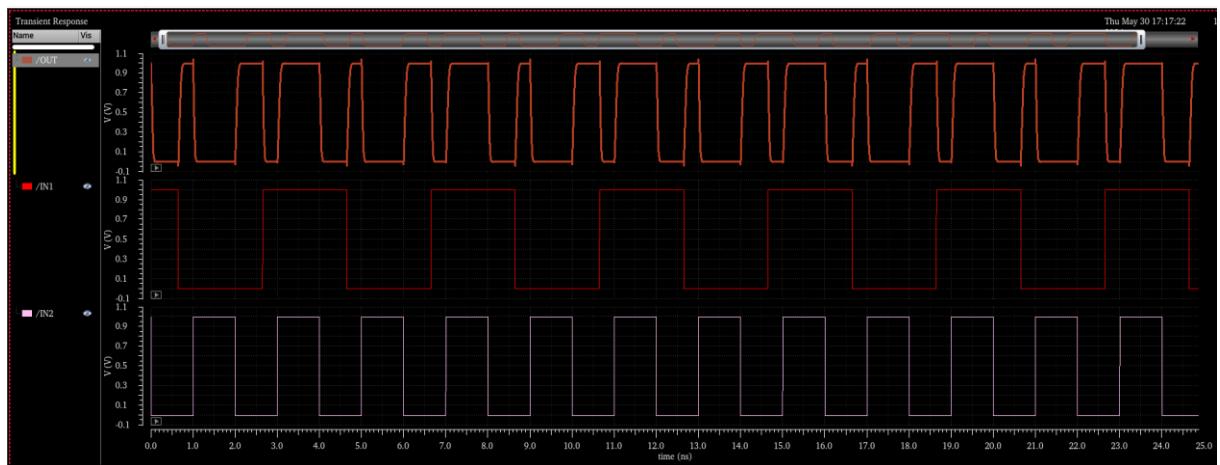
Hình 2-11 XNOR2 Schematic.



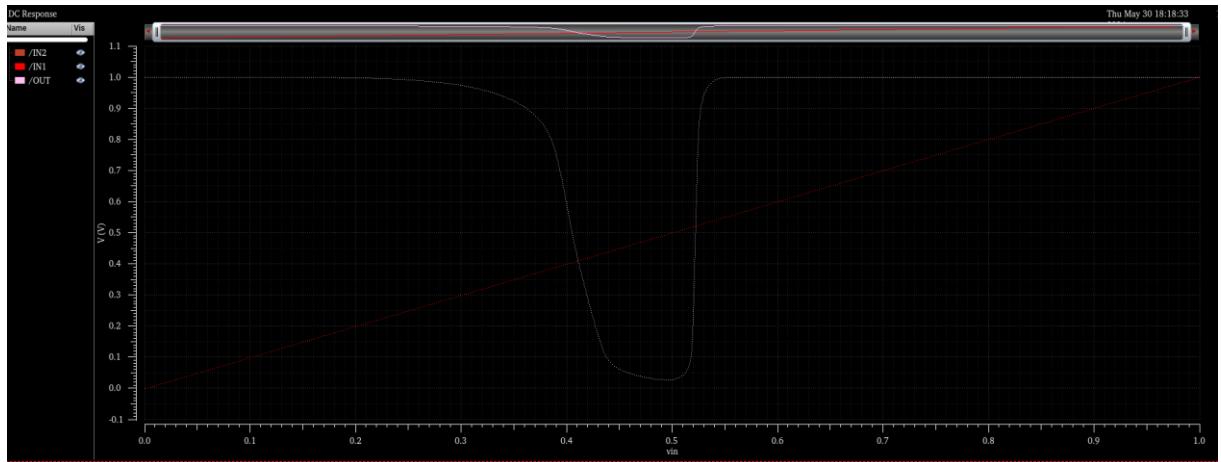
Hình 2-12 XNOR2 Symbol.



Hình 2-13 XNOR2 test bench.



Hình 2-14 XNOR2 Transient simulation.



Hình 2-15 XNOR2 DC analysis

Timing measurements:

Parameters	Results
$t_{rise}$	0.0644(ns)
$t_{fall}$	0.029(ns)
$t_{pdr}$	0.0306(ns)
$t_{pdf}$	0.0219(ns)
$t_{pd}$	0.05312(ns)
Dynamic power	14.521pW
Static power	23.256pW

Output voltage value at various input values:

V <sub>in1</sub> (V )	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
V <sub>in2</sub> (V )	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
V <sub>out</sub>	999m V	996.3m V	971.2m V	398.2m V	29m V	999m V	999m V	999 V	999 V	999 V

❖ Nhân xét: về dạng sóng  $V_{out}$  khi hoạt động chế độ DC

-Khi  $V_{in}$  chạy từ 0V-0.3V (2 ngõ vào đang ở mức 1), qua cổng XNOR2 thì ngõ ra ở mức 1 tương đương  $V_{out}$  ngược pha có giá trị xấp xỉ 0V.

-Khi  $V_{in}$  chạy từ 0.3V-1V (2 ngõ vào tiệm cận mức 1) qua công XNOR2 thì ngõ ra tiến tới ở mức 1.

-Tại  $V_{in1} = V_{in2} = 1V$  thì ngõ ra  $V_{out} = 0V$ .

→ Hoạt động của mạch: Khi 2 ngõ vào có giá trị điện áp thấp (mức logic 1) thì ngõ ra sau khi qua công XNOR2 sẽ có giá trị điện áp cao (mức logic 1). Càng tăng giá trị điện áp 2 ngõ vào thì ngõ ra sẽ tiến dần tới điện áp thấp ở khoảng 0.4 rồi cao. Khi 2 giá trị ngõ vào đồng thời ở mức logic 1 thì ngõ ra sẽ ở mức logic 1.

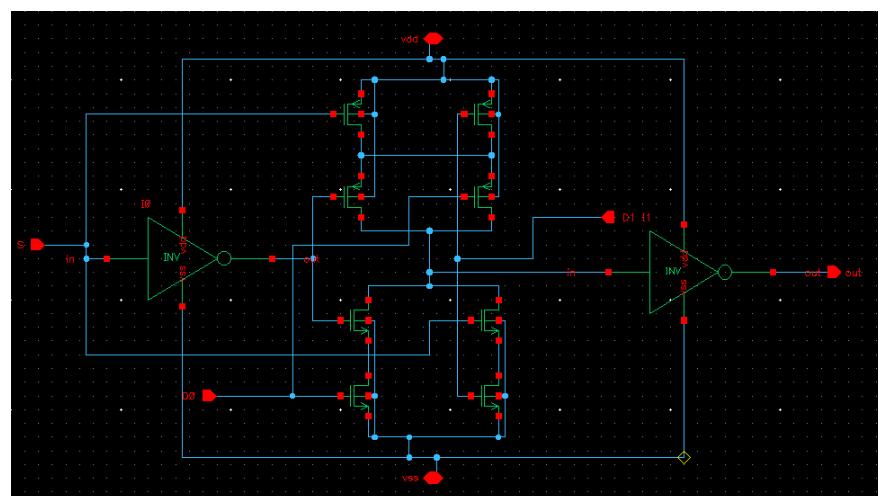
→ Thực hiện mô phỏng đáp ứng DC giống với lý thuyết hoạt động.

## 2.2. Experiment 2

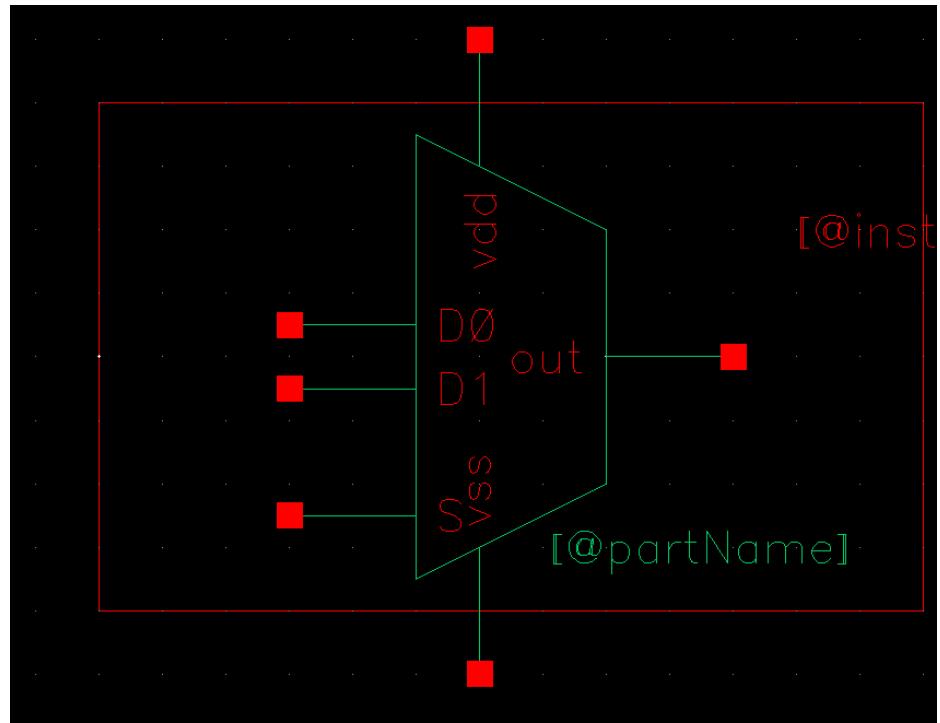
Truth table:

S	D0	D1	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

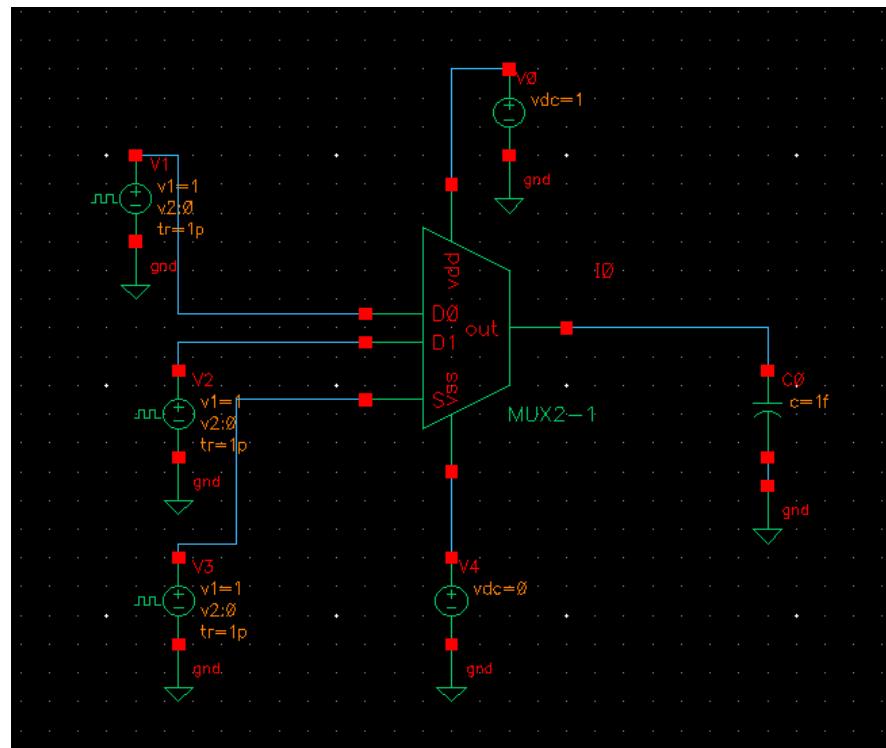
Schematic of MUX 2-1 using compound gate



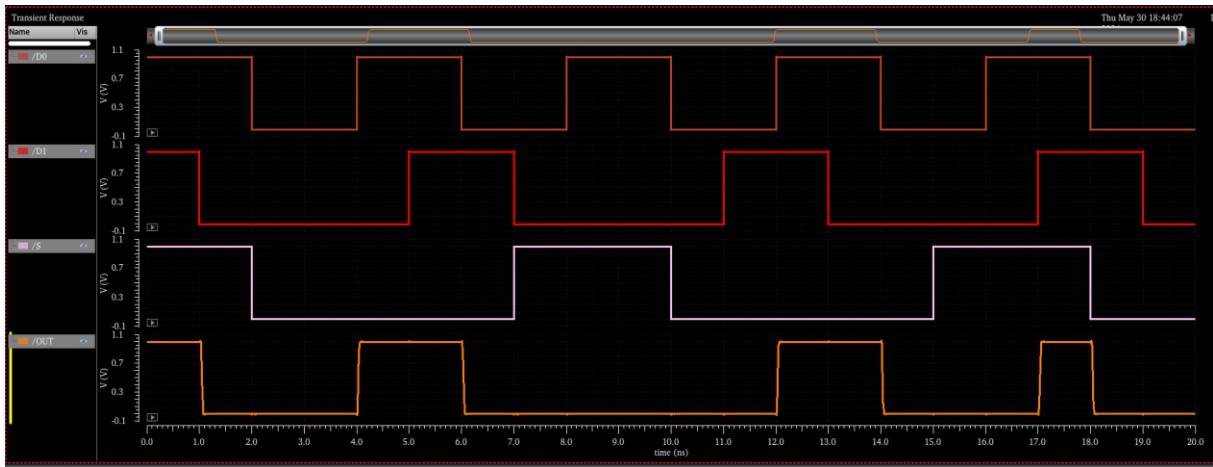
Hình 2-16 MUX 2-1 Schematic



Hinh 2-17 Symbol of MUX 2-1



Hinh 2-18 MUX 2-1 test bench



Hình 2-19 MUX 2-1 Transient simulation

Timing measurements:

Parameters	Results
$t_{rise}$	0.025(ns)
$t_{fall}$	0.029(ns)
$t_{pdr}$	0.42(ns)
$t_{pdf}$	0.28(ns)
$t_{pd}$	0.35(ns)
Power consumption	13.221pW

➔ Mạch hoạt động đúng với yêu cầu đề ra giống truth table

### 2.3. Experiment 3

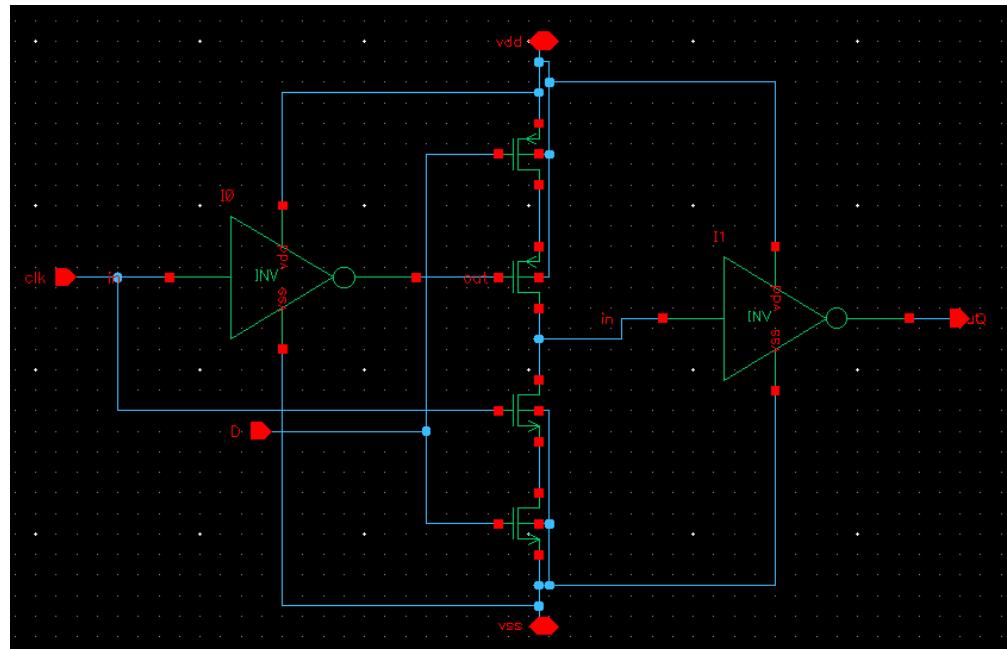
Truth table of D Latch:

<b>CLK</b>	<b>D</b>	<b>Q</b>	<b>Q'</b>
0	X	Không thay đổi	Không thay đổi
1	0	0	1
1	1	1	0

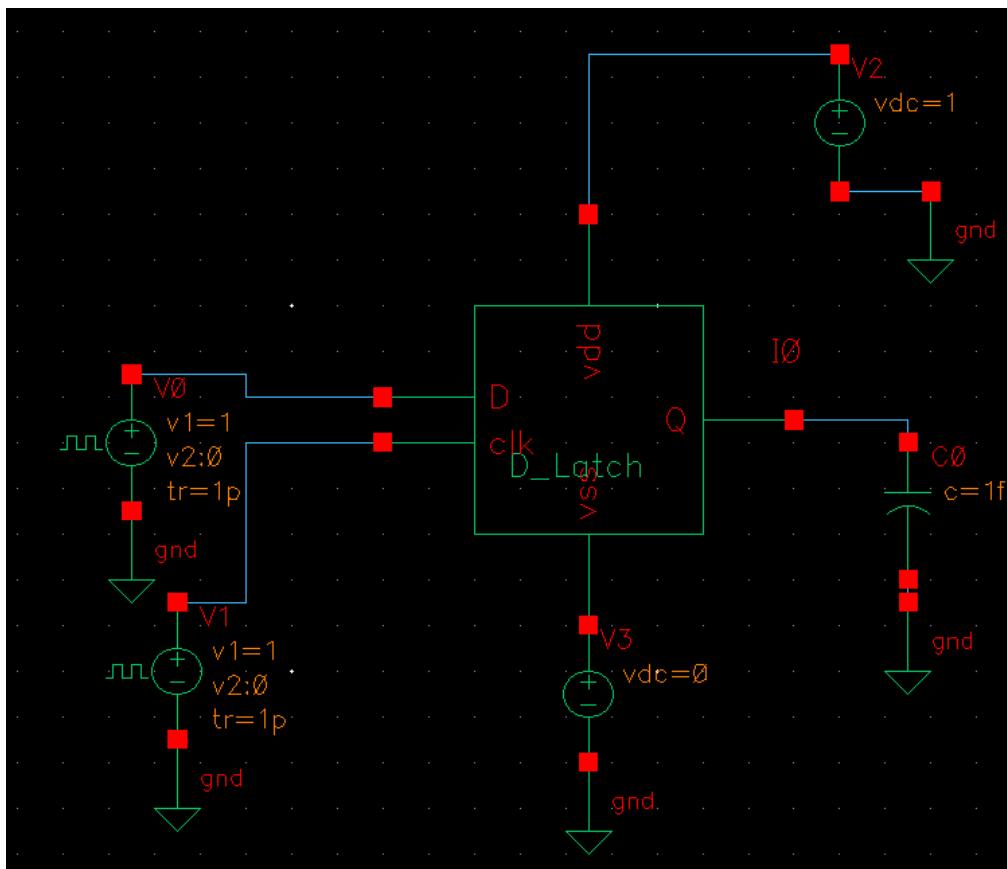
Truth table of DFFPOS :

<b>CLK</b>	<b>D</b>	<b>Q</b>	<b>Q'</b>
0, 1, ↓	X	Không thay đổi	Không thay đổi
0, 1, ↓	X	Không thay đổi	Không thay đổi
↑	0	0	1
↑	1	1	0

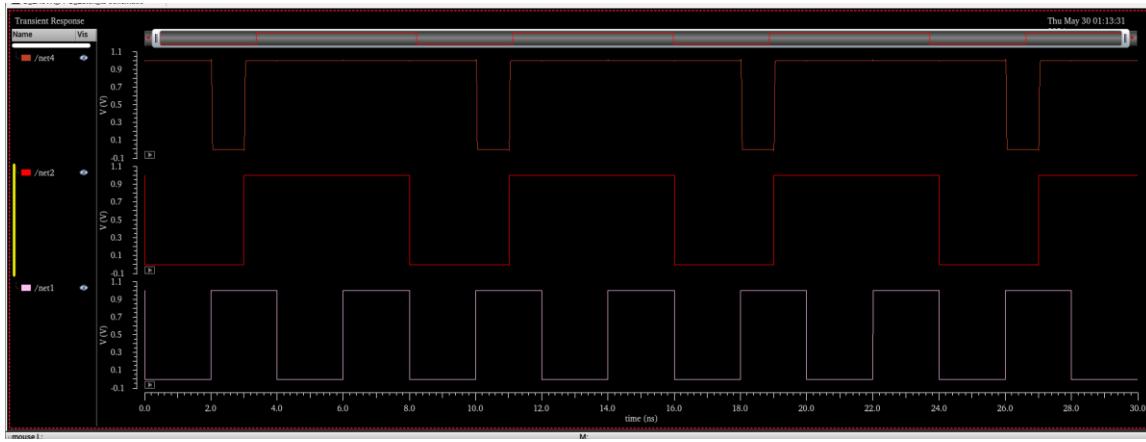
Schematic of D latch:



Hình 2-20 D latch schematic



Hình 2-21 D latch testbench

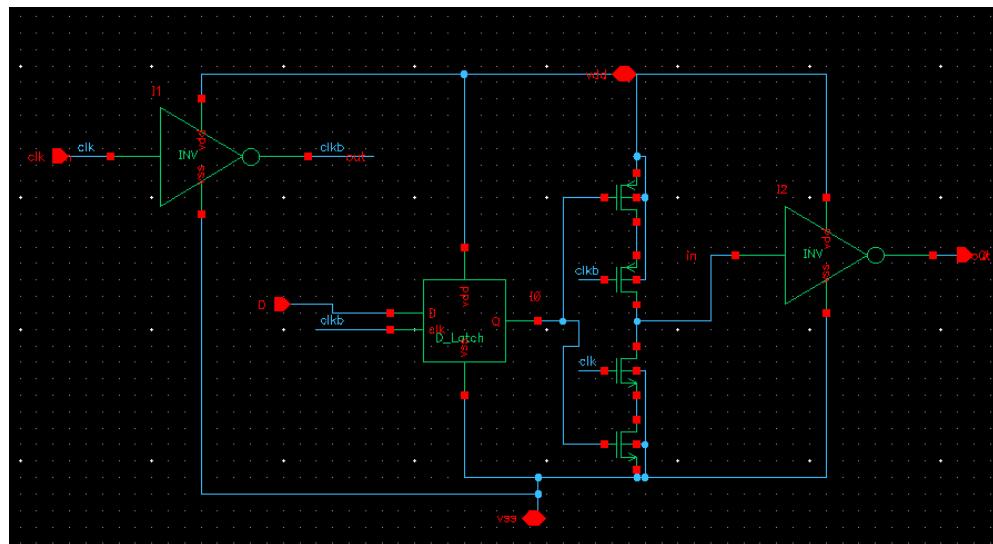


Hình 2-22 D latch Transient simulation.

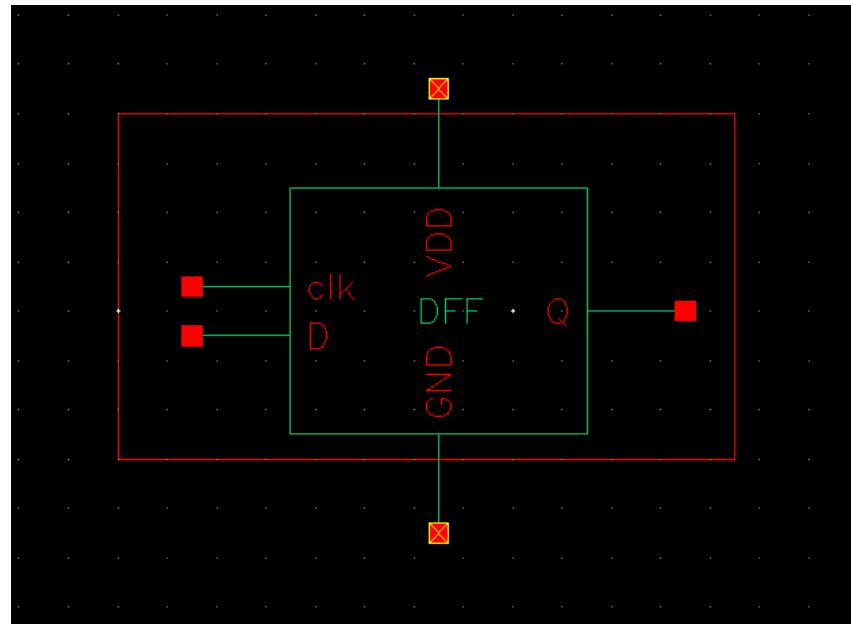
Ta thấy ngõ ra Q sẽ thay đổi phụ thuộc vào ngõ vào D chỉ khi clk được kích ở mức cao, ở mức thấp thì sẽ không thay đổi ngõ ra.

Schematic of DFFPOS từ D latch.

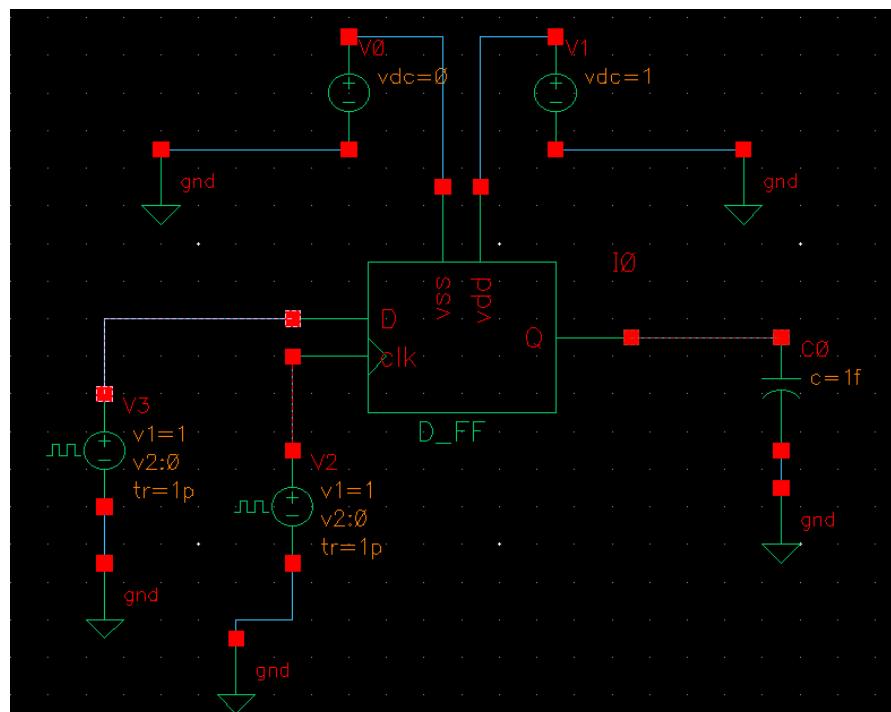
Trạng thái của ngõ ra Q sẽ thay đổi theo ngõ vào D mỗi khi chân ngõ vào clk là xung cạnh lên ( 0-1 ).



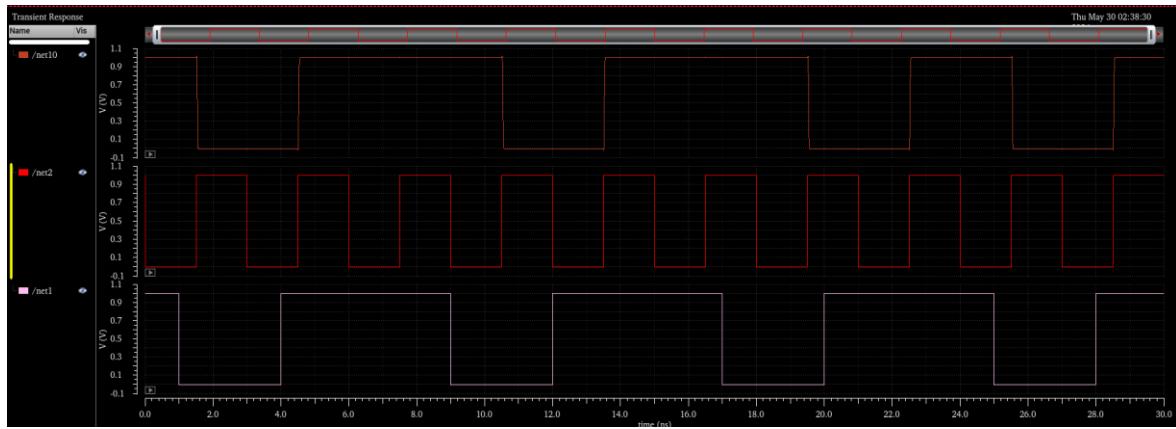
Hình 2-23 DFFPOS schematic



Hình 2-24 DFF symbol



Hình 2-25 DFFPOS test bench



Hình 2-26 DFF Transient simulation.

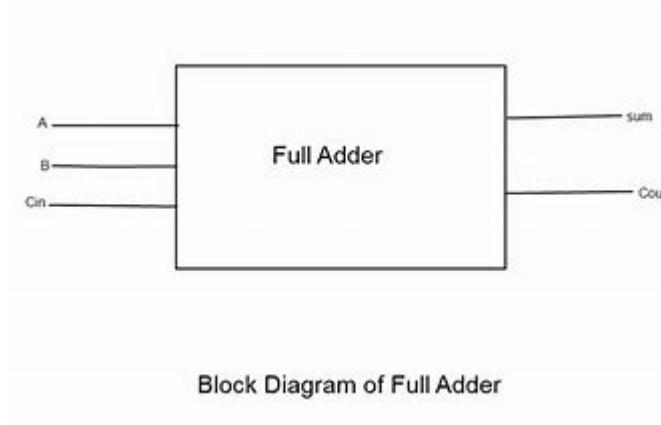
Ta thấy ngõ ra lúc này thay đổi mỗi khi có xung cạnh lên, thỏa mãn với yêu cầu đề bài.

Parameters	Results
$t_{pd}$	0.0644(ns)
$t_{cd}$	0.029(ns)
$t_{peq}$	0.0306(ns)
$t_{ccq}$	0.0219(ns)
$t_{setup}$	0.05312(ns)
$t_{hold}$	14.521pW

## Chương 3: COMBINATIONAL AND SEQUENTIAL CIRCUIT

### 3.1. Experiment 1

Block diagram of 1 bit Full Adder

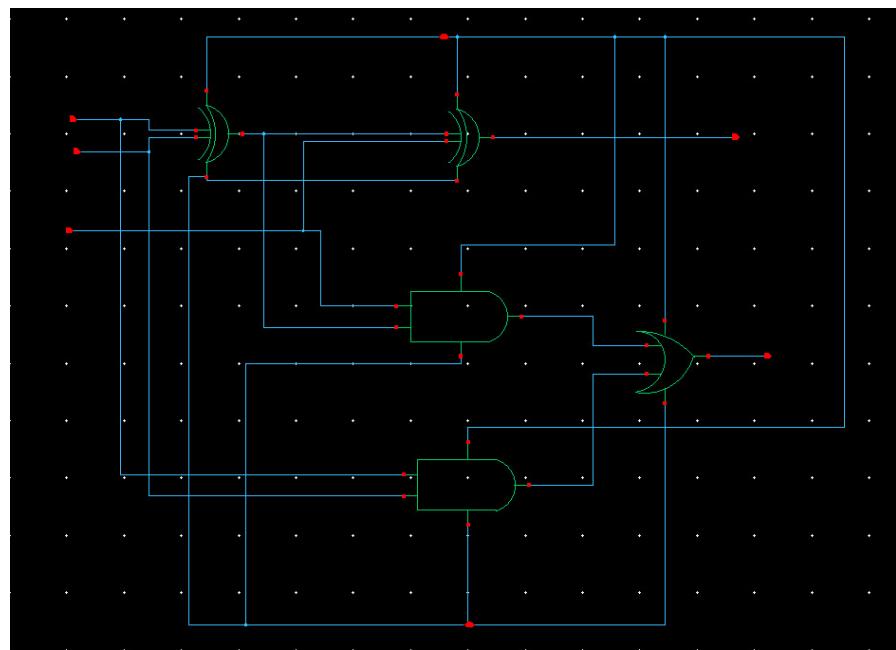


Hình 3-1 1 bit FA block diagram

Truth table:

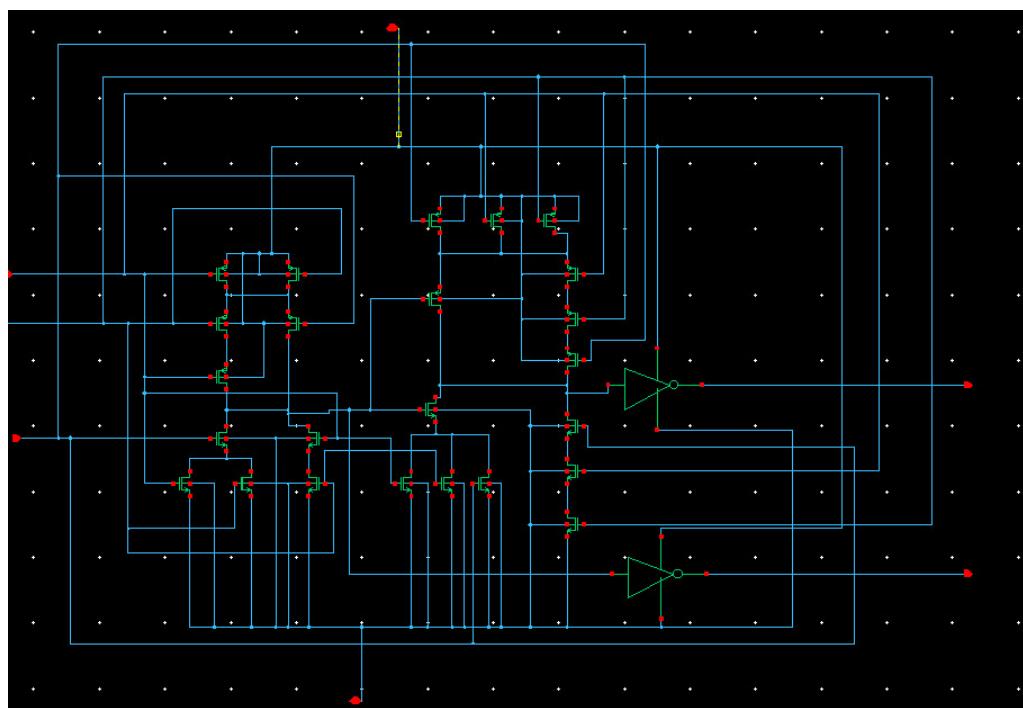
A	B	$C_{in}$	S	$C_{out}$	Carry Status
0	0	0	0	0	Delete
0	0	1	1	0	Delete
0	1	0	1	0	Propagate
0	1	1	0	1	Propagate
1	0	0	1	0	Propagate
1	0	1	0	1	Propagate
1	1	0	0	1	Generate
1	1	1	1	1	Generate

Logic circuit of 1 bit FA



Hình 3-2 Logic circuit.

Schematic of 28T FA



Hình 3-3 Schematic of 28T FA.

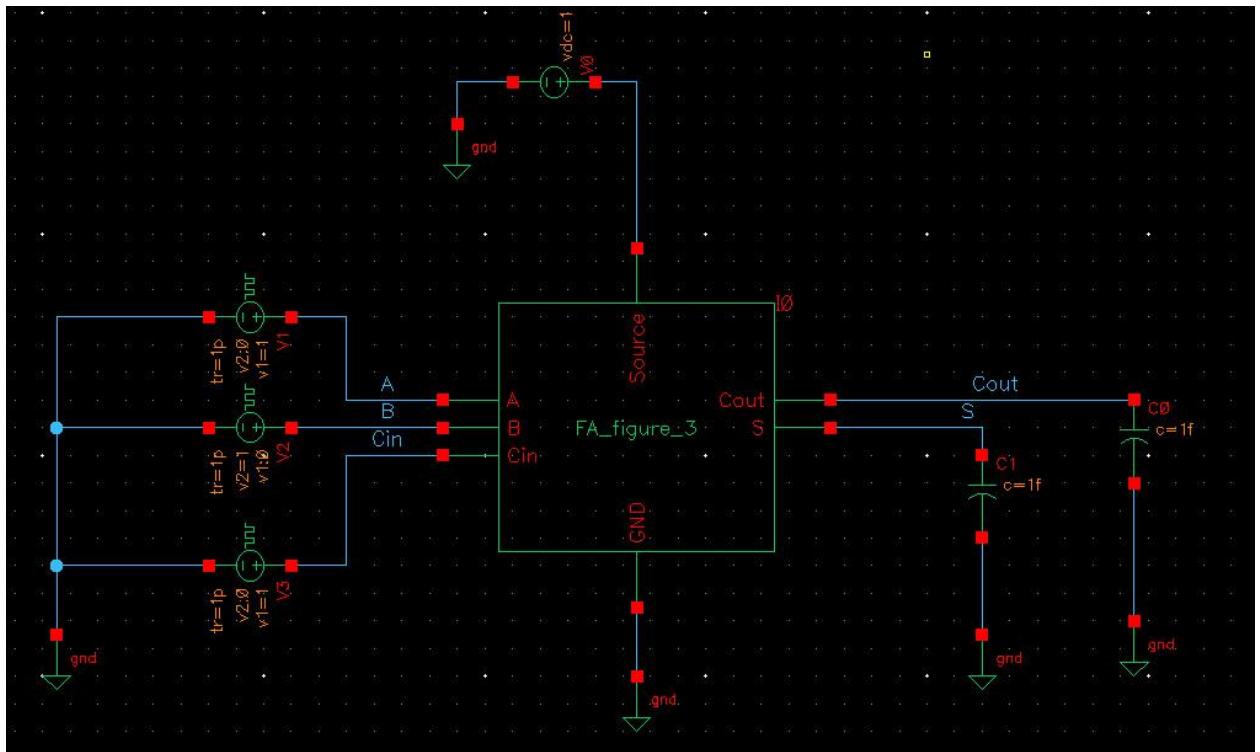
Kiểm tra:

- Dạng sóng mô phỏng:

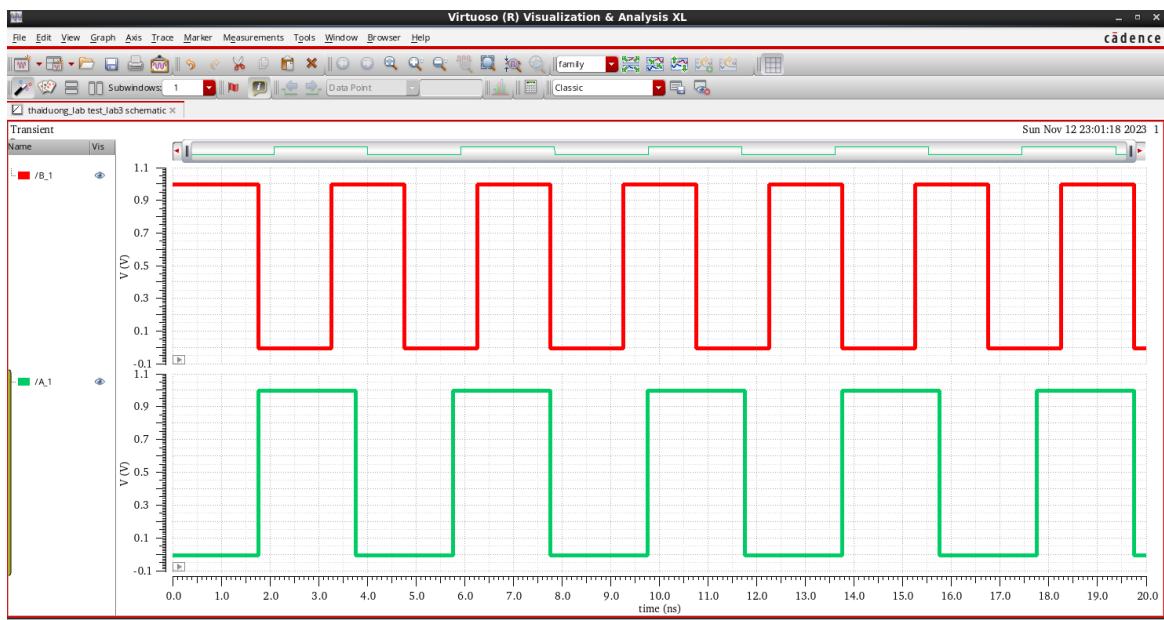
- Theo như bảng chân trị, nhóm có thể chia ra làm hai trường hợp để thực hiện mô phỏng là  $C_{in} = 0$  và  $C_{in} = 1$ . Đầu tiên, ta cài đặt dạng sóng ngõ vào dựa theo bảng số liệu sau đây

Linh kiện	Tham số	Giá trị
vdc	DC voltage	1
cap	Capacitance	1f F
Vpulse	Voltage 1	$V_A = 0, V_B = 1$
	Voltage 2	$V_A = 1, V_B = 0$
	Period	A: 4ns, B: 3ns
	Delay time	1.75ns
	Rise time	1ps
	Fall time	1ps
	Pulse width	A: 2ns, B: 1.5ns

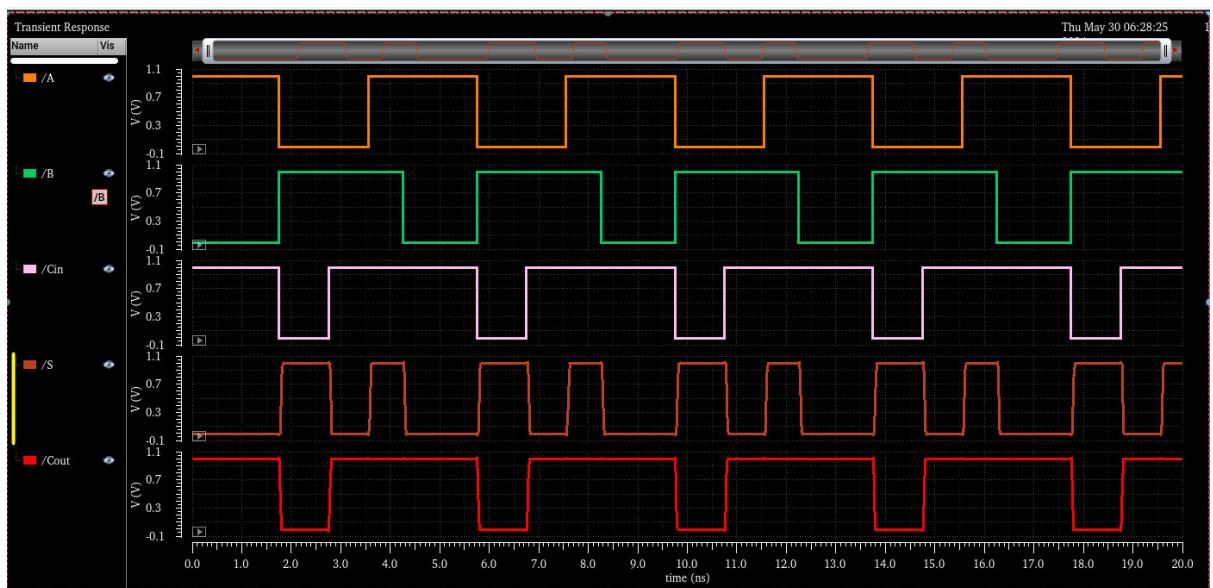
Test bench



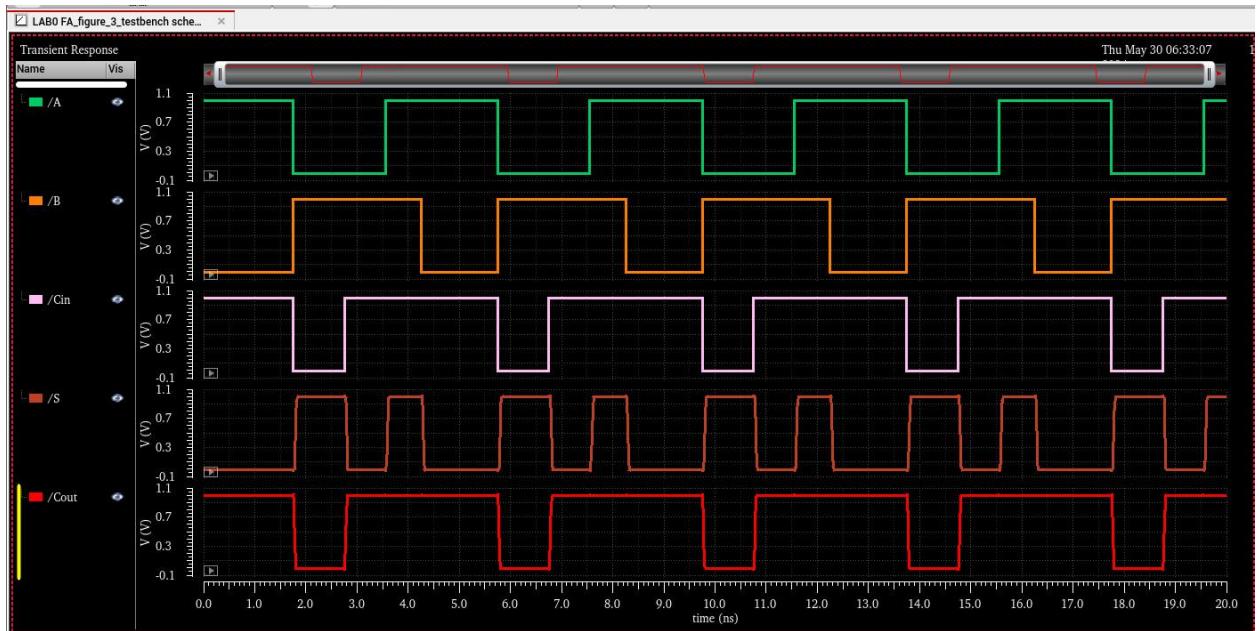
Hình 3-4 Test bench 28T FA.



Hình 3-5 Dạng sóng mô phỏng ngõ vào  $V_A$  và  $V_B$



Hình 3-6 Dạng sóng mô phỏng của trường hợp dùng 28T



Hình 3-7 Dạng sóng mô phỏng của trường hợp logic gate

**Nhận xét:** khi ngõ vào được thiết lập các thông số như vậy thì ta có thể quan sát được các trường hợp của dạng sóng tổng hợp của ngõ ra.

Nhìn chung cả 2 mạch đều chạy đúng với logic của bảng chân trị. Tuy nhiên do sử dụng các phương pháp khác nhau để vẽ mạch nên dạng sóng mô phỏng nên nhiều ảnh hưởng vào cũng khác nhau vậy nên ta có thể thấy ở dạng sóng ngõ ra của 2 phương pháp có những chỗ nhiễu không giống nhau.

Đo giá trị các thông số thời gian và công suất. Đồng thời, trình bày ngắn gọn các cách tính hoặc đo đặc các thông số này.

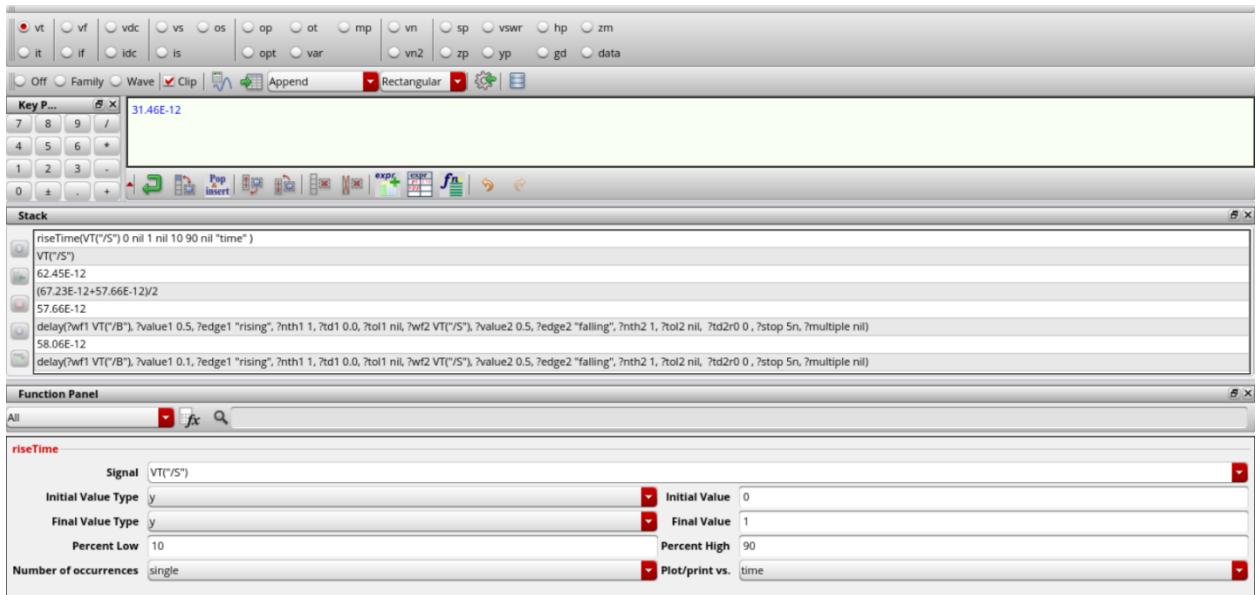
	Original FA	FA 28T
$t_{rise}$ – Rising Time (10% - 90%)	31,46ps	8,439ps
$t_{fall}$ – Falling Time (90% - 10%)	20,55ps	9,344ps
$t_{pdr}$ – Rising propagation delay (90% - 50%)	67.63ps	53,27ps
$t_{pdf}$ – Falling propagation delay (10% - 50%)	58,06ps	49,31ps
$t_{pd}$ – Average propagation delay (50% - 50%)	62,445ps	50,89ps
Power consumption	84ns	21,73ns

### **Định nghĩa:**

- *Rising time*: là thời gian mà một tín hiệu cần để thay đổi từ một giá trị mức thấp xác định đến một giá trị mức cao xác định. Rise time được định nghĩa là thời gian mà tín hiệu cần để chuyển từ 10% đến 90% của tín hiệu ngõ ra.
- *Falling time*: là thời gian mà một tín hiệu cần để giảm từ một giá trị mức cao đến một giá trị mức thấp. Falling time được định nghĩa là thời gian mà tín hiệu cần để chuyển từ 90% đến 10% của tín hiệu ngõ ra.
- *Rising propagation delay*: là thời gian mà tín hiệu cần để tăng từ một giá trị thấp lên một giá trị cao. Nó được đo từ điểm 90% của sóng tín hiệu đầu vào đến điểm 50% của sóng tín hiệu đầu ra.
- *Falling propagation delay*: là thời gian mà tín hiệu cần để giảm từ một giá trị cao xuống một giá trị thấp. Nó được đo từ điểm 10% của sóng tín hiệu đầu vào đến điểm 50% của sóng tín hiệu đầu ra.
- *Average propagation delay*: Là thời gian cần thiết để tín hiệu truyền tải. Nó được tính bằng trung bình của tổng thời gian Tpdr và Tpdf.
- *Power consumption*: bao gồm công suất tĩnh và công suất động (Pstatic + Pdynamic).

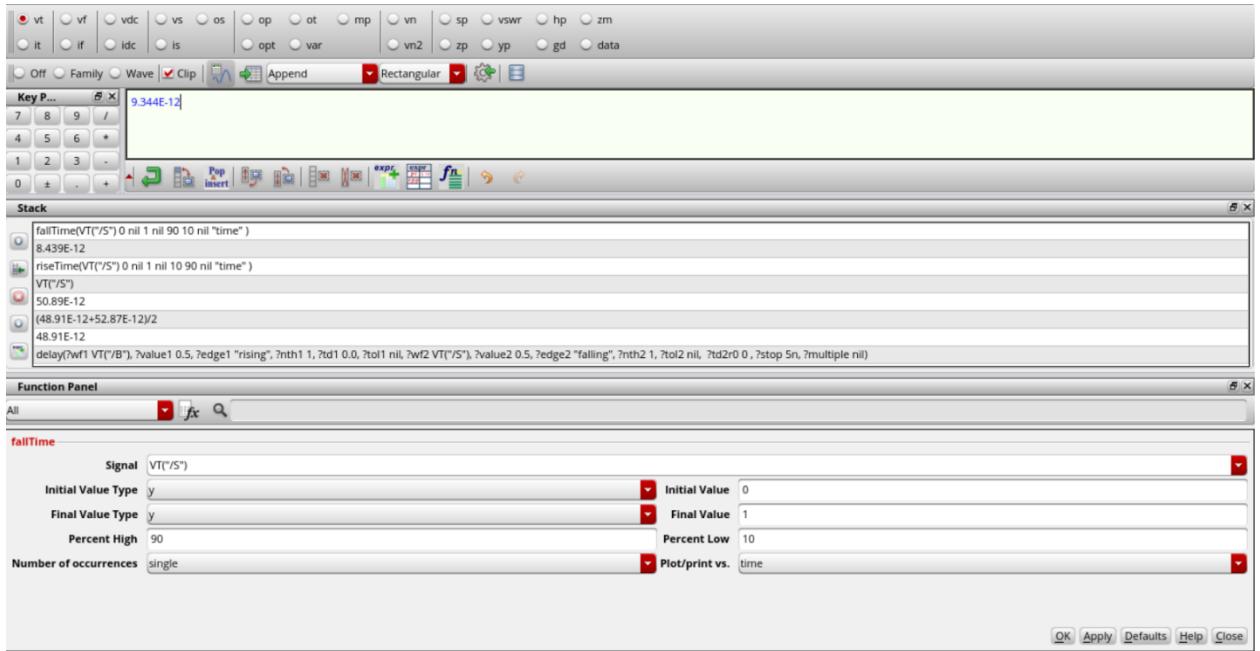
**Cách tính và đo đạc:** Trong phạm vi thí nghiệm 3 này, nhóm sẽ thực hiện việc đo đạc các thông số trên bằng tính năng Virtuoso Visualization & Analysis XL Calculator để đảm bảo độ chính xác của số liệu.

- *Rising time*: sử dụng công cụ Calculator, Signal là ngõ ra của mạch, initial value = 0, final value = 1, percent low = 10, percent high = 90. Đây là giá trị được xét chung cho cả ba bộ Original FA, FA 28T

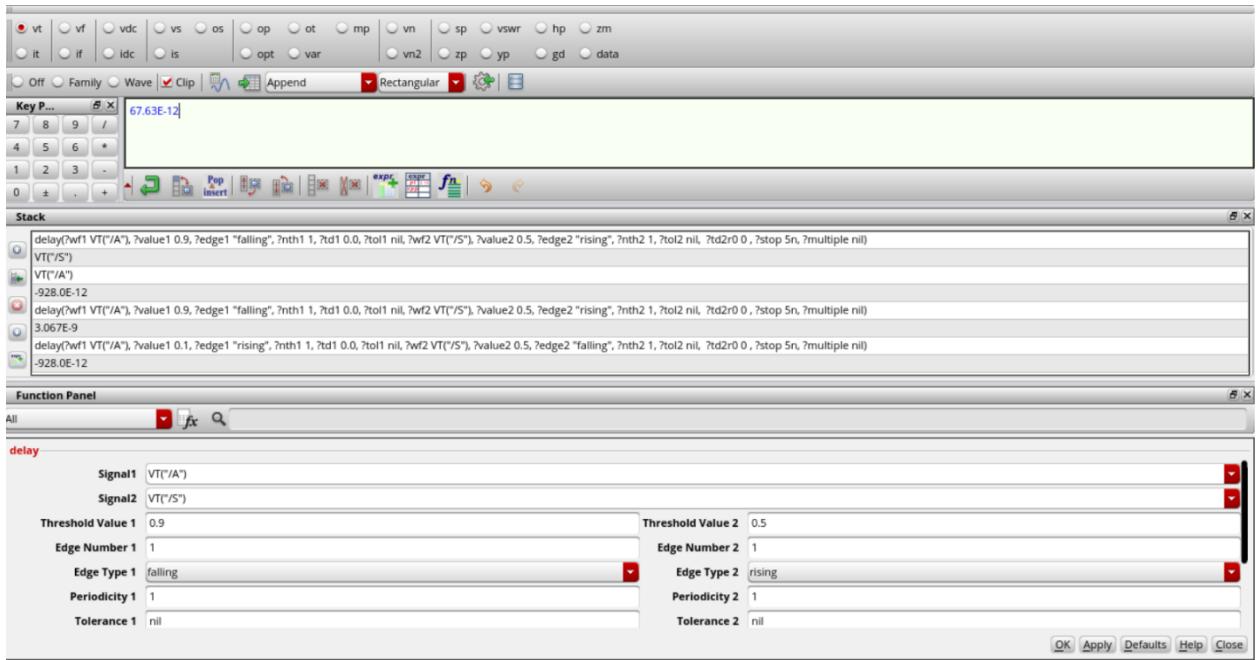


Hình 3-8 Rise time của original FA

- *Falling time*: sử dụng công cụ Calculator, Signal là ngõ ra của mạch, initial value = 0, final value = 1, percent low = 10, percent high = 90. Đây là giá trị được xét chung cho cả ba bộ Original FA, FA 28T

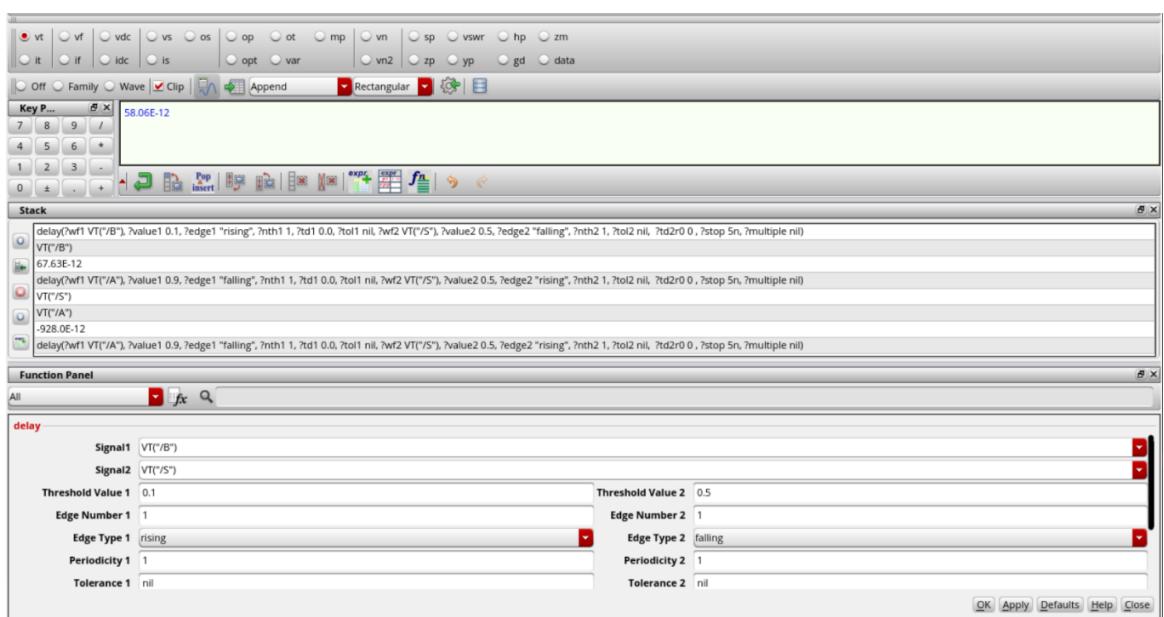


Hình 3-9 Fall time của FA28T



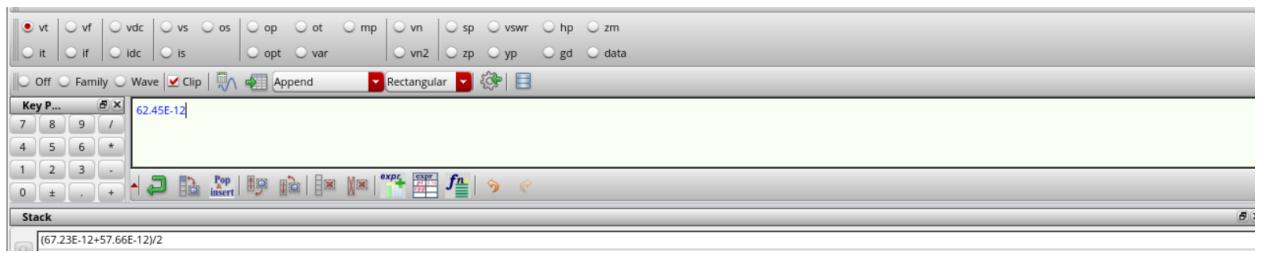
Hình 3-10  $t_{pdr}$  Original FA

- *Rising propagation delay*: sử dụng công cụ Calculator, Signal1: ngõ vào, Signal2: ngõ ra; threshold value 1: 0,9 ; threshold value 2: 0,5; edge type 1: falling ; edge type 2: rising; start1: 0; stop: 5ns
- *Falling propagation delay*: sử dụng công cụ Calculator, Signal1: ngõ vào, Signal2: ngõ ra; threshold value 1: 0,1 ; threshold value 2: 0,5; edge type 1: falling ; edge type 2: rising; start1: 0; stop: 5ns



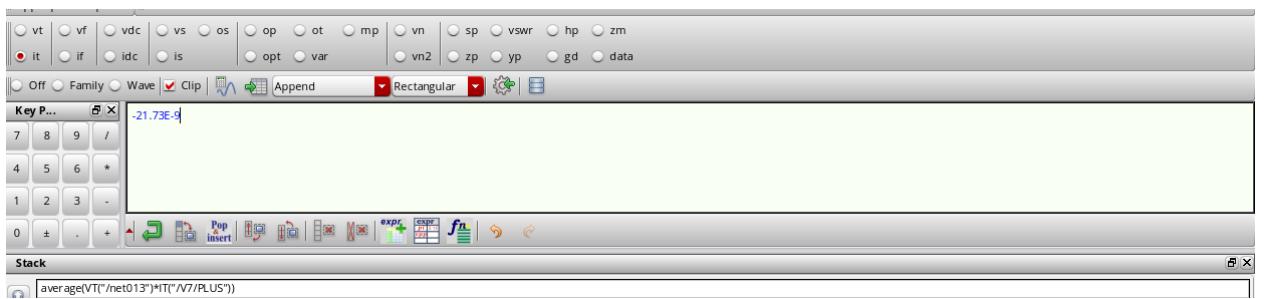
Hình 3-11  $t_{pdf}$  original FA

- Average propagation delay: lấy  $(tpdf + tpdr)/2$



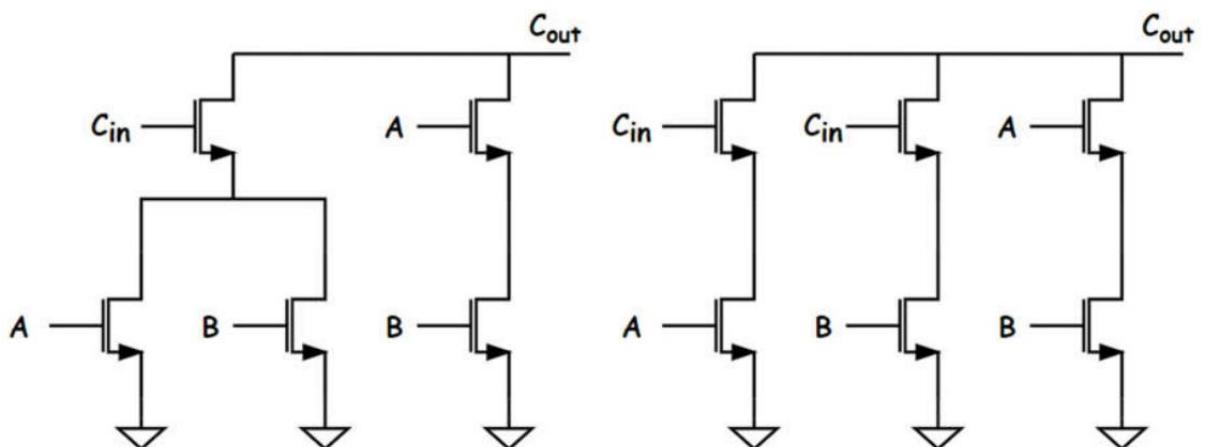
Hình 3-12  $t_{pd}$  Original FA

- Power consumption: để tính power consumption, ta nhân điện áp ở đường dây cấp nguồn và dòng điện tại nguồn như hình bên dưới (màu trắng là điểm lấy dòng it, màu đỏ lấy điện áp vt). Sử dụng công cụ Calculator, chọn function: average để tính power.



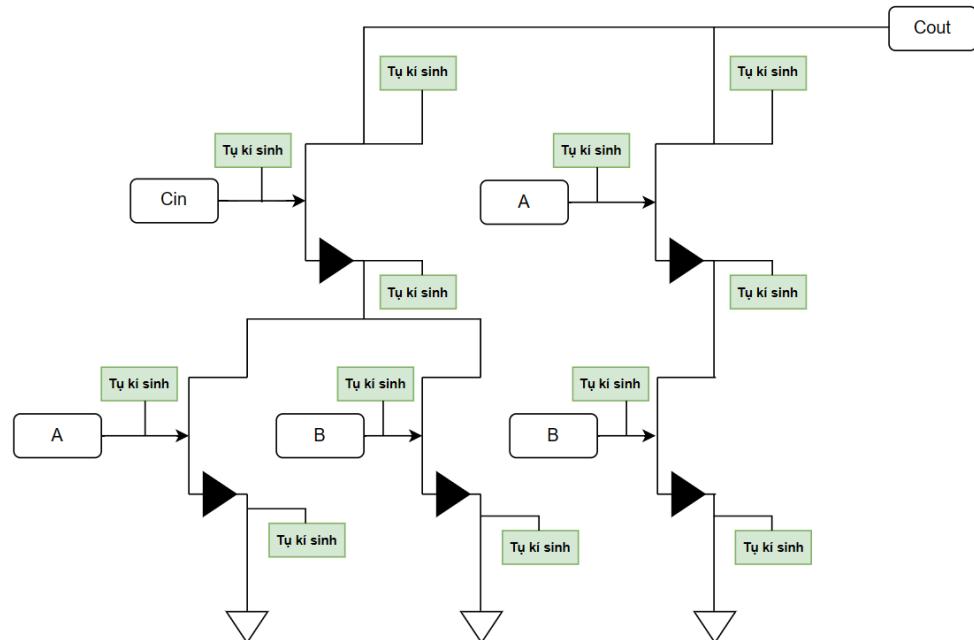
### Câu hỏi:

Trong mạch thiết kế bộ Full adder với 28 transistor, tại sao phải thực thi phần đầu tiên bằng công thức  $Cout = (A+B).Cin + AB$  thay vì  $Cout = AB + Acin + Bcin$  ?

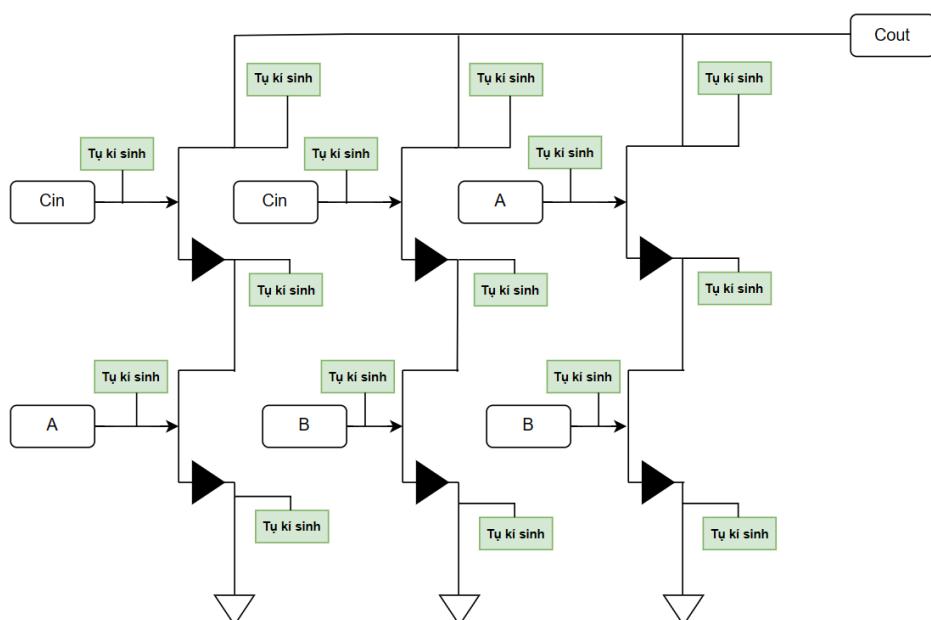


Hình 3-13 PDN.

**Trả lời:** Khi thực hiện mạch thiết kế A (mạch bên trái) thay vì mạch thiết kế B (mạch bên phải) thì giảm được 1 NMOS, nếu trong mạch sử dụng càng nhiều bộ Full adder, chúng ta sẽ giảm được diện tích mạch logic đáng kể. Ngoài ra, Mạch B tăng 1 NMOS nên làm tăng 3 tụ kí sinh ở 3 cực NMOS, điều này giúp mạch A cũng giúp giảm số lượng tụ điện kí sinh, từ đó giảm thời gian delay của mạch, được mô tả như hình sau:



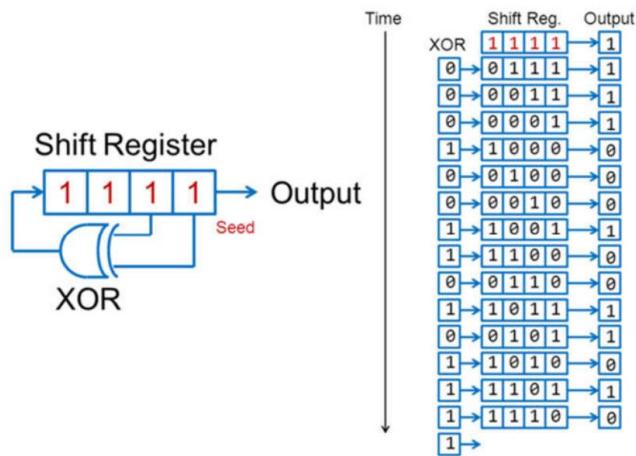
Hình 3-14 Minh họa số lượng tụ điện kí sinh trong mạch A với 12C



Hình 3-15 Hình minh họa số lượng tụ điện kí sinh trong mạch B với 15C

### 3.2 Experiment2

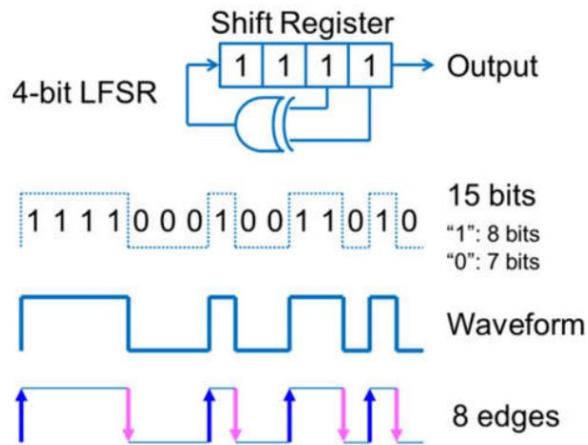
Một ví dụ về cách hoạt động của Linear Feedback Shift Register (LFSR)



Hình 3-16 Ví dụ cách hoạt động của Linear Feedback Shift Register

Cơ chế hoạt động: là một loại thanh ghi dịch có ngõ vào là một hàm tuyến tính của trạng thái trước đó. Khi thanh ghi dịch được lắp đầy bằng một chuỗi bit toàn bit 1, chuỗi bit được dịch sang bên phải 1 bit. Đồng thời, bit thứ 0 và 1 sẽ thực hiện phép XOR để tạo đầu vào cho bit MSB, với 4 bit trong thanh ghi dịch, LFSR sẽ tạo ra chuỗi 15 bit.

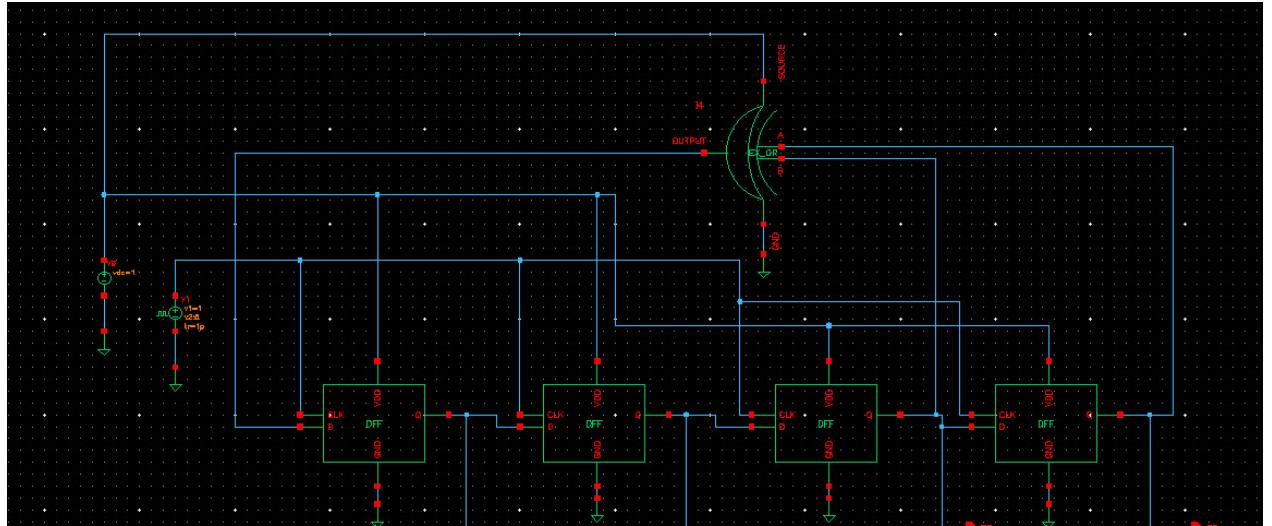
Dạng sóng lý thuyết của chuỗi bit 1111



Hình 3-17 Dạng sóng lý thuyết của chuỗi bit 1111 sau khi tạo ra bởi LFSR

Nhận xét: Dạng sóng có 8 cạnh lên theo chuỗi 15 bit NRZ, bao gồm 8 bit 1's và 7 bit 0's

### Sơ đồ nguyên lý SISO (Schematics)



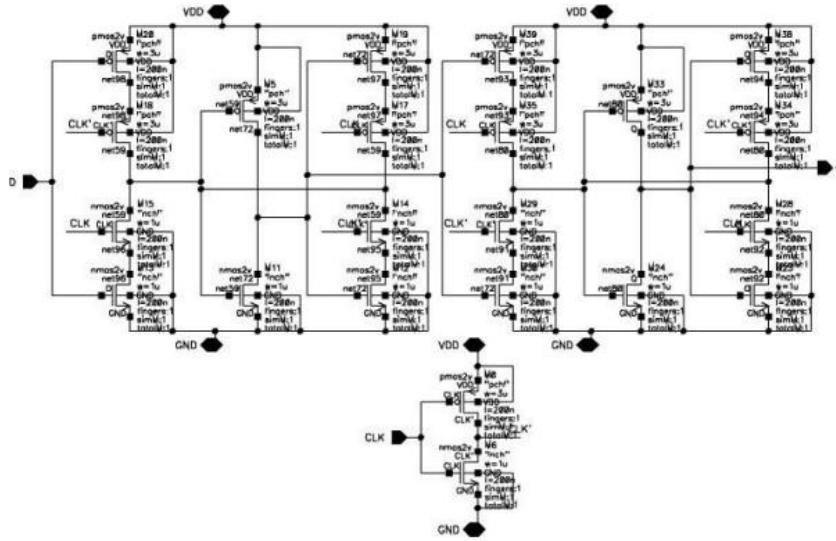
Hình 3-18 Sơ đồ nguyên lý SISO 4-Bit LFSR dùng cổng XOR thực hiện trên Cadence Virtuoso

Giải thích: 4 D flipflop sử dụng chung 1 nguồn xung CLOCK và 1 nguồn VDD = 1V. Ngoài ra, ngõ ra Q của mỗi D flipflop được nối ra chân pin để kiểm tra sự thay đổi của giá trị trong phần tử nhớ. Ngõ ra QD là dạng sóng bit tạo ra từ trạng thái ban đầu.

### Kiểm tra:

Linh kiện	Tham số	Giá trị
vdc	DC voltage	1
Vpulse	Voltage 1	1
	Voltage 2	0
	Period	20ns
	Delay time	1ns
	Rise time	1ps
	Fall time	1ps

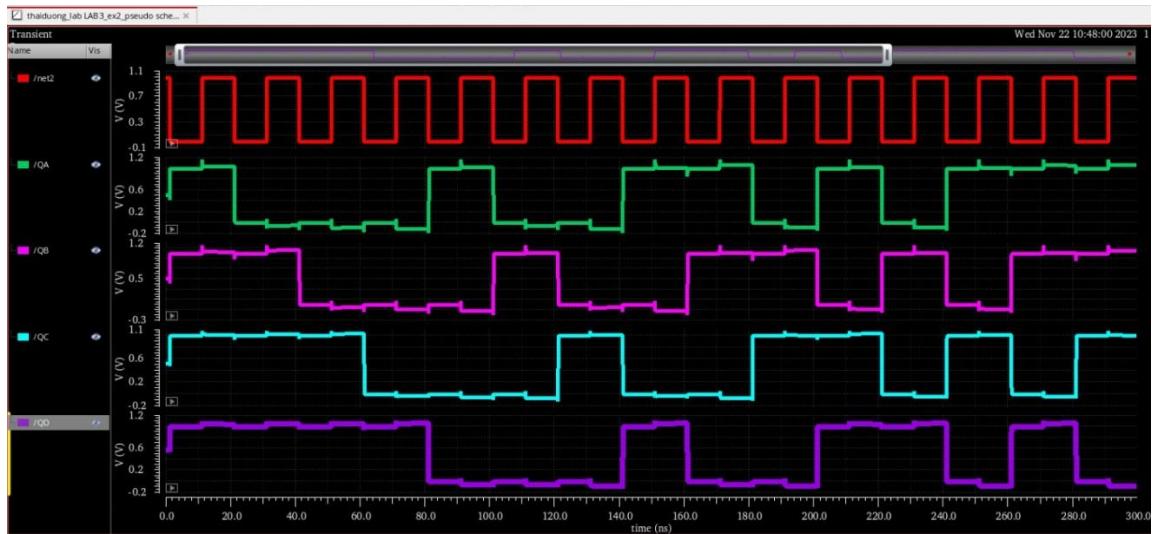
	Pulse width	10ns
--	-------------	------



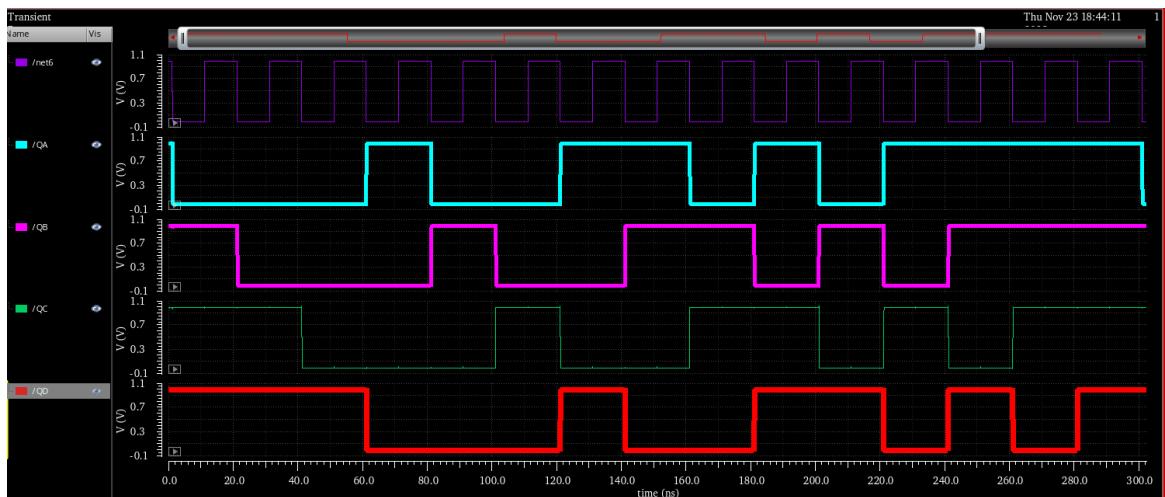
Hình 3-19 Cấu trúc D-type Flip-Flop negative edge triggered trong mạch thiết kế PRBS

Dạng sóng mô phỏng:

Nhóm 1 sẽ tiến hành thực hiện mô phỏng mạch thiết kế PRBS với 2 cấu trúc D flipflop lần lượt là: cấu trúc cạnh xuống dùng C2MOS và cấu trúc cạnh xuống DFFNEG để mô phỏng hoạt động mạch thiết kế PRBS.



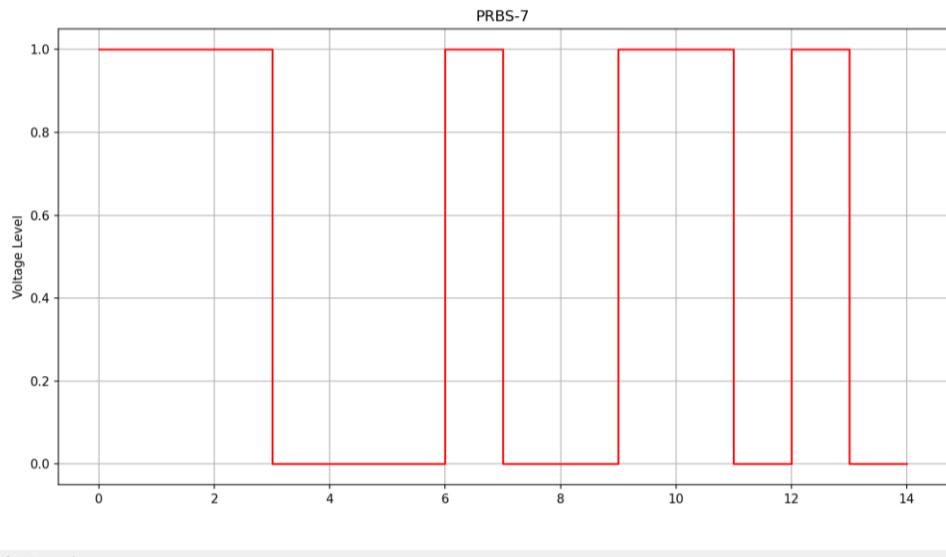
Hình 3-20 Dạng sóng mô phỏng của mạch thiết kế SISO 4-bit LFSR dùng cổng XOR và D flipflop tích cực cạnh xuống



Hình 3-21 Dạng sóng mô phỏng của mạch thiết kế SISO 4-bit LFSR dùng cổng XOR và DFFNEG

Nhận xét: Hoạt động mô phỏng của 2 mạch thiết kế hoàn toàn trùng khớp về mặt chức năng, với ngõ ra tín hiệu QD là chuỗi bit (111100010011010). Tuy nhiên, mạch thiết kế sử dụng cấu trúc C2MOS có nhiều gợn sóng tại các ngõ ra của D flipflop. Trong khi, mạch thiết kế sử dụng cấu trúc TSPC có dạng sóng nhẵn hơn. Điều này có thể được giải thích ở bài thí nghiệm 2 (so sánh mạch thiết kế D flipflop với 3 cấu trúc D flipflop khác nhau).

Dạng sóng của chuỗi bit 1111 thực hiện bằng ngôn ngữ Python



*Hình 3-22* Dạng sóng lý tưởng của chuỗi bit 1111 thực hiện bằng Python

Kết luận: Dạng sóng lý tưởng thực hiện bằng ngôn ngữ Python 3.12.0 hoàn toàn trùng hợp với chuỗi tín hiệu đầu ra (111100010011010) của dạng sóng mô phỏng của 2 mạch PRBS sử dụng 2 cấu trúc D flipflop cạnh lên và cạnh xuống.

## Chương 4: ARITHMETIC LOGIC UNIT & REGISTER FILE

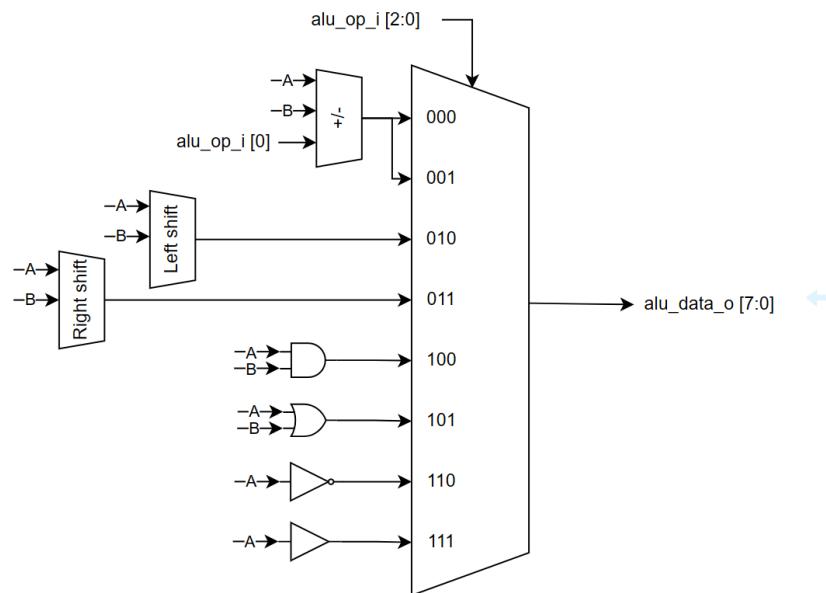
### 4.1 Experiment1

Các toán hạng trong khối ALU, bao gồm: ADD, SUB, SLL, SRL, AND, OR, NOT và FORWARDING

ALU_op_i[2:0]			Function
0	0	0	Add, A+B
0	0	1	Subtract, A-B
0	1	0	Logical shift left
0	1	1	Logical shift right
1	0	0	AND
1	0	1	OR
1	1	0	NOT
1	1	1	Forwarding input, A

Hình 4-1 Các toán hạng trong khối ALU

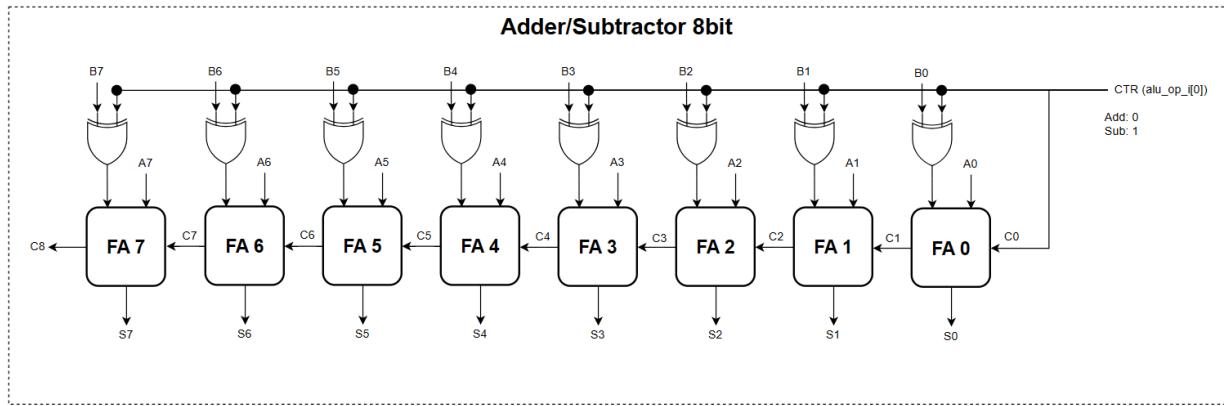
Sơ đồ khối ALU 8-bit



Hình 4-2 Block diagram ALU 8-bit

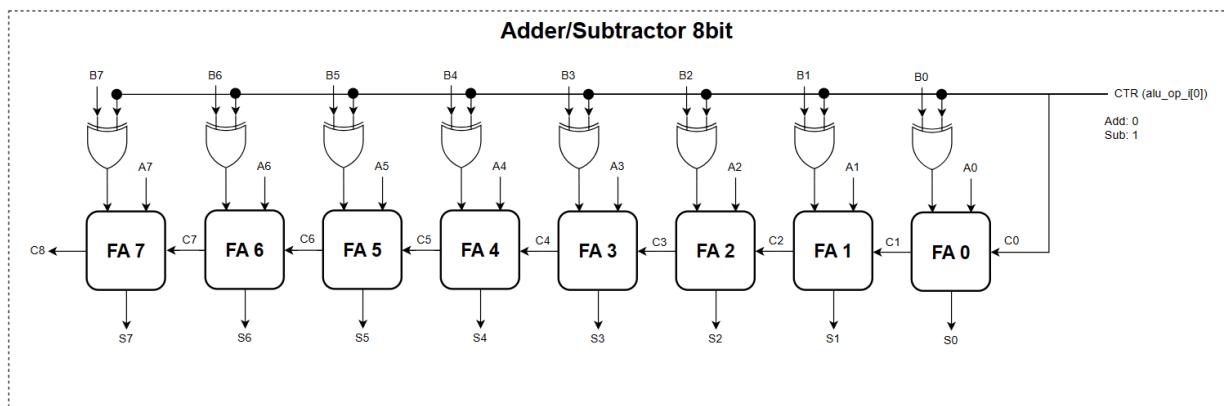
Sơ đồ logic của các thiết kế cho các toán hạng

- Toán hạng ADD:



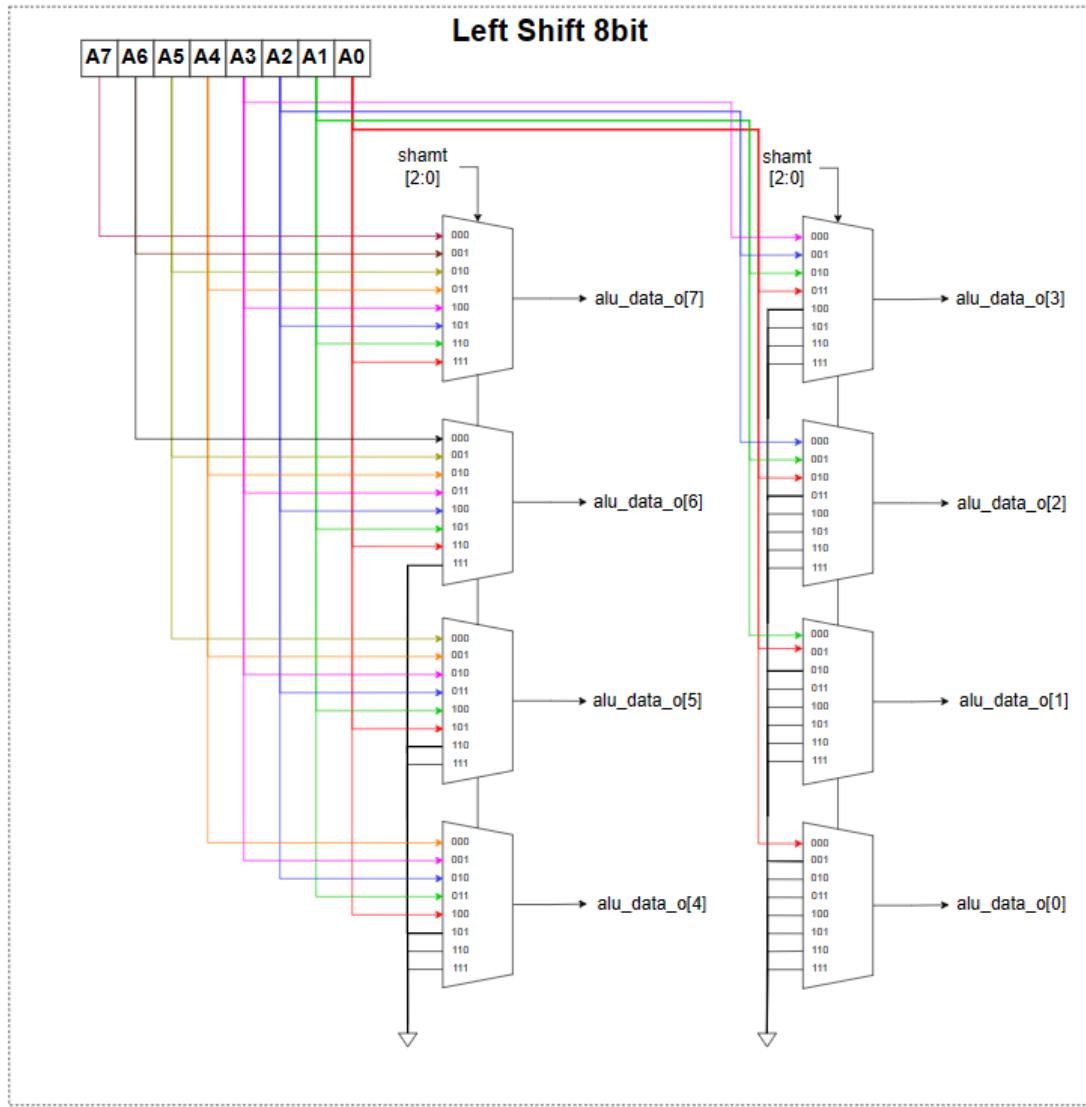
*Hình 4-3* Bộ cộng Ripple Carry Adder với CTR = 0

- Toán hạng SUB:

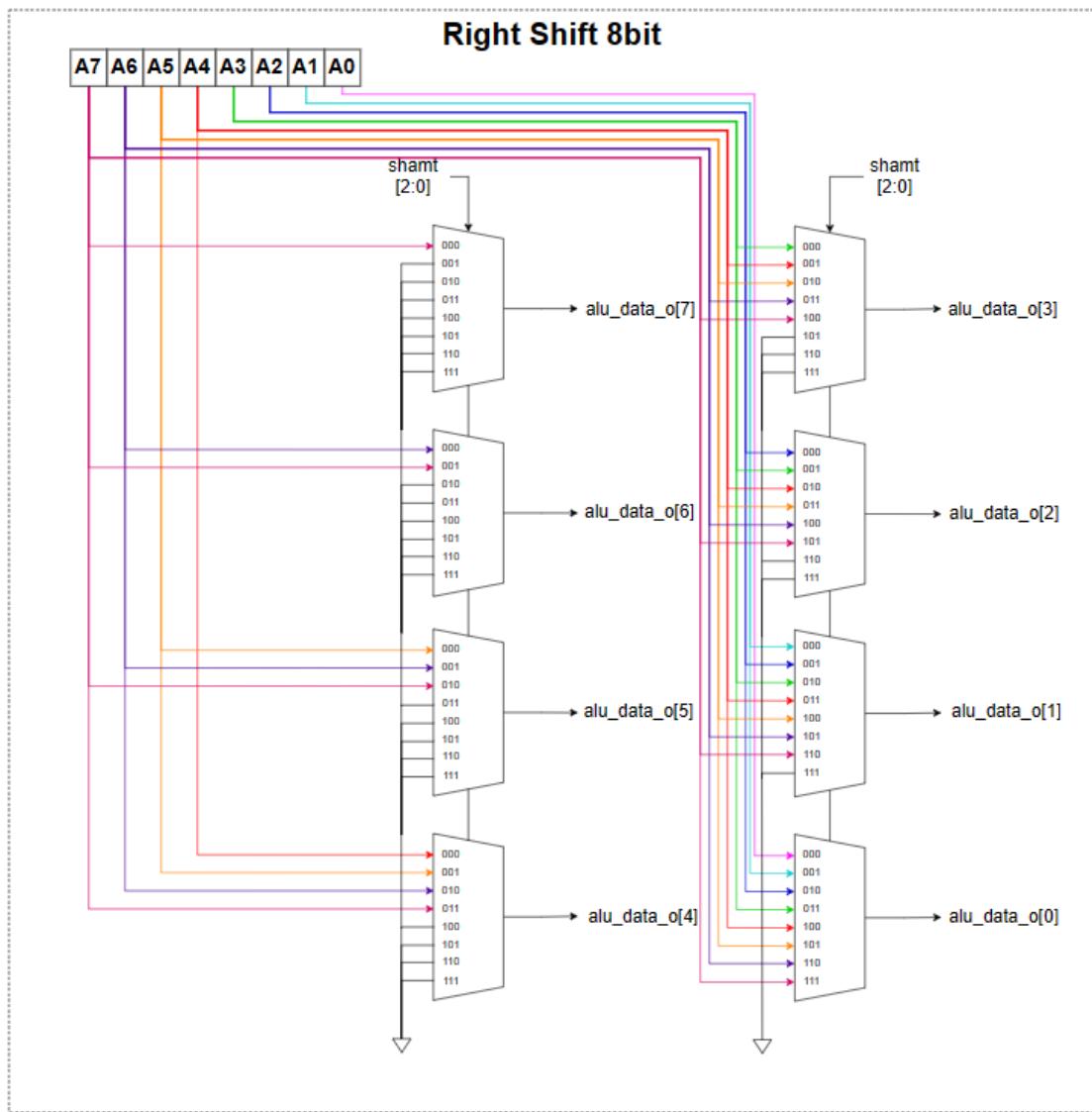


*Hình 4-4* Bộ cộng Ripple Carry Adder với CTR = 1

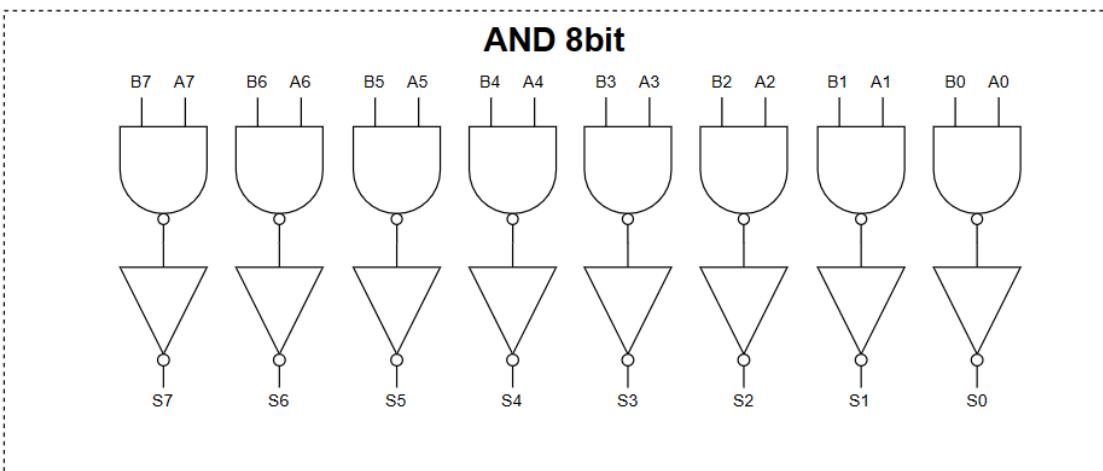
- Toán hạng SLL:



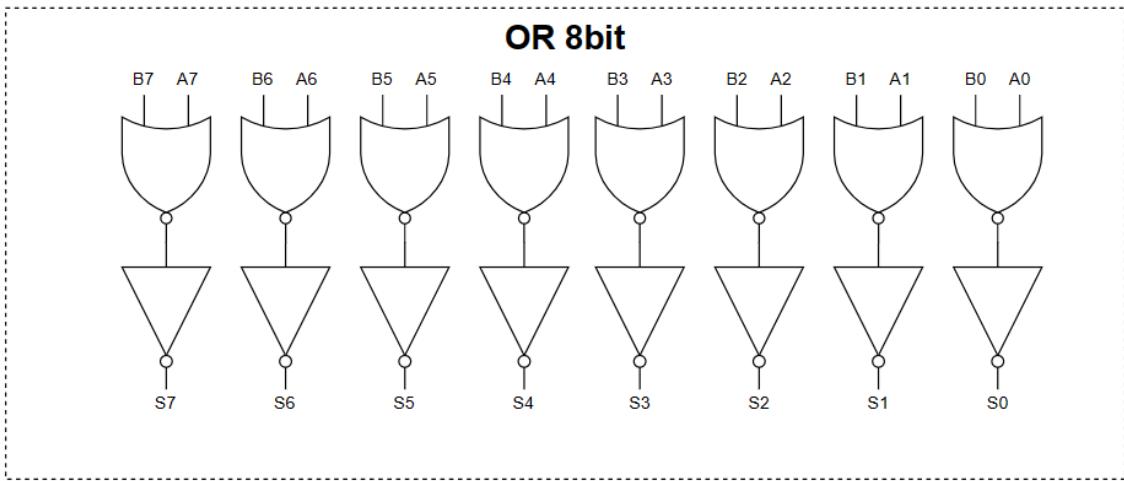
- Toán hạng SRL:



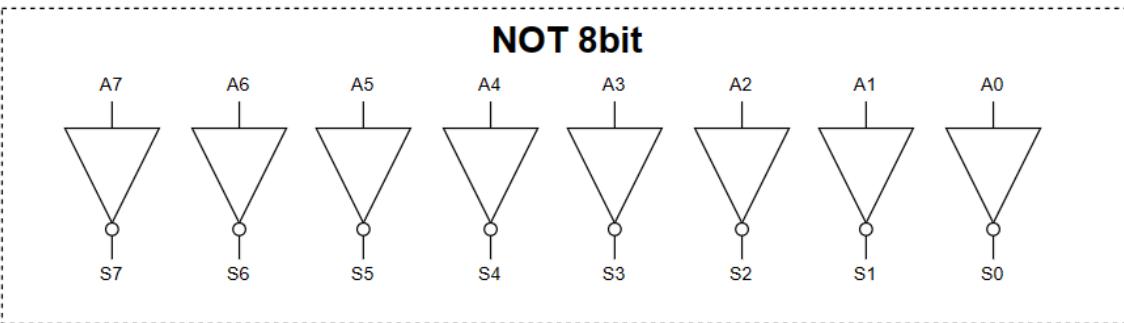
- Toán hạng AND:



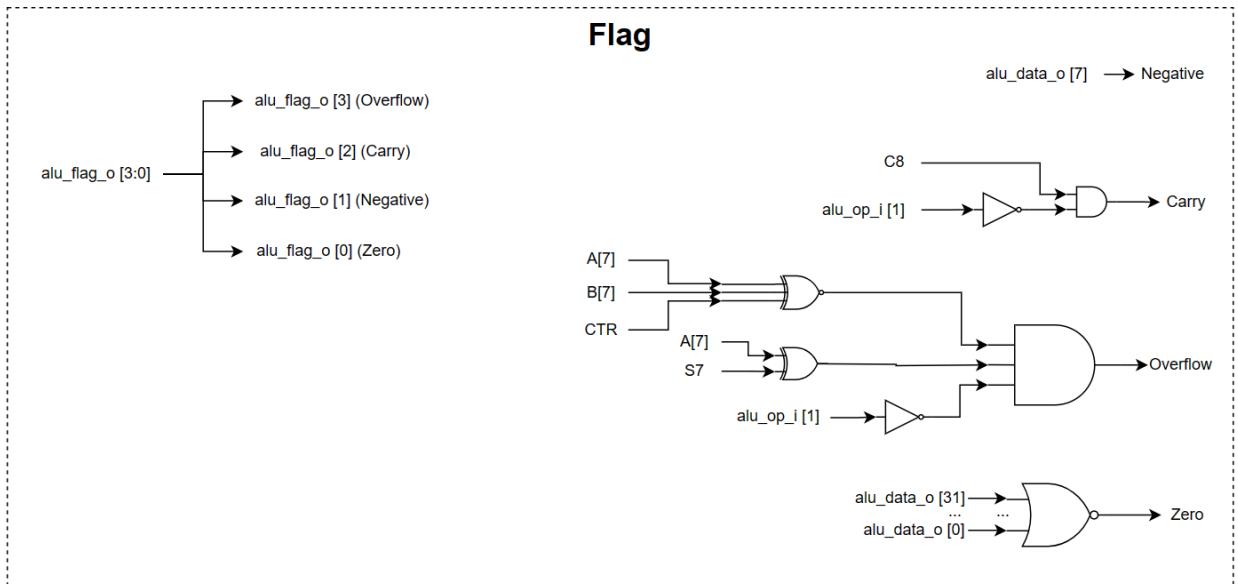
- Toán hạng OR:



- Toán hạng NOT:

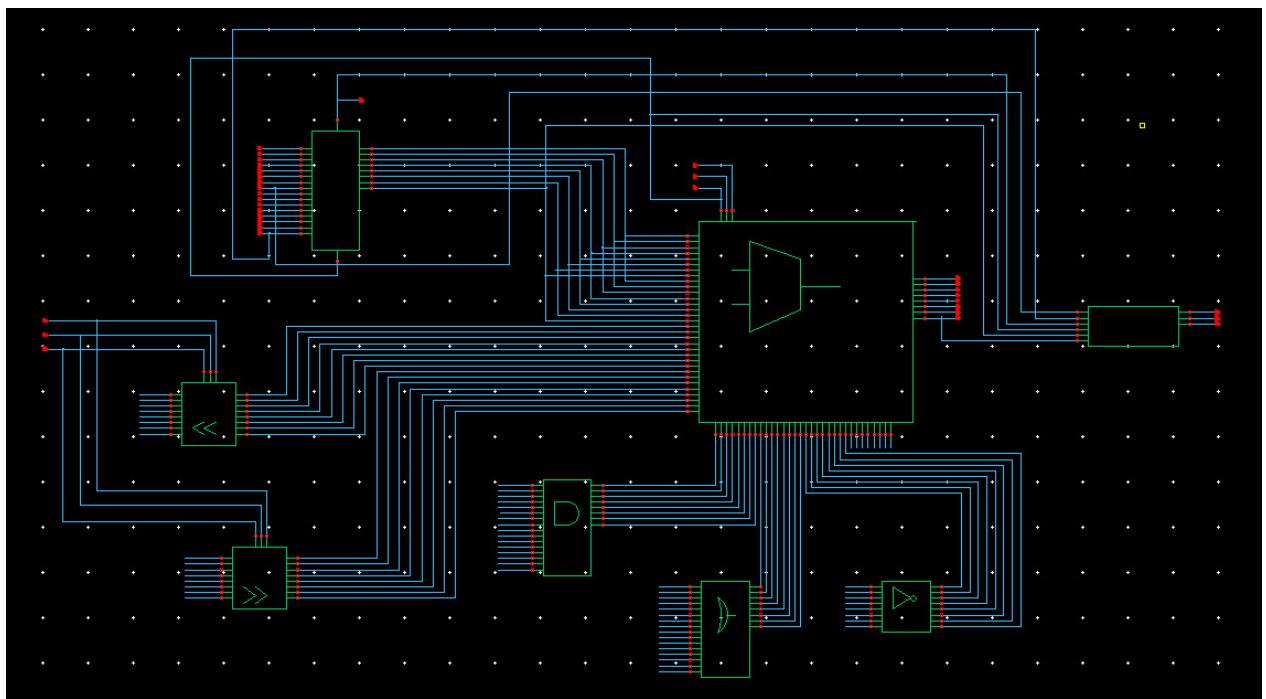
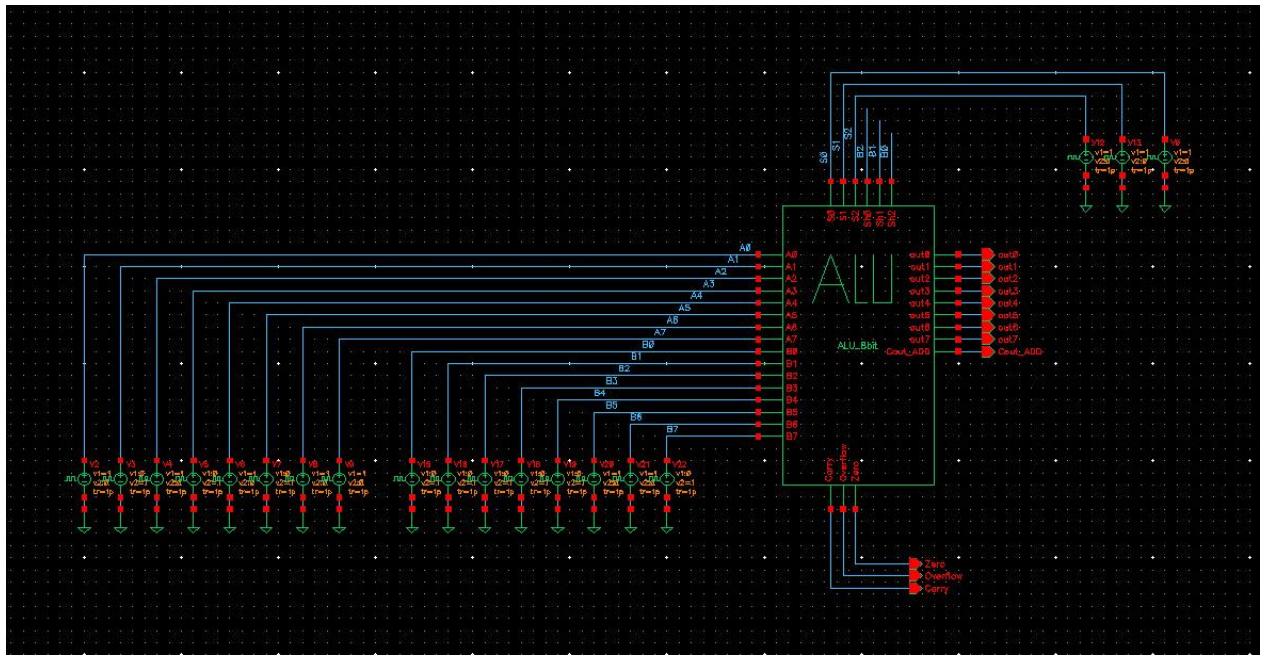


➤ Sơ đồ logic các chân ngõ ra Flags (N,C,V,Z)

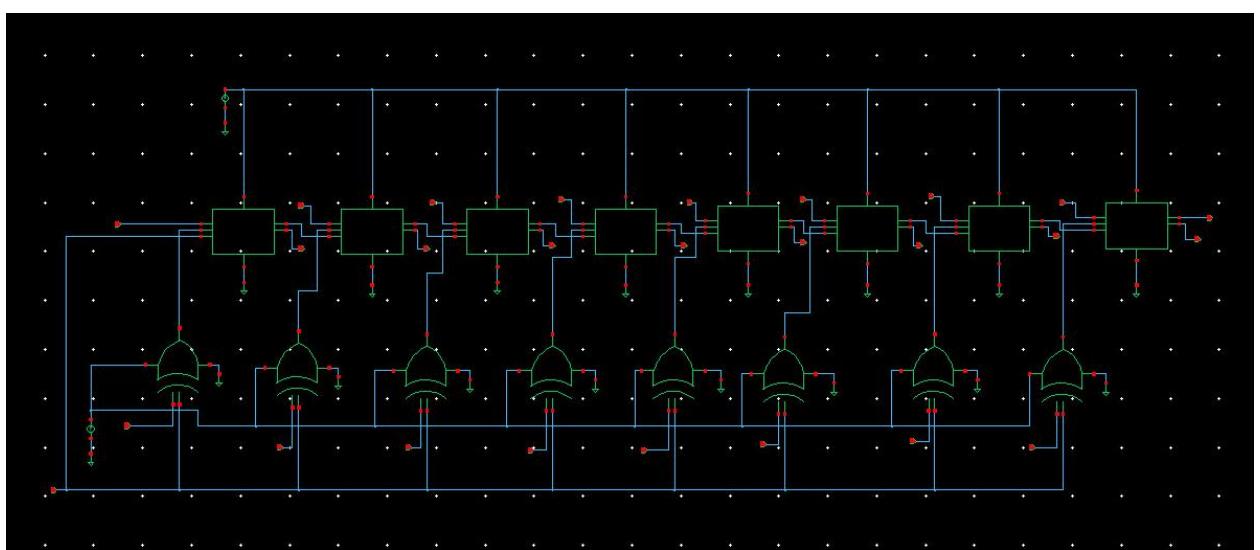
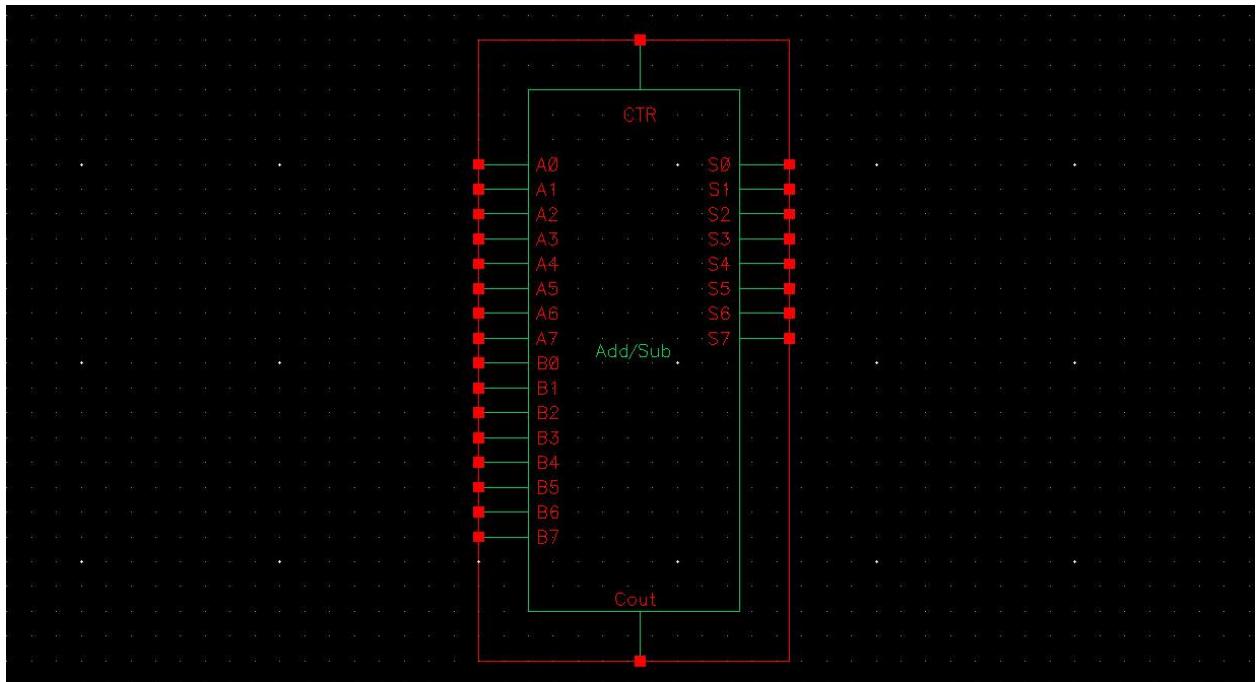


➤ Sơ đồ schematics của các thiết kế cho các toán hạng trên Cadence Virtuoso

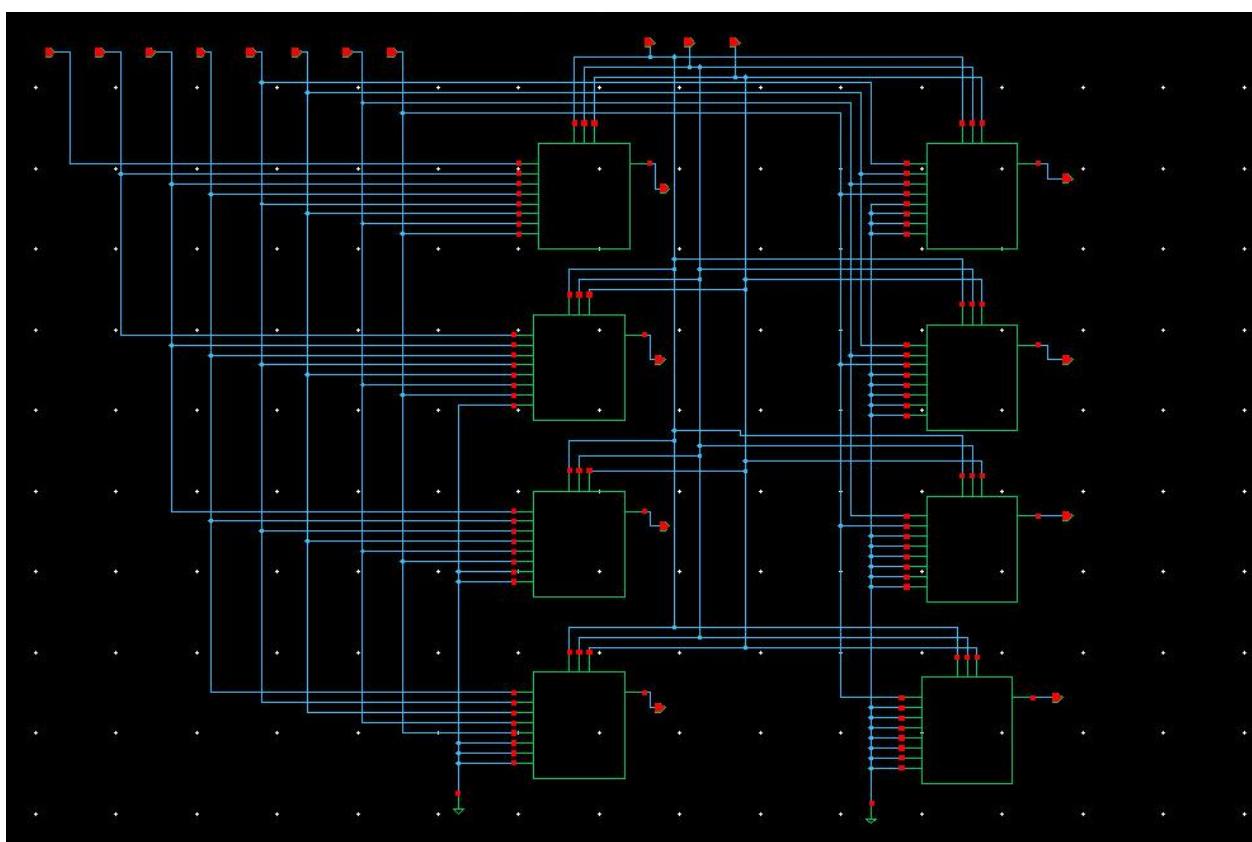
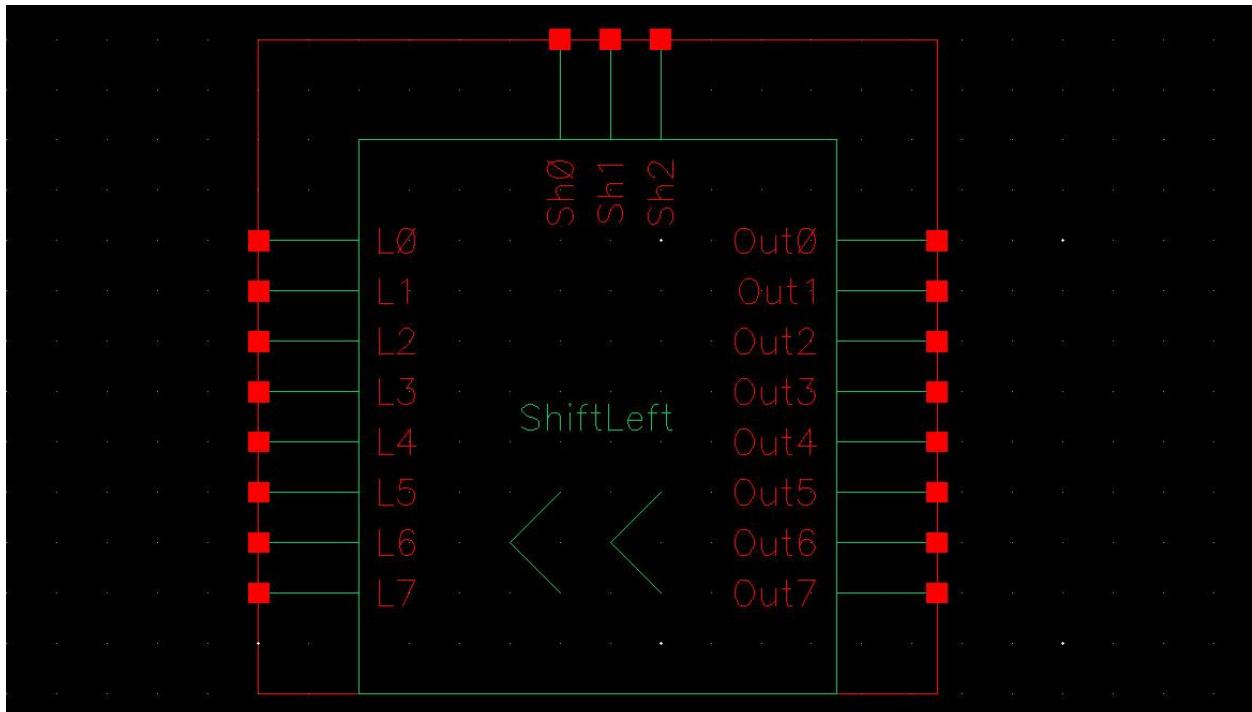
- Arithmetic Logic Unit:



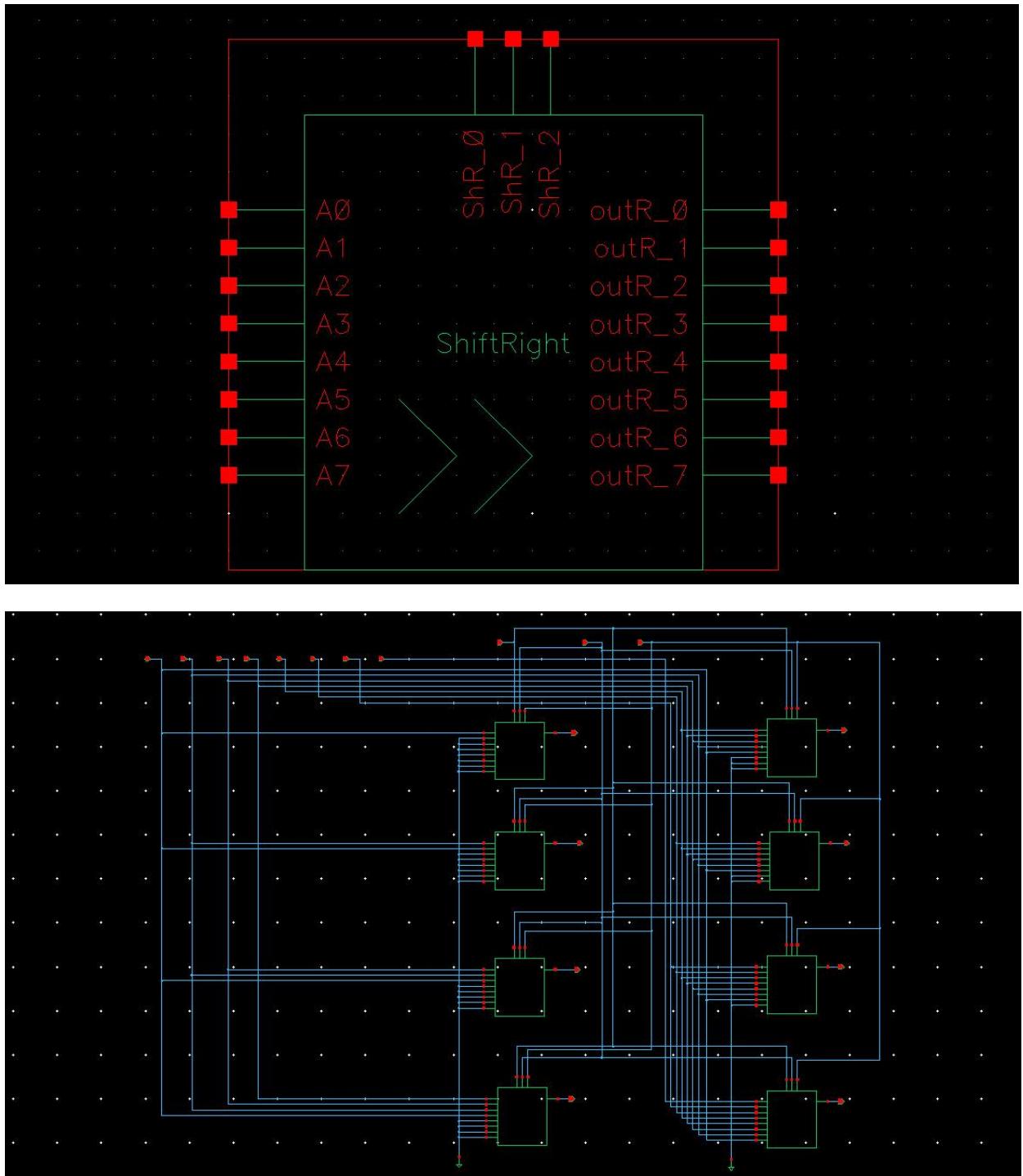
- Toán hạng ADD/SUB:



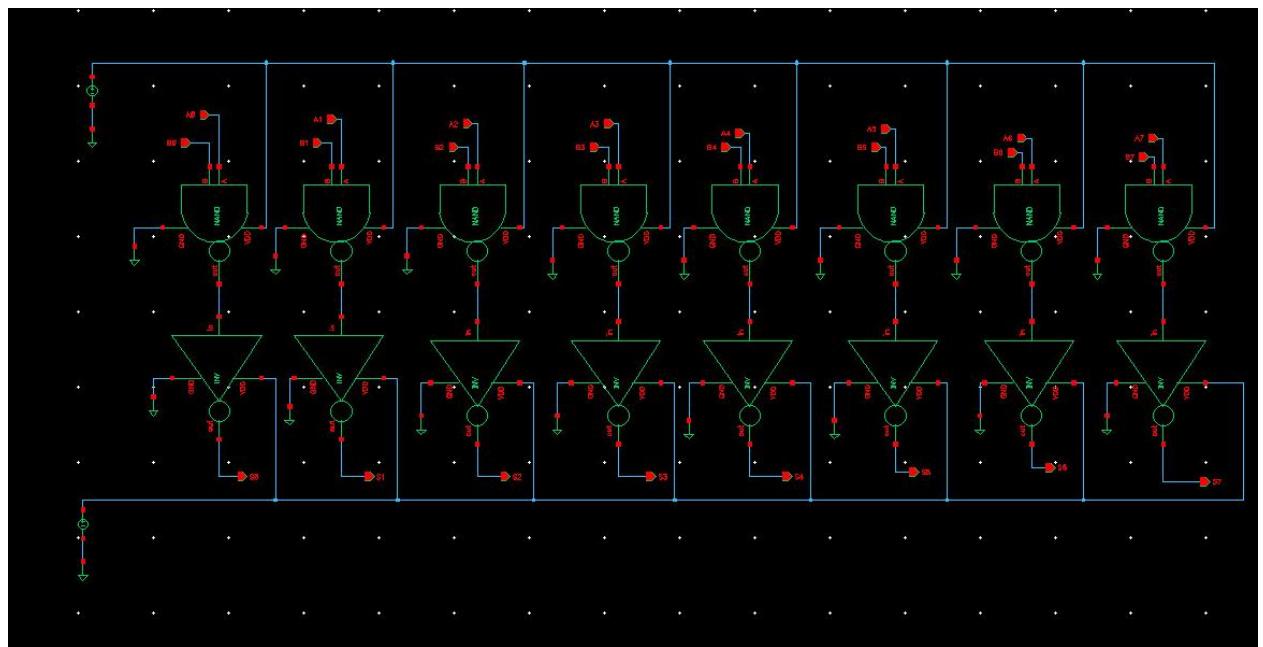
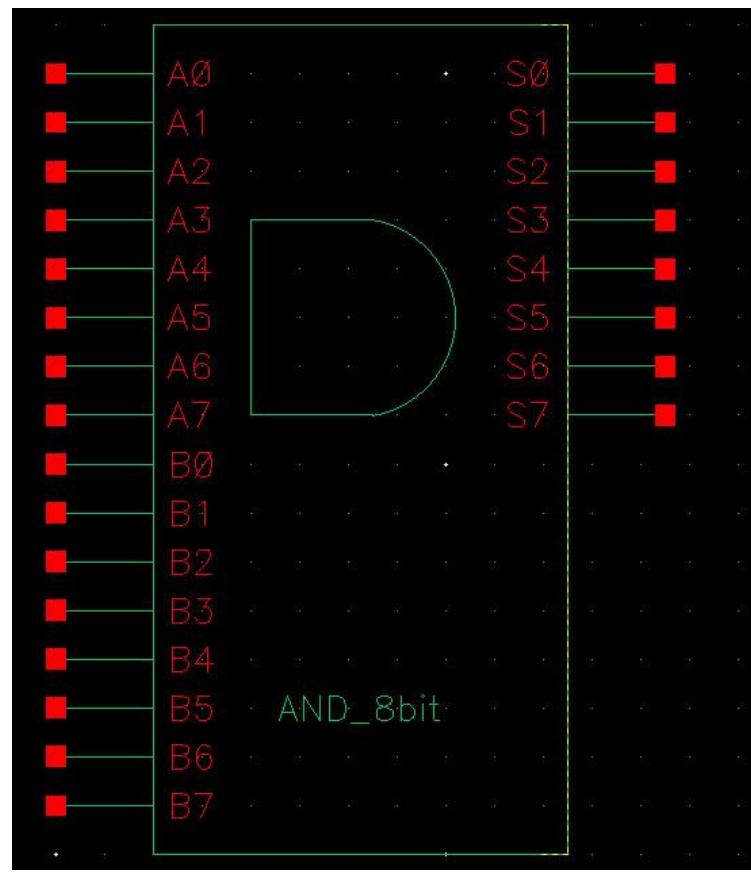
- Toán hạng SLL:



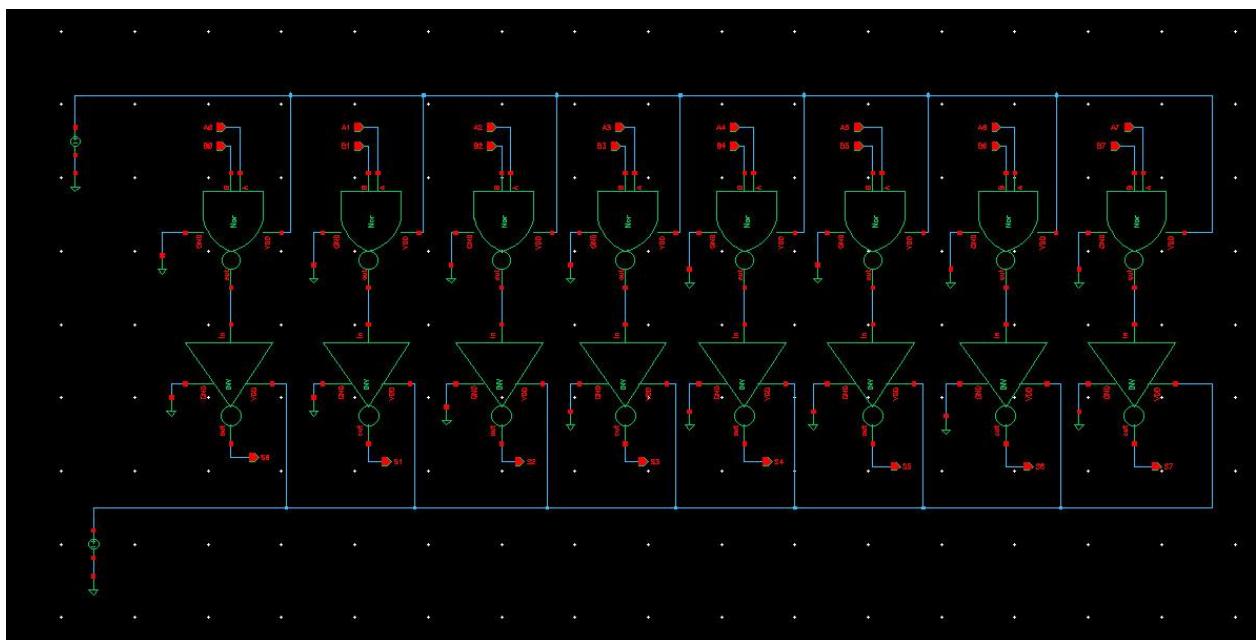
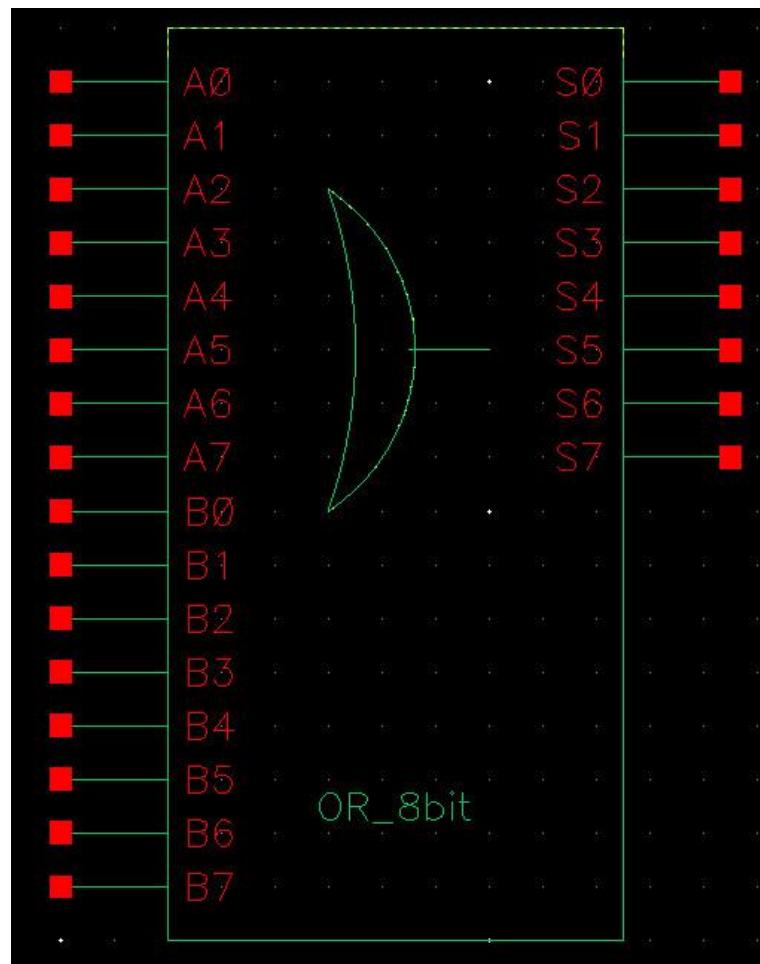
- Toán hạng SRL:



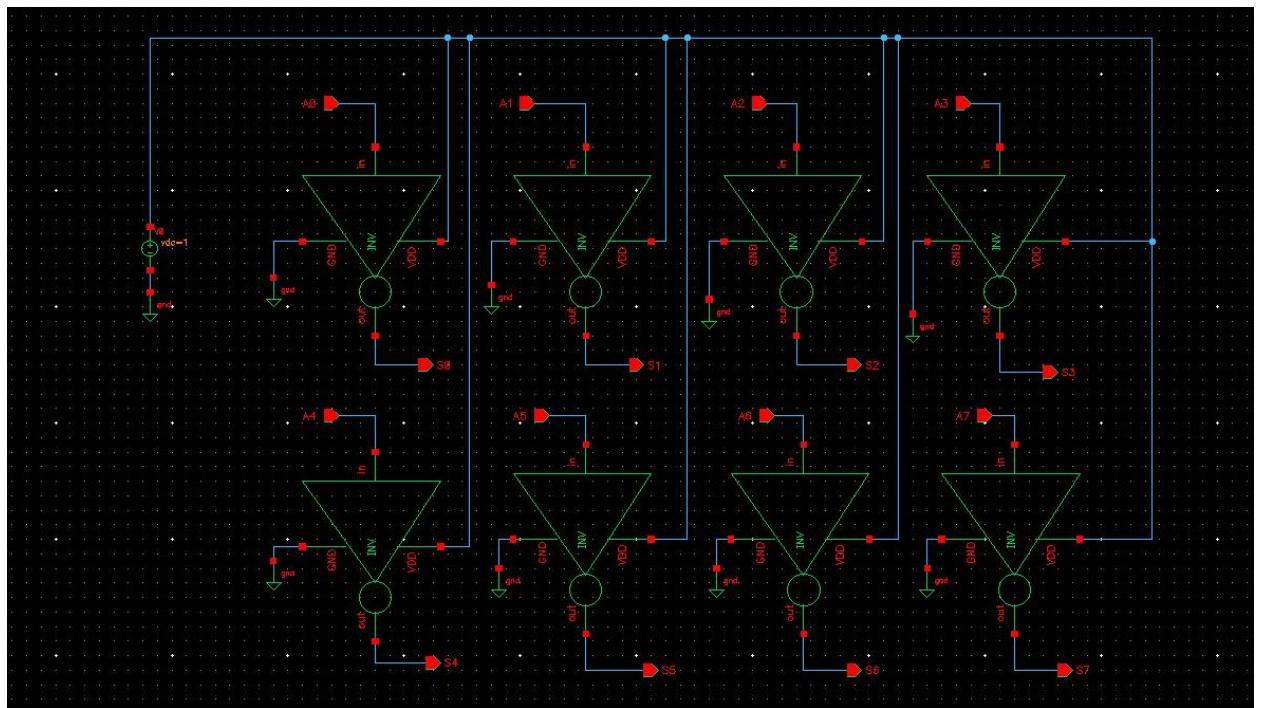
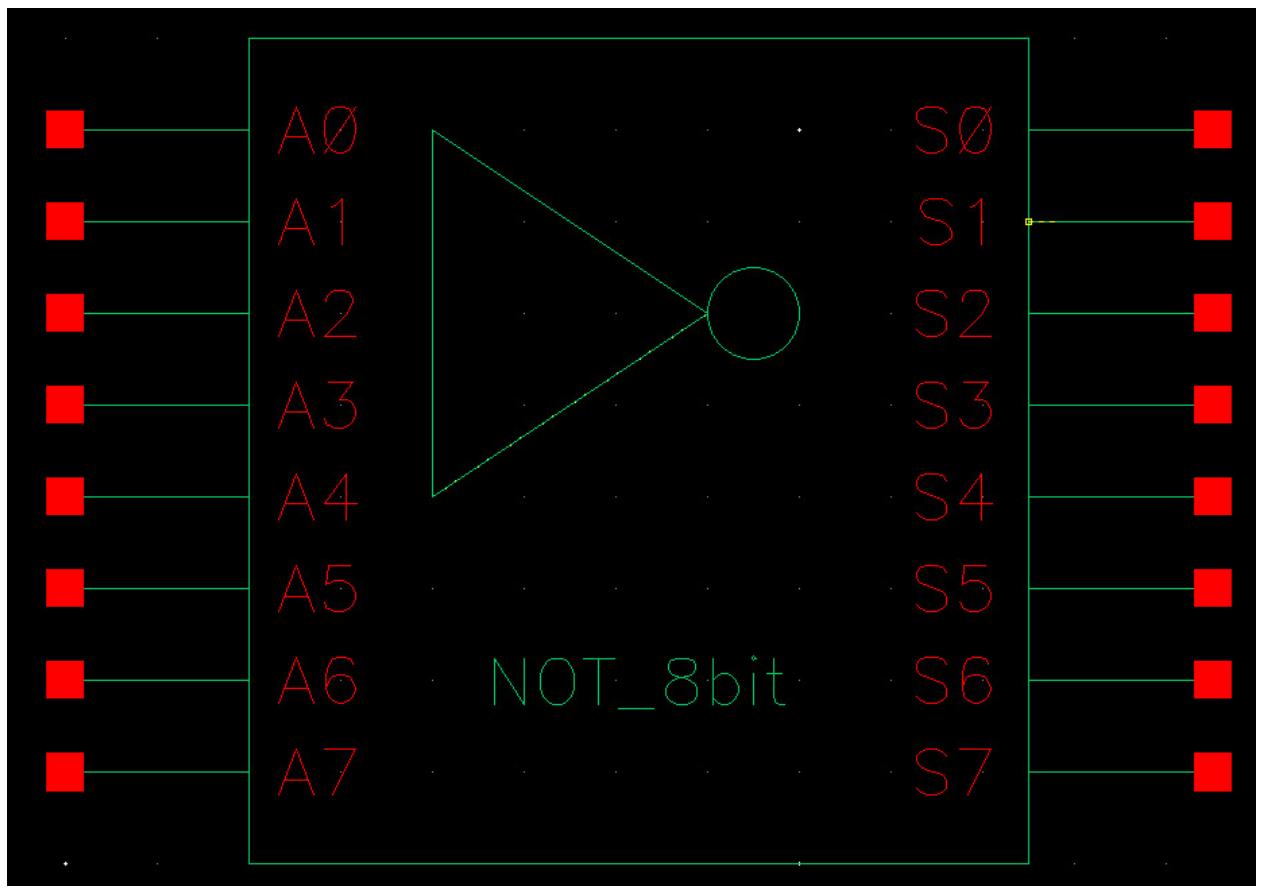
- Toán hạng AND:



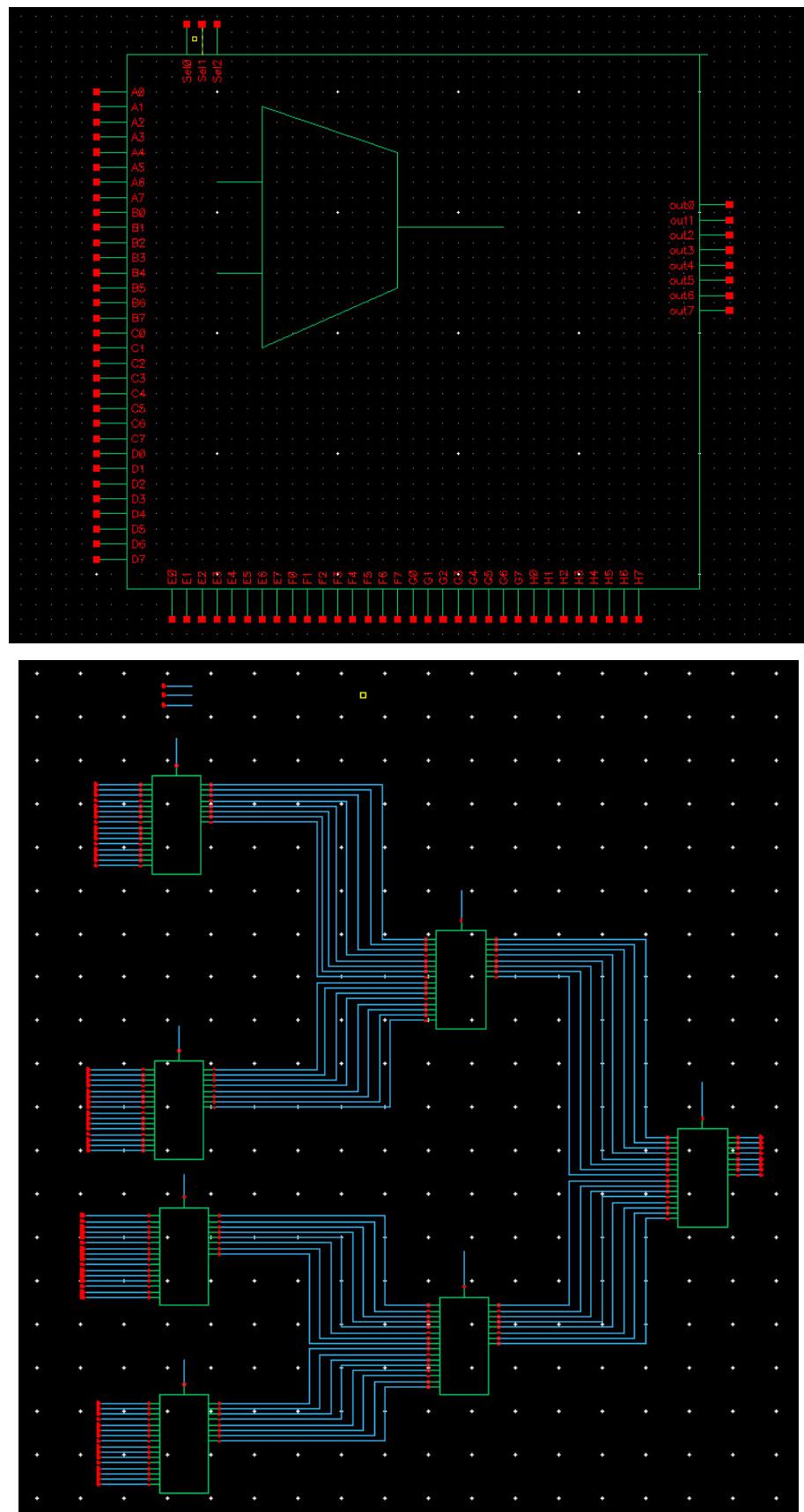
- Toán hạng OR:



- Toán hạng NOT:



-MUX 64-8:



Kiểm tra:

Đoạn code HDL sử dụng SystemVerilog để mô tả thiết kế ALU 8-bit

```
module ALU_HDL(
//inputs
input logic clk_i,
input logic [7:0] operand_a_i,
input logic [7:0] operand_b_i,
input logic [2:0] alu_op_i,
//outputs
output logic [7:0] alu_data_o,
output logic alu_overflow_o,
output logic alu_carryout_o,
output logic alu_negative_o,
output logic alu_zero_o
);

logic tmp_carry;
logic [7:0] tmp_alu_data;

typedef enum logic [2:0] {
ADD = 3'b000,
SUB = 3'b001,
SLL = 3'b010,
SRL = 3'b011,
AND = 3'b100,
OR = 3'b101,
NOT = 3'b110,
FIA = 3'b111
} alu_op_t;

//modules
adder_8bit RCA_01(
.A1 (operand_a_i),
.N1 (operand_b_i),
.CTR1 (alu_op_i[0]),
.carryout_o (tmp_carry),
.result_o (tmp_alu_data)
);

//implementation
always.comb begin
case (alu_op_i[2:0])
ADD: alu_data_o = tmp_alu_data;
SUB: alu_data_o = tmp_alu_data;
SLL: alu_data_o = operand_a_i << operand_b_i[2:0];
SRL: alu_data_o = operand_a_i >> operand_b_i[2:0];
NOT: alu_data_o = ~operand_a_i;
OR : alu_data_o = operand_a_i | operand_b_i;
AND: alu_data_o = operand_a_i & operand_b_i;
FIA: alu_data_o = operand_a_i;
default: alu_data_o = 8'h0;

```

```

endcase
end

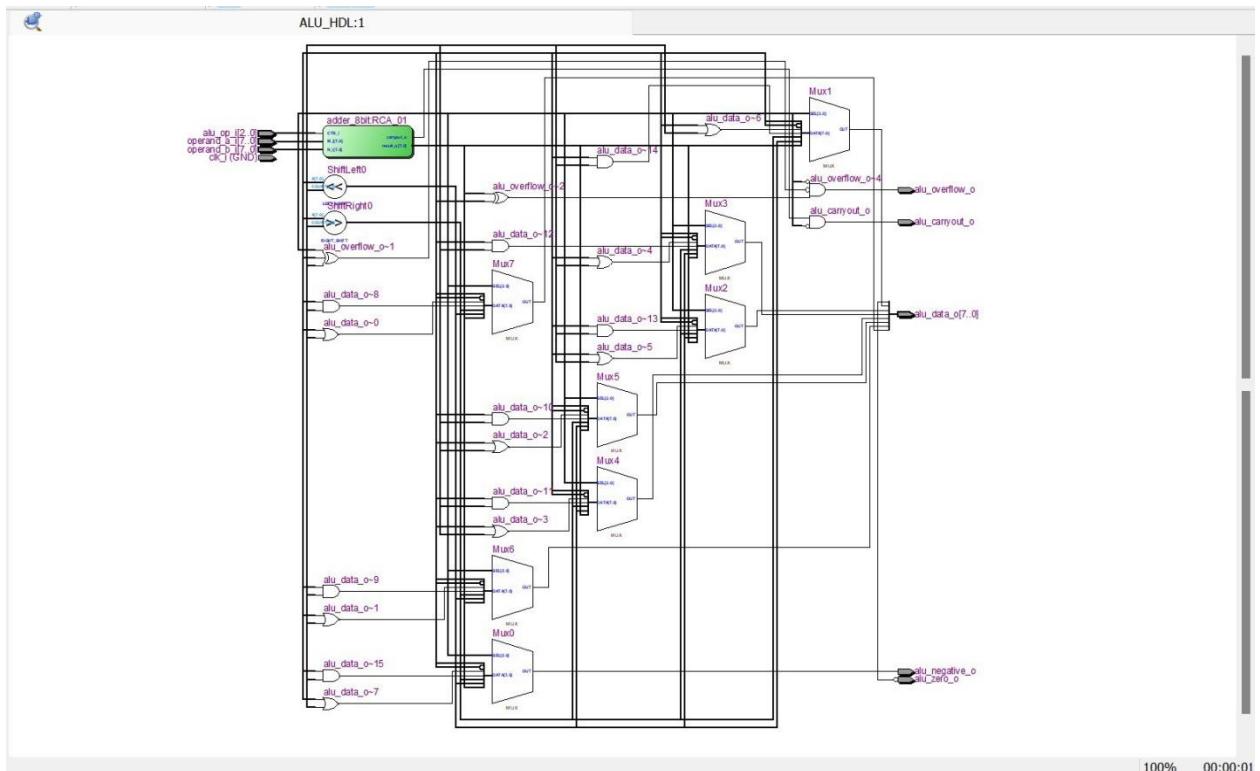
assign alu_overflow_o = -(operand_a_i[7] ^ operand_b_i[7]) ^ alu_op_i[0]) &
(operand_a_i[7] ^ tmp_alu_data[7]) & (~alu_op_i[1]);
assign alu_carryout_o = tmp_carry & (~alu_op_i[1]);
assign alu_negative_o = alu_data_o[7];
assign alu_zero_o = ~|alu_data_o[7:0];
endmodule : ALU_HDL

```

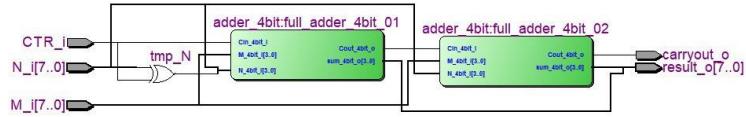
**Nhận xét:** Đoạn code trên sử dụng ngôn ngữ systemverilog và áp dụng các phép toán “không cho phép” (+,-,<<,>>). Tuy nhiên, điều này tạm chấp nhận vì kết quả đạt được sẽ tạm thời không quan tâm đến kết quả mạch tổng hợp (Synthesis) trên các phần mềm như DC compiler, mà chỉ dùng kết quả mô phỏng để so sánh với kết quả đạt được khi vẽ mạch trên phần mềm Cadence Virtuoso.Thêm vào đó, kết quả về diện tích hay timing trong mạch khi synthesis cũng sẽ không đề cập trong báo cáo này.

➤ Kết quả RTL viewer

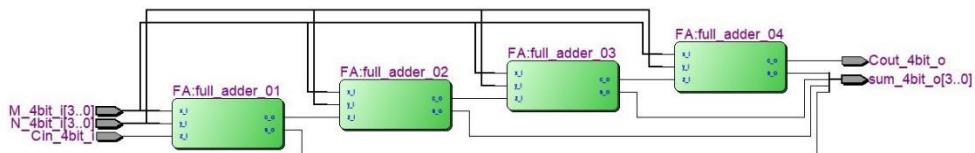
- Tổng quan ALU:



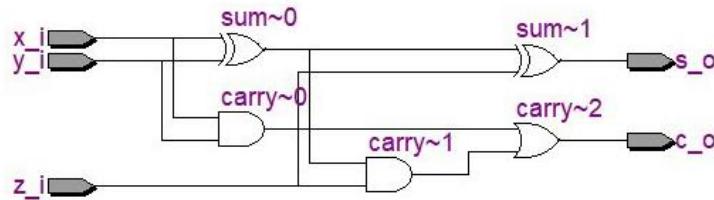
- Adder 8 bit:



- Adder 4 bit (Ripple Carry Adder):

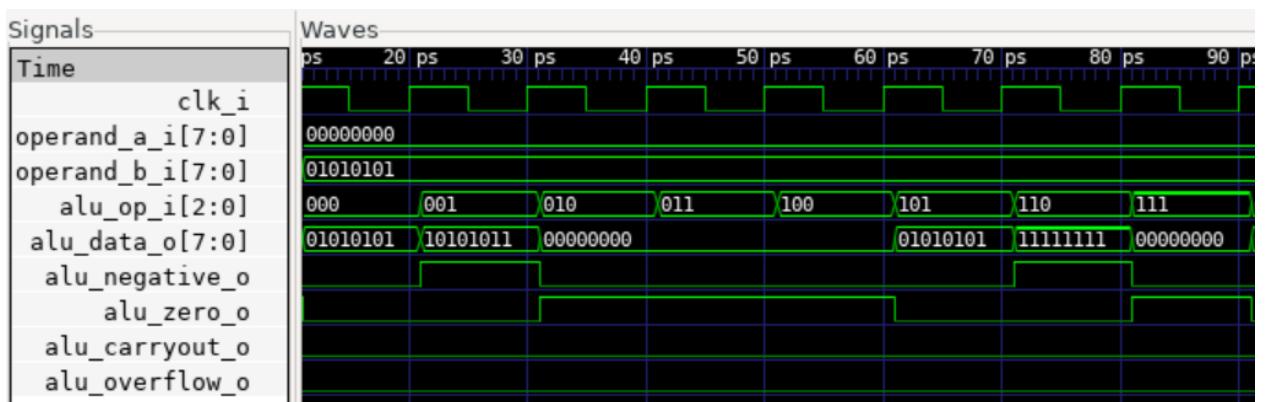


- Full adder:



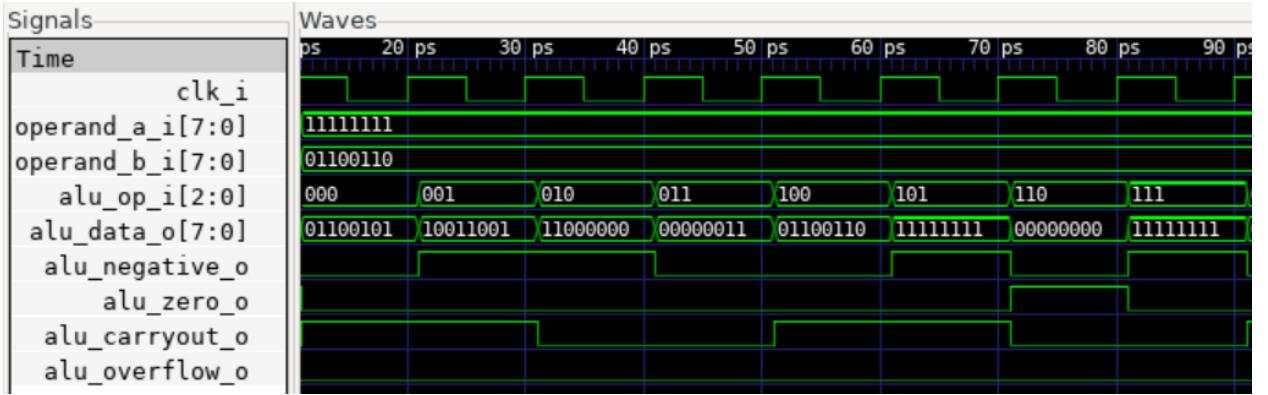
➤ Dạng sóng mô phỏng lý tưởng từ code HDL

- *TestVector01:*



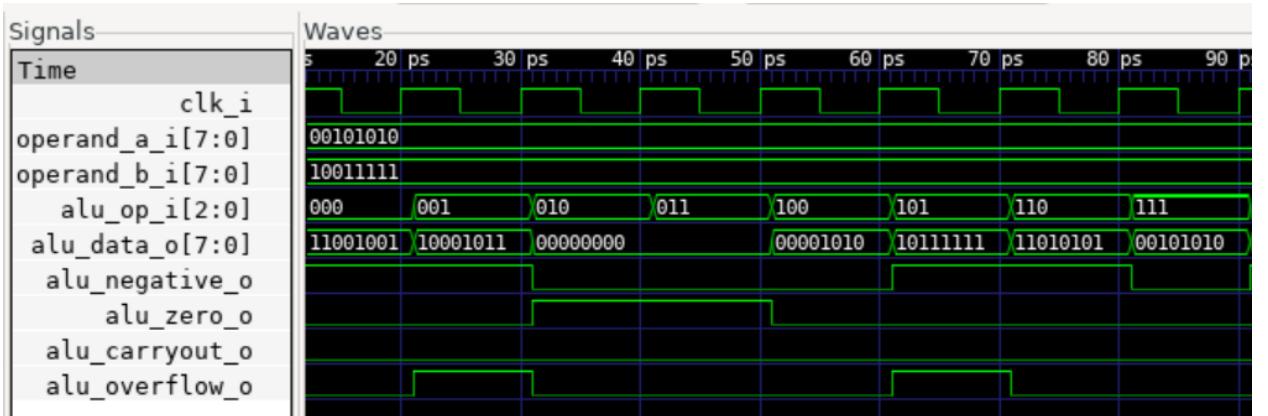
**Nhận xét:** Như kết quả mô phỏng, phép toán cộng (ADD) chính xác ( $0000\_0000 + 0101\_0101 = 0101\_0101$ ), phép toán trừ (SUB) chính xác ( $0000\_0000 - 0101\_0101 = 0000\_0000 + 1010\_1011 = 1010\_1011$ ), vì  $0000\_0000$  dịch trái và phải với sham = 101 nên kết quả SLL và SLR là  $0000\_0000$  (0 luôn = 0 với mọi giá trị sham). Phép toán AND chính xác (vì A = 0 nên alu\_data = 0), kết quả NOT chính xác ( $\sim 0000\_0000 = 1111\_1111$ ). Các kết quả đã được kiểm tra lại với kết quả hiển thị trên Casio 580VNX và website: RapidTable.

- *TestVector02:*



**Nhận xét:** Như kết quả mô phỏng, phép toán cộng (ADD) chính xác ( $1111\_1111 + 0110\_0110 = 0110\_0101$ ), phép toán trừ (SUB) chính xác ( $1111\_1111 - 0110\_0110 = 1111\_1111 + 0110\_0110 = 1001\_1001$ ), vì  $1111\_1111$  dịch trái và phải với shamt = 110 nên kết quả SLL là  $1100\_0000$  và SLR là  $0000\_0011$ . Phép toán AND chính xác ( $1111\_1111 \& 0110\_0110 = 0110\_0110 = B$ ), kết quả NOT chính xác ( $\sim 1111\_1111 = 0000\_0000$ ). Các kết quả đã được kiểm tra lại với kết quả hiển thị trên Casio 580VNX và website: RapidTable.

- *TestVector03:*



**Nhận xét:** Như kết quả mô phỏng, phép toán cộng (ADD) chính xác ( $0010\_1010 + 1001\_1111 = 1100\_1001$ ), phép toán trừ (SUB) chính xác ( $0010\_1010 - 1001\_1111 = 0010\_1010 + 0110\_0000 = 1000\_1010$ ), vì  $0010\_1010$  dịch trái và phải với shamt = 111 nên kết quả SLL là  $0000\_0000$  và SLR là  $0000\_0000$ . Phép toán AND chính xác ( $0010\_1010 \& 1001\_1111 = 0000\_1010$ ), kết quả NOT chính xác ( $\sim 0010\_1010 = 1101\_0101$ ). Các kết quả đã được kiểm tra lại với kết quả hiển thị trên Casio 580VNX và website: RapidTable.

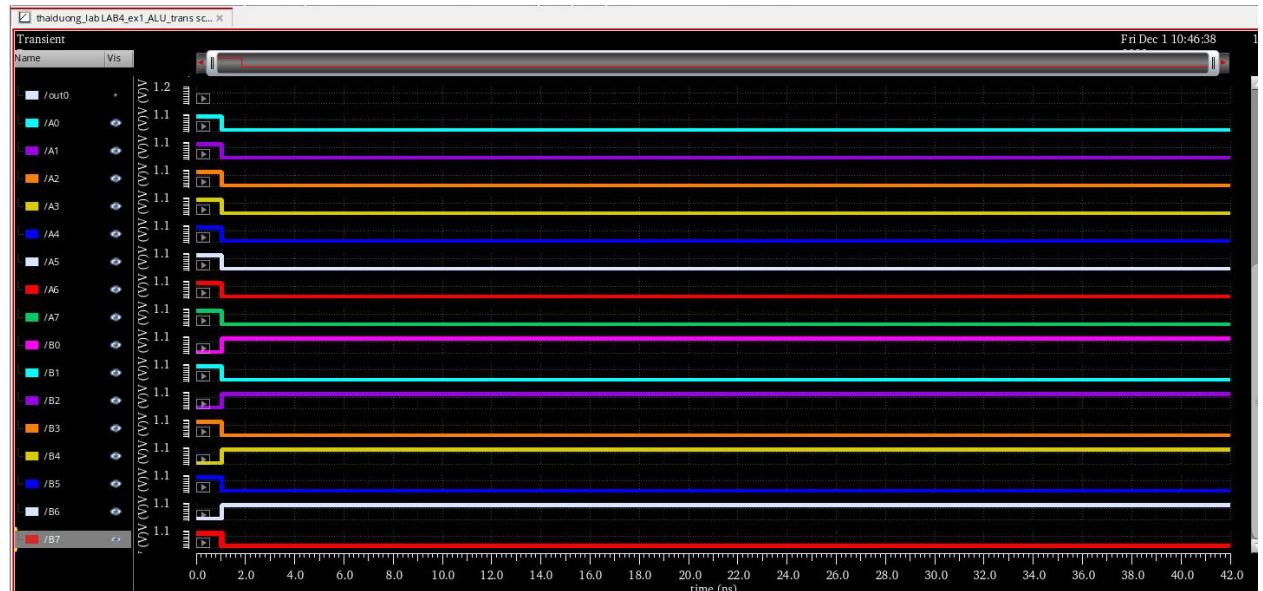
➤ Dạng sóng mô phỏng thiết kế ALU 8-bit

### *TestVector01:*

- Bảng chân trị bao gồm giá trị vào và ngõ ra:

A[7:0]	B[7:0]	Alu_op_i[2:0]	Alu_data_o[7:0]	Function
0000_0000	0101_0101	000	0101_0101	ADD
0000_0000	0101_0101	001	1010_1011	SUB
0000_0000	0101_0101	010	0000_0000	SLL
0000_0000	0101_0101	011	0000_0000	SRL
0000_0000	0101_0101	100	0000_0000	AND
0000_0000	0101_0101	101	0101_0101	OR
0000_0000	0101_0101	110	1111_1111	NOT
0000_0000	0101_0101	111	0000_0000	FWA

- Dạng sóng mô phỏng từng trường hợp





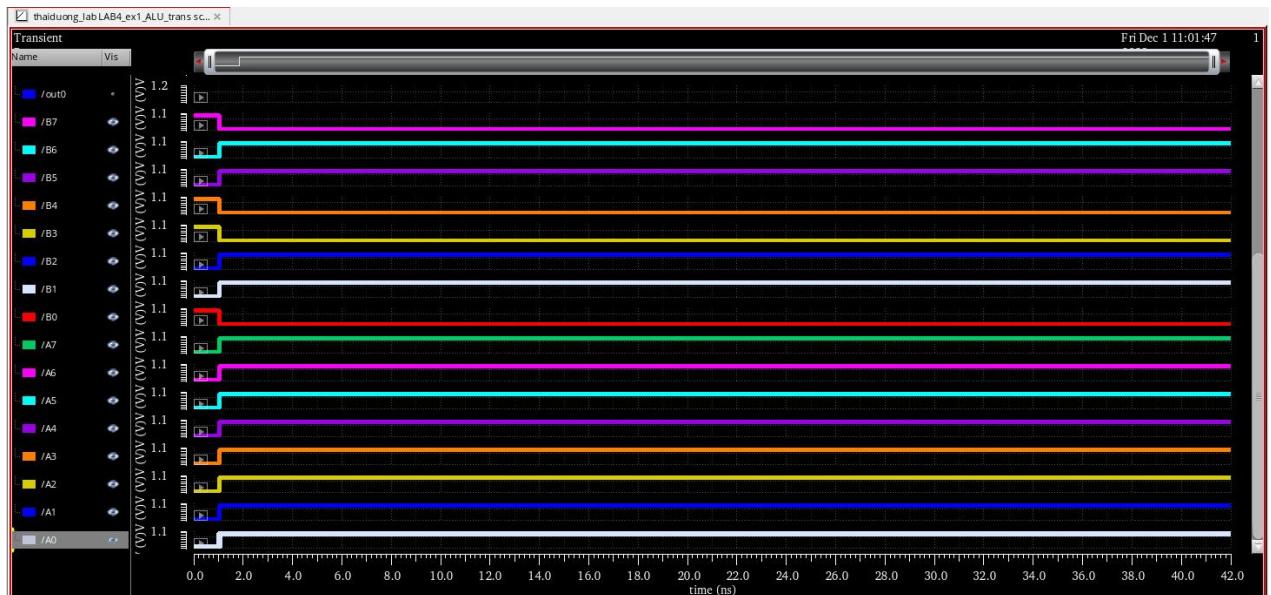
**Nhận xét:** Ảnh mô phỏng trên là giá trị ngõ vào của operand a\_i và operand b\_i, ảnh mô phỏng dưới hiển thị giá trị alu\_op\_i (S2,S1,S0) và giá trị kết quả alu\_data\_o (out7, out6, out5, out4, out3, out2, out1 và out0). Theo như trên, kết quả đạt được từ mạch thiết kế trên Cadence là hoàn toàn khớp với kết quả đạt được trên HDL code. Điều này cho thấy kết quả đạt được từ thiết kế trên Cadence là hoàn toàn chính xác vì kết quả trên HDL đã được kiểm chứng bằng các công cụ khác trong TestVector01. Tuy nhiên, dễ nhận thấy tại một số vị trí xảy ra gợn sóng do việc thiết lập giá trị rise và fall khá nhỏ.

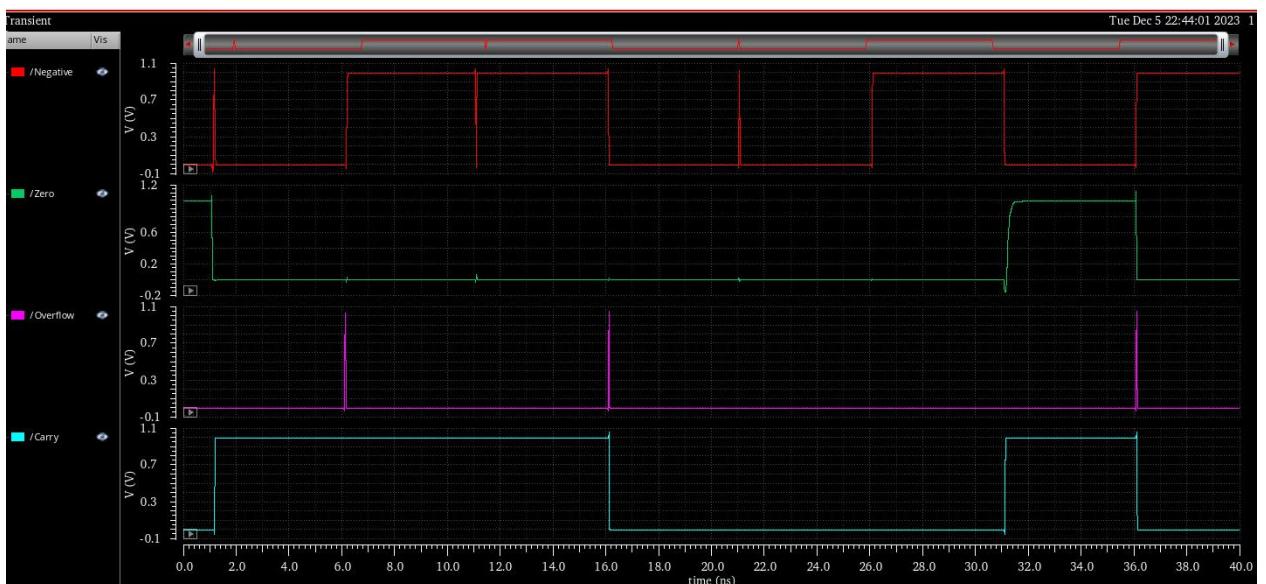
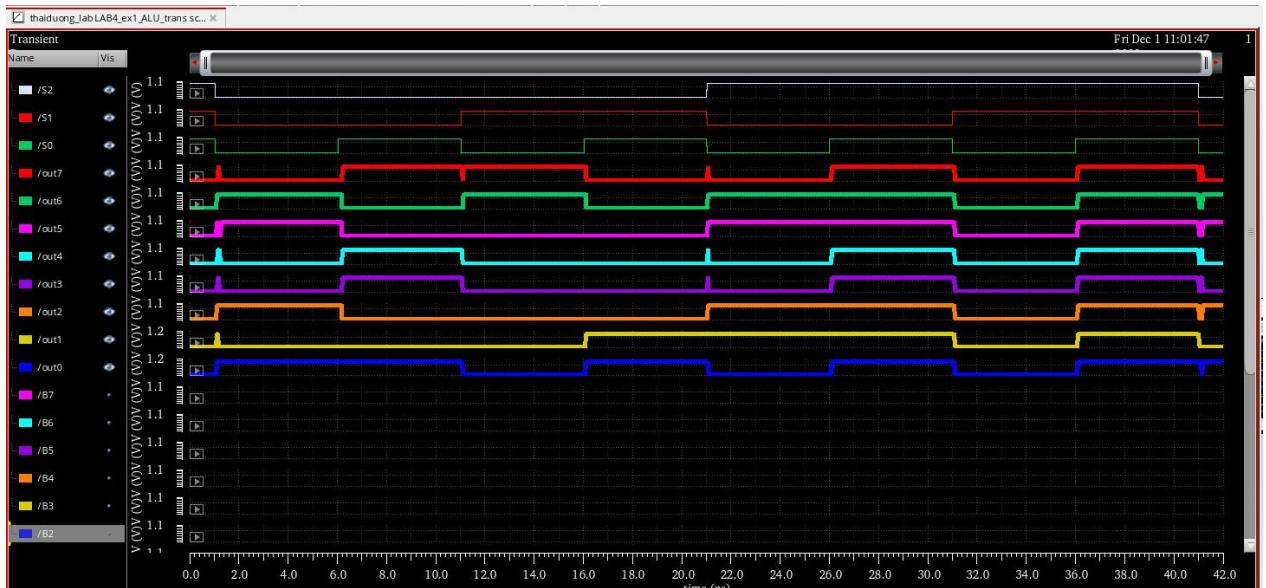
#### TestVector02:

- Bảng chân trị bao gồm giá trị ngõ vào và ngõ ra:

A[7:0]	B[7:0]	Alu_op_i[2:0]	Alu_data_o[7:0]	Function
1111_1111	0110_0110	000	0110_0101	ADD
1111_1111	0110_0110	001	1001_1001	SUB
1111_1111	0110_0110	010	1100_0000	SLL
1111_1111	0110_0110	011	0000_0011	SRL
1111_1111	0110_0110	100	0110_0110	AND
1111_1111	0110_0110	101	1111_1111	OR
1111_1111	0110_0110	110	0000_0000	NOT
1111_1111	0110_0110	111	1111_1111	FWA

- Dạng sóng mô phỏng từng trường hợp





**Nhận xét:** Ánh mô phỏng trên là giá trị ngõ vào của operand\_a\_i và operand\_b\_i, ánh mô phỏng dưới hiển thị giá trị alu\_op\_i (S2,S1,S0) và giá trị kết quả alu\_data\_o (out7, out6, out5, out4, out3, out2, out1 và out0). Theo như trên, kết quả đạt được từ mạch thiết kế trên Cadence là hoàn toàn khớp với kết quả đạt được trên HDL code. Điều này cho thấy kết quả đạt được từ thiết kế trên Cadence là hoàn toàn chính xác vì kết quả trên HDL đã được kiểm chứng bằng các công cụ khác trong TestVector02. Tuy nhiên, dễ nhận thấy tại một số vị trí xảy ra gợn sóng do việc thiết lập giá rise và fall khá nhỏ.

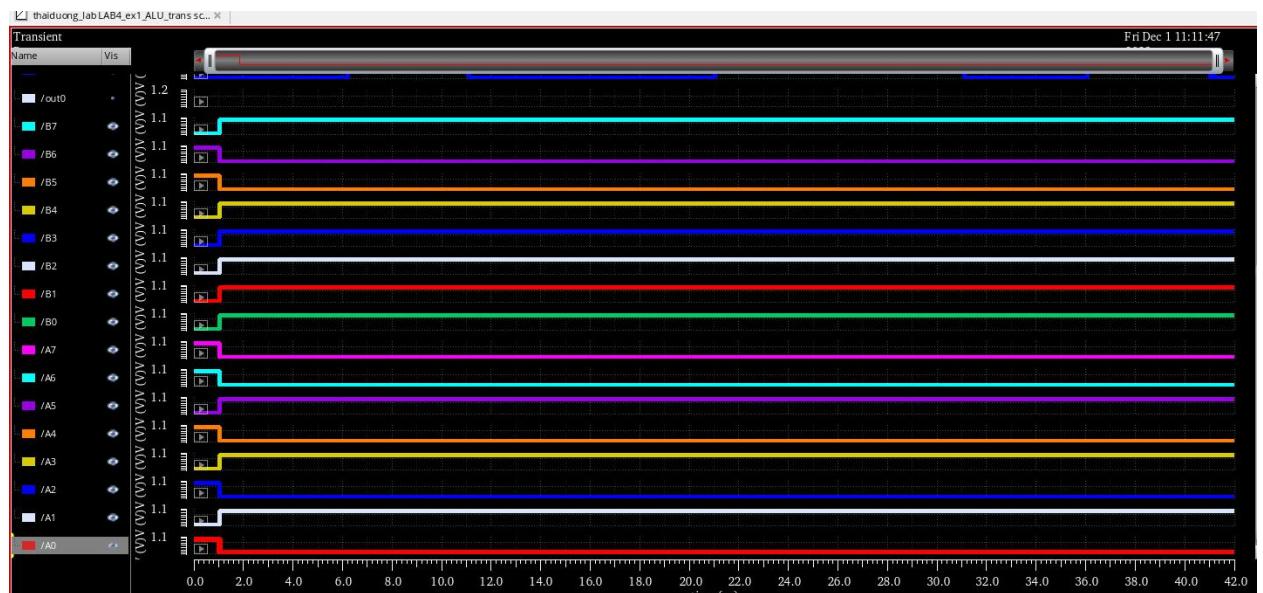
*TestVector03:*

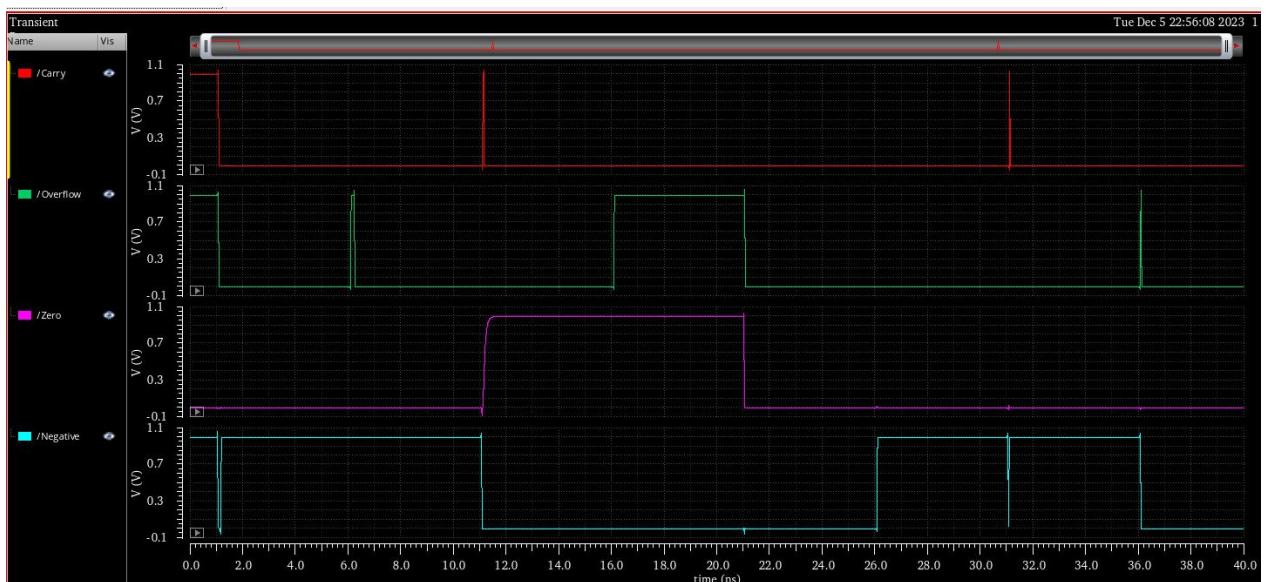
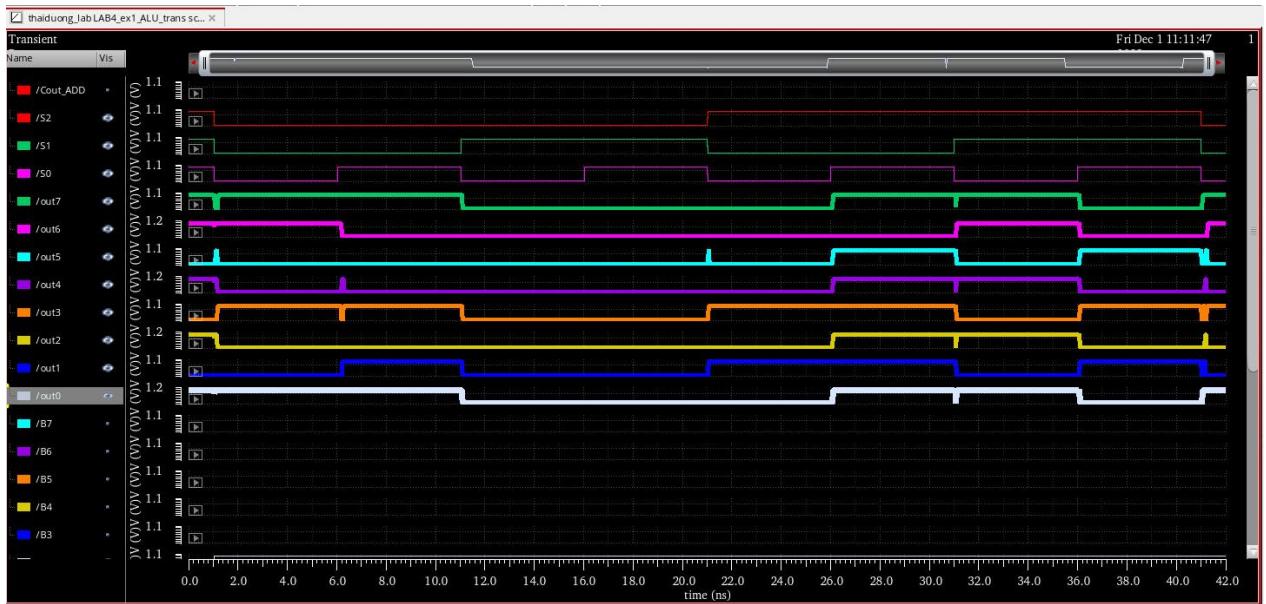
- Bảng chân trị bao gồm giá trị ngõ vào và ngõ ra:

A[7:0]	B[7:0]	Alu_op_i[2:0]	Alu_data_o[7:0]	Function
--------	--------	---------------	-----------------	----------

0010_1010	1001_1111	000	1100_1001	ADD
0010_1010	1001_1111	001	1000_1011	SUB
0010_1010	1001_1111	010	0000_0000	SLL
0010_1010	1001_1111	011	0000_0000	SRL
0010_1010	1001_1111	100	0000_1010	AND
0010_1010	1001_1111	101	1011_1111	OR
0010_1010	1001_1111	110	1101_0101	NOT
0010_1010	1001_1111	111	0010_1010	FWA

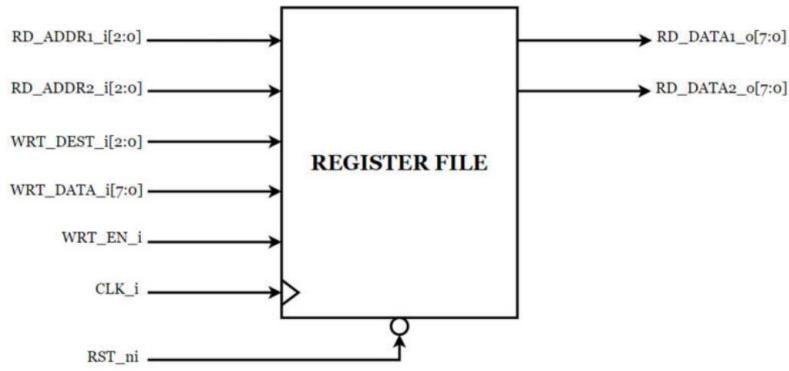
- Dạng sóng mô phỏng từng trường hợp





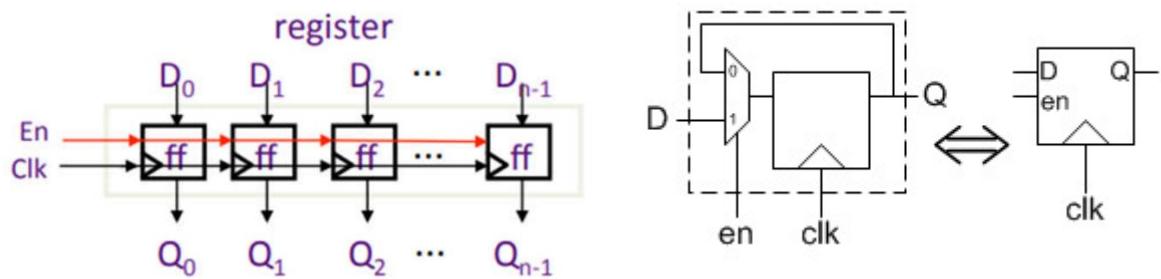
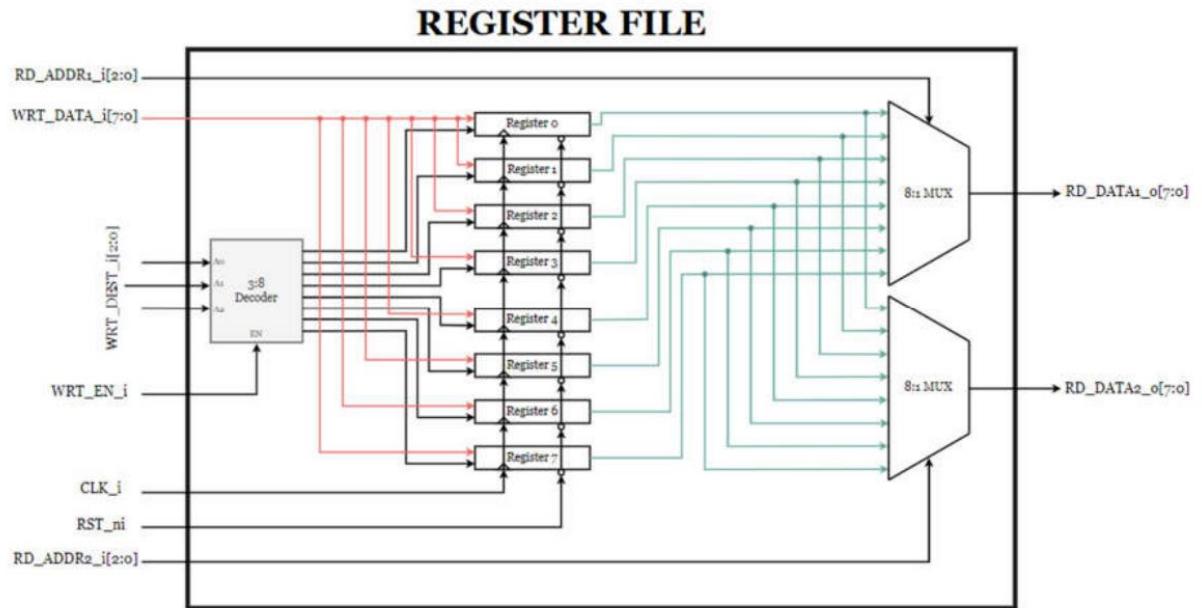
**Nhận xét:** Ánh mõ phỏng trên là giá trị ngõ vào của operand\_a\_i và operand\_b\_i, ánh mõ phỏng dưới hiển thị giá trị alu\_op\_i (S2,S1,S0) và giá trị kết quả alu\_data\_o (out7, out6, out5, out4, out3, out2, out1 và out0). Theo như trên, kết quả đạt được từ mạch thiết kế trên Cadence là hoàn toàn khớp với kết quả đạt được trên HDL code. Điều này cho thấy kết quả đạt được từ thiết kế trên Cadence là hoàn toàn chính xác vì kết quả trên HDL đã được kiểm chứng bằng các công cụ khác trong TestVector03. Tuy nhiên, dễ nhận thấy tại một số vị trí xảy ra gợn sóng do việc thiết lập giá trị rise và fall khá nhỏ.

#### 4.2. Experiment 2



*Block diagram Register File 8-bit*

- Sơ đồ khái niệm cho thiết kế Register File – RF



*Block diagram chi tiết Register File 8-bit (Tham khảo từ: Tài liệu tham khảo thí nghiệm 4)*

- Mô tả các chân ngõ vào và ngõ ra

Signal	Description
CLK_i	Positive clock
RST_ni	Low negative reset
RD_ADDR1_i[2:0]	The first address to be read from.
RD_ADDR2_i[2:0]	The second address to be read from.
WRT_DEST_i[2:0]	Address supposed to be written into the register file.
WRT_DATA_i[7:0]	Data supposed to be written into the register file.
WRT_EN_i	1: write to the Register File 0: read the Register File
RD_DATA1_o	The read data from first address.
RD_DATA2_o	The read data from second address.

### Kiểm tra:

Đoạn code HDL sử dụng SystemVerilog để mô tả thiết kế Register File 8-bit

```

module register_file(
    //-----INPPUT-----
    input logic [2:0] RD_ADDR1_i, RD_ADDR2_i, WRT_DEST_i,
    input logic WRT_EN_i, CLK_i, RST_ni,
    input logic [7:0] WRT_DATA_i,
    //-----OUTPUT-----
    output logic [7:0] RD_DATA1_o, RD_DATA2_o
);

//-----Declare Signal-----//
//Decoder//
logic [7:0] enable_register;

//register file//
logic [7:0] data_reg_0, data_reg_1, data_reg_2, data_reg_3, data_reg_4,
data_reg_5,
    data_reg_6, data_reg_7;

//-----Decoder-----//
always_comb
begin
    if (!WRT_EN_i)
        enable_register = 8'd0;
    else begin

```

```

        case(WRT_DEST_i)
        3'b000:
            enable_register = 8'd1;
        3'b001:
            enable_register = 8'd2;
        3'b010:
            enable_register = 8'd4;
        3'b011:
            enable_register = 8'd8;
        3'b100:
            enable_register = 8'd16;
        3'b101:
            enable_register = 8'd32;
        3'b110:
            enable_register = 8'd64;
        3'b111:
            enable_register = 8'd128;
        endcase
    end
end

//-----Mux-----//
//---RS1---//
always_comb
begin
    case (RD_ADDR1_i)
        3'b000:
            RD_DATA1_o = data_reg_0;
        3'b001:
            RD_DATA1_o = data_reg_1;
        3'b010:
            RD_DATA1_o = data_reg_2;
        3'b011:
            RD_DATA1_o = data_reg_3;
        3'b100:
            RD_DATA1_o = data_reg_4;
        3'b101:
            RD_DATA1_o = data_reg_5;
        3'b110:
            RD_DATA1_o = data_reg_6;
        3'b111:
            RD_DATA1_o = data_reg_7;
    endcase
end
//---RS2--//
always_comb
begin
    case (RD_ADDR2_i)
        3'b000:
            RD_DATA2_o = data_reg_0;

```

```

3'b001:
    RD_DATA2_o = data_reg_1;
3'b010:
    RD_DATA2_o = data_reg_2;
3'b011:
    RD_DATA2_o = data_reg_3;
3'b100:
    RD_DATA2_o = data_reg_4;
3'b101:
    RD_DATA2_o = data_reg_5;
3'b110:
    RD_DATA2_o = data_reg_6;
3'b111:
    RD_DATA2_o = data_reg_7;
endcase
end
//-----Memory-----
PIPO_Register register_0(
    .enable(enable_register[0]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_0)
);

PIPO_Register register_1(
    .enable(enable_register[1]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_1)
);

PIPO_Register register_2(
    .enable(enable_register[2]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_2)
);

PIPO_Register register_3(
    .enable(enable_register[3]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_3)
);

PIPO_Register register_4(

```

```

    .enable(enable_register[4]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_4)
);

PIPO_Register register_5(
    .enable(enable_register[5]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_5)
);

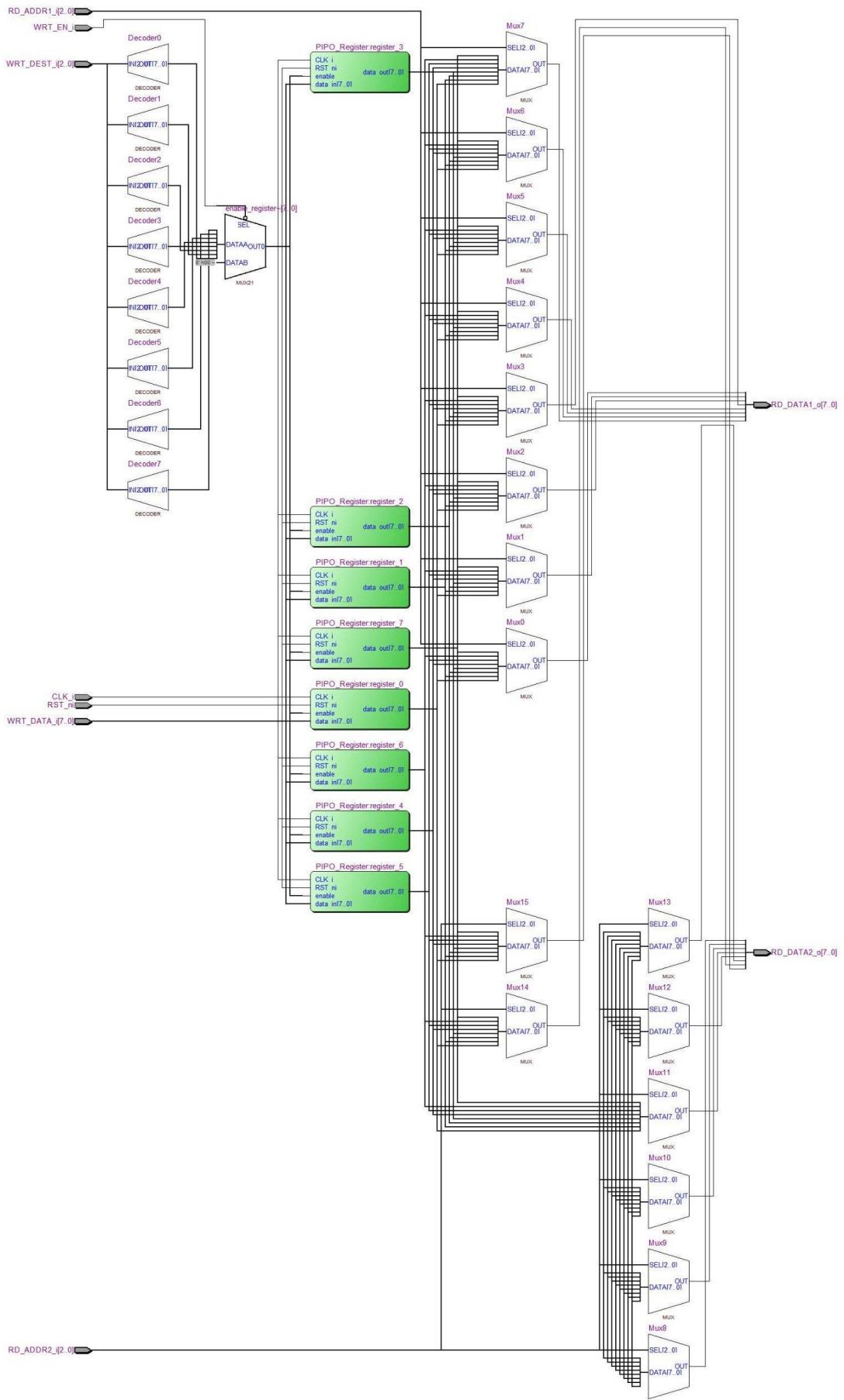
PIPO_Register register_6(
    .enable(enable_register[6]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_6)
);

PIPO_Register register_7(
    .enable(enable_register[7]),
    .CLK_i(CLK_i),
    .RST_ni(RST_ni),
    .data_in(WRT_DATA_i),
    .data_out(data_reg_7)
);

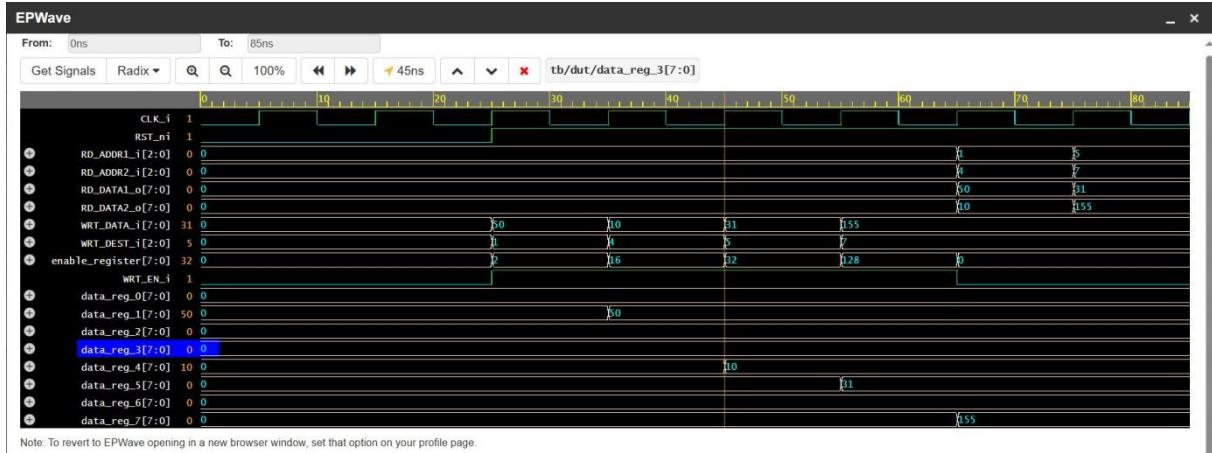
endmodule

```

Kết quả RTL viewer



## Dạng sóng mô phỏng lý tưởng từ code HDL

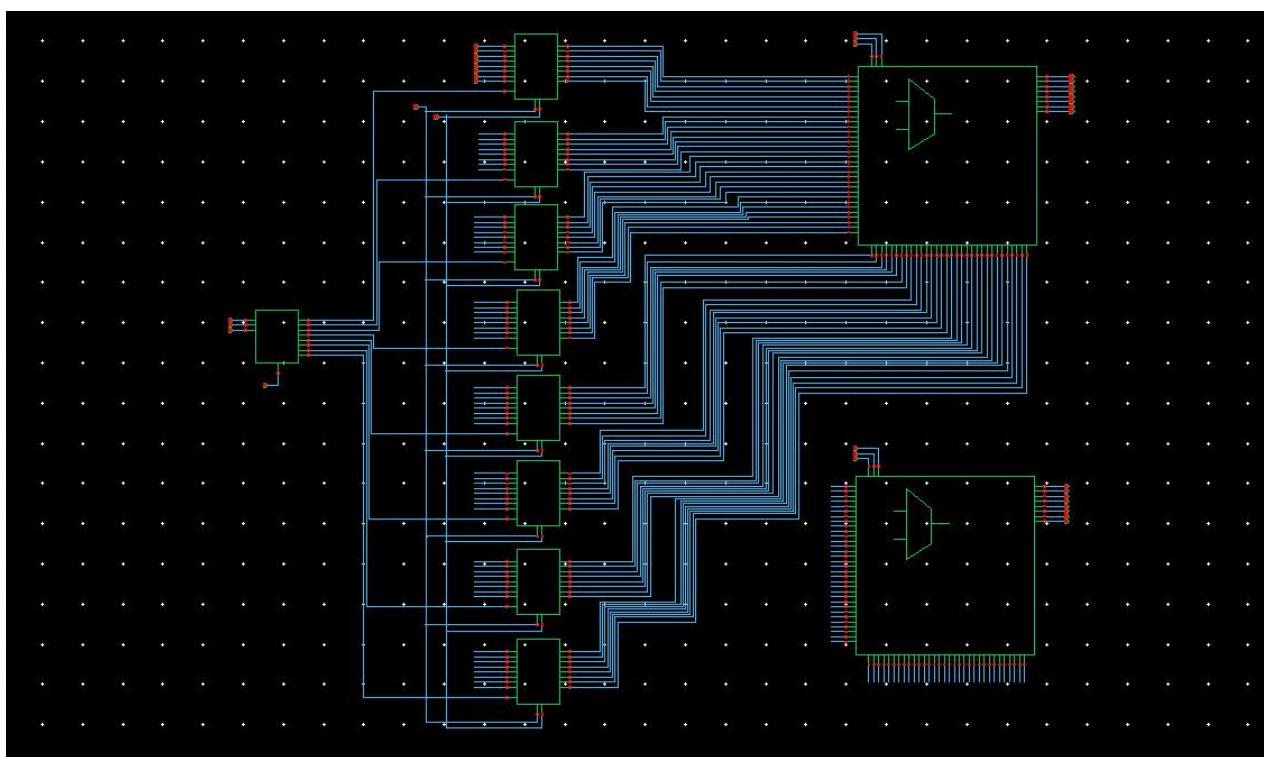
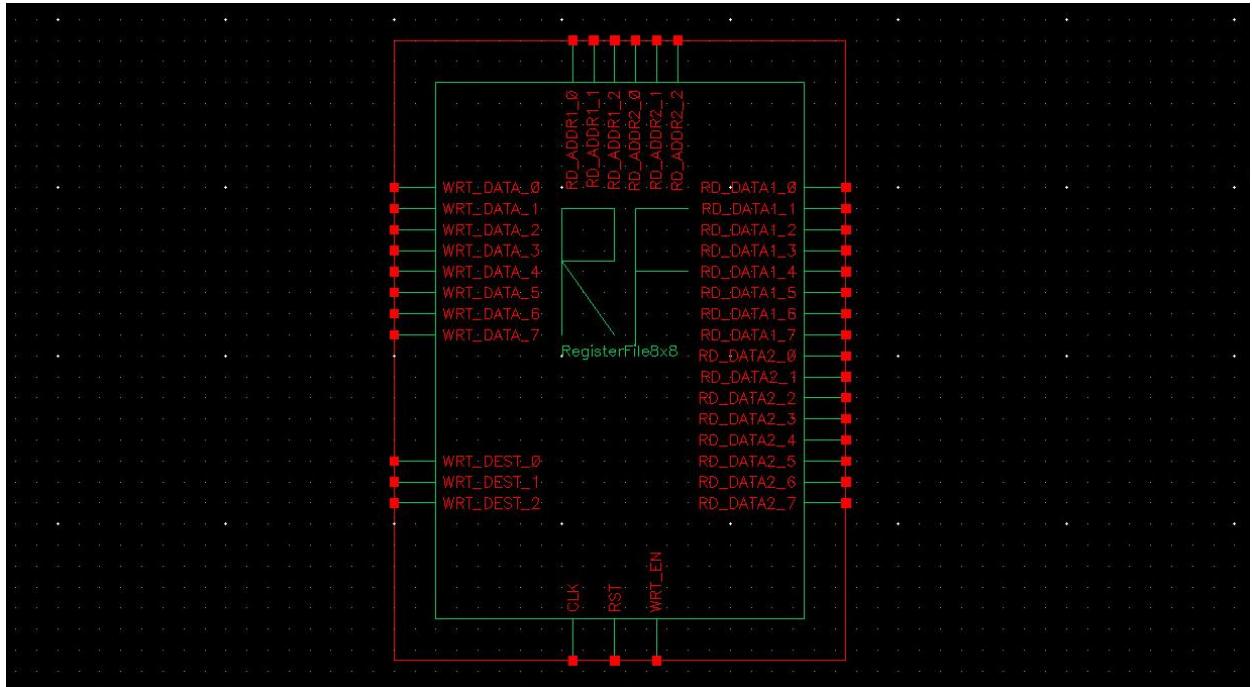


**Nhận xét:** Theo như hình trên, để dễ dàng cho việc mô phỏng và quan sát dạng sóng nên nhóm sẽ thực hiện việc ghi dữ liệu trước rồi mới đọc dữ liệu, tránh việc gây hiểu nhầm cho người đọc. Tuy nhiên, điều này không có nghĩa là khi hoạt động Register File, phải hoàn thành việc ghi dữ liệu toàn bộ 8 ô nhớ thì mới đọc dữ liệu, điều này là không đúng. Đầu tiên, nhóm sẽ ghi dữ liệu FE, FD, FC, FB, FA, F9, F8 vào lần lượt các thanh ghi (ô nhớ) từ địa chỉ 001 – 111, không ghi giá trị vào thanh ghi có địa chỉ 000, bởi vì thông thường thanh ghi này không được sử dụng cho mục đích ghi dữ liệu (theo RISC-V convention). Sau khi hoàn thành quá trình ghi dữ liệu, chúng ta tiến hành đọc dữ liệu bằng 2 ngõ ra 8-bit là rs\_data1\_o và rs\_data2\_o. Cụ thể, các địa chỉ được thiết lập ngẫu nhiên, như quan sát thấy, tại rs\_addr1\_i = 100 và rs\_addr2\_i = 101 thì rs\_data1\_o và rs\_data2\_o lần lượt đọc ra là FB và FA, điều này hoàn toàn trùng khớp với giá trị được ghi vào 2 ô nhớ 100 và 101 trước đó. Tương tự, tại rs\_addr1\_i = 010 và rs\_addr2\_i = 001 thì rs\_data1\_o và rs\_data2\_o lần lượt đọc ra là FD và FE.

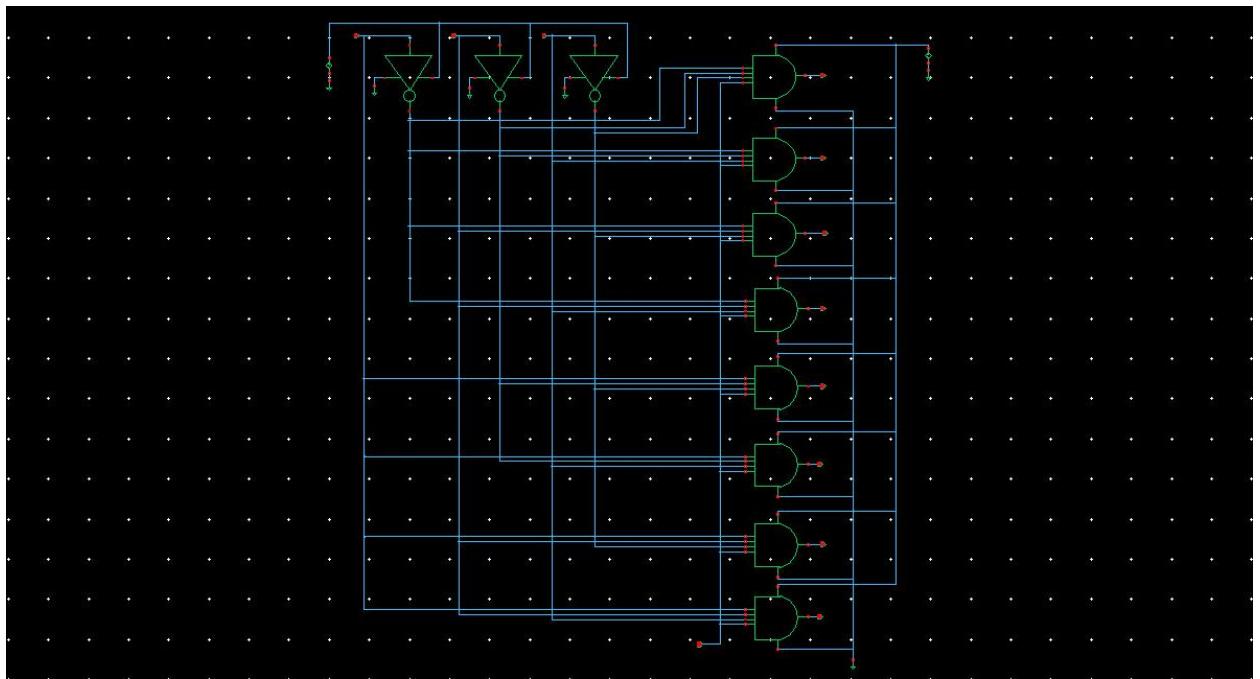
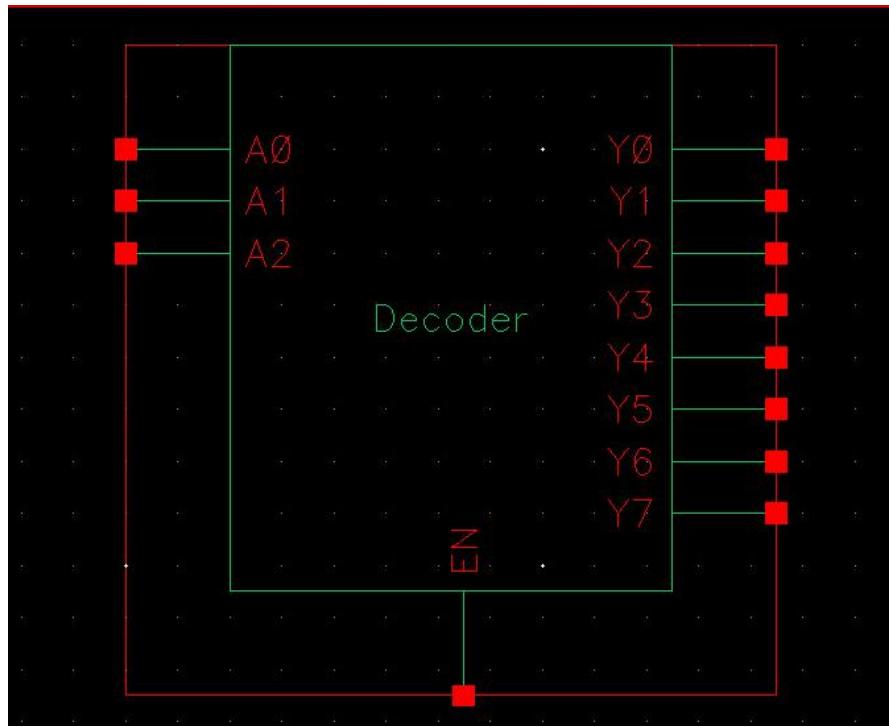
**Kết luận:** Dạng sóng mô phỏng Register File trên ngôn ngữ phần cứng systemverilog hoàn toàn chính xác theo cơ chế hoạt động của Register File trong tài liệu thí nghiệm.

➤ Sơ đồ nguyên lý thiết kế Register File 8-bit

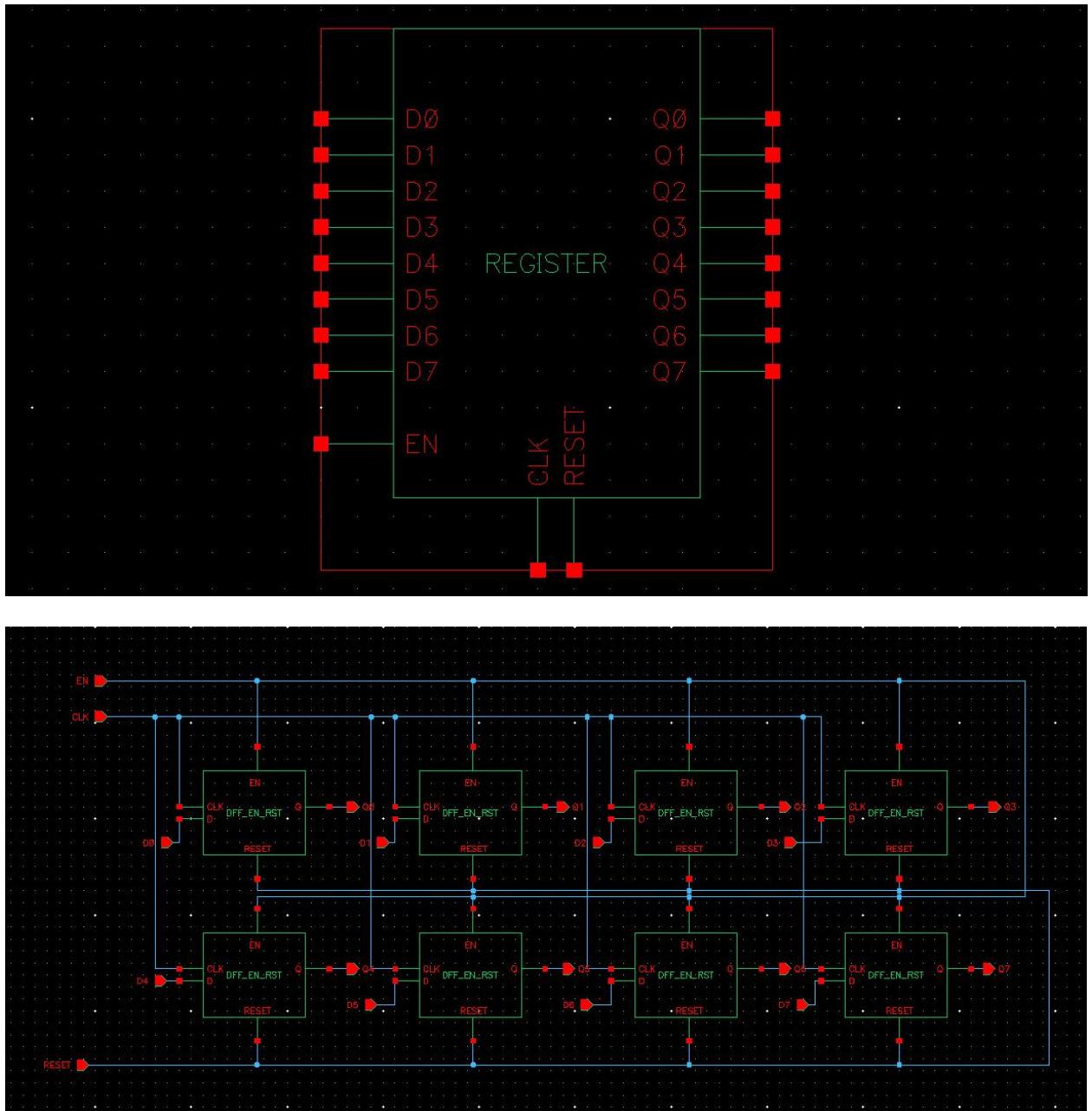
- Tổng quan register file:



- Decoder 3 to 8:

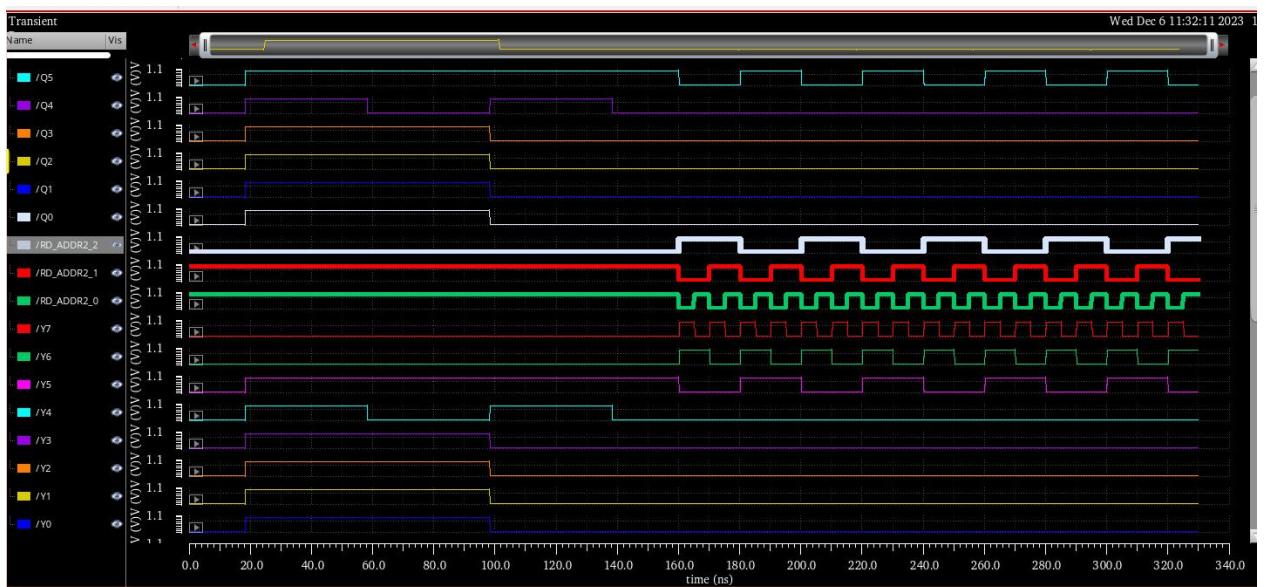
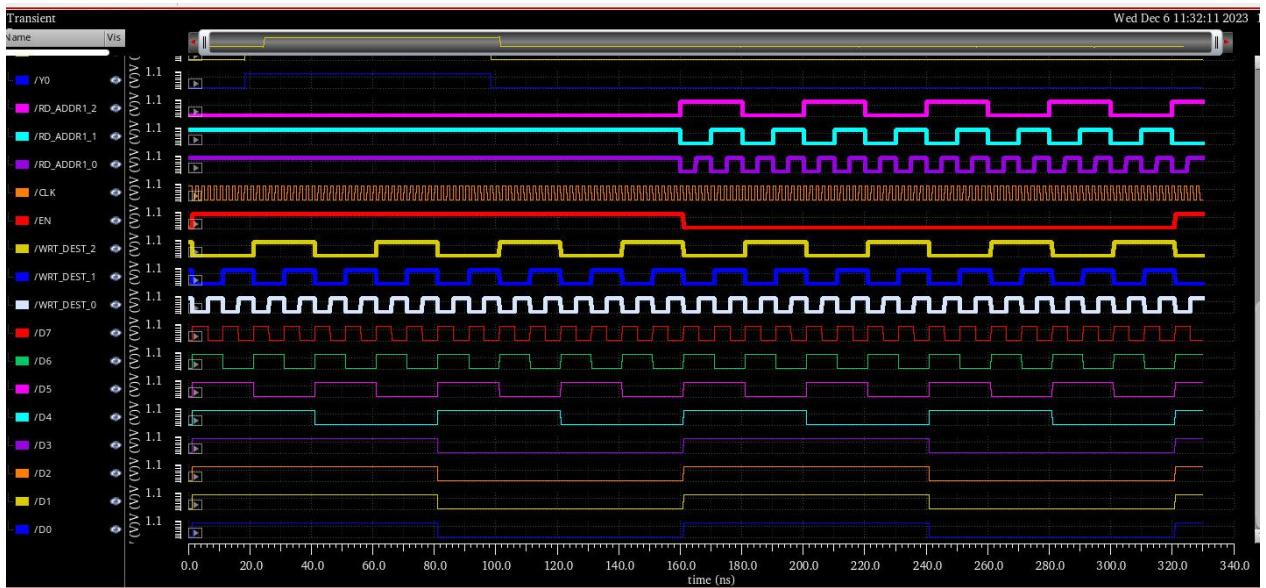


- 8 thanh ghi (ô nhớ):



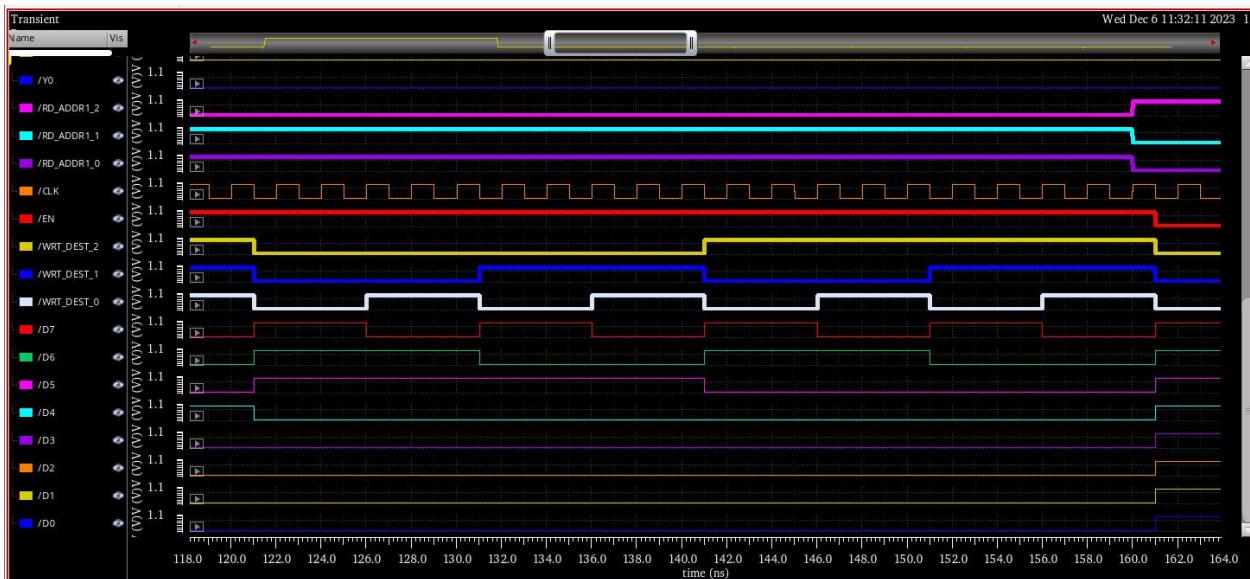
➤ Dạng sóng mô phỏng thiết kế Register File 8-bit

- Tổng quan:



**Nhận xét:** Theo như hình trên, để dễ dàng cho việc mô phỏng và quan sát dạng sóng nên nhóm sẽ thực hiện việc ghi dữ liệu trước rồi mới đọc dữ liệu, tránh việc gây hiểu nhầm cho người đọc. Tuy nhiên, điều này không có nghĩa là khi hoạt động Register File, phải hoàn thành việc ghi dữ liệu toàn bộ 8 ô nhớ thì mới đọc dữ liệu, điều này là không đúng.

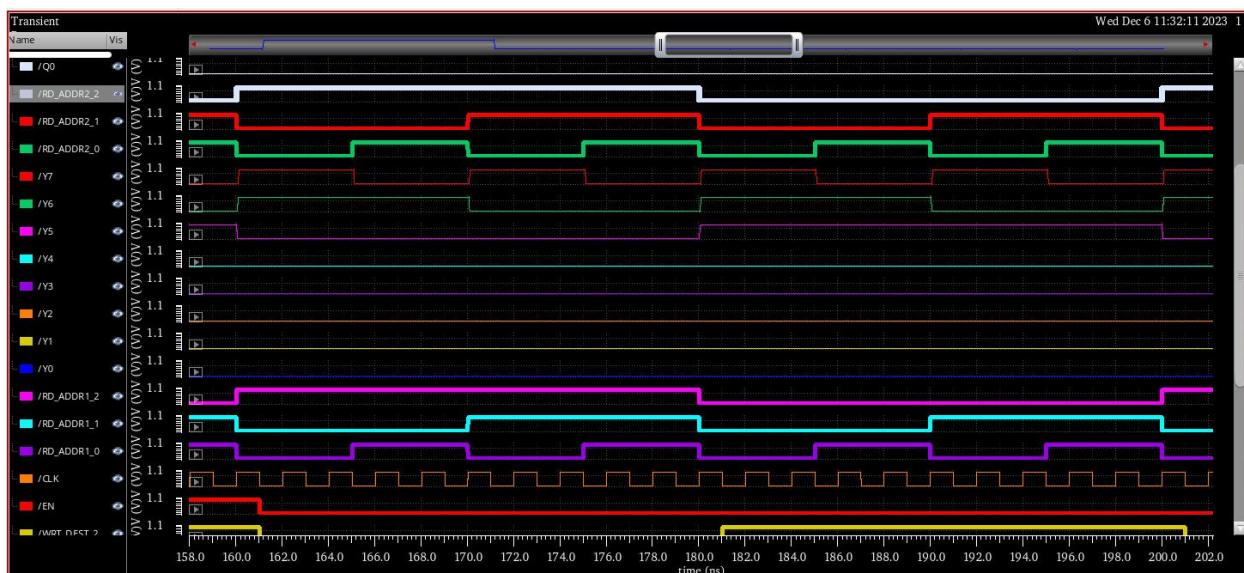
- Quá trình ghi dữ liệu:



**Nhận xét:** Theo hình mô phỏng trên, nhóm sẽ hiển thị lại kết quả dữ liệu ghi vào 8 thanh ghi như bảng bên dưới. Quá trình ghi dữ liệu được thực thi khi tín hiệu EN được lên mức 1, các ô nhớ dữ liệu sẽ không cập nhật dữ liệu ghi vào khi tín hiệu EN xuống mức 0. Điều này là hoàn toàn đúng với nguyên lý hoạt động của Register File. Ngoài ra, chúng ta sẽ quan sát waveform trong khoảng thời gian mà địa chỉ thanh ghi đích (wrt\_dest\_i) từ 000 – 111, tương ứng với việc ghi dữ liệu vào 8 thanh ghi. Sau khi hoàn tất, tín hiệu EN sẽ xuống mức 0 và đạt được dữ liệu như bảng bên dưới.

Địa chỉ	000	001	010	011	100	101	110	111
Dữ liệu	E0	60	B0	20	D0	40	80	00

- Quá trình đọc dữ liệu thanh ghi nguồn rs1:



**Nhận xét:** Theo hình mô phỏng trên, nhóm sẽ hiển thị lại kết quả dữ liệu đọc từ 8 thanh ghi từ thanh ghi nguồn rs1 như bảng bên dưới. Quá trình ghi dữ liệu được thực thi khi tín hiệu EN được xuống mức 0. Dựa theo kết quả từ bảng bên dưới, chúng ta thấy rằng giá trị đọc ra từ các ô nhớ là hoàn toàn trùng khớp với giá trị đã ghi vào các ô nhớ. Điều này cho thấy kết quả mô phỏng hoàn toàn chính xác với hoạt động của Register File. Ngoài ra, chúng ta sẽ quan sát waveform trong khoảng thời gian mà địa chỉ thanh ghi nguồn rs1 (rd\_addr1\_o) từ 000 – 111, tương ứng với việc đọc dữ liệu từ 8 thanh ghi.

Địa chỉ	000	001	010	011	100	101	110	111
Dữ liệu	E0	60	B0	20	D0	40	80	00

- Quá trình đọc dữ liệu thanh ghi nguồn rs2:



**Nhận xét:** Theo hình mô phỏng trên, nhóm sẽ hiển thị lại kết quả dữ liệu đọc từ 8 thanh ghi từ thanh ghi nguồn rs2 như bảng bên dưới. Quá trình ghi dữ liệu được thực thi khi tín hiệu EN được xuống mức 0. Dựa theo kết quả từ bảng bên dưới, chúng ta thấy rằng giá trị đọc ra từ các ô nhớ là hoàn toàn trùng khớp với giá trị đã ghi vào các ô nhớ. Điều này cho thấy kết quả mô phỏng hoàn toàn chính xác với hoạt động của Register File. Ngoài ra, chúng ta sẽ quan sát waveform trong khoảng thời gian mà địa chỉ thanh ghi nguồn rs2 (rd\_addr2\_o) từ 000 – 111, tương ứng với việc đọc dữ liệu từ 8 thanh ghi.

Địa chỉ	000	001	010	011	100	101	110	111
---------	-----	-----	-----	-----	-----	-----	-----	-----

Dữ liệu	E0	60	B0	20	D0	40	80	00
---------	----	----	----	----	----	----	----	----

## ONE MORE THING

After completing this lab, please answer the following questions.

- How much time did you spend completing this lab?

=> Nhóm 1 dành khoảng 2 tuần để hoàn thành bài lab 4 (1 tuần dành 4-5 buổi, mỗi buổi 3 tiếng để thực hiện). Điều này tương đương khoảng 30 tiếng hoàn thành tổng cộng.

- Is there any problem when you do? Please let me know

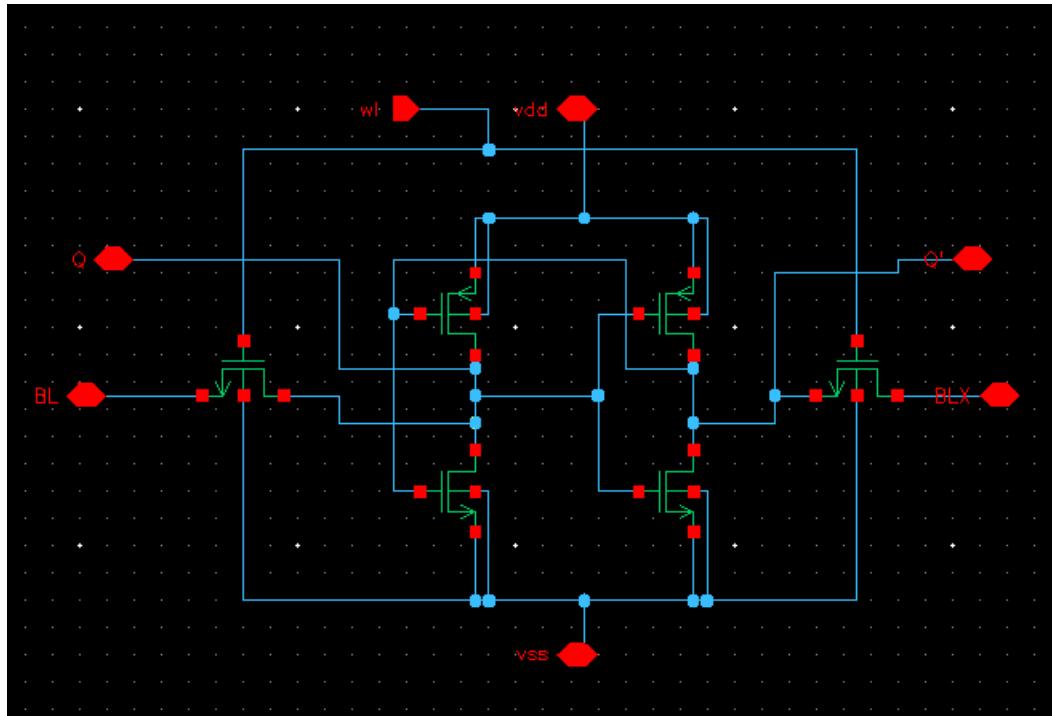
=> Không có vấn đề gì xảy ra đối với phần mềm Cadence trong thời gian hoàn thành lab 4 vì nhóm 1 đã tải và sử dụng phần mềm trên máy ảo. Sở dĩ nhóm phải tải và sử dụng phần mềm trên máy ảo là bởi vì server sau khi được update xong thì có 2 trên 3 thành viên của nhóm không đăng nhập vào được. Việc tìm cách để cố gắng để đăng nhập vào server khá mất thời gian và công sức nên nhóm đã quyết định tải và sử dụng phần mềm trên máy ảo.

=> Tuy nhiên, việc sử dụng máy ảo để chạy phần mềm xảy ra 1 vấn đề nhỏ đó là các thành viên không kết nối các thiết kế của mình với nhau được, dẫn đến việc chỉ có 1 trong ba thành viên của nhóm sử dụng phần mềm thiết kế (làm tổn rất nhiều thời gian). Việc này nếu các thành viên trong nhóm đăng nhập được vào server và cùng nhau làm thì sẽ có thể giảm thiểu được khoảng phân nửa thời gian thực hiện. Vậy nên, nhóm mong rằng bộ môn có gắng xây dựng server ổn định hơn để cho các khóa sau có thể giảm thiểu tối đa trở ngại khi đăng nhập vào server như nhóm 1 đã gặp phải.

## Chương 5: INTRODUCTION TO MEMORY CIRCUIT DESIGN

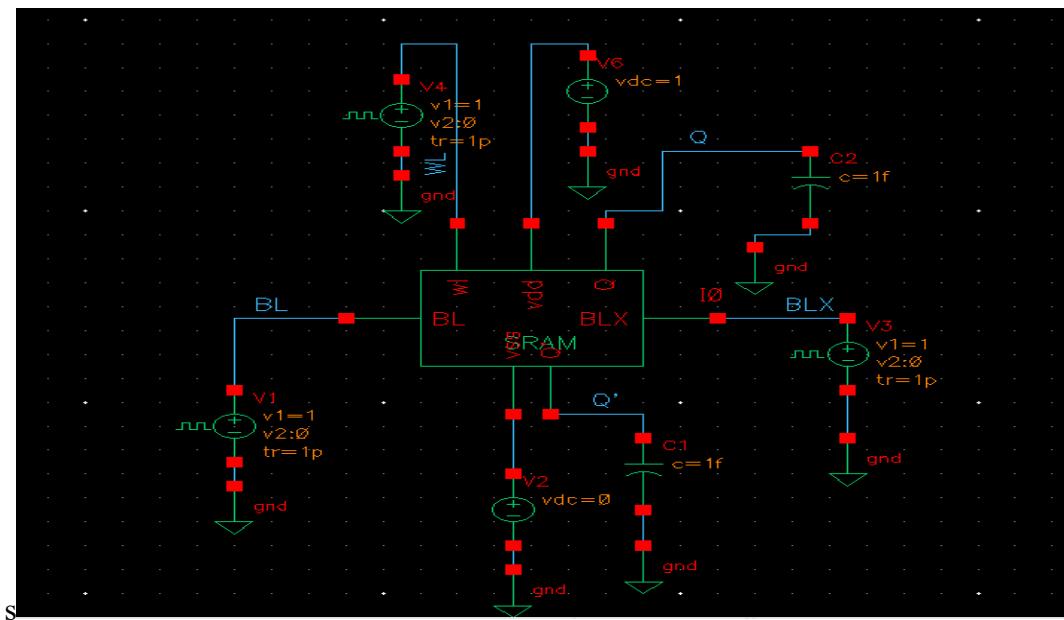
### 5.1. Experiment 1

Schematic của SRAM cơ bản

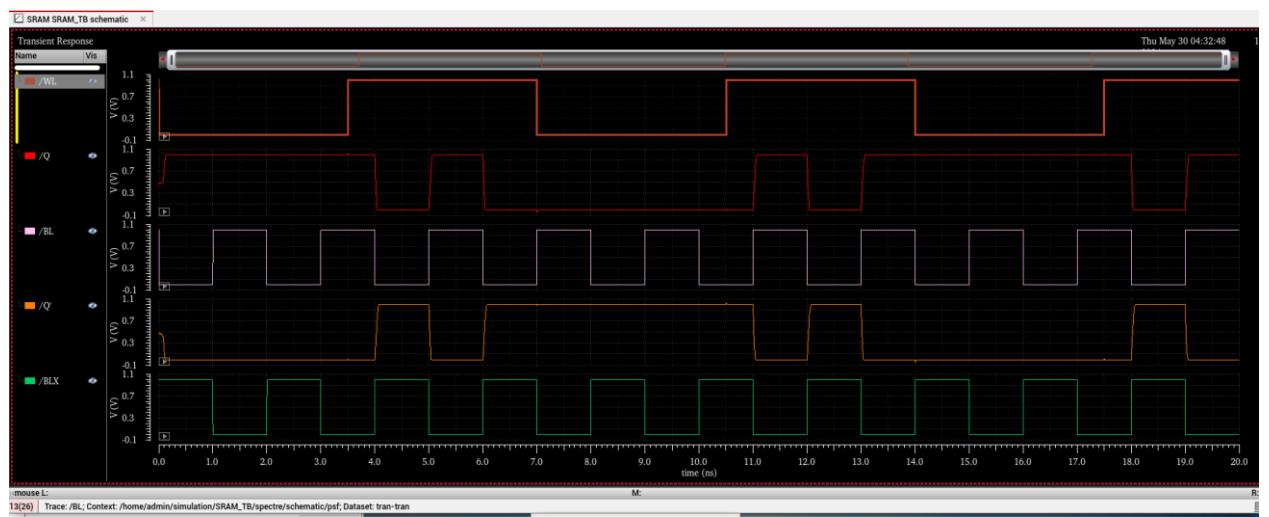


Hình 5-1 Schematic của SRAM

Tiến hành ghi các giá trị vào SRAM:

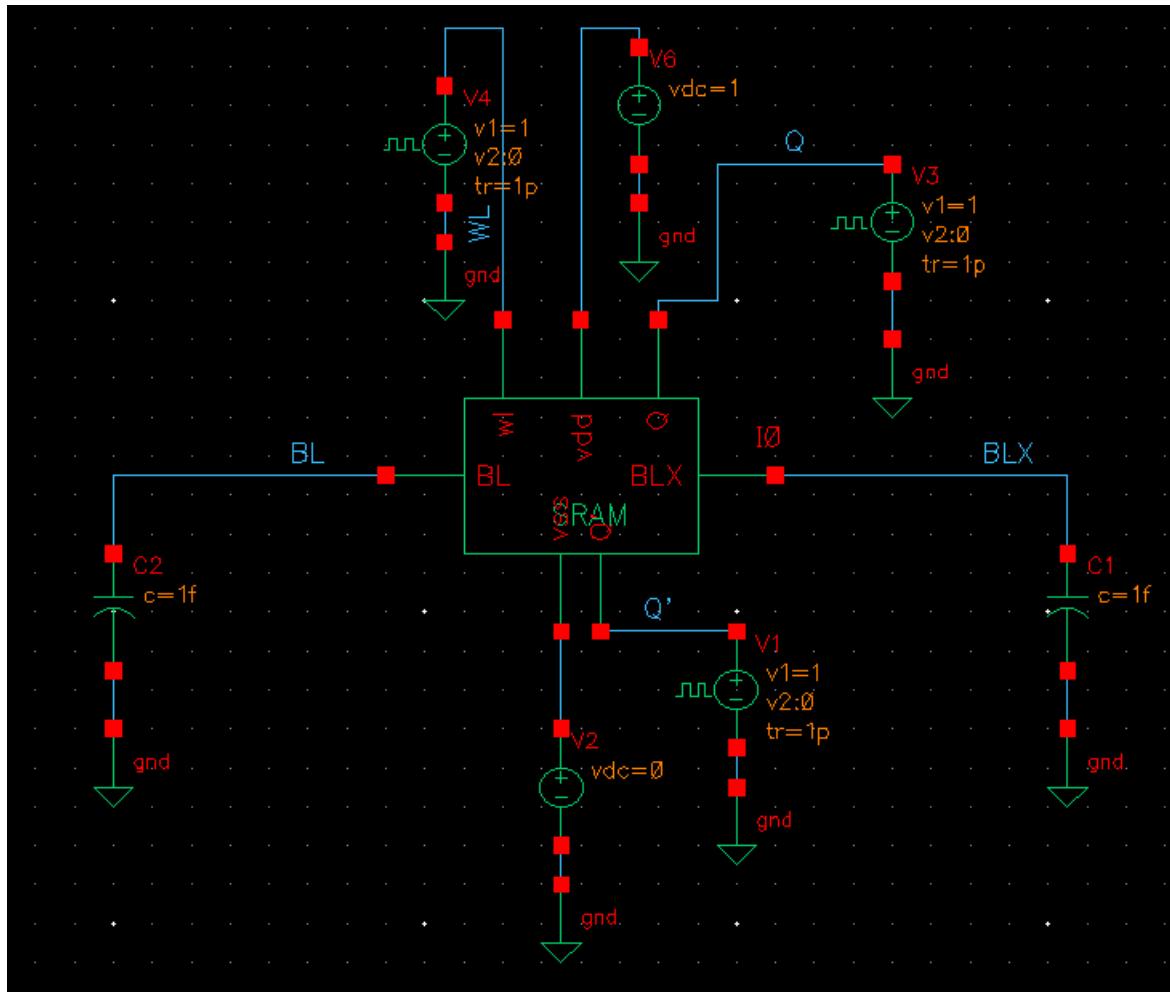


Hình 5-2 Test bench for Write to SRAM.

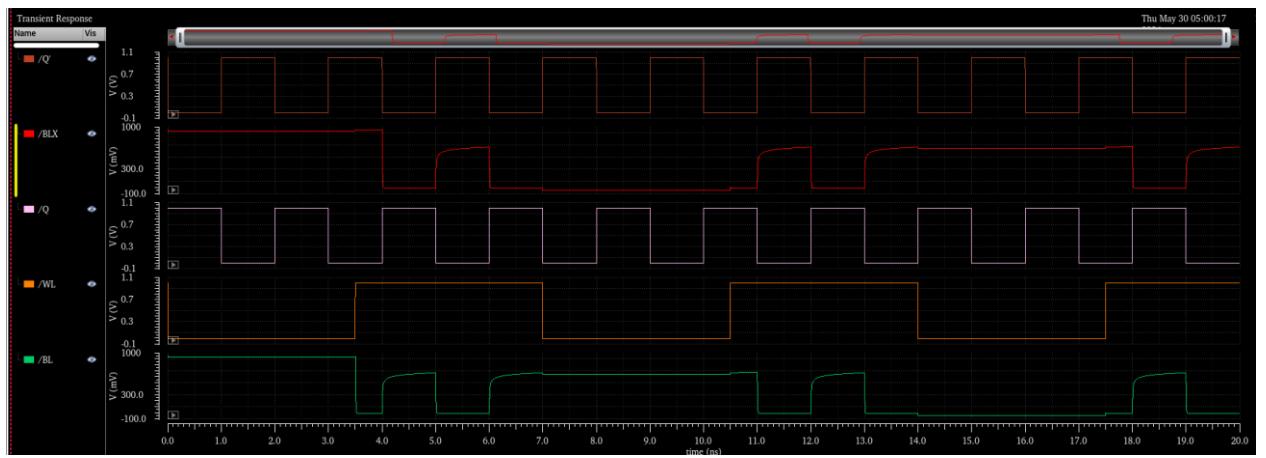


Hình 5-3 Result of Write.

Sram read

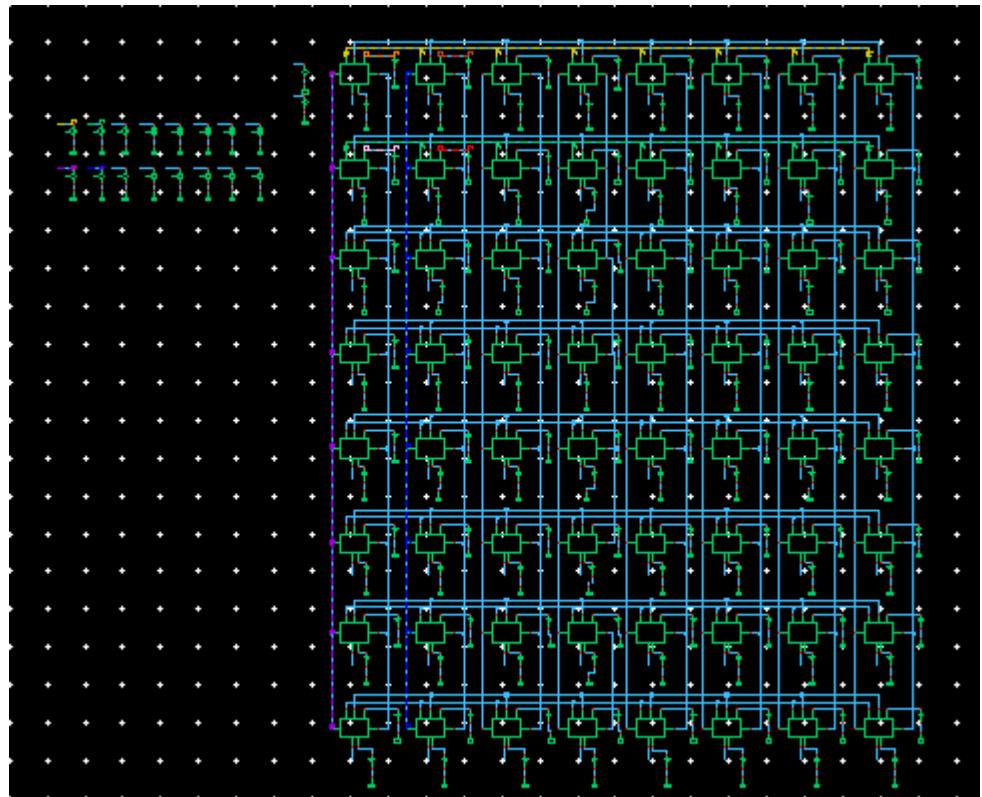


Hình 5-4 Read test bench.

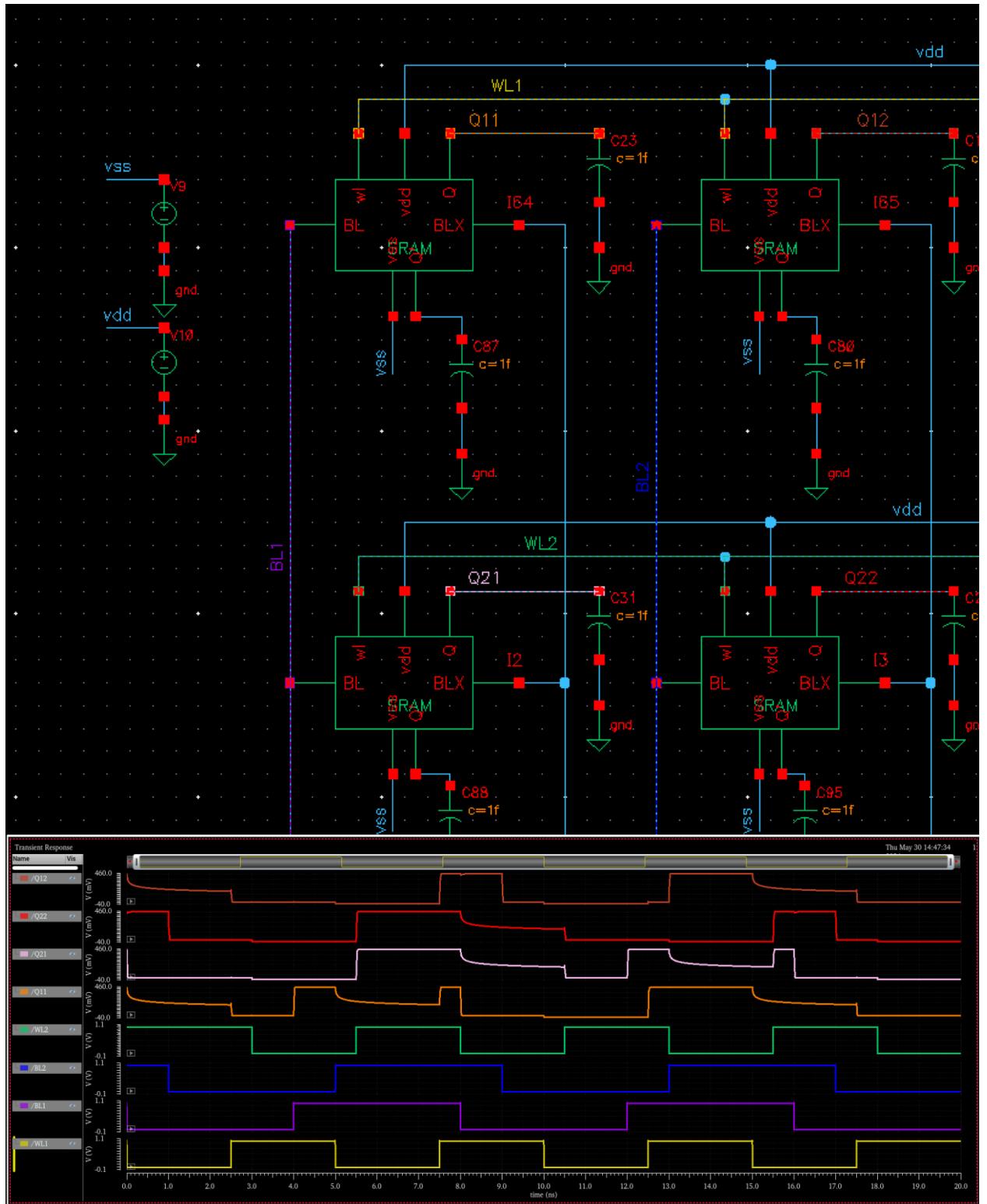


Hình 5-5 Result of read.

## SRAM 8x8



Hình 5-6 8x8 Sram.



Hình 5-7 Kết quả của Test Cell 11 12 21 22.

➔ Các quá trình ghi, đọc Sram đúng với lý thuyết từ 1 cell tới 8x8

## Chương 6: IC DESIGN FLOW

### 6.1 Giới thiệu chung

#### 6.1.1 Định nghĩa về ASIC

Mạch tích hợp được làm từ wafer silicon, với mỗi wafer chứa hàng trăm khuôn. ASIC (Application Specific Integrated Circuit) là một mạch tích hợp dành riêng cho ứng dụng. Một mạch tích hợp được thiết kế được gọi là ASIC nếu chúng ta thiết kế ASIC cho ứng dụng cụ thể. Ví dụ về ASIC bao gồm: chip được thiết kế cho vệ tinh, chip được thiết kế cho ô tô, chip được thiết kế như một giao diện giữa bộ nhớ và CPU... Ví dụ về loại vi mạch không được gọi là ASIC bao gồm bộ nhớ, bộ vi xử lý...



Các ASIC

Có các loại ASIC bao gồm:

- + *Full-Custom ASIC*: Đối với loại ASIC này, kỹ sư sẽ thiết kế tất cả hoặc một số logic cell, layout cho một chip đó. Người thiết kế không sử dụng được các công xác định trước trong thiết kế. Mọi phần của thiết kế đều được làm từ đầu.
- + *Standard Cell ASIC*: Người thiết kế sử dụng các logic cell được thiết kế trước như công AND, công NOR, v.v ... Các công này được gọi là Standard Cell. Lợi thế của Standard Cell là các nhà thiết kế tiết kiệm thời gian, tiền bạc và giảm rủi ro bằng cách sử dụng thư viện Standard Cell được thiết kế trước và thử nghiệm trước. Ngoài ra mỗi Standard Cell có thể được tối ưu hóa riêng biệt. Thư viện Standard Cell được thiết kế bằng cách sử dụng phương pháp tùy chỉnh đầy đủ, nhưng bạn có thể sử dụng các thư viện đã được thiết kế sẵn này trong

thiết kế. Cách thiết kế này cung cấp cho một nhà thiết kế sự linh hoạt giống như thiết kế tùy chỉnh (Full-Custom ASIC), nhưng giảm rủi ro.

+ *Gate Array ASIC*: Trong loại ASIC này, các bóng bán dẫn được xác định trước trong tám silicon. Mẫu bóng bán dẫn được xác định trước trên mảng cổng được gọi là mảng cơ sở và phần tử nhỏ nhất trong mảng cơ sở được gọi là cell cơ sở. Các bộ cục cell cơ sở giống nhau cho mỗi cell, chỉ có sự liên kết giữa các cell và bên trong các cell được tùy chỉnh.

Khi thiết kế một chip, các mục tiêu sau được xem xét:

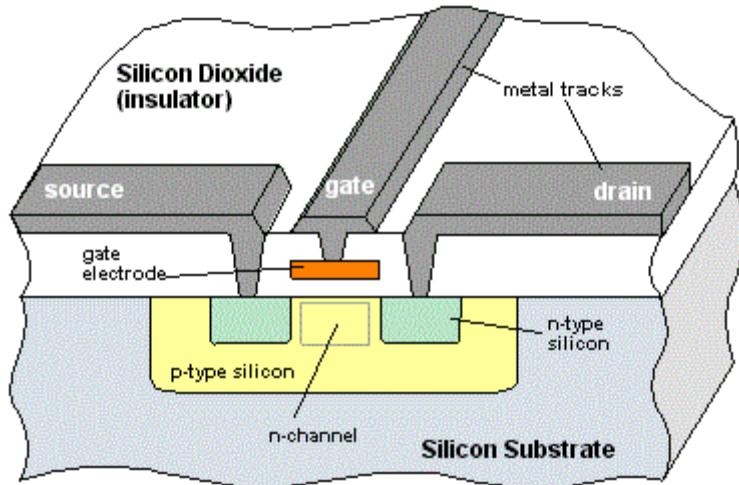
1. *Tốc độ*.
2. *Diện tích*.
3. *Công suất*.
4. *Thời gian đưa ra thị trường*.

Để thiết kế một ASIC, người ta cần phải hiểu rõ về công nghệ CMOS. Một số phần tiếp theo cung cấp một cái nhìn cơ bản về công nghệ CMOS.

### 6.1.2 Công nghệ CMOS

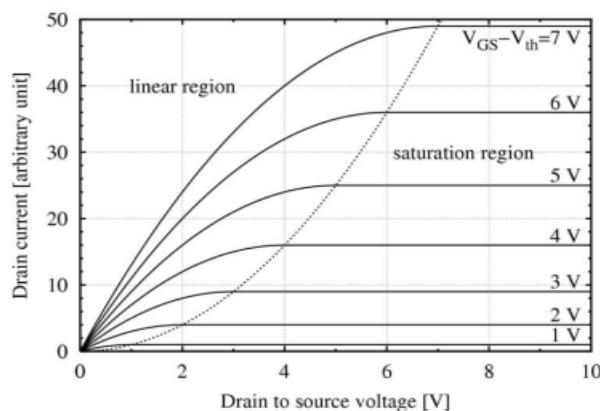
#### 6.1.2.1 MOS Transistor

MOSFET là viết tắt của Metal Oxide Semiconductor field effect transistor (transistor hiệu ứng trường). MOS là phần tử cơ bản trong thiết kế của một mạch tích hợp quy mô lớn. Nó được điều khiển bằng điện áp. Những bóng bán dẫn này được hình thành bao gồm một lớp bán dẫn, thường là một lớp mảnh hoặc wafer tạo ra bởi tinh thể silicon; một lớp của SiO<sub>2</sub> và một lớp kim loại. Các lớp này được tạo theo mô hình sẵn cho phép các bóng bán dẫn được hình thành trong vật liệu bán dẫn. Bóng bán dẫn MOS bao gồm ba cực: Nguồn, Cổng và Máng. Cực nguồn và Máng khá giống nhau và được dán nhãn tùy thuộc vào những gì chúng được kết nối. Nguồn là thiết bị đầu cuối, hoặc nút, hoạt động như nguồn của các hạt mang điện tích, rời khỏi nguồn và đi đến máng. Trong trường hợp MOSFET kênh N (NMOS), nguồn là cực âm hơn của các thiết bị đầu cuối; trong trường hợp của MOSFET kênh P (PMOS), nguồn là cực dương hơn. Diện tích dưới cổng oxit là được gọi là “kênh”. Dưới đây là hình của một Transistor MOS.



*Cấu tạo của MOS transistor*

Các bóng bán dẫn thường cần đặt ngưỡng điện áp ban đầu để hình thành kênh. Khi không có kênh hình thành, bóng bán dẫn được cho là ở trong vùng tắt (cut off region). Các điện áp mà tại đó bóng bán dẫn bắt đầu dẫn điện (một kênh bắt đầu hình thành giữa nguồn và công) được gọi là điện áp ngưỡng (Voltage Threshold). Bóng bán dẫn tại thời điểm này được cho là trong vùng tuyến tính (linear region). Bóng bán dẫn được cho là đi vào vùng bão hòa (saturation region) khi ở đó không còn thêm điện tích đi từ cực nguồn đến cực máng.



*Đặc tuyến điện áp dòng Id và Vds.*

Công nghệ CMOS được tạo thành từ cả hai loại NMOS và CMOS. Thiết bị bán dẫn bù oxit kim loại (Complementary Metal-Oxide Semiconductors- CMOS) là thiết bị phổ biến nhất được sử dụng ngày nay với mật độ cao và có số lượng lớn bóng bán dẫn được tìm thấy trong nhiều loại mạch. Cấu trúc CMOS phổ biến vì công suất tiêu thụ thấp, tốc độ xử lý tín hiệu hoạt động cao và dễ thực hiện các cấp độ của bóng bán dẫn. Các mạng bóng bán dẫn kênh p và kênh n bù sung nhau được sử dụng để kết nối đầu ra của thiết bị logic với nguồn cung

cấp  $V_{DD}$  hoặc  $V_{SS}$  để đặt trạng thái logic đầu vào. Các bóng bán dẫn MOSFET có thể được coi như các công tắc đơn giản và các công tắc phải được bật để cho phép dòng điện chạy giữa cực Nguồn và Máng.

\* *Ví dụ:* Tạo công Inverter CMOS chỉ cần một PMOS và một NMOS. Bóng bán dẫn NMOS cung cấp kết nối công tắc ON nối với đất khi đầu vào là logic cao. Tụ điện tải đầu ra được xả và đầu ra được điều khiển đến mức logic 0. Bóng bán dẫn PMOS ON cung cấp kết nối với nguồn điện  $V_{DD}$  khi đầu vào của mạch Inverter ở mức logic thấp. Tụ điện tải đầu ra được tính vào  $V_{DD}$ . Đầu ra được điều khiển đến logic 1.

Điện dung tải đầu ra của công logic bao gồm:

+ *Điện dung ký sinh:* Điện dung giữa cực Cổng và Máng  $C_{gd}$  (của cả NMOS và PMOS).

+ *Điện dung bên ngoài:* Điện dung của dây kết nối và cá đầu vào điện dung của Fan out.

=> Trong CMOS, chỉ có một trình điều khiển, nhưng cực Cổng có thể điều khiển nhiều công nhất có thể. Trong công nghệ CMOS, đầu ra luôn điều khiển ngõ vào công CMOS khác.

Các hạt mang điện tích cho bóng bán dẫn PMOS là lỗ trống và hạt mang điện cho NMOS là các electron. Độ linh động của các electron gấp hai lần độ linh động của các lỗ trống. Vì điều này ngõ ra rise và fall time là khác nhau. Để làm cho nó giống nhau, tỷ lệ W/L của PMOS bóng bán dẫn được tạo ra gấp đôi so với bóng bán dẫn NMOS. Bằng cách này, PMOS và các bóng bán dẫn NMOS sẽ có cùng khả năng dẫn điện. Trong thư viện Standard cell, chiều dài “L” của bóng bán dẫn luôn không đổi. Các giá trị chiều rộng “W” được thay đổi khác nhau cho mỗi công. Điện trở tỷ lệ với  $(L/W)$ , vì vậy nếu chiều rộng càng tăng thì điện trở càng giảm.

#### 6.1.2.2 Công suất trong vi mạch CMOS

Phần lớn điện năng tiêu thụ trong vi mạch CMOS là do quá trình nạp và xả của tụ điện. Phần lớn vấn đề thiết kế vi mạch CMOS có công suất thấp là thất thoát năng lượng. Các nguồn tiêu thụ công suất chính là:

**Dynamic Switching Power:** do sạc và xả của tụ điện.

+ Sự chuyển đổi đầu ra từ thấp đến cao lấy năng lượng từ nguồn điện

+ Quá trình chuyển đổi từ cao xuống thấp làm tiêu hao năng lượng được lưu trữ trong bóng bán dẫn CMOS.

- + Với tần số f, của quá trình chuyển đổi từ thấp đến cao, tổng công suất tiêu hao sẽ là:  $P = Điện dung tải * V_{dd} * V_{dd} * f$

**Dòng ngắn mạch:** Xảy ra khi rise và fall time ở ngõ vào của cổng là lớn hơn thời gian rise và fall time của ngõ ra.

**Dòng bị rò rỉ:** Được gây ra bởi hai lý do

- + Rò rỉ diode phân cực ngược trên bóng bán dẫn ở cực Máng: Điều này xảy ra trong thiết kế CMOS, khi một bóng bán dẫn tắt và bóng bán dẫn hoạt động sẽ sạc lên/xuống cực Máng bằng cách sử dụng điện thế lớn của bóng bán dẫn khác. Ví dụ: Hãy xem xét một inverter có điện áp đầu vào cao, đầu ra thấp có nghĩa là NMOS đang bật và PMOS tắt. Phần lớn PMOS là kết nối với VDD. Do đó có một điện áp xả thành bulk VDD, gây ra dòng rò diode.
- + Rò rỉ ở ngưỡng phụ thông qua kênh tới bóng bán dẫn/thiết bị “OFF”.

#### 6.1.2.3 Cổng truyền CMOS

Một bóng bán dẫn PMOS được kết nối song song với một bóng bán dẫn NMOS để tạo thành một cổng truyền dữ liệu (Transmission Gate). Cổng truyền chỉ truyền giá trị ở đầu vào đến đầu ra. Nó bao gồm cả NMOS và PMOS vì bóng bán dẫn PMOS truyền tín hiệu “1” và bóng bán dẫn NMOS truyền tín hiệu “0”. Những lợi thế của việc sử dụng cổng truyền là:

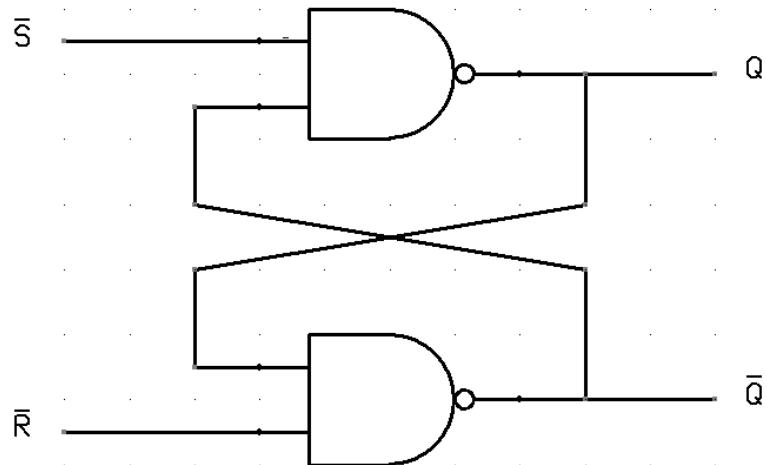
- + Nó cho thấy những đặc điểm tốt hơn một công tắc.
- + Điện trở của mạch giảm, vì các tranzistor được mắc song song.

\* *Phản tử tuần tự (Sequential Element):*

Trong CMOS, một phản tử lưu trữ giá trị logic (bằng cách có một vòng phản hồi) được gọi là phản tử tuần tự. Một ví dụ đơn giản nhất về một phản tử tuần tự sẽ là hai bộ inverter kết nối trở lại với nhau. Có hai loại phản tử tuần tự cơ bản:

#### **Latch**

Hai con inverter được kết nối ngược trở lại với nhau, khi kết nối với cổng truyền, với một đầu vào điều khiển, tạo thành một Latch. Khi đầu vào điều khiển là mức cao (logic “1”), cổng truyền được bật và bất kỳ giá trị nào ở đầu vào “D” đều chuyển đến đầu ra. Khi đầu vào điều khiển ở mức thấp, cổng truyền bị tắt và các bộ inverter được kết nối trở lại để giữ giá trị. Chốt được gọi là transparent gate vì khi đầu vào “D” thay đổi, ngõ ra cũng thay đổi tương ứng.



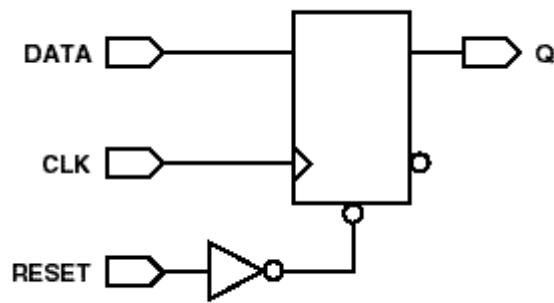
*Latch*

### *Flip-Flop*

Một flip flop được cấu tạo từ hai latch nối tiếp nhau. Latch đầu tiên là được gọi là master latch và Latch thứ hai được gọi là slave latch. Sự kiểm soát ngõ vào cho công truyền trong trường hợp này được gọi là clock.

+ Khi ngõ vào clock ở mức cao, công truyền của latch đầu tiên là được bật và đầu vào “D” được chốt bởi 2 inverter được kết nối với nhau (về cơ bản giống như một con latch hay transparent gate). Ngoài ra, do sự đảo clock đầu vào khi đến công truyền của latch thứ hai, công truyền của latch này không “bật” và nó giữ giá trị trước đó.

+ Khi clock ở mức thấp, latch thứ hai được bật và sẽ cập nhật giá trị ở đầu ra với những gì mà chốt chính được lưu trữ khi đầu vào ở clock cao. Slave latch sẽ giữ giá trị mới này ở đầu ra bất kể những thay đổi ở đầu vào ở master latch khi clock ở mức thấp. Khi clock lên mức cao trở lại, giá trị ở đầu ra của slave latch được lưu trữ và quá trình được lặp lại một lần nữa.



X8598

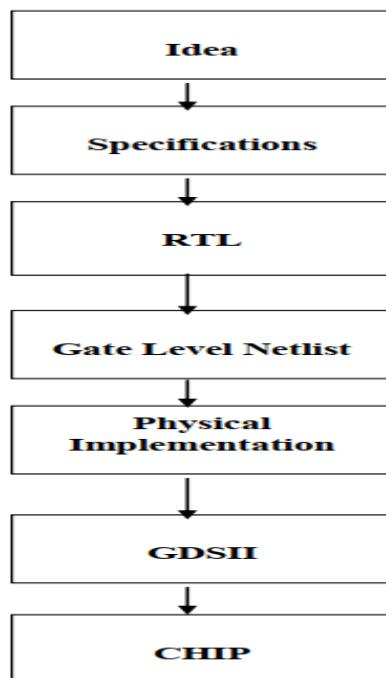
*D Flip-Flop*

### 6.1.3 Tổng quan về ASIC flow

#### 6.1.3.1 Giới thiệu

Để thiết kế một con chip, người ta cần có ý tưởng (Idea) về chính xác thứ mà người ta muốn thiết kế. Bước đầu tiên để biến ý tưởng thành một con chip là đưa ra các đặc tả kỹ thuật (System Specification).

Bước tiếp theo trong quy trình là đưa ra Sự mô tả cấu trúc và chức năng (Structural and Functional Description). Có nghĩa là tại thời điểm này, người ta phải quyết định loại kiến trúc (cấu trúc) bạn muốn sử dụng cho thiết kế, ví dụ như RIS /CISC, ALU, pipelining, v.v. Để dễ dàng hơn trong việc thiết kế một hệ thống phức tạp; nó thường được chia thành nhiều các hệ thống con. Chức năng của các hệ thống con này phải phù hợp với các thông số kỹ thuật. Tại điểm này, mối quan hệ giữa các hệ thống con khác nhau và với hệ thống cấp cao nhất là cũng được xác định.



*Simple ASIC Design Flow*

Ở các hệ thống con, khi hệ thống cấp cao nhất đã được xác định, cần phải được thực hiện. Nó là được thực hiện bằng cách sử dụng hàm biểu diễn logic (Biểu thức Boolean), Finite state machine, Combinatorial, Logic tuần tự, Schematic, v.v .... Bước này được gọi là Thiết kế logic ở mức thanh ghi (Register Transfer Level- RTL). Về cơ bản, RTL mô tả một số hệ thống con. Nó phải phù hợp với mô tả chức năng. RTL thường được thể hiện trong ngôn ngữ Verilog hoặc VHDL. Verilog và VHDL là ngôn ngữ mô tả phần cứng. Mô tả phần cứng ngôn

ngữ (HDL) là một ngôn ngữ được sử dụng để mô tả một hệ thống kỹ thuật số, ví dụ: một công tắc mạng, bộ vi xử lý hoặc bộ nhớ hoặc một flip-flop đơn giản. Điều này có nghĩa là, bởi sử dụng HDL người ta có thể mô tả bất kỳ phần cứng nào (kỹ thuật số) ở bất kỳ cấp độ nào. Việc xác minh chức năng/logic được thực hiện ở giai đoạn này để đảm bảo được thiết kế RTL phù hợp với ý tưởng.

Sau khi xác minh chức năng (Functional Verification) hoàn tất, RTL được chuyển đổi thành Netlist cấp cổng (Gate Level Netlist). Bước này được gọi là Logic/RTL synthesis. Điều này được thực hiện bởi tool tổng hợp như Trình biên dịch thiết kế (Design Compiler- Synopsys), Trình biên dịch RTL (RTL Compiler- Cadence) vv ... Một tool tổng hợp lấy mô tả phần cứng RTL và một thư viện standard cell làm đầu vào và xuất một file netlist cấp cổng ở đầu ra. Thư viện standard cell là khối xây dựng cơ bản cho thiết kế vi mạch ngày nay. Các constraint như timing, diện tích, khả năng kiểm tra, và công suất được xem xét. Các tool tổng hợp cố gắng đáp ứng các constraint, bằng cách tính toán chi phí triển khai khác nhau. Sau đó, nó cố gắng tạo ra một cách triển khai cấp cổng tốt nhất cho một tập hợp các ràng buộc nhất định và quy trình mục tiêu. Kết quả ở file netlist cấp cổng một mô tả cấu trúc hoàn thiện chỉ với các standard cell ở các nhánh của thiết kế. Tại giai đoạn này, nó cũng được xác minh xem chuyển đổi cấp cổng đã được chính xác chưa thực hiện bằng cách thực hiện loạt mô phỏng.

Bước tiếp theo trong quy trình ASIC là thực hiện dưới cấp độ vật lý (Physical Implementation) của file netlist dưới dạng cổng. Netlist cấp cổng được chuyển đổi thành biểu diễn hình học. Hình học đại diện không là gì khác ngoài layout của thiết kế. Layout được thiết kế theo các quy tắc thiết kế được chỉ định trong thư viện. Các quy tắc thiết kế không có gì khác ngoài các hướng dẫn dựa trên về những giới hạn của quá trình chế tạo. Bước Physical Implementation bao gồm ba bước chính như: Floor planning->Placement->Routing. File được tạo ở đầu ra của bước này là tệp GDSII. Đây là tệp được xưởng đúc sử dụng để chế tạo ASIC. Bước này được thực hiện bằng các công cụ như Blast Fusion (Magma), IC Compiler (Synopsys), và Encounter (Cadence), v.v.... Xác minh vật lý (Physical Verification) được thực hiện để xác minh xem layout có được thiết kế theo các đúng các rules hay không.

Để bất kỳ thiết kế nào hoạt động ở một tốc độ cụ thể, phải thực hiện phân tích thời gian. Chúng ta cần kiểm tra xem thiết kế có đáp ứng yêu cầu tốc độ được đề cập trong Specification ban đầu hay không. Điều này được thực hiện bởi tool STA (Static Timing Analysis tool), ví dụ: Primetime (Synopsys). Nó xác nhận hiệu suất thời gian của một thiết kế bằng cách kiểm tra thiết kế cho tất cả trường hợp ví dụ như vi phạm về thời gian có thể xảy ra; setup và hold time...

Sau khi PD, Verification và STA xong, layout đã sẵn sàng để chế tạo. Các dữ liệu layout được chuyển đổi thành mặt nạ in quang khắc. Sau khi chế tạo, tám wafer được cắt thành từng chip. Mỗi chip đều được đóng gói và kiểm tra.

## 6.2 Front-end

Sử dụng ngôn ngữ thiết kế phần cứng (VHDL, Systemverilog...) để hiện thực các chức năng logic của thiết kế. Lúc này ta không cần quan tâm đến cấu tạo chi tiết của mạch mà chỉ chú trọng vào chức năng của mạch dựa trên kết quả tính toán cũng như sự luân chuyển dữ liệu giữa các thanh ghi (register). Đây là thiết kế mức độ chuyển đổi thanh ghi (RTL – Register Transfer Level). Sau đó thiết kế RTL sẽ được mô phỏng để kiểm tra xem có thỏa chức năng của mạch hay không. Các CADs phổ biến dùng để thiết kế và mô phỏng RTL là: NC-Verilog, NC – VHDL (Cadence), ModelSim (Mentor Graphics), VCS (Synopsys)...

Tiếp theo, thiết kế RTL được tổng hợp (synthesis) thành các cổng (gate) cơ bản: NOT, NAND, NOR... Quá trình này được thực hiện với sự trợ giúp của các CADs chuyên dụng. Phổ biến hơn cả là Design Compiler (Synopsys), Synlify (Synplicity), XST (Xilinx). Kết quả của quá trình tổng hợp không phải là duy nhất và tùy thuộc vào CADs và thư viện các cổng và macro của nhà sản xuất chip.

Các bước cơ bản của quá trình front-end trong IC Design:

- Idea/System Specification
- RTL design/ System – level design
- RTL Verification

### 6.2.1 System Specification

Đầu tiên, dựa trên các yêu cầu của khách hàng đề ra, những yếu tố ban đầu và chức năng chính của thiết kế được hình thành. Ở bước này, các chức năng được đặc tả ở dưới dạng các hình vẽ sơ đồ khói, giao diện cũng như những hiệu năng ban đầu mong muốn đạt được.

Các đặc tả không có gì khác ngoài:

- Mục tiêu và constraint của thiết kế.
- Chức năng (chip sẽ làm gì).
- Các số liệu về hiệu suất như tốc độ và công suất.
- Các constraint về công nghệ như kích thước và diện tích (kích thước vật lý).
- Công nghệ chế tạo và kỹ thuật thiết kế.

Gần gũi nhất đối với các bản đặc tả này có lẽ là các bản “**DataSheet**” của các IP hay các thiết bị. Có thể hiểu rằng các bản “**DataSheet**” là phiên bản hoàn thiện của các bản đặc tả khi mà chip đã được sản xuất. Do đó, các thông số hiệu năng trên các “**DataSheet**” cũng như giao diện, kiến trúc khói là chính xác và đã được đảm bảo. Ngược lại ở các bản đặc tả, những giao diện, kết nối và những thông số này ở dạng ước lượng hay mong muốn đạt được và có thể sẽ được chỉnh sửa nhiều lần trong quá trình thiết kế. Dựa vào quy mô thiết kế mà có thể hiểu đặc tả chức năng được chia làm hai loại mà tạm gọi là “**General Specification**” và “**Internal Specification**”.

**General Specification:** Các đặc tả chức năng chỉ ở dạng khói. Các kết nối giữa các khói cũng như giao diện cấp độ chip được liệt kê chi tiết. Các thông số hiệu năng được đặc tả cho từng khói. Nói chung, đặc tả theo khói được thể hiện ở loại này.

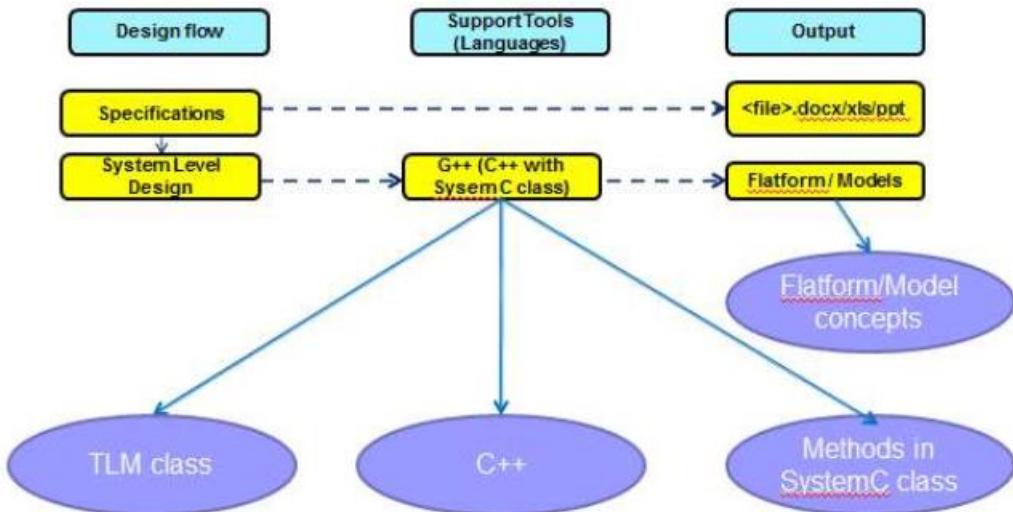
**Internal Specification:** Đặc tả theo dạng này dành cho các kỹ sư thiết kế phần cứng mức độ RTL. Các đặc tả này được chi tiết hóa đến các mức cổng và FlipFlop. Do đó, các chức năng chính của thiết kế cũng được chi tiết hóa một cách rõ ràng với các bảng giá trị cũng như những liên kết sâu bên trong các khối chính trong thiết kế.

**Ví dụ:** Về một Internal Design Specification của mạch một bộ full adder 1 bit được liệt kê các thông tin cơ bản như

### 6.2.2 *Design process*

#### 6.2.2.1 Microarchitecture and system – level design

Quá trình thiết kế chip bắt đầu với bước thiết kế ở mức độ hệ thống (system – level design) và thiết lập vi cấu trúc (microarchitecture) cho chip. Trong các công ty thiết kế vi mạch, ban quản lý và thường là bộ phận phân tích sẽ soạn thảo một đề xuất cho nhóm thiết kế bắt đầu thiết kế một con chip mới để phù hợp với một phân khúc trong ngành. Ở giai đoạn này, các nhà thiết kế cấp trên sẽ họp để quyết định cách thức hoạt động của con chip. Bước này là nơi quyết định chức năng và thiết kế của vi mạch. Các nhà thiết kế vi mạch sẽ vạch ra các yêu cầu chức năng, bàn kiểm tra xác minh và phương pháp thử nghiệm cho toàn bộ dự án, sau đó sẽ biến thiết kế sơ bộ thành đặc điểm kỹ thuật cấp hệ thống có thể được mô phỏng bằng các mô hình đơn giản sử dụng các ngôn ngữ như C ++ và MATLAB và các công cụ mô phỏng.



*Các vấn đề trong bước thiết kế System – level Design*

#### a. 2.1.1. C++ trong thiết kế phần cứng

Là một trong những ngôn ngữ lập trình hướng đối tượng cấp cao. C++ kế thừa toàn bộ các thư viện C chuẩn kết hợp với đặc tính hướng đối tượng, nó trở thành một ngôn ngữ được đánh giá mạnh mẽ trong những công đoạn mô phỏng hay mô tả hành vi cho phần cứng. Trong thiết kế phần cứng, C++ đã trở thành công cụ quen thuộc trong các môi trường kiểm tra thiết kế vì tính linh động và thời gian chạy rất ngắn của chúng.

Ở đây, những ưu điểm của C++ được khai thác trong lĩnh vực phần cứng. Một ví dụ minh họa dễ hiểu cho thấy việc thiết kế behavior của một IP thông thường, sau khi hiểu được đặc tả của thiết kế, một người kỹ sư có kinh nghiệm chỉ mất khoảng phân nữa thời gian để thiết kế và kiểm tra behavior của nó so với người thiết kế bằng HDL.

Do đó, C++ được ưa chuộng trong mô hình kiểm tra thiết kế phần cứng. Nó mô tả hành vi của phần cứng và so sánh kết quả của nó với kết quả phần cứng cần kiểm tra và cho những kết quả so sánh cuối cùng.

#### b. 2.1.2. SystemC Class

Mặc dù C++ được đánh giá là ngôn ngữ mạnh mẽ trong việc hỗ trợ thiết kế các mô hình mô phỏng hành vi của phần cứng nhằm kiểm tra và đánh giá so với các phần cứng thật. Tuy nhiên C++ vẫn còn tồn tại nhiều hạn chế như sau.

- Hạn chế trong việc mô phỏng các tác vụ song song và tính ưu tiên giữa các tác vụ.
- Hạn chế trong việc mô phỏng các hiệu năng như thời gian, hiệu suất.

- Hạn chế trong việc mô phỏng các giao thức bắt tay của phần cứng.

#### c. 2.1.3. Mô hình truyền tải – Transaction Level Model – TLM

Với C++ và lớp SystemC, các mô phỏng hành vi phần cứng gần như hoàn thiện. Tuy nhiên vẫn còn một số hạn chế về các giao thức giao tiếp giữa các khối kiến trúc với nhau. Tiêu biểu cho điều này là các giao thức Bus nói riêng. Nếu sử dụng ngôn ngữ để mô tả các giao thức bắt tay sao cho giống như các giao thức mà phần cứng thể hiện, đó là một điều hết sức khó khăn với những kỹ sư mô phỏng hành vi. Một khía cạnh khác là sự thay đổi trong giao thức. Hơn nữa hiệu năng về mô phỏng thời gian thực cũng sẽ bị hạn chế khi theo phương thức cũ này. Chính vì những hạn chế này, mô hình TLM được phát triển bởi Thorsten Grötker vào năm 2000, một trong những người đứng đầu của bộ phận R&D thuộc công ty Synopsys. Bằng ngôn ngữ C++ với sự hỗ trợ của lớp SystemC, mô hình TLM được ra đời nhằm thống nhất mô phỏng các giao thức. Nói một cách đơn giản, TLM được xem như một lớp dữ liệu mới nhằm hỗ trợ C++ trong vấn đề mô hình hóa các giao thức truyền nhận và bắt tay giữa các IP hay các core xử lý. Tương tự lớp SystemC, TLM cho phép người dùng sử dụng những đối tượng mà nó cung cấp để phát triển các phương pháp đánh giá riêng cho IP cũng như hệ thống trong quá trình bắt tay hay truyền tải dữ liệu.

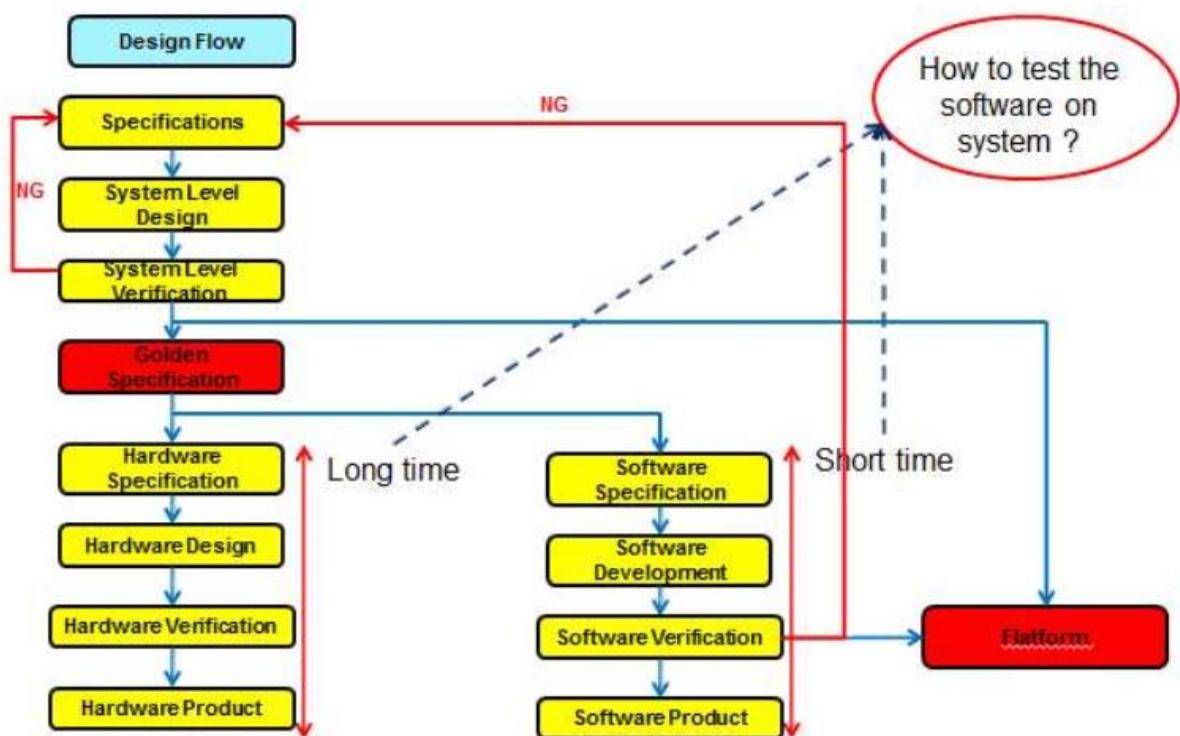
#### d. 2.1.4. Platform and SystemC Model

**Platform:** Khái niệm platform được giới thiệu trong bước thiết kế này nhằm ám chỉ toàn bộ mô hình phần cứng được mô hình hóa bằng một gói phần mềm mà gói phần mềm này được phát triển dựa trên nền tảng C++ và SystemC. Khi một hệ thống mới ra đời, những thay đổi đáng kể hay không đáng kể sẽ được cập nhật. Dựa trên những đặc tính cập nhật này, gói phần mềm cũ sẽ được cải tiến sao cho mô hình hóa giống với hệ thống mới. Công việc này sẽ tốn rất ít thời gian vì việc cải tiến hành vi phần mềm trên nền tảng C++ là một công việc dễ dàng. Khái niệm Platform có thể hiểu giống như khái niệm một “khuôn mẫu” có sẵn để sản xuất các mặt hàng và người sản xuất có thể chế biến thay đổi các mẫu cụ thể dựa trên khuôn mẫu tổng quát này.

**SystemC Model:** Ám chỉ phương pháp mô hình hóa phần cứng trên nền tảng C++ với sự hỗ trợ bởi các đối tượng trong lớp SystemC. Để hiểu hơn về tính liên kết giữa các khái niệm trên, hình 1.8 được giới thiệu. Trong hình, một MCU (micro controller) tổng quan với các IP tiêu biểu bên trong được giới thiệu.

- Các hành vi của các IP cũng như CPU, DMAC,... được mô hình hóa trên nền tảng ngôn ngữ C++ với sự hỗ trợ của lớp SystemC.
- Các khối IP và CPU được liên kết với nhau thông qua bus. Bus này được mô hình hóa bởi mô hình TLM được giới thiệu ở trên.

Khi các mô hình được phát triển và được nối với nhau, chúng hình thành nên một Platform hay ta nói một MCU giả lập được phát triển trên nền tảng ngôn ngữ lập trình cấp cao. Ở phương diện thiết kế có thể xem đây là một gói phần mềm. Như vậy, để trả lời cho các câu hỏi đã nêu từ trước là tại sao cần bước thiết kế SLD trong quy trình phát triển ASIC, thêm một hình vẽ nữa được thể hiện



Vai trò của System – level Design trong quy trình phát triển ASIC

#### 6.2.2.2 RTL design

Sau khi hoàn thành bước SLD nhằm đảm bảo các đặc tả là chính xác và khả thi, bước thiết kế RTL được thực hiện. RTL (Register Transfer Level) được hiểu là thiết kế ở cấp độ thanh ghi. Ở cấp độ thiết kế này, các luồng dữ liệu luân chuyển bên trong các khối kiến trúc được làm rõ ở cấp độ từ thanh ghi này qua thanh ghi khác.

Ngôn ngữ được sử dụng ở cấp độ thiết kế này là Verilog hoặc VHDL. Dựa trên các đặc tả có từ trước, kỹ sư thiết kế sử dụng ngôn ngữ Verilog/VHDL để mô hình hóa lại kiến trúc phần cứng. Thuyết minh tiếp cận ngôn ngữ Verilog là ngôn ngữ sẽ được sử dụng để phát triển đề tài vì tính tiện dụng và gần gũi của nó đối với người lập trình. Verilog, được chuẩn hóa theo IEEE 1364, là một ngôn ngữ mô tả phần cứng (HDL) được sử dụng để mô hình hóa các hệ thống điện tử. Nó thường được sử dụng trong việc thiết kế và kiểm tra mạch kỹ thuật số ở cấp độ truyền dữ liệu giữa thanh ghi (RTL level). Nó cũng được sử dụng trong việc kiểm tra hành vi các vi mạch tương tự và mạch tín hiệu hỗn hợp (số và tương tự).

Verilog là ngôn ngữ mô tả phần cứng đầu tiên được phát minh. Nó được tạo ra bởi Phil Moorby và Prabhu Goel trong mùa đông năm 1983/1984. Hiện nay, đã có các phiên bản 95, 2001 và 2005. Ngoài ra, một ngôn ngữ khác là System Verilog đã xuất hiện dựa trên nền tảng của Verilog nhằm cung cấp nhiều tác vụ và hàm hệ thống phục vụ cho việc viết các trường hợp kiểm tra lớn.

Có một lưu ý về ngôn ngữ thiết kế phần cứng và vai trò của thiết kế ở cấp độ RTL. Mục đích chính của cấp độ này là mô hình hóa phần cứng bằng ngôn ngữ phần cứng nhưng sản phẩm coding phải có khả năng tổng hợp xuống lớp cổng được. Ngôn ngữ phần cứng chỉ mang ý nghĩa công cụ nghĩa là kỹ sư có thể sử dụng ngôn ngữ này tùy biến. Hay nói cách khác có thể viết mô phỏng phần cứng với nhiều cách thức, đoạn code khác nhau. Mặc dù về chức năng chúng vẫn đảm bảo chạy đúng như đặc tả nhưng chỉ có những đoạn code thỏa mãn các ràng buộc thì mới có thể tổng hợp được. Do đó khi sử dụng ngôn ngữ phần cứng vào mục đích thiết kế phần cứng cần phải chú ý những tiêu chí sau

- Phiên bản ngôn ngữ phần cứng mà công cụ biên dịch hỗ trợ.
- Các ràng buộc và điều kiện để có thể tổng hợp xuống lớp cổng.
- Cách thức viết để có thể dễ dàng hiểu và tái sử dụng.

### 6.2.3 *RTL Verification*

Sau khi kiến trúc được thiết kế ở cấp độ RTL dưới dạng các đoạn mã Verilog/VHDL, chúng sẽ được kiểm tra và đánh giá các chức năng logic. Có rất nhiều mô hình kiểm tra các đoạn code RTL, ở đây tạm chia làm 3 mô hình chính.

*Unit Test:* Kích thích trực tiếp vào ngõ vào của thiết kế nhằm tạo ra các trường hợp cần kiểm tra.

*Combination Test:* Nối khối kiến trúc cần kiểm tra (Design Under Test - DUT) với một số khối liên kết khác trong hệ thống. Mô hình hóa sao cho các ngõ vào của DUT nhận các tín hiệu từ các khối liên kết. Việc thay đổi các tín hiệu này từ các khối liên kết cũng chính là các trường hợp cần kiểm tra của DUT.

*System Test:* Kết nối khối kiến trúc vào trong một hệ thống hoàn chỉnh. Viết các đoạn chương trình tạo hoạt động cho hệ thống sao cho DUT được sử dụng trong đoạn chương trình đó. Các trường hợp kiểm tra phụ thuộc vào đoạn chương trình mà hệ thống chạy trên nó.

Để hiểu rõ hơn về các mô hình kiểm tra này, từng mô hình được rút trích và phân tích.

#### 6.2.3.1 3.1 Unit Test

Thông thường sau khi thiết kế ở cấp độ RTL, Unit Test được thực hiện luôn bởi những kỹ sư phát triển code Verilog. Các trường hợp kiểm tra được liệt kê dưới dạng các file tài liệu. Sau đó, dựa trên các tài liệu này, các trường hợp kiểm tra được phát triển cùng với môi trường để thực hiện công đoạn kiểm tra này.

Để hiểu rõ hơn về quy trình kiểm tra Unit Test này, một môi trường và cách thức thực thi mà thuyết minh sẽ tiếp cận được giới thiệu.

- *Môi trường thiết kế và kiểm tra:* Hệ điều hành Linux.
- *Ngôn ngữ thiết kế được chọn:* Verilog.
- *Công cụ hỗ trợ:* Phần mềm VCS của Synopsys và phần mềm DVE.
- *Môi trường soạn thảo code Verilog:* Trình soạn thảo VI trong Linux.

Các trường hợp kiểm tra cũng như cách kích thích các tín hiệu ngõ vào của thiết kế được mô tả bởi ngôn ngữ Verilog và cũng trên trình soạn thảo VI. Từ “TestBench” được sử dụng để ám chỉ đoạn mã code Verilog mà trong đó chứa cách thức kiểm tra, thiết kế cần kiểm tra và các trường hợp kiểm tra thiết kế đó.

##### a. 3.1.1 Linux và trình soạn thảo VI

Linux là tên gọi của một hệ điều hành máy tính và cũng là tên hạt nhân của hệ điều hành. Có rất nhiều phiên bản sau này dựa trên nhân của nó. Nó có lẽ là một ví dụ nổi tiếng nhất của phần mềm tự do và của việc phát triển mã nguồn mở... Trong môi trường công

nghiệp, đặc biệt là môi trường lập trình thiết kế phần cứng, Linux là hệ điều hành được dùng rộng rãi. Gần như tất cả các công đoạn phát triển vi mạch đều được thực hiện trên môi trường này. Với môi trường thiên về hỗ trợ các tác vụ dựa trên các lệnh, Linux phù hợp cho những tác vụ có tính lặp đi lặp lại hay những mã lệnh hồi quy...

Trình soạn thảo VI cũng là môi trường tương tác giữa người dùng với máy tính. Thông qua trình soạn thảo này, người dùng có thể viết những câu lệnh để gọi các phần mềm hay soạn thảo những đoạn code trên chính giao diện này. Việc sửa lỗi cũng như gỡ lỗi được hỗ trợ tích cực bởi trình soạn thảo này bởi các tác vụ bằng lệnh nhanh nhẹn. Có thể nói sử dụng trình soạn thảo VI trên nền Linux là một trong những kỹ năng không thể thiếu của kỹ sư thiết kế phần cứng.

### b. 3.1.2 VCS và DVE

VCS là một trong những công cụ mà Synopsys hỗ trợ để kiểm tra hành vi, chức năng về tính logic các thiết kế phần cứng cấp độ RTL. Có thể dùng cụm từ “Dynamic Verification” cho công việc mà phần mềm VCS hỗ trợ. Có thể hiểu rằng phần mềm chỉ hỗ trợ kiểm tra tính logic của thiết kế mà không kiểm tra hay đánh giá các hiệu năng về thời gian, diện tích, năng lượng ... VCS có thể hỗ trợ cho nhiều loại ngôn ngữ ở cấp độ RTL cùng một lúc như Verilog, VHDL hay System Verilog.

Có hai trạng thái sử dụng VCS đó là trạng thái sử dụng GUI hoặc trạng thái COMMAND LINE. Ở trạng thái sử dụng GUI, người dùng tương tác trực tiếp lên giao diện của phần mềm để thực hiện ý đồ của mình. Trong trạng thái COMMAND LINE, tất cả các thao tác được thực thi bằng các lệnh, người dùng thông qua các file tường thuật (report file) để hiểu và nắm bắt quát trình thực hiện. Trong môi trường công nghiệp, trạng thái COMMAND LINE luôn được khuyến khích khuyên dùng. Hình vẽ dưới đây mô tả cách gọi phần mềm VCS và sử dụng ở trạng thái COMMAND LINE cũng như các tùy chọn khi sử dụng phần mềm này.

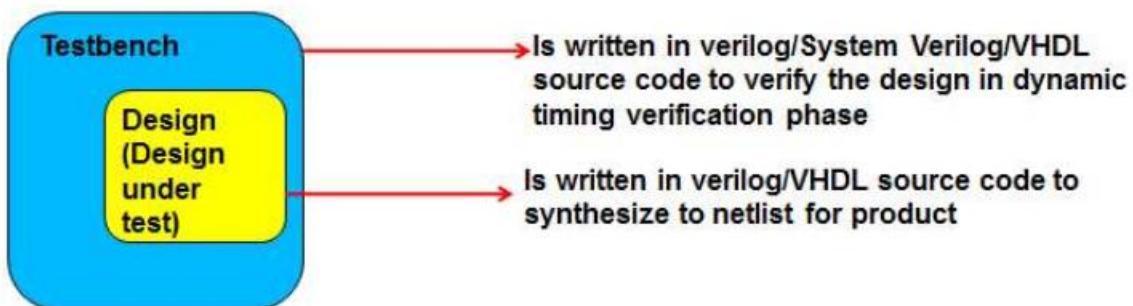
source /home/lampham/synopsys/licenses/linux/bin/s		
vcs <mode option> <debug option> <file option> <version option> ...		
Option Name	Command	Notes
Mode	-R	Command Mode
	-gui	GUI Mode
Debug	-debug	Export the waveform to debug
	-debug_all	Export the waveform to debug, line debug
File	-f <file>	Compile all files listed in the file
	-o <file>	Export the executed file (sim.v is default)
	-l <file>	Export the log file
Version	+v2K	Add the version 2000
Others	-sverilog	Compile the mis_language (Verilog & System Verilog)

### *Trạng thái COMMAND LINE gọi phần mềm VCS*

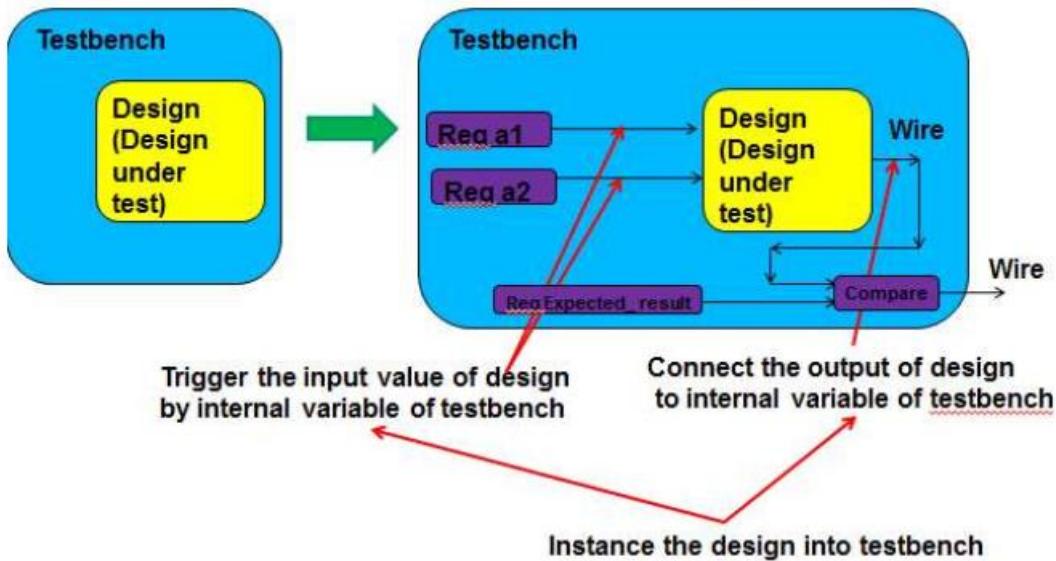
Sau khi hoàn tất thiết kế (thiết kế viết bằng ngôn ngữ Verilog) và môi trường để kiểm tra thiết kế (Môi trường được viết bằng ngôn ngữ Verilog bao gồm các trường hợp kiểm tra cách thức kiểm tra thiết kế). Phần mềm VCS được gọi ra và thực thi nhằm kiểm tra sự tương đồng và chính xác của thiết kế so với mong muốn. Người dùng quan sát các file tường thuật để nắm bắt các lỗi mà thiết kế vi phạm. Tuy nhiên, một số lỗi đặc biệt ví dụ như “Rising” (lỗi một tín hiệu được kích thích từ hai nguồn tín hiệu khác nhau) khó có thể kiểm tra bằng mắt qua các file tường thuật. Để hỗ trợ cho việc gỡ lỗi được tiện lợi và nhanh hơn, phần mềm DVE được kết hợp sử dụng để xem các file dạng sóng tín hiệu mà phần mềm VCS tạo ra. Dựa trên giao diện DVE, dạng sóng của tín hiệu được tường thuật trực quan. Điều này giúp người dùng nhận ra các lỗi đặc biệt nhanh chóng. Hình vẽ dưới đây là một trong những giao diện của DVE.

#### c. 3.1.3. *Testbench*

TestBench là tên gọi được nhiều người ám chỉ đến môi trường kiểm tra thiết kế cấp độ RTL hay cũng được ám chỉ đến file (viết bằng Verilog) mà trong đó các trường hợp kiểm tra và thiết kế cũng như cách thức kiểm tra được thể hiện. Để hiểu rõ khái niệm này, một mô hình trùu tượng được thể hiện qua hình các hình minh họa sau đây



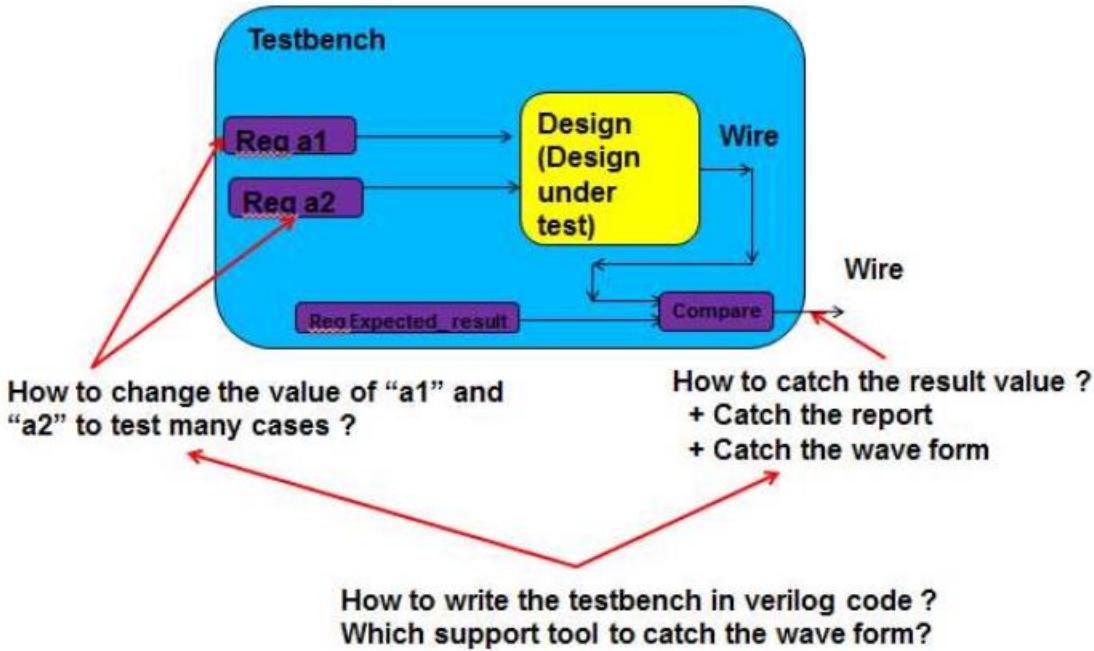
*Ngôn ngữ sử dụng để viết thiết kế và testbench*



### Cách thức kích thích ngõ vào của thiết kế

Phân tích hình cho thấy cách thức mà testbench được sử dụng để kích thích ngõ vào của thiết kế. Các ngõ vào của thiết kế được kết nối với các biến “reg – dạng thanh ghi”, các ngõ ra của thiết kế được liên kết với biến “wire – dạng dây”. Các biến này được khai báo bên trong file testbench (Viết bằng Verilog). Khi muốn kiểm tra các trường hợp mong muốn, các biến thanh ghi ngõ vào được thay đổi giá trị nhằm tạo các giá trị kiểm tra trên ngõ vào thiết kế. Các ngõ ra của thiết kế sau đó được cập nhật và so sánh với giá trị mong muốn. Kết quả so sánh sẽ được người dùng bắt lấy và xem xét thông qua các file tường thuật hoặc giảng đồ xung.

Có một số vấn đề cần quan tâm là khi số lượng trường hợp kiểm tra lớn thì cách thức thực hiện như thế nào trong file testbench được xem là phù hợp. Một vấn đề nữa là cách thức mà người dùng sử dụng để lấy được thông tin so sánh kết quả như là vào thời điểm nào thì hợp lý...



### Hai vấn đề cần quan tâm trong thiết kế Testbench

Một ví dụ cho thấy các bước cần thực hiện và một đoạn code mẫu khi viết một file testbench. Các bước liệt kê có thể xáo trộn thứ tự do ngôn ngữ lập trình phần cứng không thực thi theo thứ tự từ trên xuống mà song song theo thời gian. Khi nhìn vào các khoản mục “Declare Testcases” và “Catch the output” sẽ thấy được hai vấn đề cần quan tâm được nêu ở trên.

Testbench feature
Module Declaration
Parameter Declaration
Internal variable Declaration
Instance design
Clock generation
Test Case
Catching the output
Export the waveform

```

//===== File : t_exampl1_and_gate.v
//===== Version: 1.0
//===== Update Note: Fist version

module t_exampl1_and_gate;

//parameter define
parameter DATA_WIDTH = 8;

//inputs define
reg t_system_clock;
reg t_system_rst_n;
reg [DATA_WIDTH-1:0] t_second_data_in;
reg [DATA_WIDTH-1:0] t_first_data_in;

//output define
wire [DATA_WIDTH-1:0] t_data_out_and_gate;

//clock define
always begin
    t_system_clock = 1'b0;
    # 50;
    t_system_clock = 1'b1;
    # 50;
end

//instance define
exampl1_and_gate exampl1_and_gate_01(
    .system_clock(t_system_clock),
    .system_rst_n(t_system_rst_n)
);

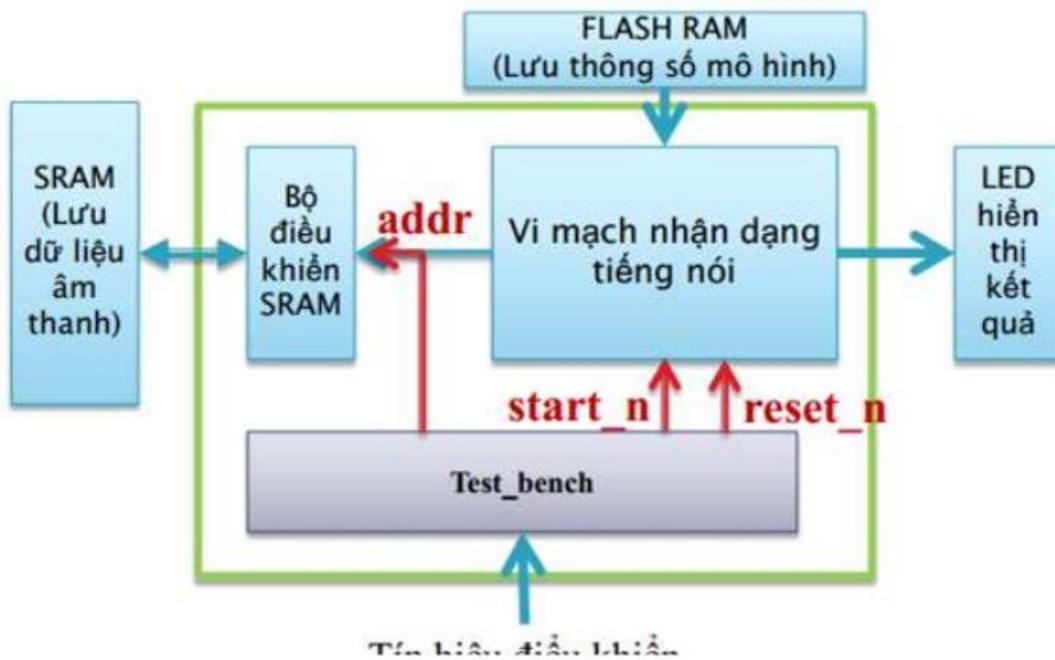
initial begin
    $monitor("time=%d, system_clock=%b,
        system_rst_n=%b, t_first_data_in=%b,
        t_second_data_in=%b, data_out_and_gate=%b \n",
        $time, t_system_clock, t_system_rst_n,
        t_first_data_in, t_second_data_in,
        t_data_out_and_gate);
end

```

Ví dụ các thành phần trong file testbench

### 6.2.3.2 3.2. Combination Test

Mô hình kiểm tra này như đã giới thiệu ở trên cần có các kiến trúc phụ tương tác. Các IP sau khi được phát triển sẽ kết nối với các IP có sẵn tạo nên một môi trường kiểm tra. Các ngõ vào của IP cần kiểm tra được kích thích bởi các kiến trúc phụ trợ này. Có thể xem việc tiếp cận FPGA như một trong những phương thức kiểm tra của mô hình này. Sau khi thiết kế được phát triển ở cấp độ RTL (Code Verilog) hoàn tất, nó sẽ được nhúng lên trên kít FPGA, các tín hiệu ngõ ra và ngõ vào của thiết kế được kết nối với các thiết bị trên kít FPGA. Người dùng điều khiển các thiết bị trên kít FPGA nhằm kích thích ngõ vào của thiết kế đã nhúng trên FPGA. Kiểm tra ngõ ra bằng cách kiểm tra các thiết bị trên kit mà đã kết nối với ngõ ra của thiết kế. Thuyết minh cũng giới thiệu mô hình kiểm tra trên FPGA đã tiếp cận và đạt thành công.



*Mô hình kiểm tra thiết kế vi mạch nhận dạng giọng nói trên FPGA*

Đầu tiên các file dữ liệu được lưu trữ trên SRAM, để lấy được dữ liệu này phải sử dụng bộ điều khiển SRAM. Nối các chân điều khiển SRAM với các SWITCH có thể dễ dàng điều khiển SRAM. Tương tự như mô hình Unit Test, thông số mô hình được lưu trên FLASH RAM.

Khi người thử nghiệm đọc từ muốn nhận dạng, dữ liệu sẽ được lấy mẫu và trích đặc trưng sau đó lưu lại lên SRAM. Sau thao tác này người dùng sử dụng SWITCH nối với tín hiệu

START nhằm khởi động quá trình nhận dạng. Sau khi SWITCH được kích hoạt, kết quả nhận dạng sẽ hiện ra đèn LED. Nhờ vào các led này, việc kiểm tra kết quả được tường minh.

### 6.2.3.3 3.3. System Test

Mô hình này rất thường được sử dụng trong môi trường công nghiệp vì tính tái sử dụng của nó. Khi một IP hay kiến trúc bất kỳ được hoàn tất, luôn luôn có sẵn một hệ thống mà trước đó đã được phát triển. IP này được kết nối vào hệ thống phục tạp có sẵn này và kiểm tra. Các trường hợp kiểm tra lúc này không được viết thông thường mà phải thông qua một CPU điều khiển.

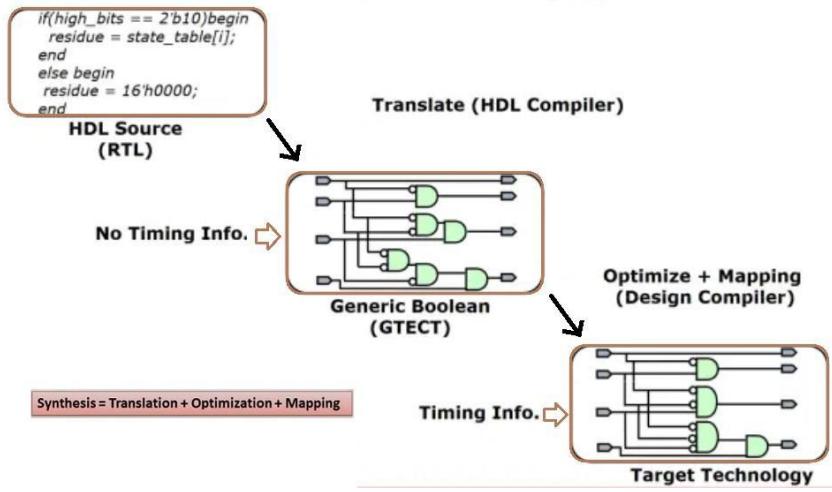
Trong môi trường học thuật ở Việt Nam hiện nay, môi trường sử dụng phần mềm Quatus kết hợp với phần mềm Nios của công ty Alter nhằm giả lập một hệ thống đầy đủ trên FPGA có ý nghĩa tương tự. Một hệ thống giả lập được thực hiện với core CPU NIOS. Việc kiểm tra IP thông qua việc nối IP đó vào môi trường giả lập và lập trình cho core CPU NIOS này thực hiện. Trong quá trình thực hiện các lệnh, các luồng dữ liệu sẽ tương tác với IP nhằm kiểm tra các chức năng IP.

### 6.2.4 Synthesis

#### 6.2.4.1 Khái niệm về Synthesis

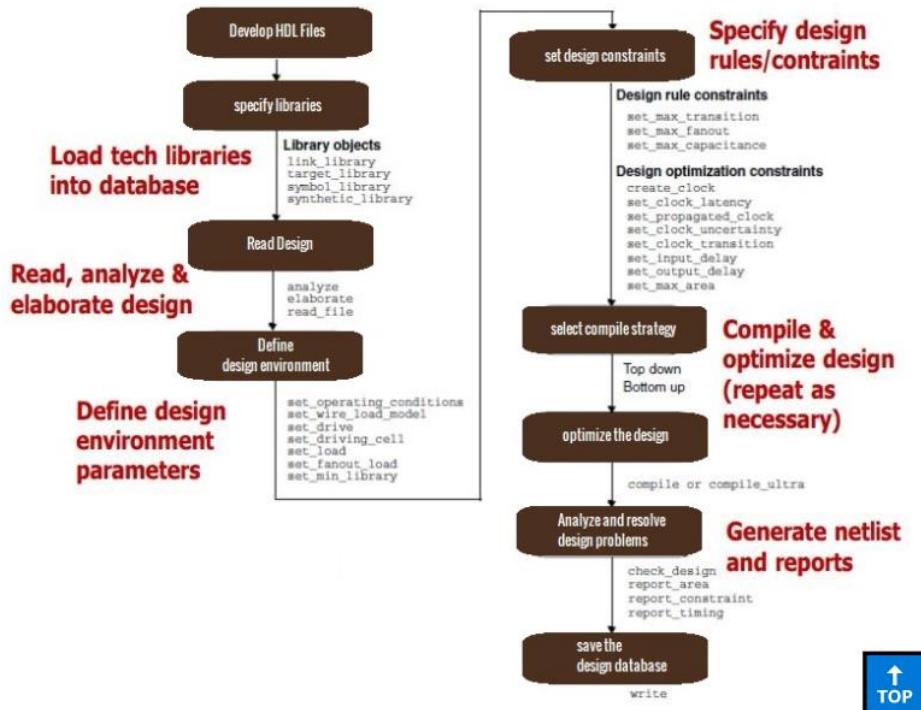
##### *Synthesis*

- Là quá trình chuyển mức từ cao hơn (RTL) sang mức thấp hơn có thể thực hiện được. Đây là quá trình chuyển đổi RTL sang netlist cấp cổng (gate-level netlist).
- Quá trình Synthesis có thể được tối ưu hóa cho Tốc độ (Timing)/ Diện tích/ Khả năng kiểm tra (DFT)/ Công suất (DFP)/ Thời gian chạy.
- Trình biên dịch thiết kế (Design Compiler- DC) từ Synopsys và trình biên dịch RTL (RTL compiler) từ Cadence là những công cụ được sử dụng rộng rãi để thực hiện bước Synthesis.



### Synthesis

#### Synthesis Flow



#### Mục tiêu của Synthesis

- Xuất ra file netlist cấp công, chèn các Clock Gates, tối ưu logic, chèn DFT Logic.
- Các liên hệ logic giữa RTL và Netlist phải được duy trì

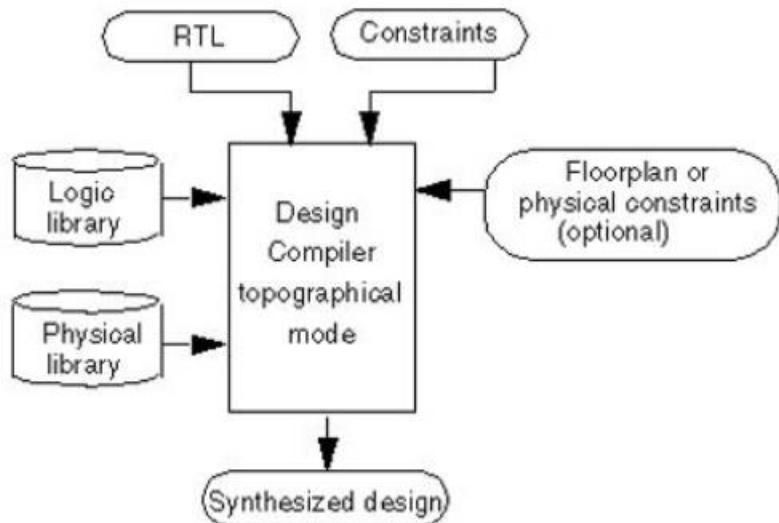
#### Input của Synthesis

- RTL: Files HDL.
- Các thư viện.
- Các constraints.

- UPF: Công suất dự định.

*Output của Synthesis*

- Netlist
- Báo cáo (QoR, Diện tích, Timing...)
- UPF: Định dạng thông nhất về công suất.



*Input và Output của tool DC*

#### 6.2.4.2.4.2. File HDL

*Khởi tạo file HDL và thiết lập thư viện*

- Các tệp đầu vào HDL được viết bằng ngôn ngữ VHDL hoặc Verilog hoặc System Verilog.
- Cách phân vùng và mã hóa HDL ảnh hưởng trực tiếp đến quá trình synthesis và tối ưu hóa.
- Các thư viện dựa trên công nghệ bán dẫn được sử dụng.

Có nhiều loại thư viện khác nhau với các định nghĩa khác nhau: link libraries (Thư viện để tạo ra các loại liên kết với nhau), target libraries (thư viện cell của bạn), symbol libraries (Thư viện biểu tượng chứa định nghĩa của các biểu tượng đồ họa đại diện cho các cell trong sơ đồ thiết kế), synthetic libraries (thư viện DesignWare), v.v.

*Đọc file*

Có hai phương pháp để đọc file thành tool DC. Một là sử dụng các lệnh phân tích và xây dựng; còn lại là sử dụng lệnh **read\_file**.

#### *Phân tích và xây dựng:*

- Lệnh phân tích thực hiện những việc sau:
  - + Đọc các tệp nguồn HDL, thực hiện kiểm tra cú pháp HDL và kiểm tra các rule của Synopsys.
  - + Kiểm tra các tập tin để tìm lỗi mà không xây dựng logic chung cho thiết kế.
  - + Khởi tạo các đối tượng HDL ở dạng trung gian.
  - + Lưu trữ các tệp trung gian ở một vị trí được chỉ định bởi lệnh `define_design_lib`.
- Lệnh xây dựng thực hiện những việc sau:
  - + Chuyển thiết kế thành biểu diễn GTECH của nó.
  - + Cho phép thay đổi các giá trị tham số được xác định trong mã nguồn.
  - + Thay thế các toán tử số học HDL trong mã bằng các thành phần thiết kế.
  - + Thực hiện liên kết tự động.

#### **6.2.4.3 4.3. Các ràng buộc về môi trường thiết kế**

Khi thiết kế đã được đọc xong, bạn cần xác định được môi trường thiết kế và các ràng buộc thiết kế.

*Môi trường thiết kế (Design environment):* Nó bao gồm các điều kiện hoạt động (Operating Conditions), mô hình tải dây (Wire Load Models) và các yêu cầu về giao diện hệ thống (System Interface).

*Điều kiện hoạt động:* Nó bao gồm các yêu cầu về quy trình, điện áp và nhiệt độ. Tác động của mỗi loại trong số này có thể có đối với chip cần được xem xét trong quá trình synthesis và phân tích timing.

Hầu hết các thư viện đều có cài đặt mặc định cho các điều kiện hoạt động được chỉ định rõ ràng bằng cách sử dụng lệnh **set\_operating\_condition**. Điều kiện hoạt động được đặt trong tệp **dc\_setup.tcl** bằng lệnh **set\_operating\_condition**.

*Mô hình tải dây:* Nó cho phép tool DC ước tính ảnh hưởng của chiều dài dây và fanout đối với điện trở, điện dung và điện tích của net. DC sử dụng các giá trị này để tính toán delay của dây. Thiết kế thường sử dụng zero WLM khi synthesis được thực hiện bằng tool DC.

**Giao diện hệ thống:** Thông tin liên quan đến việc điều khiển logic bên ngoài và nhận các tín hiệu từ ASIC của bạn được thu thập thông qua các ràng buộc này. Nó bao gồm cường độ đầu vào (**set\_driving\_cell**), điện dung của tải (**set\_load**), tải ở fanout (**set\_fanout\_load**), v.v.

**report\_lib, report\_design** là một số lệnh để xem các ràng buộc về môi trường của tải.

### **Đường dẫn thời gian (Timing Path)**

Công cụ phân tích timing sẽ tìm và phân tích tất cả các đường dẫn về timing trong thiết kế. Mỗi đường timing sẽ có điểm bắt đầu và điểm kết thúc.

- Điểm bắt đầu của đường dẫn là một chân clock của một phần tử tuần tự hoặc một cổng đầu vào của thiết kế.
- Điểm kết thúc của đường dẫn là một chân đầu vào dữ liệu của một phần tử tuần tự hoặc một cổng đầu ra của thiết kế.

Có bốn loại đường dẫn thời gian như thể hiện trong hình trên:

- + *Đường dẫn 1:* bắt đầu từ một cổng đầu vào và kết thúc ở đầu vào dữ liệu của một phần tử tuần tự.
- + *Đường dẫn 2:* bắt đầu tại chân clock của một phần tử tuần tự và kết thúc ở đầu vào dữ liệu của một phần tử tuần tự.
- + *Đường dẫn 3:* bắt đầu tại chân clock của một phần tử tuần tự và kết thúc tại một cổng đầu ra.
- + *Đường dẫn 4:* bắt đầu ở một cổng đầu vào và kết thúc ở một cổng đầu ra.

Phân tích thời gian tĩnh (Static timing analysis) là một phương pháp xác nhận hiệu suất về mặt thời gian của một thiết kế bằng cách kiểm tra tất cả các đường dẫn có thể có để xác định vi phạm thời gian trong các điều kiện xấu nhất. Setup và hold là những kiểm tra timing quan trọng nhất được thực hiện.

+ *Setup:* Thời gian mà dữ liệu sẽ ổn định trước khi có cạnh tích cực của đồng hồ được gọi là *setup time*.

+ *Hold:* Thời gian mà dữ liệu sẽ ổn định sau khi có cạnh tích cực của đồng hồ được gọi là *hold time*.

+ Đối với tất cả các timing path, slack được tính toán trong khi thực hiện thiết lập và kiểm tra timing. Các công thức tính toán slack để thiết lập và lưu giữ được đưa ra dưới đây

*Setup slack*= Data Required Time - Data Arrival Time.

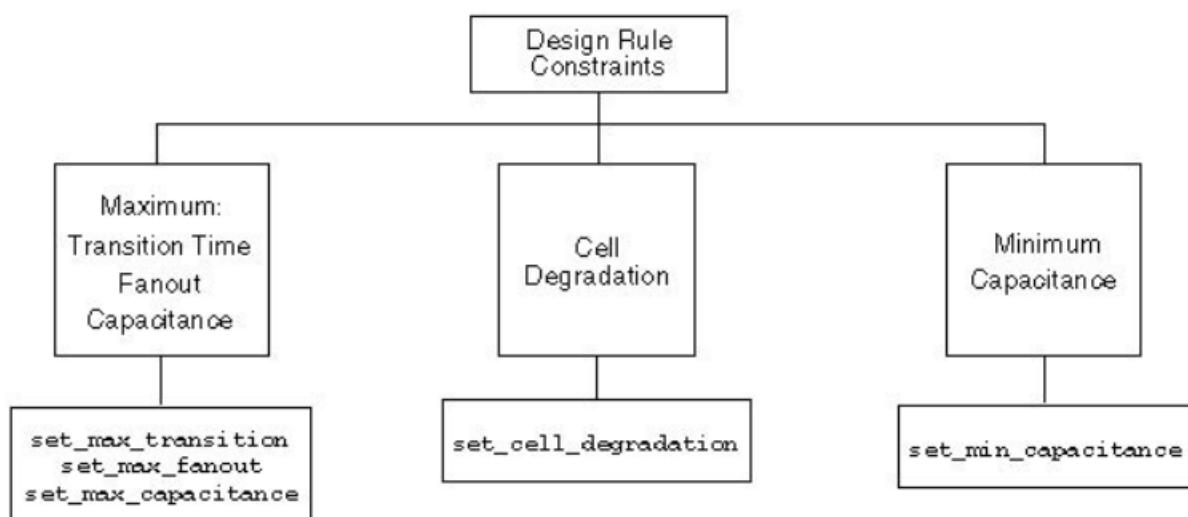
*Hold slack*= Data Arrival Time - Data Required Time.

### Constraints

Khi trình biên dịch thiết kế (DC) tối ưu hóa thiết kế của bạn, nó sử dụng hai loại constraints:

- + *Ràng buộc quy tắc thiết kế (Design Rule Constraints)*: Thư viện logic xác định các ràng buộc mặc định này. Những constraint này là bắt buộc để thiết kế hoạt động chính xác. Chúng áp dụng cho bất kỳ thiết kế nào có sử dụng thư viện. Theo mặc định, các constraint về quy tắc thiết kế có mức ưu tiên cao hơn các ràng buộc về mặt tối ưu hóa.
- + Các ràng buộc tối ưu hóa (Optimization Constraints): Bạn xác định các constraint rõ ràng này. Các constraint tối ưu hóa áp dụng cho thiết kế mà bạn đang làm việc trong suốt thời gian của phiên dc\_shell và thể hiện các mục tiêu của thiết kế. Trong quá trình tối ưu hóa, trình biên dịch thiết kế cố gắng đáp ứng các mục tiêu này, nhưng quy trình này sẽ không vi phạm quy tắc thiết kế nào. Để tối ưu hóa một thiết kế một cách chính xác, bạn phải thiết lập các constraint có tính thực tế.

Trình biên dịch thiết kế cố gắng đáp ứng cả constraint về quy tắc thiết kế và tối ưu hóa, nhưng các constraint về quy tắc thiết kế được ưu tiên hơn.



## *Design Rule Constraints*

Mục tiêu của Design Compiler là đáp ứng tất cả các constraints. Tuy nhiên, theo mặc định, nó ưu tiên cho các constraints về quy tắc thiết kế vì constraints này là các yêu cầu chức năng cho thiết kế. Sử dụng mức ưu tiên mặc định, trình biên dịch thiết kế sửa lỗi vi phạm quy tắc thiết kế ngay cả khi bạn vi phạm các constraints về delay và area.

### *Timing Constraints*

Để thiết lập chính xác các ràng buộc về timing, bạn cần chỉ định những điều sau:

- + Clock.
- + Yêu cầu về I/O timing
- + Yêu cầu về delay của đường dẫn tổ hợp.
- + Ngoại lệ về timing.

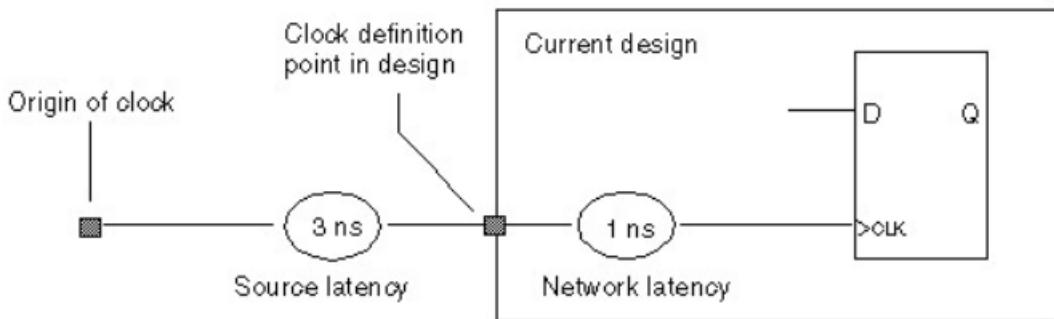
#### \* *Clocks*:

- Để xác định clock, lệnh **create\_clock** được sử dụng và lệnh **create\_generated\_clock** được sử dụng để xác định clock được tạo bên trong. Một số thông số kỹ thuật của clock bao gồm nguồn (clock source), chu kỳ clock (clock period), duty cycle...

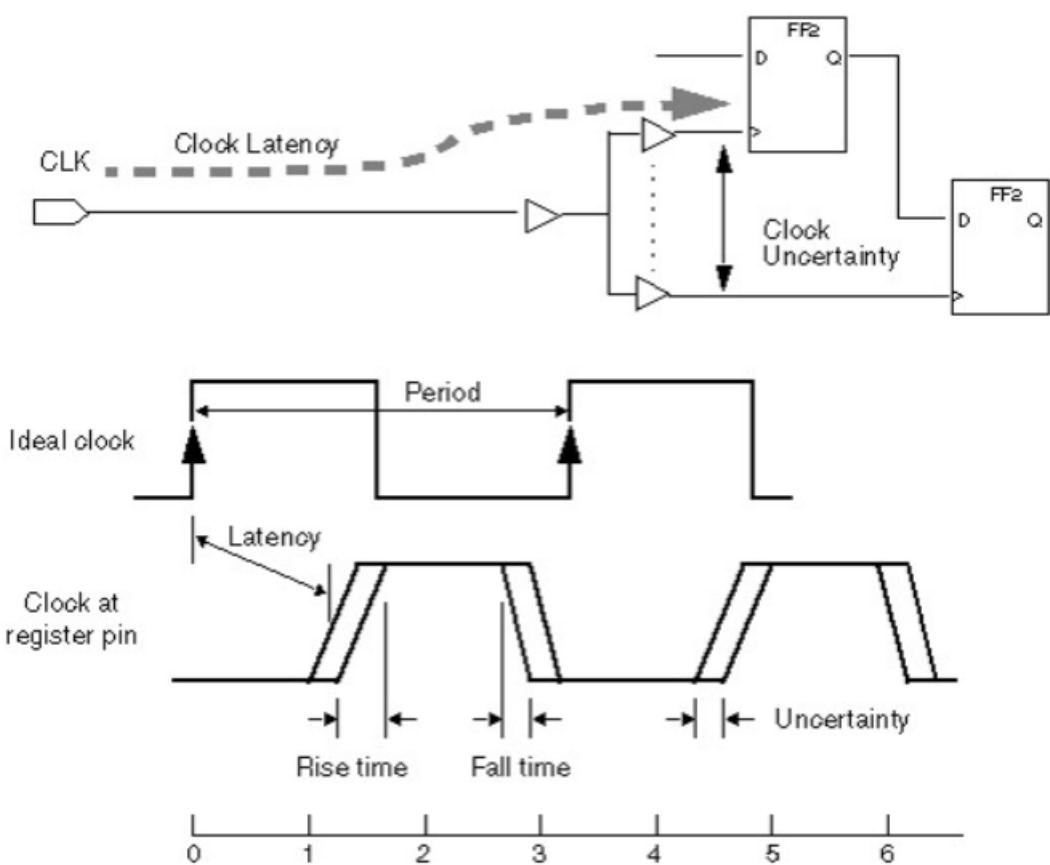
- Ví dụ về cách tạo clock và ngõ ra clock chia cho 2 được tạo ra dưới đây:

```
create_clock-name SYSCLK_NAME -period 2-waveform {0 1} [get_ports SYSCLK]
create_generated_clock-name DIVIDE -source (get_ports SYSCLK) split_by 2 [get_pins
FF1 / Q]
```

- *Clock latency*: Độ trễ (latency) là khoảng thời gian cần thiết để tín hiệu clock được truyền từ nguồn xung clock ban đầu đến các phần tử tuần tự trong thiết kế. Nó bao gồm hai thành phần, độ trễ nguồn và độ trễ mạng. Độ trễ nguồn (source latency) là độ trễ từ nguồn xung clock ban đầu đến điểm xác định xung clock trong thiết kế. Độ trễ mạng (network latency) là độ trễ từ điểm xác định clock đến chân clock của thanh ghi.



*Source Latency và Network Latency*



*Đồ thị giải thích clock latency, uncertainty và transition time*

- *Uncertainty*: Độ không chắc chắn là sự khác biệt lớn nhất giữa sự xuất hiện của các tín hiệu clock tại các thanh ghi. Đây còn được gọi là skew. Skew càng lớn, càng khó đáp ứng các constraints về thời gian.
- *Thời gian chuyển tiếp* (Transition time): Thời gian chuyển tiếp là khoảng thời gian cần thiết để một tín hiệu thay đổi từ mức logic thấp đến mức logic cao (rise time), hoặc từ mức logic

cao đến mức logic thấp (fall time). Thời gian chuyển tiếp của tín hiệu tại đầu vào của cell ảnh hưởng đến độ trễ đầu ra của cell và thời gian chuyển tiếp của tín hiệu đầu ra.

- Các lệnh **set\_clock\_latency**, **set\_clock\_uncerturance** và **set\_clock\_transition** được sử dụng để chỉ định latency, uncertainty và transition time tương ứng.
- Lệnh **get\_clocks** đưa ra danh sách các clock được xác định trong thiết kế hiện tại.
- Lệnh **report\_clock** cung cấp thông tin chi tiết về clock được xác định trong thiết kế hiện tại.

### **IO Timing**

Cần xác định các yêu cầu về timing cho các đầu vào và đầu ra của cổng, nếu không DC sẽ giả sử tín hiệu đến cổng đầu vào tại thời điểm 0 và không ràng buộc bất kỳ đường dẫn nào kết thúc ở cổng đầu ra.

- Các lệnh **set\_input\_delay** và **set\_output\_delay** được sử dụng để hạn chế độ trễ của đầu vào và đầu ra của cổng.
- Lệnh **set\_input\_delay** được sử dụng để chỉ định lượng thời gian được sử dụng bởi cổng logic bên ngoài. Sau đó DC tính toán lượng thời gian còn lại cho logic bên trong và cố gắng đáp ứng nó.

+ **set\_input\_delay 4.5 -clock CLK1 [get\_ports IN1]**

+ **set\_input\_delay 2.3 -clock CLK2 -add\_delay (get\_ports IN1}**

- Lệnh **set\_output\_delay** được sử dụng để chỉ định lượng thời gian mà logic bên ngoài cần. Sau đó DC tính toán lượng thời gian còn lại cho logic bên trong và cố gắng đáp ứng nó.

+ **set\_output\_delay 4.3 -clock CLK1 [get\_ports OUT1]**

### **Combinational Delay**

- Để ràng buộc một đường dẫn tổ hợp, các lệnh **set\_max\_delay** và **set\_min\_delay** có thể được sử dụng.

- Lệnh **set\_max\_delay** cho phép bạn chỉ định độ trễ tối đa của timing path, tức là từ điểm bắt đầu đến điểm cuối. DC sẽ cố gắng làm cho đường dẫn nhỏ hơn giá trị delay được chỉ định.

+ **set\_max\_delay 10.0 -to [get\_ports Y]**

- Lệnh **set\_min\_delay** cho phép bạn chỉ định độ trễ tối thiểu của một đường dẫn thời gian. Nếu một đường dẫn vi phạm yêu cầu được đưa ra trong lệnh **set\_min\_delay**, DC sẽ cố gắng đáp ứng bằng cách thêm delay (sử dụng buffer).

+ **set\_min\_delay 10.0 -from [get\_ports {AB}] -to [get\_ports Z]**

### **Diện tích**

- DC sẽ thực hiện tối ưu hóa diện tích đến mức tối thiểu đến mức giới hạn diện tích được thiết lập. Chỉ định diện tích có thể được giới hạn bằng cách sử dụng lệnh **set\_max\_area**.
- Hầu hết các trường hợp có giới hạn diện tích tối đa bằng 0, trong trường hợp này DC cố gắng đạt được diện tích tốt nhất có thể với tác động đến thời gian chạy.
- Một tùy chọn khác có thể mang lại thời gian chạy tối ưu và chất lượng kết quả tốt là đặt diện tích tối đa là khoảng 90% diện tích tối thiểu.

+ **set\_max\_area X**

#### **6.2.4.4 4.4. Compile**

- Quá trình tối ưu hóa được thực hiện thông qua trình biên dịch (compile\_ultra). Tối ưu hóa bao gồm ba bước: Tối ưu hóa cấp độ kiến trúc, logic và cấp cổng. Tối ưu hóa dựa trên cách mã hóa HDL và các constraints được thiết lập.
  - + *Kiến trúc (Architectural level)*: Giai đoạn này bao gồm các nhiệm vụ như chia sẻ các biểu thức chung, chia sẻ tài nguyên và lựa chọn các thành phần thiết kế. Sau khi tối ưu hóa giai đoạn này, thiết kế được thể hiện ở định dạng GTECH.
  - + *Mức logic (Logic Level)*: Nó có thể được mô tả như một thao tác sử dụng phương trình Boolean.
  - + *Mức cổng (Gate Level)*: Trong giai đoạn này, logic thực sự được chuyển đổi thành các cổng. Tối ưu hóa này có bốn quy trình: Lập bản đồ, tối ưu hóa delay, sửa quy tắc thiết kế và tối ưu hóa diện tích.
- Các tùy chọn lệnh biên dịch khác nhau có thể được sử dụng cho các yêu cầu cụ thể của thiết kế.
  - Một số tip để có kết quả chất lượng tốt hơn là:
    - + Bỏ nhóm các phân cấp không cần thiết.
    - + Bỏ nhóm các khối nhỏ hơn để cho phép chia sẻ tối ưu hóa qua các ranh giới.
    - + Sử dụng lệnh **group\_path** để cô lập đường dẫn IO.
    - + Đặt các ngoại lệ về thời gian bằng cách sử dụng các ký tự đại diện có thể ảnh hưởng đến thời gian chạy, hãy cố gắng tránh nó càng nhiều càng tốt.

- Ví dụ về biên dịch ultra:

```
compile_ultra -gate_clock-scan-no_autoungroup -no_seq_output_inversion -spg
```

\* *Tạo báo cáo:*

- **check\_timing:** Lệnh này kiểm tra các vấn đề ràng buộc như xung clock không xác định, arrival time của ngõ không xác định và các ràng buộc ngõ ra không xác định.

- **report\_qor:** Lệnh này báo cáo chi tiết nhóm đường dẫn timing và số lượng cell, cùng với các thông kê thiết kế hiện tại như diện tích tổ hợp, không tổ hợp và tổng diện tích. Lệnh này cũng báo cáo công suất tĩnh, vi phạm quy tắc thiết kế và biên dịch chi tiết về thời gian.

- **report\_constraint:** Lệnh này cho bạn biết liệu thiết kế có đáp ứng các ràng buộc về timing, diện tích, công suất và quy tắc thiết kế hay không.

- **report\_timing:** Lệnh này cung cấp thông tin chi tiết từng điểm về đường dẫn clock và đường dẫn dữ liệu có skew tồi tệ nhất, cùng với tính toán delay và skew. Bạn có thể kiểm soát phạm vi của thiết kế được báo cáo, số lượng đường dẫn để báo cáo và các loại thông tin đường dẫn được bao gồm cả trong báo cáo. Thông tin này rất hữu ích trong việc xác định nguyên nhân vi phạm về timing và cách khắc phục chúng.

- **report\_clock\_timing:** Báo cáo độ trễ của clock, thời gian chuyển tiếp và các đặc tính skew tại các chân clock được chỉ định của các phần tử tuần tự trong network.

\* *Viết tệp:*

- Khi thiết kế được biên dịch và xác minh thành công sau khi phân tích, bước tiếp theo và cuối cùng là viết các file đầu ra. Lệnh ghi được sử dụng để ghi ra các file cần thiết:

```
write -format [Verilog | ddc VHDL] -hierarchy -output [name]
```

### 6.3 Back-end

#### 6.3.1 Physical Design

Hai vấn đề chính trong Physical design:

- *Thiết kế layout:*

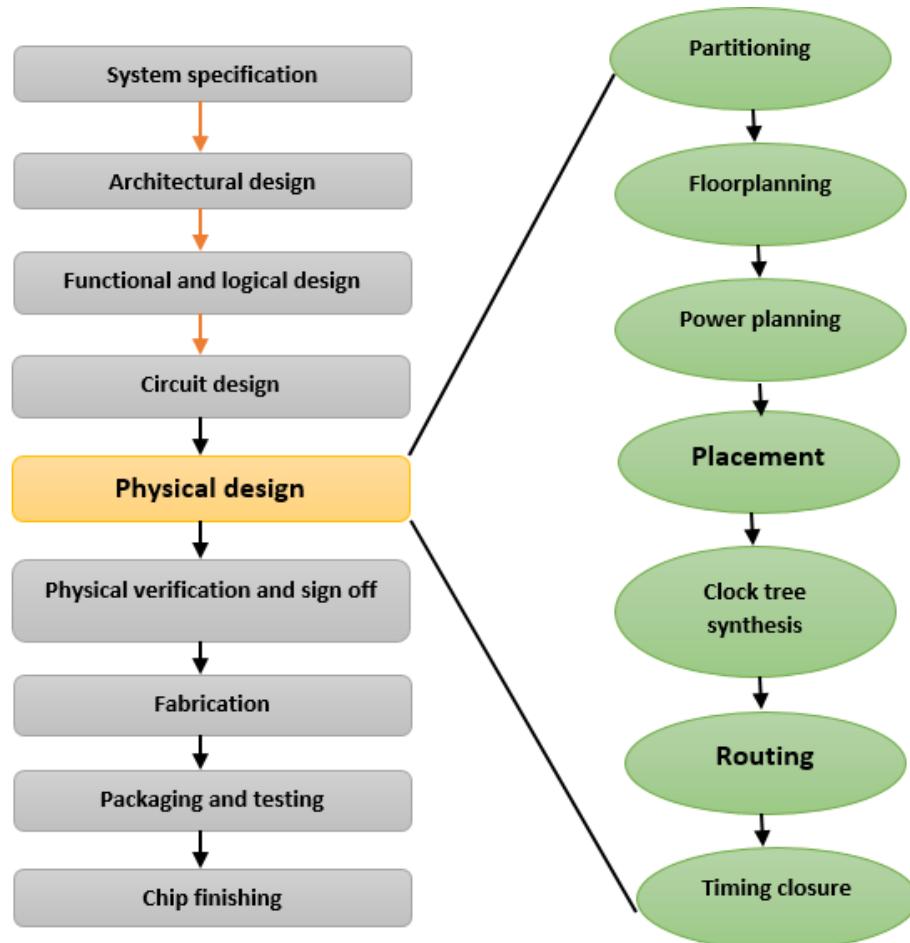
+ Netlist thu được trong qua trình frontend được dùng để tạo layout cho chip. Ở giai đoạn này các linh kiện (transistor, điện trở, tụ điện, cuộn cảm) và các liên kết giữa chúng sẽ được tạo hình (hình dạng thực tế của các linh kiện và dây dẫn trên wafer trong quá trình sản xuất). Việc thiết kế tuân theo các qui luật (design rules) mà nhà sản xuất đưa ra. Có hai loại qui luật

thiết kế là: lamda ( $\lambda$ ) và qui luật tuyệt đối. Với qui luật lamda thì các kích thước như độ dài, rộng phải là bội số của lamda, trong khi qui luật tuyệt đối sử dụng các kích thước là hằng số.

+ Thiết kế số được hỗ trợ lớn bởi CADs, từ việc sử dụng lại thư viện các cells cơ bản cho đến place and route tự động. Chip analog đòi hỏi các thiết kế chính xác và các kỹ thuật chuyên biệt để đảm bảo tương thích (matching) giữa các linh kiện nhạy cảm, chống nhiễu (noise) và đáp ứng tần số.

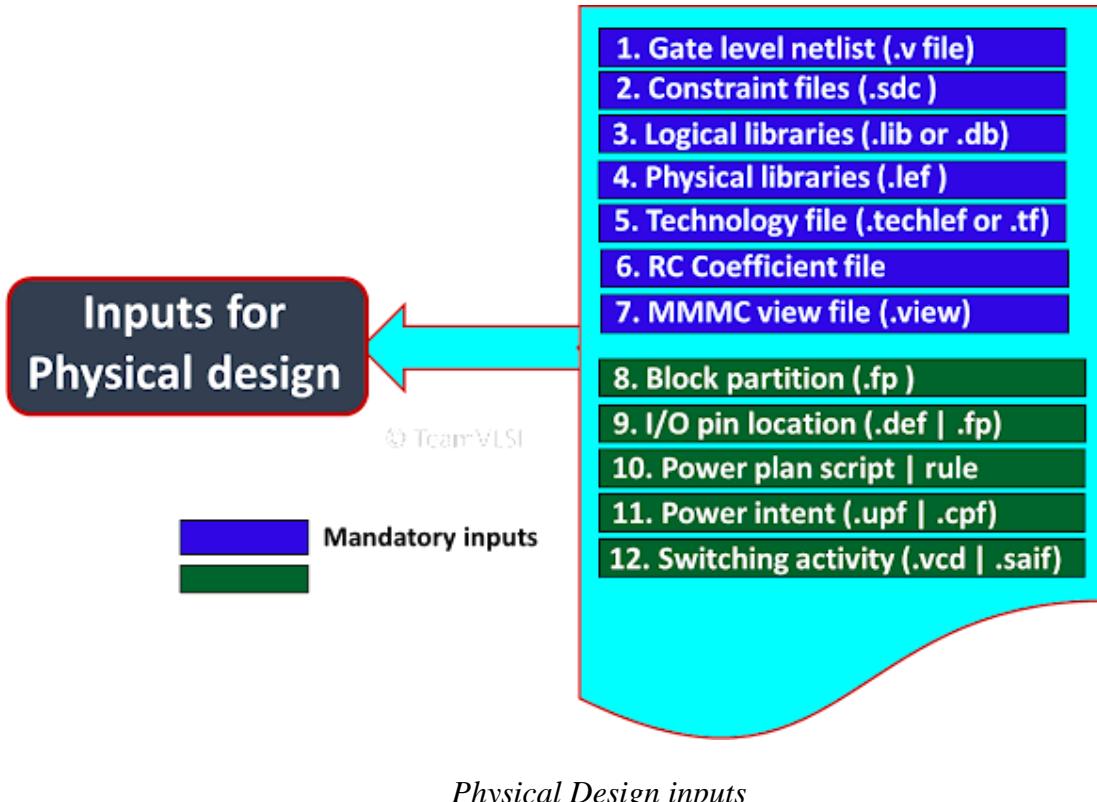
- Kiểm tra DRC và LVS: Sau khi layout chip và hoàn tất kiểm tra qui luật thiết kế (DRC - design rule check), layout được export thành file netlist để đem so sánh với netlist thu được trong quá trình thiết kế luận lý để kiểm tra tính đồng nhất của chúng. Nếu không có sự tương đồng giữa 2 netlist thì phải kiểm tra và sửa lại layout cho đến khi tương đồng. Sau đó các toàn bộ quá trình thiết kế vật lý sẽ được Tape out ra 1 file (\*.gds hay \*.gds2) và gửi đến nhà máy sản xuất.

- *Physical design flow:*



### 6.3.2 Tóm tắt nội dung các bước trong PD flow

#### 6.3.2.1 Physical Input



Trong các files Inputs của Physical design, bước đầu tiên là các files liên quan đến thiết kế bao gồm files netlist ở cấp độ cổng và files liên quan đến constraint. Chúng được team synthesis chuyển tới.

##### - Gate level netlist:

Đây là file netlist tổng hợp. Team synthesis thực hiện tổng hợp dựa trên code RTL với các thư viện standard cell và các constraint và chuyển đổi code RTL thành file netlist dưới dạng cổng dựa trên các standard cell có sẵn.

##### - Constraint file:

Constraint file thường được gọi là file SDC bởi đó là phần mở rộng của file. Về cơ bản, nó chứa:

- + Đơn vị (Thời gian, Điện dung, Điện trở, Điện áp, Dòng điện, Công suất).
- + Giao diện hệ thống (Driving cell, tải).

+ Các ràng buộc (constraint) liên quan đến quy tắc thiết kế (số lượng fanout tối đa).

+ Ràng buộc về timing (Định nghĩa clock, độ trễ của clock, trễ đầu vào/ đầu ra).

Một trong số các ngõ vào yêu cầu liên quan đến thư viện các standard cell, bao gồm:

- *Logical libraries:*

Các thư viện logic còn được gọi là thư viện timing hoặc thư viện chức năng vì có các chức năng, timing và thông tin hiệu năng của các cell. File này về cơ bản chứa thông tin của các standard cells hoặc macros:

+ Timing chi tiết của standard cells/ macros.

+ Setup và hold time của standard cells/ macros

+ Chi tiết chức năng của standard cells/ macros.

+ Diện tích standard cells/ macros

+ Hướng chân và điện dung

+ Công suất bị rò rỉ của standard cells/ macros.

Các logical libraries có thể định dạng theo .lib cho công cụ Cadence hoặc ở dạng .db cho công cụ Synopsys.

- *Physical libraries:*

Các physical libraries chứa góc nhìn tóm tắt của layout cho các standard cells/ macros. File LEF về cơ bản chứa:

+ Kích thước của cell (Chiều cao và chiều rộng)

+ Tính đối xứng của cell

+ Tên, hướng, cách sử dụng, hình dạng, lớp của các chân (pins)

+ Vị trí các chân (Pins)

Physical library ở định dạng file (.lef) cho tool Cadence hoặc định dạng .CELL và .FRAM cho tool Synopsys.

- *Technology file:*

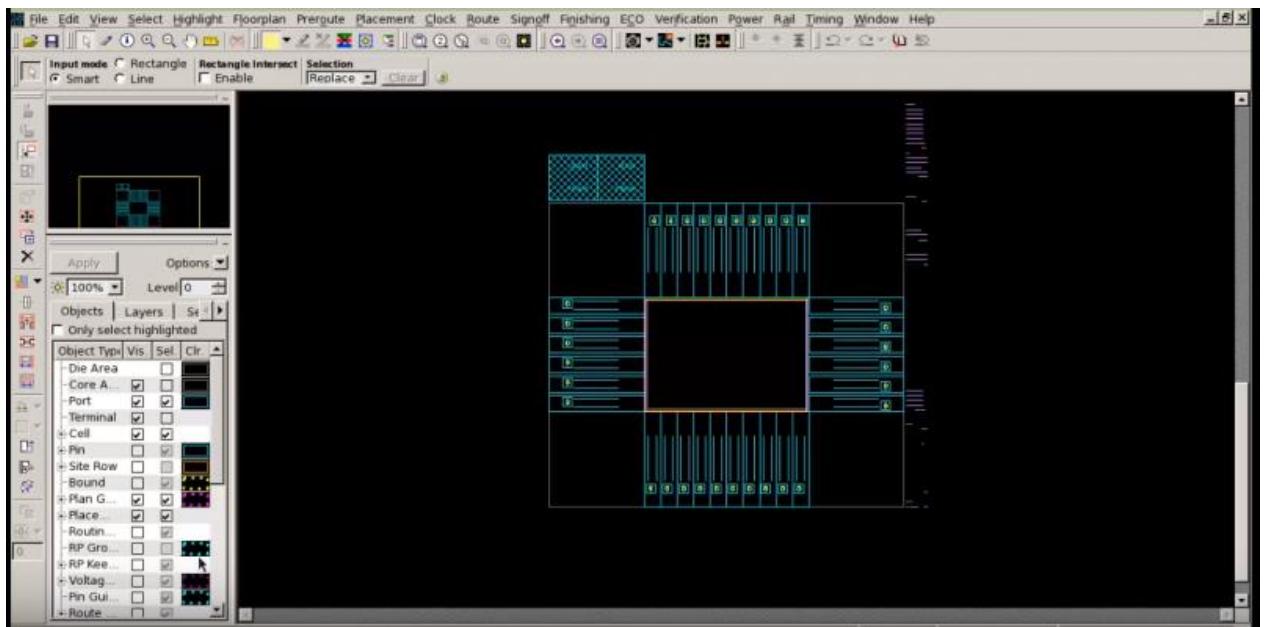
Technology file là phần input quan trọng nhất cho tool PD. Technology file chứa thông tin chi tiết về tất cả các lớp metal, via và các quy tắc thiết kế của chúng. Tệp này ở định dạng mã ASCII và về cơ bản chứa các thông tin sau:

- + Các grid định nghĩa sẵn
  - + Tên các lớp (poly, contact, metal, via).
  - + Các loại và hướng của metal.
  - + Độ cao
  - + Chiều rộng
  - + Khoảng cách
  - + Điện trở
  - File chứa hệ số RC: File TLU là viết tắt của “Table Look-up” hay còn gọi là bảng tra cứu được sử dụng để ước tính và trích xuất RC.
  - File MMMC: File Multi-Mode Multi-Corner được sử dụng để tạo các góc nhìn phân tích khác nhau dựa trên các độ trễ và chế độ constraint khác nhau. Độ trễ được xác định trên thư viện và RC.
- => Tập hợp các tệp trên là cần thiết để bắt đầu cho PD.

### 6.3.2.2        2.2. Floor planning

#### ***Khái niệm Floor planning***

- **Floorplanning** là quá trình lặp đi lặp lại (cả trong FLOW) để lên kế hoạch cho việc layout IC như:
- + Kích thước và hình dạng lõi IC (hay còn gọi là die) và các hàng cho placement (tạo ra các tracks để đặt các Standard cell).
- + Đặt các: Macro, IO, corner..
- + Xác định vị trí các Filler Pad cell (tìm kiếm trên gg để biết nó là gì nha, hoặc hôm khác mình giới thiệu).
- + Tạo các constraints(ràng buộc) để đặt Standard cell ví dụ như blockages cho Macro.
- + Power grid (rings, straps, rails).



*Floor planning trên tool*

### ***Inputs của floorplan***

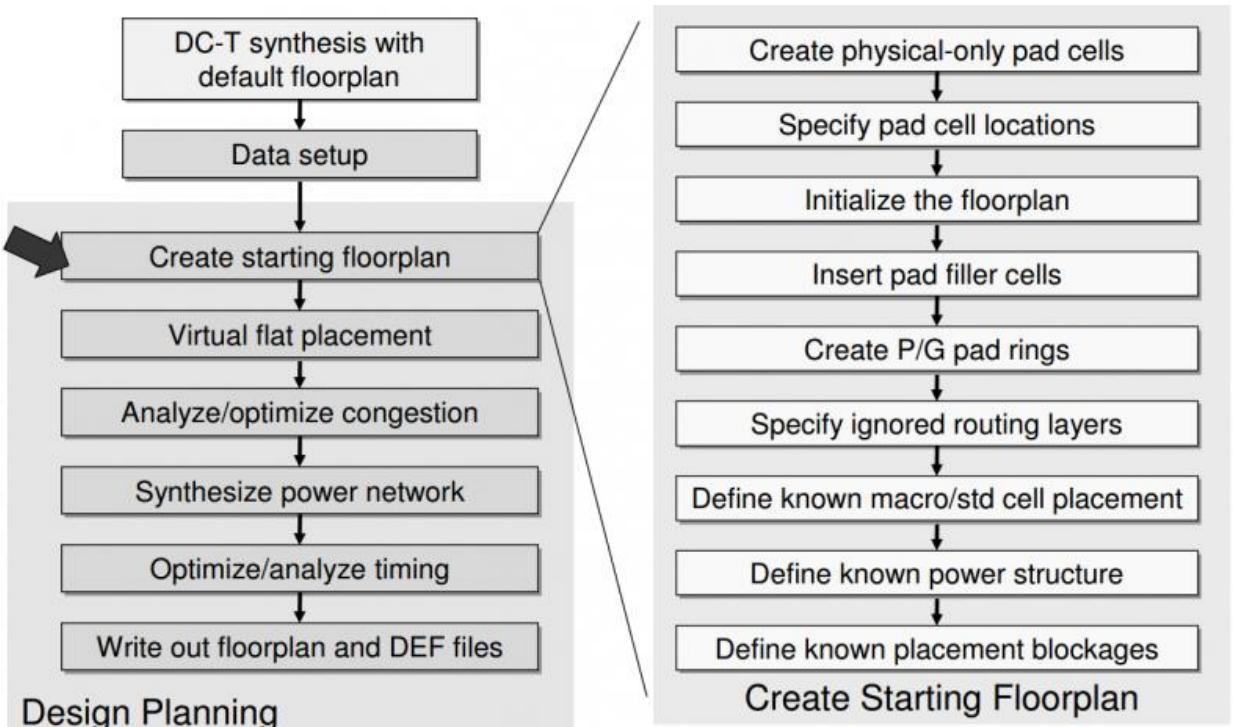
- Synthesized netlist(.v,.vhdl): code Verilog được tổng hợp thành file Netlist
- Design Constrains (SDC): Các ràng buộc của thiết kế như: input delay, output delay, latency, skew, max fanout...
- Thông tin phần vùng cho thiết kế như: kích thước IC, thông tin P/G,..
- IO placement file: Thông tin vị trí đặt các IO, pad cell...
- Macro placement file: thông tin về Macro như: P/G, kích thước Macro..
- Các File thư viện như: Technology File (thư viện công nghệ), Standard Cell Library

### ***Output của floorplan***

- + Diện tích Die/Block.
- + I/O pad được đặt.
- + Các macro được đặt.
- + Power grid được thiết kế.
- + Diện tích standard cell được đặt.

### ***Thiết kế Floorplan***

- Các bước floorplan:



*Floor planning flow*

#### *Các bước quan trọng trong Floorplanning:*

**Core Boundary:** Floorplan xác định kích thước và hình dạng của chip/block. Thiết kế số mức top sẽ có dạng hình chữ nhật hoặc hình vuông hoặc hình chữ L tùy vào yêu cầu. Core boundary xác định Area(diện tích) mà chúng ta sẽ đặt các Standard Cell và các khối IP khác. Chúng ta có thể có không gian route bên ngoài core boundary. Đối với một chip đầy đủ, chúng ta cũng sẽ có IO buffers và IO pads đặt bên ngoài Core boundary .

Trong công cụ PnR, Floorplanning có thể được kiểm soát bởi các thông số khác nhau:

- **Aspect ratio:** Đây là tỷ số chiều cao chia cho chiều rộng, xác định floorplan là hình vuông hay hình chữ nhật. Aspect ratio bằng 1 sẽ cho chúng ta floorplan hình vuông.

- **Core utilization:** Core utilization = (standard cell area + macro cells area)/ total core area. Tỷ lệ sử dụng core là 0.8 có nghĩa là 80% diện tích dùng để place các cell, trong khi 20% cho việc route.

**IO Placement/Pin placement:** Nếu chúng ta đang làm một thiết kế digital-top, chúng ta cần phải đặt thêm IO pads và IO buffers của chip. Một chip hình chữ nhật hoặc hình vuông có pad ở bốn bên. Chúng ta có thể lấy các bên và vị trí tương đối của PADs từ tool. Các pad này có chức năng kết nối các Power và Ground của IO lại với nhau.

**Macro placement:** Khi chúng ta đã có sẵn kích thước và hình dạng của floorplan, bước tiếp theo là đặt các Macro vào bên trong core. Có thể đặt tự động các Macro bằng lệnh hoặc bằng

tay. Tốt nhất là cả hai. Cũng có thể thêm Blockage xung quanh Macro để tránh các Cell đặt gần các IO của Macro, có thể dễ dẫn đến congestion trong khi Route.

**Creating Power Rings and Straps:** Ở giai đoạn này, ta xác định các trunks cung cấp Power cho lõi. Nhưng cũng phải đảm bảo rằng tất cả các macros đều có đủ các rings/straps xung quanh nó để nối vào PG trunks. Một cấu trúc Power tốt sẽ thực hiện lặp đi lặp lại để tối ưu và phân tích IR drop ở giai đoạn sau.

#### 6.3.2.3        **2.3. Placement**

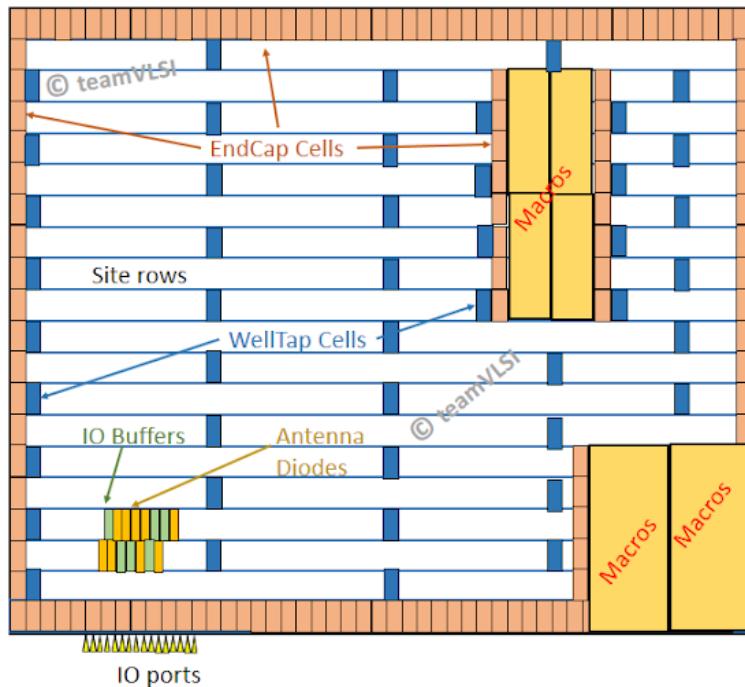
##### ***Khái niệm về Placement***

Placement là quá trình đặt các standard cell bên trong khuôn của lõi chip ở một vị trí tối ưu. Tool sẽ cố gắng đặt standard cell theo cách mà thiết kế phải có ở việc giảm tối thiểu congestion và timing tốt nhất. Mỗi tool PnR cung cấp các lệnh khác nhau để người dùng có thể tối ưu hóa thiết kế theo cách tốt hơn về timing, congestion, diện tích và công suất theo yêu cầu của họ. Placement không chỉ đặt các standard cell có trong netlist mà còn đặt nhiều physical cell và thêm buffer/ inverter theo yêu cầu để đáp ứng các yêu cầu về timing, DRV và yêu cầu của việc sản xuất chip.

##### ***Các bước cơ bản của Placement***

1. **Pre Placement**
2. **Initial Placement / Course Placement / Global Placement**
3. **Legalization**
4. **HFNS (High Fanout Net Synthesis)**
5. **Iteration for Congestion, Timing, DRV, and Power Optimization**
6. **Multibit flop conversion**
7. **Timing optimization iterations**
8. **Scan-Chain Reorder**
9. **Tie Cell insertion**
10. **Save Design**

*Placement step*



### *Pre-placement*

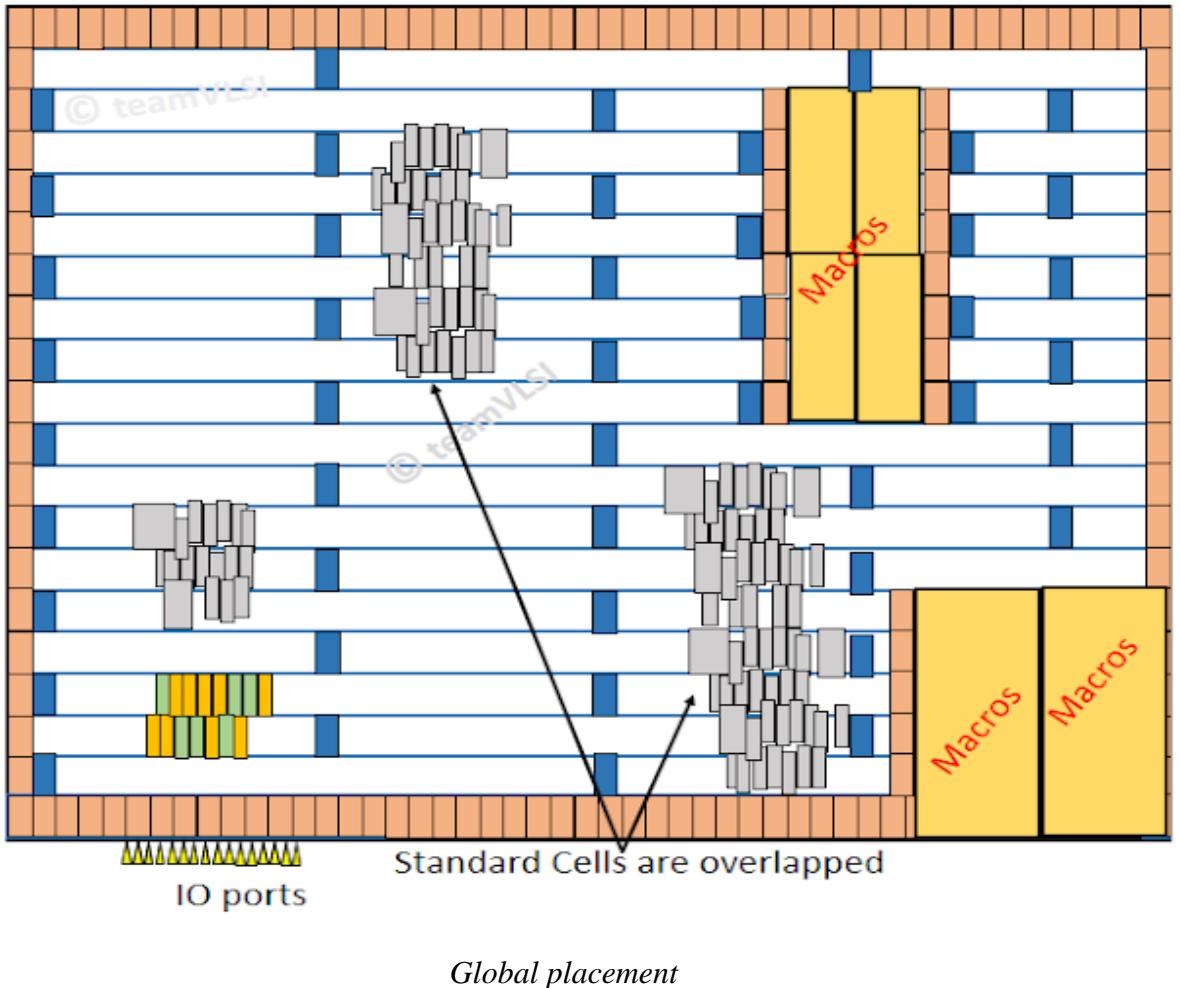
Trước khi bắt đầu bước placement các standard cell trong thực tế có trong netlist tổng hợp, chúng ta cần đặt các physical cells khác nhau như end-cap cell, well-tap cell, IO buffer, điốt chống hiệu ứng antenna và spare cell có ở trong hình 1. Thường sẽ được thực hiện:

- + Thêm điốt Antenna và buffer ở block level ports.
- + Thêm Tap cell để tránh vấn đề latch-up.
- + Chèn Spare cell
- + Nếu thiết kế yêu cầu, thêm các cell đặt biệt.
- + Chèn tụ de-coupling (hiếm khi ở thiết kế hiện tại)

### *\* Course placement:*

- + Khi giai đoạn Pre-placement đã được hoàn thành, chúng ta có thể bắt đầu placement các standard cell nhưng trước đó, chúng ta phải cung cấp tất cả các cài đặt vị trí và tối ưu hóa chính xác mà chúng ta muốn được áp dụng cho tool để thực hiện.
- + Sau khi cung cấp tất cả các cài đặt vị trí này, chúng ta có thể gọi lệnh vị trí (**place\_op\_design**). Đầu tiên, tool thực hiện global placement, trong đó tool xác định vị trí gần đúng của từng cell theo các ràng buộc về timing, congestion và nhiều mức điện áp đặt vào. Bất kỳ macro nào được đặt sẽ hoạt động như một khối placement.

- + Trong giai đoạn này, tool sẽ không kiểm tra bất kỳ sự chồng lấn nào của các trường hợp. Ở hình 3.2.7, có thể thấy rằng các standard cell được đặt ở một vị trí gần đúng nhưng chưa được tối ưu.



#### \* Legalization:

- + Trong giai đoạn global placement, sẽ xảy ra sự chồng lấn. Trong bước này, tool sẽ di chuyển các khối ở những nơi gần đó để khắc phục sự chồng lấn.
- + Để khớp các chân nguồn thích hợp như chân vdd của standard cell phải ở trên vdd rail và đối với vss là trên vss rail. Quá trình này được gọi là Legalization.

#### \* HFNS (High Fanout Net Synthesis)

- + Ban đầu, có một số nets có số lượng fanout rất cao. Chúng ta có constraint về số lượng fanout tối đa, vì vậy chúng ta cần phân phối các phần chìm của nets cho các trình điều khiển khác nhau.

+ Quá trình thêm buffer và tách fanout được gọi là tổng hợp fanout (HFNS). Vì vậy, trong bước này, tất cả các nets có fanout cao sẽ được tổng hợp lại.

\* *Iteration for Congestion, Timing, DRV and Power Optimization*

+ Trước tiên, tool thực hiện một global route sớm và ước tính tình trạng congestion trong thiết kế.

+ Tiếp theo, tool bắt đầu trích xuất RC để tính toán độ trễ cho thiết lập phân tích. Công cụ cố gắng giảm thiểu thiết lập WNS và TNS trong bước này. Tương tự, công cụ cũng cố gắng giảm thiểu VNDCCH và Power trong giai đoạn này.

\* *Multibit flop conversion:*

+ Nếu người dùng bật chuyển đổi flip flop nhiều bit trong flow này thì trước tiên tool sẽ kiểm tra các flip flop nhiều bit có sẵn trong thư viện.

+ Tool xem xét mức độ quan trọng timing liên quan đến một bit lỗi đơn lẻ và tập hợp các constraint người dùng để cho chuyển đổi nhiều bit và dựa trên các constraints mà tool chuyển đổi một bit thành nhiều bit.

\* *Timing optimization iteration:*

+ Đây là một bước dài trong đó công cụ cố gắng giảm thiểu WNS và TNS của mỗi nhóm đường dẫn trong những lần lặp khác nhau.

+ Trong trường hợp kết quả không tốt sau giai đoạn này, chúng ta có thể tiếp tục tối ưu hóa cho timing. Tương tự, đối với congestion, chúng ta có sửa chữa các vùng tắc nghẽn, sau đó là tối ưu hóa tăng để có được kết quả tốt hơn. Nhưng các bước bổ sung này có thể sẽ làm tăng về timing.

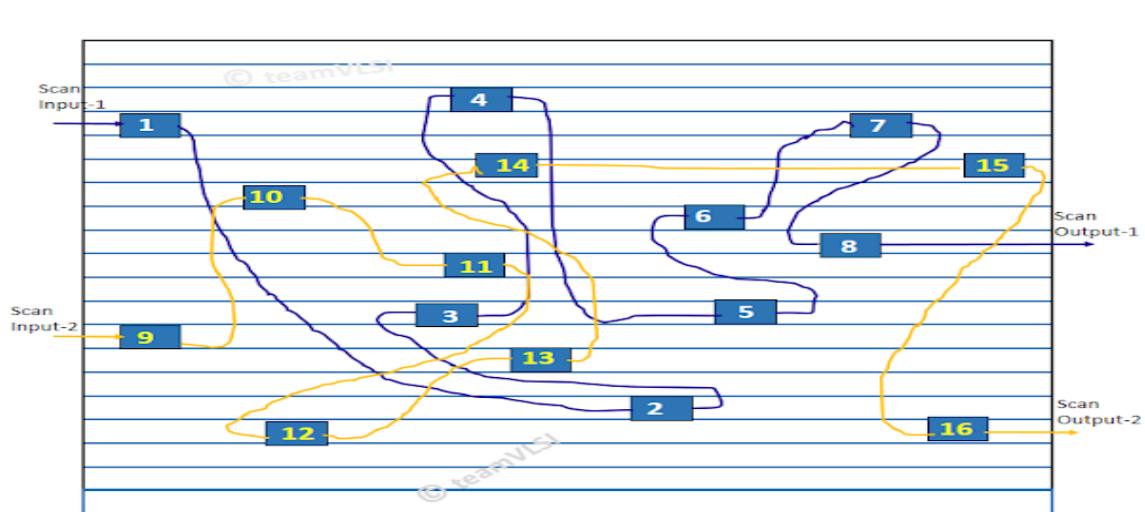
\* *Scan-chain reorder:*

+ Đây là bước để kết nối lại những scan chain trong thiết kế để tối ưu cho việc routing bằng cách sắp xếp lại các kết nối để cải thiện congestion cũng như timing.

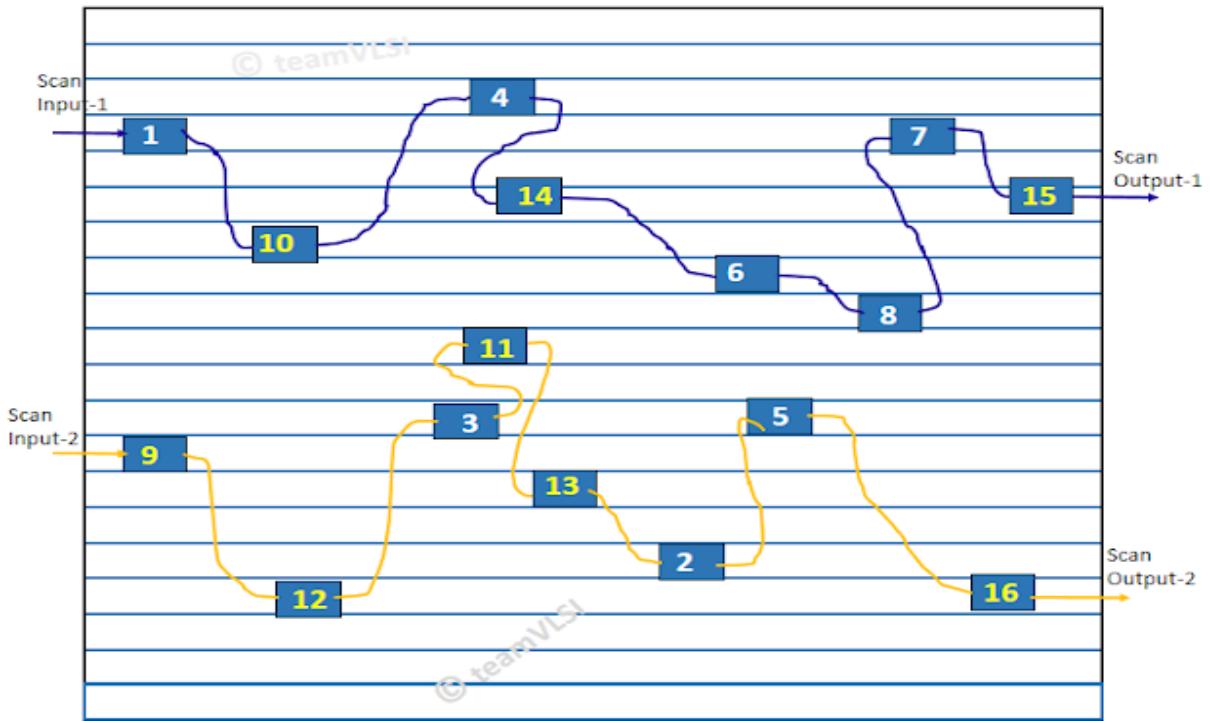


*Chuỗi Scan Chain trước khi placement*

- + Scan chain được thực hiện tùy ý trong quá trình synthesis. Sau khi sắp xếp và tối ưu hóa, chúng ta có một vị trí cho mỗi lần quét, vì vậy nó cần được sắp xếp lại để có khả năng định tuyến tốt hơn.



*Scan Chain sau khi placement*



*Scan Chain sau khi được sắp xếp lại*

- *Chèn Tie Cell:*

- + Có một số input không sử dụng của cổng logic trong netlist được gắn với vdd hoặc vss. Chúng ta không thể để bất kỳ đầu vào nào của standard cell là thả tự do, nó phải được gắn vdd hoặc vss.
- + Việc kết nối input của cổng logic là không được khuyến khích khi kết nối trực tiếp với vdd hoặc vss, do đó ta nên tie high và tie low cell trong thư viện. Vì vậy, trong bước này tool đặt các tie high và tie low cell, về cơ bản nó là một ngõ ra của logic cell và nó kết nối input của cổng logic với vdd hoặc vss tương ứng.
- *Lưu thiết kế:* Cuối cùng, chúng ta lưu cơ sở dữ liệu và sẽ sử dụng cơ sở dữ liệu này trong giai đoạn tiếp theo, đó là CTS.

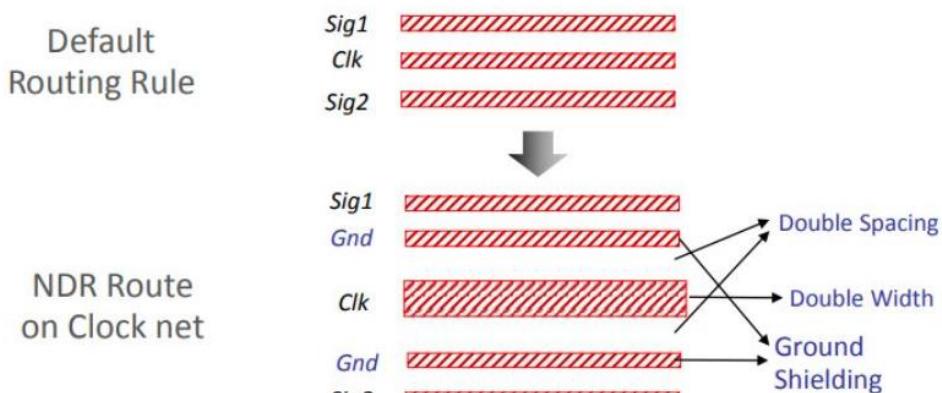
*Output của placement*

- Báo cáo về congestion.
- Báo cáo về timing.
- Thiết kế với tất cả standard cell được đặt trong diện tích của core.
- Placement DEF file.

### 6.3.2.4 2.4. Clock Tree Synthesis (CTS)

#### Tối ưu trước khi CTS

- Đặt các chỉ thị tối ưu hóa: don't\_use, size\_only
- Thực hiện HFNS:
  - + Các nets có fanout cao được tổng hợp trước khi CTS.
  - + HFNS là buffer của các nets có fanout cao. Thường thì net có Fanout cao có thể có số lượng Fanout hơn 1000 thường sẽ được đặt lại hoặc xóa.
- Đặt CTS routing rules: Non Default Rules (NDR).
- \* Non Default Rules (NDR):
  - + Quy tắc routing do người dùng xác định nằm ngoài rules mặc định.
  - + Thường được sử dụng để làm cứng các nets nhạy cảm như nets của clock.
  - + NDRs làm cho Clock routes ít nhạy cảm hơn với các hiệu ứng Cross Talk hoặc EM.
  - + Khoảng cách gấp đôi hoặc ba lần để tránh crosstalk
  - + NDRs sẽ cải thiện về mặt delay.

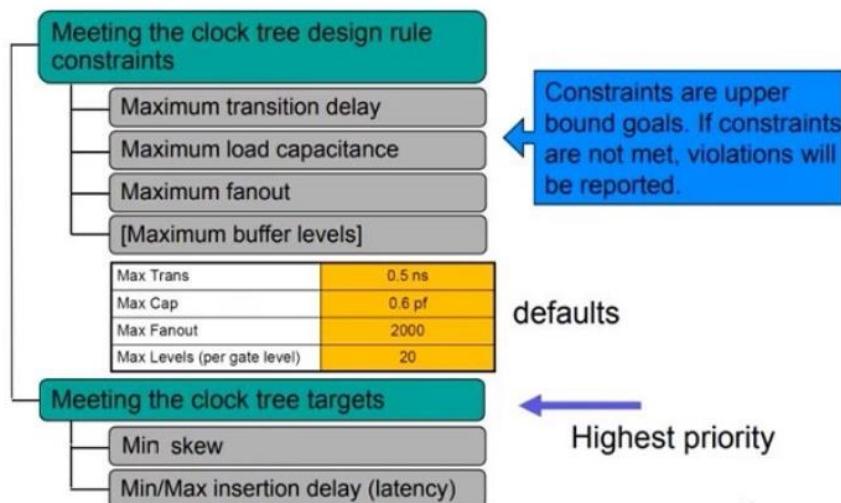


#### Định nghĩa về CTS

- CTS là một trong những bước quan trọng nhất trong PnR. CTS QoR quyết định về timing và công suất. Trong hầu hết IC clocks sẽ tiêu thụ 30-40% tổng công suất. Vì vậy, kiến trúc clock hiệu quả, cài đặt clock và clock tree giúp giảm thiểu điện năng tiêu thụ.

- Quá trình phân phối xung clock và cân bằng tải được gọi là CTS. Đây cũng là quá trình chèn các bộ buffer hay inverter theo các đường clock của thiết kế ASIC để đạt được zero skew hay balanced skew. Điểm bắt đầu CTS là một nguồn clock duy nhất và điểm kết thúc CTS là các chân clock của lần lượt các cell.

- *Mục tiêu của CTS:*



### *Inputs của CTS*

- Technology file (.tf).
- Netlist.
- SDC.
- Library files (.lib & .lef).
- TLU+ file.
- Placement DEF file.

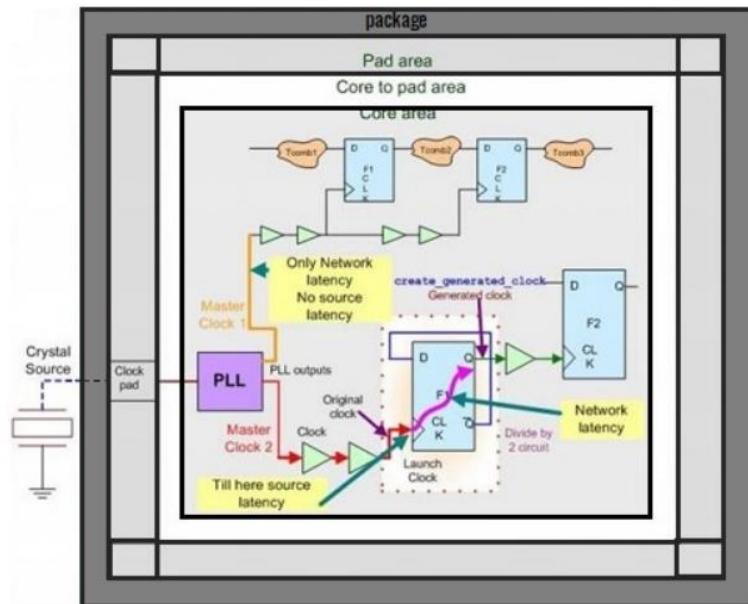
\* *Bước kiểm tra cần được thực hiện trước khi CTS:*

- + Kiểm tra legalization, nguồn, standard cells rain, cũng như kiểm tra các kết nối PG.
- + Timing QoR, DRVs
- + Congestion: chạy CTS trên thiết kế qua những vùng có congestion có thể tạo ra nhiều tắc nghẽn hơn và các vấn đề khác (Noise / IR).
- + Xóa trạng thái Don't\_use trên buffer và inverter.
- + Check các trạng thái don't\_touch, don't\_size trên thành phần clock.

### *Các thuật ngữ thường được sử dụng trong CTS*

\* *Clock Latency*:

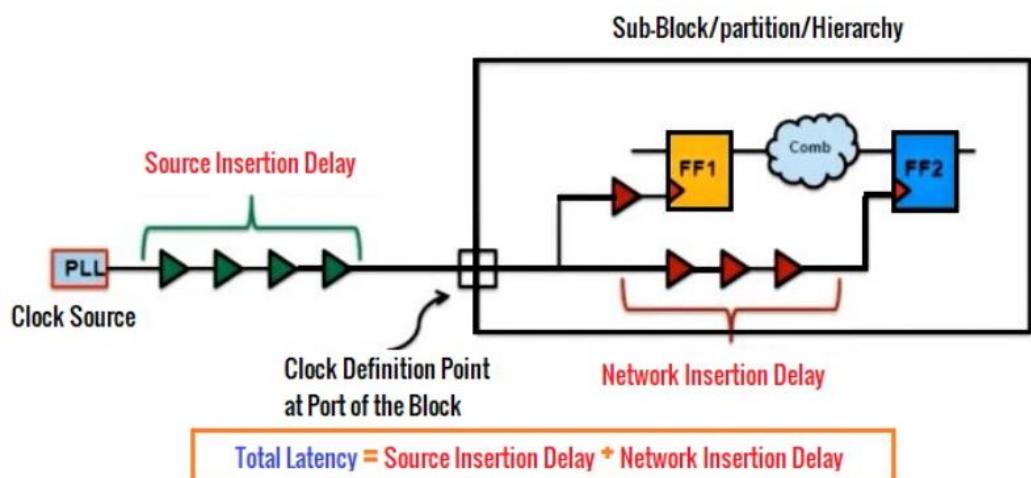
- Tổng thời gian thực hiện của tín hiệu clock khi tiếp cận ngõ vào thanh ghi
- Source latency là khoảng thời gian từ clock nguồn đến clock tại chân
- Network latency là khoảng thời gian từ clock tại chân đến clock leaf cell ở thiết kế.



Hình 3.2.13 Clock latency

\* *Insertion delay (ID)*:

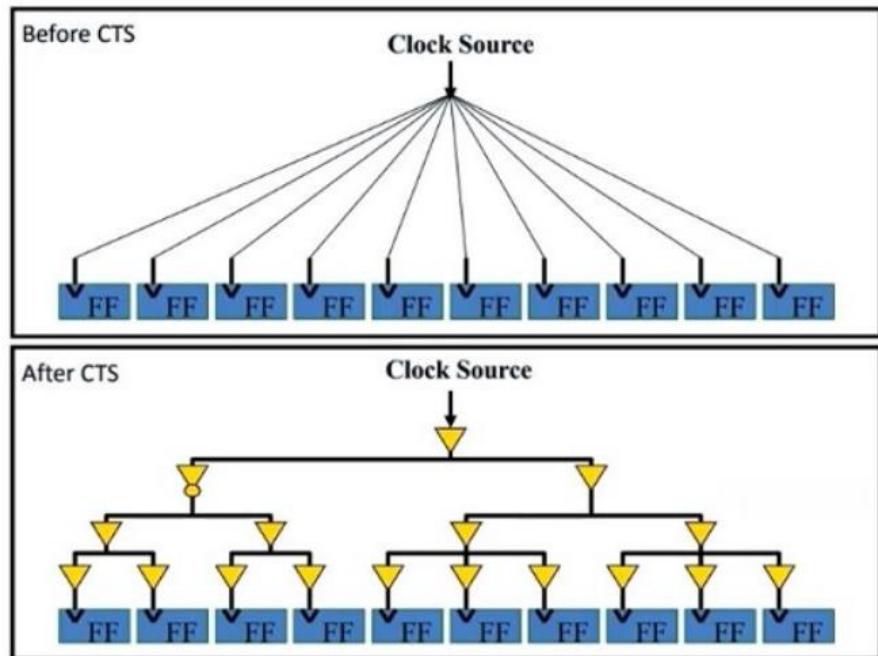
- ID là clock latency nhưng đó là khi clock tree được tổng hợp
- ID là độ trễ trong thiết kế thực và clock latency là thời gian delay ảo.
- Latency là đích đến của công cụ, là độ trễ đạt được sau bước CTS.
- Source và Network Latency: Là clock gốc và clock tạo ra



### Công thức tính total latency

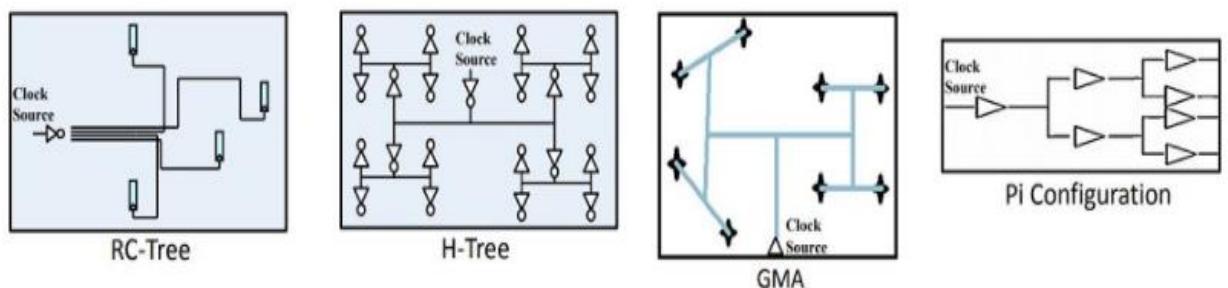
\* Clock Tree:

- Clock Tree là một phần tử Clock Source đến Clock Sinks.
- CTS là bước để tạo ra các đường clock từ Clock Source đến Clock Sinks.
- Tất cả chân clock của Flip Flop phải được xem xét như Clock Sinks khi CTS kết thúc.



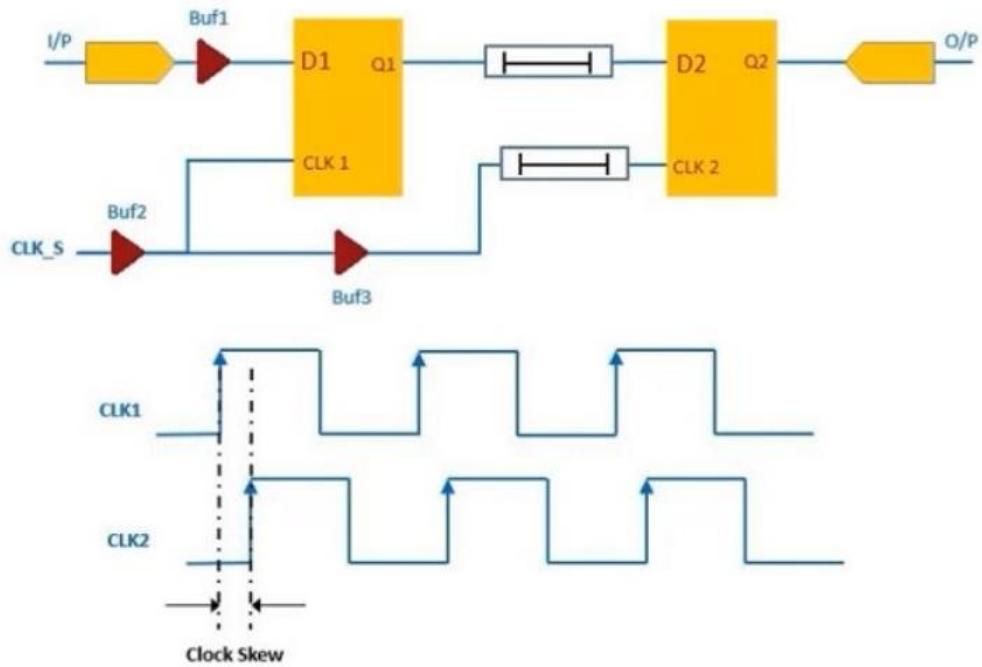
Hình 3.2.15 Trước và sau CTS

- Các thuật toán CTS thường sử dụng: *RC-tree*, *H-tree*, *GMA*, *Pi Configuration*



Các thuật toán thường được sử dụng trong CTS

\* Clock skew:



*Clock skew*

- Clock skew là khoảng thời gian khác biệt giữa việc truyền dữ liệu từ 1 clock đến 2 điểm riêng biệt.
- Positive skew: Là trường hợp capture clock đến trễ hơn clock phát ra.
- Negative skew: Là trường hợp capture clock đến sớm hơn clock được phát ra
- Local skew: Là khoảng thời gian clock di chuyển tại 2 pin liên tiếp.
- Global skew: Được định nghĩa như sai lệch giữa độ trễ của đường dẫn clock ngắn nhất và dài nhất đạt được giữa lần lượt 2 phần tử.

### ***CTS Flow***

- + Kiểm tra và tối ưu các vị trí Macro.
- + Đọc CTS SDC: Clock Tree bắt đầu ở chân clock được xác định bởi SDC và kết thúc ở stop pin của flip flop.
- Tạo ra CTS file:
- + Max. Skew; Max. and Min. Insertion Delay; Max. Transition, Capacitance, Fanout.
- + No. Buffer levels; Danh sách Buffer/ Inverter.
- Biên dịch CTS bằng cách sử dụng CTS Spec. file.

- Đặt các Clock Tree cell.

\* Các phân tích cần thực hiện từ CTS:

- Báo cáo về Timing (Cả setup và hold time).
- Nếu timing không đáp ứng thì các đường clock được nhóm lại.
- Báo cáo về ID, skew và check các mục đích cần đạt.
- Báo cáo về mục tiêu DRV (Fanout, diện dung).
- Kiểm tra các Leaf Cell (Clock Sinks) đã đạt được.
- Thực hiện QoR.
- Kiểm các constraint về design rule.
- Kiểm tra các constraint về routing.
- Báo cáo về công suất và diện tích.

### ***Output của CTS***

- Báo cáo về timing, congestion, skew, insertion delay.
- CTS DEF file.

#### **6.3.2.5            2.5. Routing**

##### ***Định nghĩa về Routing***

- Tạo kết nối vật lý giữa các chân tín hiệu bằng cách sử dụng các lớp metal được gọi là Routing. Routing là giai đoạn sau CTS và đã tối ưu hóa khi xác định các đường dẫn chính xác cho kết nối của các standard cell, macro và các chân I/O.

- Sau CTS, chúng ta có thông tin của tất cả các cell được đặt, các blockage, buffer/ inverter của clock tree và các chân I/O. Tool dựa vào thông tin này để hoàn thành tất cả các kết nối được xác định trong file netlist sao cho:

+ Giảm thiểu vi phạm DRC khi routing.

+ Thiết kế với 100% được định tuyến, đạt mức vi phạm LVS tối thiểu.

+ Giảm thiểu vi phạm về SI.

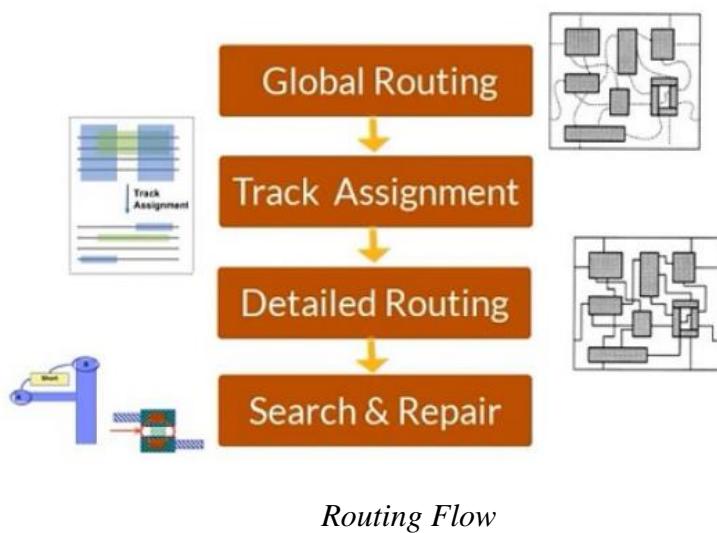
+ Không có hoặc giảm thiểu số lượng điểm tắc nghẽn.

+ Timing DRCs và QoR được đáp ứng tốt.

\* *Đối tượng của Routing:*

- + Đáp ứng về Skew.
- + Loại bỏ mạch bị hở.
- + Đường dẫn định tuyến phải được đáp ứng setup và hold timing.
- + DRVs lớn nhất. Điện dung phải ở dưới mức giới hạn.
- + Các metal phải đáp ứng yêu cầu DRC từ nhà máy sản xuất chip.

**Routing Flow**



\* *Global Routing:*

*Global Routing* là bước đầu tiên của Routing bằng cách route các đường dẫn thô trước. Trước tiên phân vùng được routing thành các ô chữ nhật được gọi là gcell, và quyết định các đường đi theo ô cho tất cả các net trong khi tối ưu hóa một số chức năng mục tiêu nhất định (ví dụ: tổng chiều dài dây và thời gian của mạch).

Blockage, pins và routing tracks bên trong mỗi ô quyết định khả năng routing cho mỗi gcell. Sau đó, tất cả các net được chỉ định cho gcell được ghi nhận và nhu cầu cho track trong mỗi gcell được tính toán và báo cáo.

- Global routing được thực hiện qua 2 công đoạn chính:

+ Giai đoạn routing ban đầu, trong đó các nets không kết nối được định tuyến cho mỗi gcell đã được tính toán.

+ Các giai đoạn route lại, trong đó vùng congestion các gcell với các nets bị tràn được giảm thiểu bằng cách tách và route lại đường net.

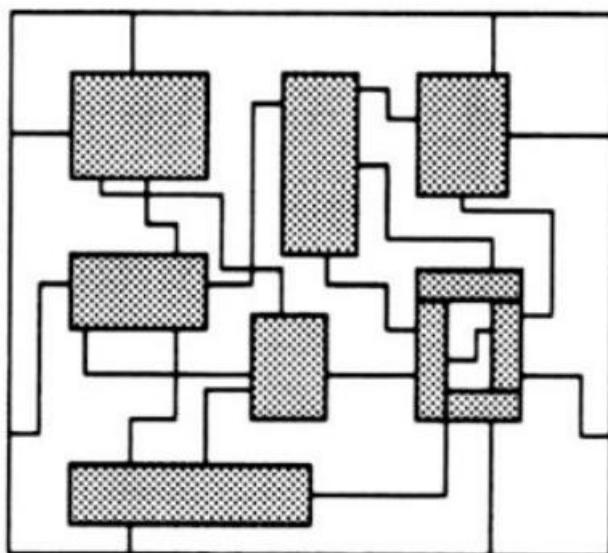
\* *Track Assignment:*

- Lấy từ layout đã được thực hiện sau giai đoạn Global Routing, phân bổ mỗi net thông qua các track và layer được chỉ định.
- Mặc dù tất cả các net đã được route, ở giai đoạn này chưa cần tuân thủ DRC rules, SI và tính vi phạm về timing.
- Giúp giảm thiểu số lượng via sử dụng.

\* *Detail Routing:*

*Detail Routing* sử dụng tool route thực sự sau các bước Global routing và Track Assignment. Đây là bước thực sự đặt metal để kết nối các chân một cách hợp lý với net và các chân khác trong thiết kế.

Mục tiêu chính của Detail Routing là hoàn thành tất cả các kết nối được yêu cầu mà không để lại các lỗi về hở mạch cũng như khoảng cách. Quá trình detail routing bắt đầu với việc chia một khối thành các khu vực cụ thể được gọi là Sbox, được biểu thị chung dưới dạng gcell.



*Detail Routing*

\* *Search and Repair:*

Được tiến hành sau mỗi lần Detail Routing. Trong bước này các lỗi về ngắn và hở mạch được xác định vị trí và đi route lại. Mục tiêu là fix tất cả các lỗi có thể có.

### 6.3.2.6 2.6. Design for manufacturing (DFM)

#### **Định nghĩa về DFM**

- Design for manufacturing (DFM) đề cập đến các hành động được thực hiện trong giai đoạn PD để đảm bảo thiết kế có thể được sản xuất chính xác.

- Ở phạm vi lớn hơn, hầu hết các lỗi trong quá trình sản xuất vi mạch là do các bước của quy trình vượt quá khả năng chịu đựng, tức là sự thay đổi level của macro hoặc các hạt bụi ngẫu nhiên làm gián đoạn quá trình quang khắc (lithography) qua lớp mặt nạ hoặc đục nhúng trong một lớp của tám wafer. Quy trình phản hồi kiểm soát và phòng sạch có hiệu quả trong việc kiểm soát hai cơ chế này tương ứng và tạo ra năng suất cao.

- Hiệu ứng nhiễu xạ trở nên đáng kể khi ánh sáng tương tác với các vật thể và khe tiếp cận các kích thước của bước sóng ánh sáng. Sản xuất chất bán dẫn đã vượt xa ngưỡng đó. Từ 130nm đến 65nm, các công nghệ nâng cao độ phân giải, bao gồm hiệu chỉnh tiệm cận quang học (còn được gọi là kỹ thuật quang khắc có tính toán), có thể xử lý các biến dạng do hiệu ứng nhiễu xạ gây ra.

- Thật không may cho các nhà thiết kế vi mạch, ở bước RET không thể sửa chữa tất cả các vấn đề một khi bạn đi vào các nút nhỏ nhất. Do đó, các xưởng sản xuất IC phải thêm các quy tắc thiết kế và các nhà thiết kế cần thực hiện các thay đổi thiết kế, loại bỏ hoặc sửa đổi các tính năng trong bộ cục mà không thể sản xuất chính xác. Tại mỗi nút, các quy tắc DFM trở nên phức tạp hơn và phạm vi ảnh hưởng sẽ mở rộng theo phạm vi.

#### **DFM Scoring**

- Thực hiện DFM cuối cùng tập trung vào việc cải thiện năng suất và ngăn ngừa các khuyết tật có thể xuất hiện sau khi sản phẩm được giao cho khách hàng.

- DFM Scoring là một phương pháp sử dụng một tập hợp các phép đo của layout vật lý để xác định xem thiết kế có đủ tốt để đạt được lợi nhuận chấp nhận hay không.

- Các phương pháp này có xu hướng theo yêu cầu bởi nhà máy sản xuất chip, bởi vì chúng dựa trên các rule thiết kế do nhà máy quy định, dựa trên kinh nghiệm và dữ liệu của họ từ các chip thử nghiệm và chip sản xuất tại mỗi công nghệ khác nhau.

### 6.3.2.7 2.7. Static Timing Analysis (STA)

#### **Định nghĩa về STA**

- Phân tích timing tĩnh là một phương pháp xác định sự đáp ứng về mặt timing của thiết kế bằng cách kiểm tra tất cả các đường đi trong thiết kế có vi phạm timing hay không trong điều kiện xấu nhất. Phân tích timing chỉ quan tâm đến độ trễ xấu nhất có thể xảy ra trong mạch số chứ không quan tâm đến chức năng của mạch số.

- So với việc mô phỏng (simulation) mạch số thì phân tích timing được thực hiện nhanh hơn và toàn diện hơn:

+ Nhanh hơn là bởi vì phân tích timing không cần sử dụng đến nhiều các test vector khác nhau như khi mô phỏng chức năng (Test vector là các giá trị đầu vào của thiết kế mà người mô phỏng sẽ xây dựng để kiểm tra chức năng thiết kế)

+ Toàn diện hơn là vì nó kiểm tra trường hợp timing xấu nhất cho tất cả các điều kiện logic có thể chứ không chỉ hạn chế bởi một tập các test vector cố định nào.

- Các ràng buộc (constraint) về timing, diện tích và năng lượng tiêu thụ là ba yếu tố ảnh hưởng đến quá trình tổng hợp và thiết kế vật lý (quá trình layout) của mạch số. Timing là ràng buộc quan trọng nhất của thiết kế. Diện tích lớn hay công suất cao thì chức năng của chip vẫn đảm bảo nhưng timing không đúng thì chip sẽ hoạt động sai chức năng.

- Giá trị đầu vào của một cell đồng bộ có thể thay đổi trong quá trình hoạt động của chip, mấu chốt của việc kiểm tra timing là để đảm bảo một cell đồng bộ luôn bắt được dữ liệu đúng của đầu vào.

\* *Mục đích của STA:*

+ Đầu tiên, STA tính toán độ trễ đường dẫn dựa trên tool tối ưu. Sau đó dựa vào thông số độ trễ đường dẫn, tool sẽ chọn cell từ thư viện timing để tạo ra mạch để đáp ứng yêu cầu về thời gian.

+ Thứ hai, STA phân tích timing của mạch để xác minh rằng liệu mạch có thể hoạt động tại tần số được chỉ định.

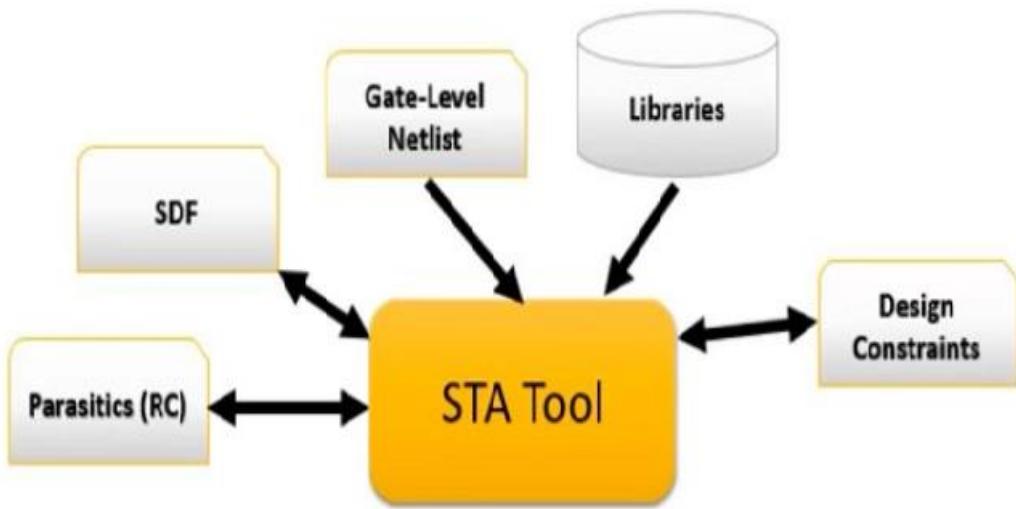
\* *Các bước chính của STA:*

+ Chia thiết kế thành tập hợp các đường dẫn để tính timing.

+ Tính toán độ trễ mỗi đường dẫn.

+ Kiểm tra tất cả các đường dẫn để xem xét nếu tất cả các constraints đều đáp ứng.

\* *STA inputs và outputs:*

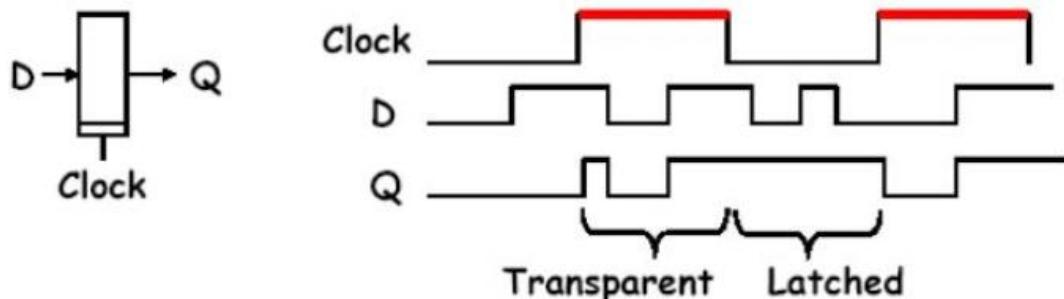


*STA inputs và outputs*

### Các thuật ngữ thường sử dụng cho STA

\* *Transparent Latch*:

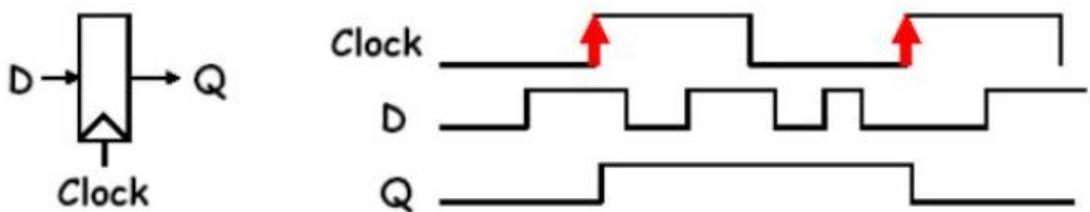
- Q Ghi lại dữ liệu D khi clock ở mức cao, chốt lại khi clock ở mức thấp.



Hình 3.2.28 Transparent Latch

\* *D-type Register hay Flip-flop*:

- Q được giữ lại trạng thái D khi clock có cạnh lên, và khi có clock cạnh xuống, Q tự do.



Hình 3.2.29 D-Flipflop

\* *Delay*:

- + Thời gian một tín hiệu cần để truyền qua một cell hoặc net.
- + Độ trễ đường dẫn thực tế là tổng của độ trễ cell và net suốt đường timing path.
- *Intrinsic delay*: Bên trong cell từ chân input đến chân output do điện dung bên trong.

- *Propagation delay*:

- + Độ trễ của một cell đối với sự thay đổi của tín hiệu đầu vào dẫn đến sự thay đổi ở tín hiệu đầu ra như một chức năng của Input Slew và tải đầu ra.
- + Propagation delay có thể từ thấp đến cao (TPLH) và cao đến thấp (PHL).
- + Propagation delay lớn nhất (Clock đến Q) được xem xét để kiểm tra Setup.

- *Contamination delay*:

- + Trường hợp trễ tốt nhất hợp lệ từ input đến output.
- + Là propagation delay nhỏ nhất (clock đến Q).

- *Net delay*

- + Tổng thời gian để sạc/xả tất cả các ký sinh có trong net nhất định.

\* *Định nghĩa clock trong STA*:

- *Synchronous clocks*:

- + 2 clock là đồng bộ w.r.t. lẫn nhau.
- + Các Timing path được khởi chạy bởi một clock và được giữ bởi một clock khác.

- *Asynchronous Clock*:

- + 2 clock không đồng bộ w.r.t. lẫn nhau.
- + Nếu không có quan hệ timing, STA không thể được áp dụng, vì vậy tool sẽ không kiểm tra vấn đề timing.

- *Generated Clock*:

- + Clock được tạo từ nguồn xung dưới dạng bội số của tần số xung nguồn.
- + Tần số có thể là bội số hoặc có thể là số chia cho xung nhịp nguồn.

- *Virtual Clock*:

- + Tồn tại nhưng không được liên kết với bất kỳ chân hoặc cổng nào của thiết kế.

+ Được sử dụng làm tham chiếu trong STA để chỉ định độ trễ đầu vào và tải đầu ra liên quan đến clock (cần thiết để sửa lỗi Vi phạm Input2Reg và Reg2Output).

+ Bằng cách xác định các ràng buộc clock ảo IO có thể được xác định liên quan đến clock ảo này mà không cần thông số kỹ thuật về cổng hoặc chân nguồn.

\* *Slack*:

- Khoảng thời gian khác biệt của required time và arrival time.

- Nếu arrival time sớm hơn required time thì slack là giá trị dương. Nếu arrival time trùng với required time thì slack bằng 0 và ràng buộc timing là vừa đủ thỏa mãn (barely MET). Nếu arrival time trễ hơn required time thì slack âm.

- *Positive Slack*: thể hiện rằng rằng thời gian đến tại nút đó có thể được tăng lên mà không ảnh hưởng đến độ trễ tổng thể của mạch điện

- *Negative Slack*: thể hiện rằng một đường dẫn quá chậm và đường dẫn phải tăng tốc nếu toàn bộ mạch hoạt động ở tốc độ mong muốn

#### 6.3.2.8        **2.8. Physical Verification**

Layout sẽ sẵn sàng sau giai đoạn routing. Một số kiểm tra mà chúng ta phải thực hiện ngay sau khi hoàn thành layout để kiểm tra xem layout có hoạt động như thiết kế hay không. Các kiểm tra này được gọi là kiểm tra Signoff.

- *Xác minh vật lý (Physical Verification)*

- *Phân tích timing (Timing Analysis)*

- *Kiểm tra tương đương logic (Logical Equivalence Checking- LEC)*

#### **Physical Verification**

- *Layout versus Schematic (LVS)*:

+ LVS là một kiểm tra quan trọng trong giai đoạn Physical Verification. Công cụ LVS tạo ra một danh sách layout bằng cách trích xuất các hình học. Danh sách layout netlist này được so sánh với danh sách schematic netlist của cùng một giai đoạn để xác minh xem chúng có khớp về mặt chức năng hay không. Nếu hai danh sách netlist khớp nhau, thì báo cáo về LVS sẽ không có lỗi nào.

- *Design Rule Check (DRC)*

+ Kiểm tra DRC xác định xem layout có đáp ứng các quy tắc nhất định do nhóm chế tạo chỉ định hay không. Một số quy tắc là khoảng cách giữa các lớp kim loại, quy tắc chiều rộng tối thiểu, quy tắc via, v.v. Đầu vào cho tool DRC là một file về design rule.

+ Kiểm tra DRC không có gì khác ngoài kiểm tra vật lý về chiều rộng kim loại, độ cao và khoảng cách yêu cầu đối với các lớp kim loại khác nhau liên quan đến quy trình chế tạo khác nhau. Nếu chúng ta cung cấp kết nối vật lý cho các thành phần mà không xem xét các quy tắc DRC, thì nó sẽ dẫn đến lỗi chức năng của chip. Vì vậy DRC phải không có lỗi trước khi đưa vào chế tạo.

## 6.4 Manufaturing

### 6.4.1 Các bước cơ bản trong Manufacturing

#### 6.4.1.1 Xử lí Wafer (Wafer Processing)

Wafer là một miếng silicon mỏng chừng 30 mil (0.76 mm) được cắt ra từ thanh silicon hình trụ. Thiết bị này được sử dụng với tư cách là vật liệu nền để sản xuất vi mạch tích hợp (người ta “cấy” lên trên đó những vật liệu khác nhau để tạo ra những vi mạch với những đặc tính khác nhau. Xử lý Wafer là bước tinh chế (xử lý hóa học) cát ( $\text{SiO}_2$ ) thành Silic nguyên chất (99.9999%). Silic đã tinh lọc được nung chảy và trở thành thỏi hình trụ. Silic nguyên chất sẽ được pha thêm tạp chất là các nguyên tố nhóm 3 hoặc nhóm 5. Những thỏi silic đó sẽ được cắt thành các tấm tròn đường kính 200mm(8 inch) hoặc 300mm(12 inch) với bề dày cỡ 750um và được đánh bóng cho đến khi chúng có bề mặt hoàn hảo, nhẵn bóng như gương.

#### **Đúc phôi**

Đầu tiên, cát cần được đun nóng để tách carbon monoxide và silicon, và quá trình này được lặp lại cho đến khi thu được silicon ở cấp electron có độ tinh khiết cực cao (EG-Si). Silicon có độ tinh khiết cao tan chảy thành chất lỏng, và sau đó đông đặc thành dạng rắn đơn tinh thể được gọi là "phôi", đây là bước đầu tiên trong sản xuất chất bán dẫn. Độ chính xác chế tạo của phôi silicon (trụ silicon) rất cao, đạt cấp độ nano.

#### **Cắt phôi**

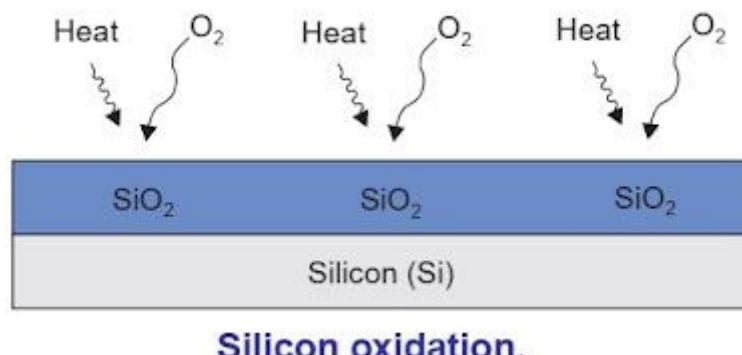
Sau khi hoàn thành bước trước, bạn cần dùng cưa kim cương cắt bỏ cả hai đầu của phôi, sau đó cắt thành từng lát có độ dày nhất định. Đường kính của lát phôi xác định kích thước của tấm wafer. Các tấm wafer lớn hơn và mỏng hơn có thể được chia thành nhiều đơn vị hơn, giúp giảm chi phí sản xuất. Sau khi cắt phôi silicon, cần phải thêm dấu "vùng phẳng (flat area)" hoặc "vết lõm (indent)" trên lát cắt, để thuận tiện cho việc định hướng xử lý dựa vào đó như một tiêu chuẩn trong các bước tiếp theo.

## **Đánh bóng bì mặt Wafer**

Miếng mỏng thu được thông qua quá trình cắt nói trên được gọi là "khuôn (die)", tức là "wafer thô" chưa qua xử lý. Bì mặt khuôn không bằng phẳng, không thể in trực tiếp các họa tiết mạch lên đó. Vì vậy, trước tiên cần phải loại bỏ các khuyết tật bì mặt thông qua quá trình mài và ăn mòn hóa học, sau đó tạo thành bì mặt nhẵn thông qua đánh bóng và sau đó làm sạch các chất bẩn còn sót lại.

### **6.4.1.2 Quá trình oxi hóa (Oxidation)**

- Trong quá trình chế tạo mạch tích hợp người ta thường phải dùng lớp SiO<sub>2</sub> trên bì mặt tinh thể Si.. Lớp SiO<sub>2</sub> này có hệ số dẫn nở nhiệt gần bằng hệ số giãn nở nhiệt của Si, với hằng số điện môi 3.9~ 4, có tác dụng bảo vệ bì mặt các linh kiện bán dẫn dưới tác dụng của môi trường bên ngoài, che chắn bì mặt Si trong quá trình khuếch tán định xử các tạp chất như P và B. Ngoài ra lớp SiO<sub>2</sub> còn được sử dụng làm cực (gate) cửa cho bóng bán dẫn (transistor). Có nhiều phương pháp tạo ra lớp SiO<sub>2</sub> nhưng phương pháp được sử dụng rộng rãi nhất để nhận lớp SiO<sub>2</sub> là phương pháp ôxy hoá ở nhiệt độ cao (khoảng 1000C -1100C).



*Oxidation*

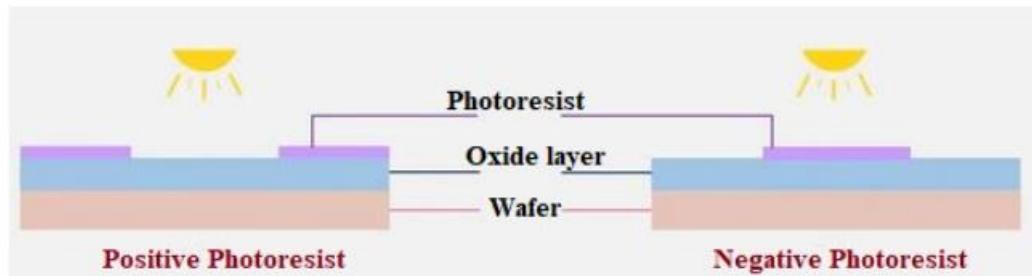
### **6.4.1.3 Photomask**

Photomask là việc sử dụng ánh sáng để "in" các mẫu mạch lên tấm wafer. Chúng ta có thể hiểu nó là những bộ phận bán dẫn vẽ trên bì mặt của tấm wafer. Độ mịn của mẫu mạch càng cao thì khả năng tích hợp của chip sản phẩm càng cao, điều này chỉ có thể đạt được thông qua công nghệ photomask tiên tiến. Cụ thể, nó có thể được chia thành ba bước: phủ lớp cản quang (photoresist coating), phơi sáng (exposure) và phát triển (development).

#### ***Phủ lớp cản quang (Photoresist coating):***

Bước đầu tiên để vẽ một mạch điện trên tấm wafer là phủ chất cản quang lên lớp oxit. Lớp cản quang thay đổi các đặc tính hóa học của tấm wafer để trở thành "tấm giấy ảnh". Lớp cản

quang trên bề mặt tám wafer càng mỏng thì lớp phủ càng đồng đều và các mẫu có thể in được càng mịn.

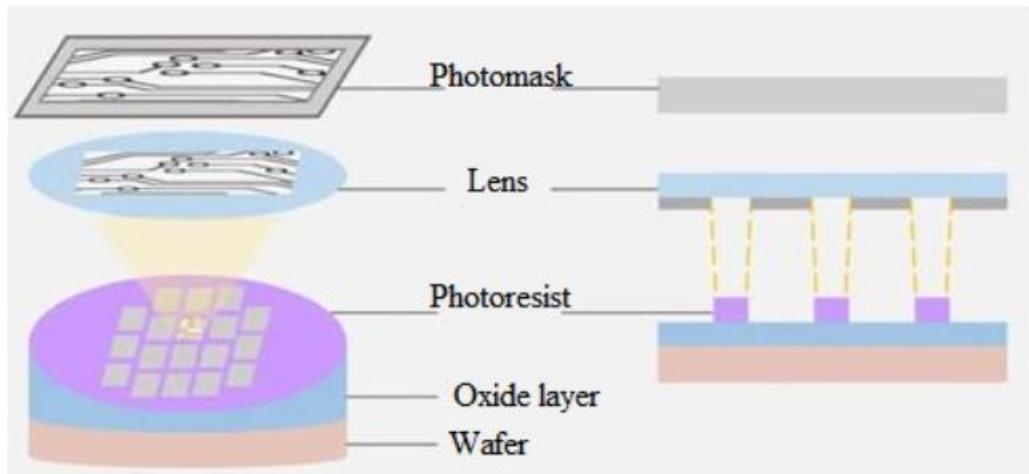


*Coating Photoresist*

Theo sự khác biệt khả năng phản ứng của tia UV, keo cản quang có thể được chia thành hai loại: keo dương và keo âm. Phần trước sẽ phân hủy và biến mất sau khi tiếp xúc với ánh sáng, để lại hình dạng của các vùng không nhận được.

### *Phơi sáng (exposure)*

Sau khi phủ màng cản quang lên tám wafer, mạch có thể được in bằng cách kiểm soát sự chiếu xạ ánh sáng. Quá trình này được gọi là "phơi sáng (exposure)". Chúng ta có thể truyền ánh sáng qua thiết bị phơi sáng một cách có chọn lọc. Khi ánh sáng đi qua mặt nạ có chứa mẫu mạch, mạch có thể được in trên tám wafer được phủ một lớp film cản quang bên dưới.



*Exposure*

### *Phát triển (Development)*

Bước sau khi phơi sáng là rải bộ phận phát quang lên tám wafer, để loại bỏ chất cản quang ở khu vực không bị che phủ bởi mẫu, để mẫu mạch in có thể lộ ra. Sau khi phát triển xong, nó

cần được kiểm tra bằng các thiết bị đo lường khác nhau và kính hiển vi quang học để đảm bảo chất lượng của bản vẽ sơ đồ mạch điện.

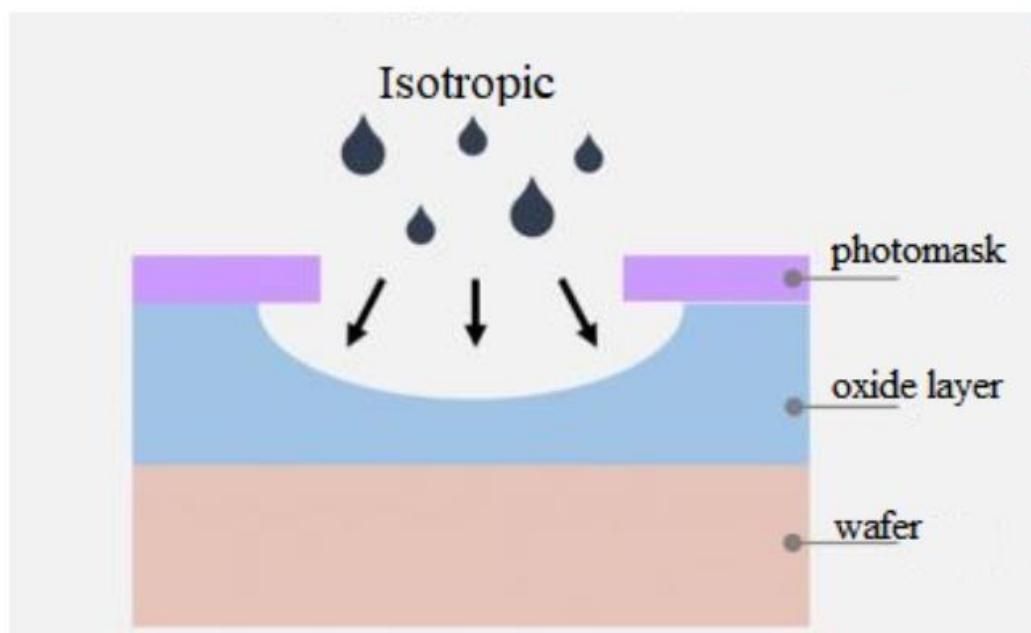
#### 6.4.1.4 Khắc (Etching)

- Khi wafer đã có lớp SiO<sub>2</sub> phủ lên rồi thì bước kế tiếp cũng giống như quá trình làm bo mạch điện (PCB). Sau khi dùng mực cảm quang in lên rồi thì sau đó sẽ đến bước "tẩy" phần không cần tới.

Đây là bước của khâu Etching. Có hai phương pháp ăn mòn chính là : ăn mòn ướt (wet etch) và ăn mòn khô (dry etch).

##### **Khắc ẩm (Wet Etching)**

Khắc ẩm sử dụng dung dịch hóa học để loại bỏ màng oxit có ưu điểm là chi phí thấp, tốc độ ăn mòn nhanh và năng suất cao. Tuy nhiên, ăn mòn theo phương pháp có đặc điểm là đẳng hướng, tức là tốc độ của nó là như nhau theo bất kỳ hướng nào. Điều này sẽ làm cho mặt nạ và màng oxit khắc không hoàn toàn thẳng hàng



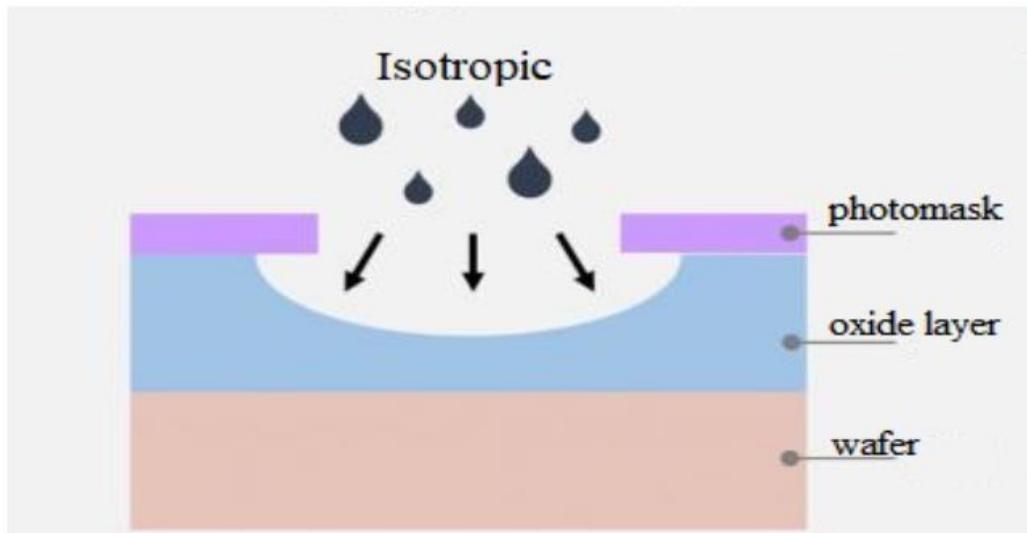
*Wet Etching Method*

##### **Khắc khô (Dry Etching)**

- Khắc khô có thể được chia thành ba loại khác nhau:

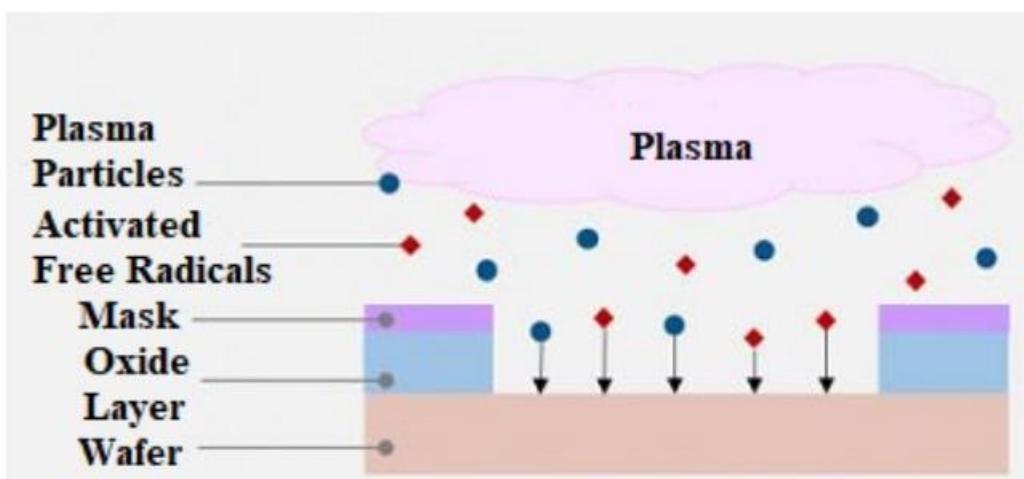
+ Loại thứ nhất là ăn mòn hóa học, sử dụng khí ăn mòn (chủ yếu là HF). Giống như phương pháp khắc ẩm, phương pháp này cũng có tính đẳng hướng

+ Phương pháp thứ hai là phương pháp phún xạ vật lý (physical sputtering), đó là các ion trong plasma được sử dụng để tấn công và loại bỏ các lớp oxit dư thừa. Là một phương pháp ăn mòn dị hướng, nó có tốc độ khắc khác nhau theo hướng ngang và dọc. Tuy nhiên, nhược điểm của phương pháp này là tốc độ ăn mòn chậm, do hoàn toàn dựa vào phản ứng vật lý bởi va chạm ion.



*Physical Sputtering*

+ Phương pháp thứ ba là khắc ion phản ứng (Rie). Nó kết hợp hai phương pháp đầu tiên, đó là trong khi sử dụng plasma để khắc vật lý các ion, và quá trình ăn mòn hóa học được thực hiện với các gốc tự do được tạo ra sau khi kích hoạt plasma. Ngoài tốc độ khắc vượt qua hai phương pháp đầu tiên, RIE có thể sử dụng các đặc điểm của dị hướng ion để đạt được quá trình khắc hoa tiết có độ nét cao.



*Reactive Ion Etching (RIE)*

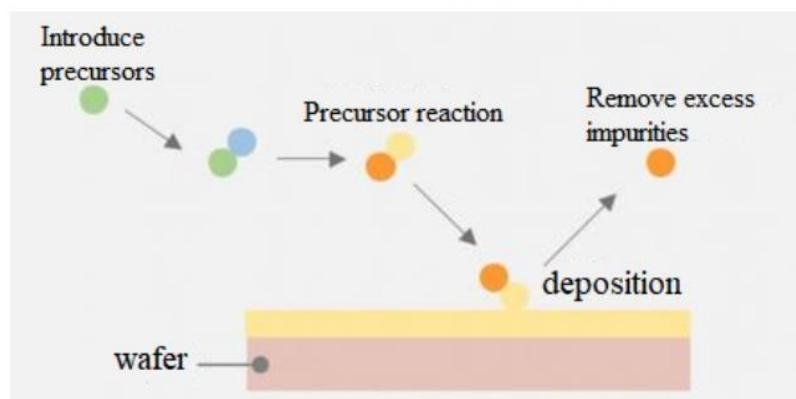
#### 6.4.1.5 Lăng đọng film (Film Deposition)

Để tạo ra các thiết bị siêu nhỏ bên trong chip, chúng ta cần liên tục lăng đọng các lớp màng mỏng và loại bỏ các phần thừa bằng cách ăn mòn, và thêm một số vật liệu để tách các thiết bị khác nhau. Mỗi bóng bán dẫn hoặc memory cell được xây dựng từng bước thông qua quá trình trên. “Màng mỏng” ở đây là “màng” có độ dày nhỏ hơn 1 micro mét và không thể được sản xuất bằng các phương pháp gia công cơ học thông thường. Quá trình đưa một màng mỏng có chứa đơn vị phân tử hoặc nguyên tử mong muốn lên tấm wafer là "deposition".

Để hình thành cấu trúc bán dẫn nhiều lớp, trước tiên chúng ta cần chế tạo một ngăn xếp, tức là xếp xen kẽ nhiều màng mỏng kim loại (dẫn điện) và điện môi (cách điện) lên bề mặt của tấm wafer, sau đó lặp lại quá trình ăn mòn để loại bỏ các phần thừa và tạo thành cấu trúc ba chiều. Các công nghệ có thể được sử dụng trong quá trình lăng đọng bao gồm lăng đọng hơi hóa học (chemical vapor deposition-CVD), lăng đọng lớp nguyên tử (atomic layer deposition-ALD) và lăng đọng hơi vật lý (physical vapor deposition-PVD). Các phương pháp sử dụng các công nghệ này có thể được chia thành lăng đọng khô (dry deposition) và lăng đọng ẩm (wet deposition).

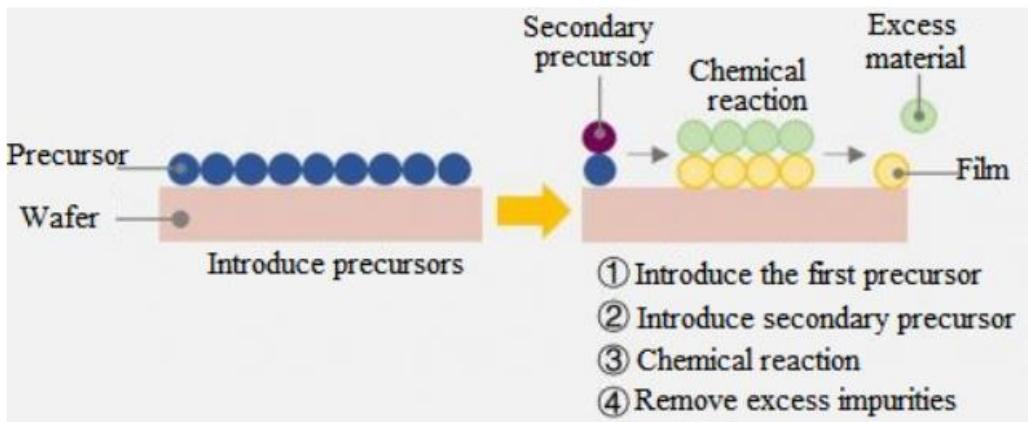
##### CVD

Trong quá trình lăng đọng hơi hóa học, tiền chất khí phản ứng hóa học trong buồng phản ứng và tạo ra một màng mỏng gắn trên bề mặt của tấm wafer và các sản phẩm phụ được hút ra khỏi buồng. Sự lăng đọng hơi hóa chất được tăng cường plasma yêu cầu sử dụng plasma để tạo ra khí phản ứng. Phương pháp này làm giảm nhiệt độ phản ứng và rất thích hợp cho các cấu trúc nhạy cảm với nhiệt độ.



Chemical Vapor Deposition

## **ALD**



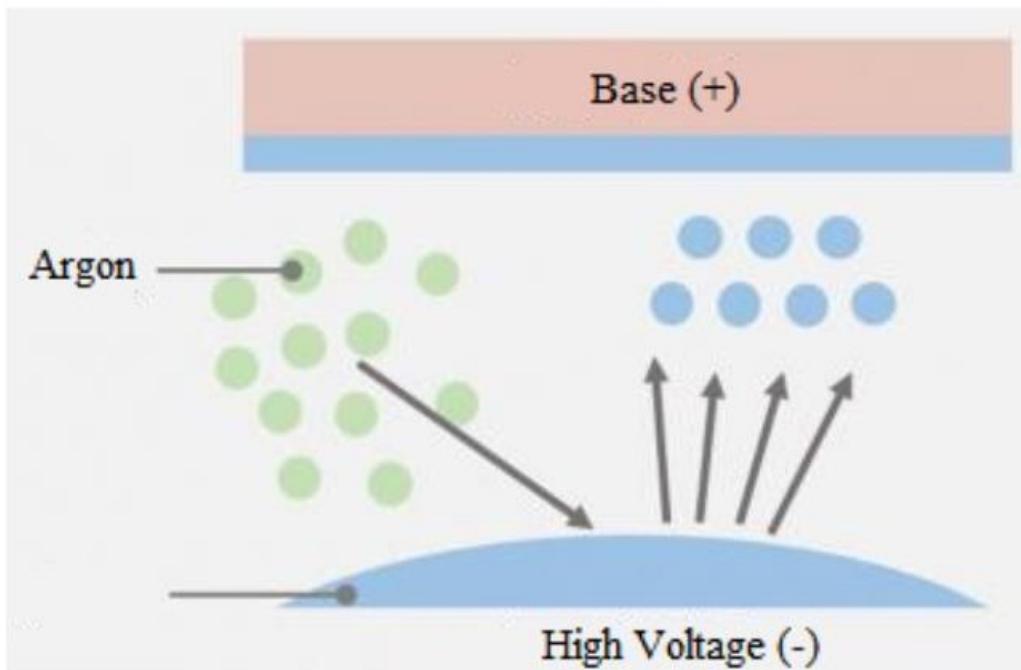
### *Atomic Layer Deposition*

Sự lăng đọng lớp nguyên tử tạo thành một màng mỏng bằng cách chỉ lăng đọng một vài lớp nguyên tử tại một thời điểm. Chìa khóa của phương pháp này là lặp lại các bước độc lập theo một thứ tự nhất định và duy trì sự kiểm soát tốt. Phủ tiền chất lên bề mặt tám wafer là bước đầu tiên, sau đó các khí khác nhau được đưa vào để phản ứng với tiền chất tạo thành các chất cần thiết trên bề mặt tám wafer.

## **PVD**

Sự lăng đọng hơi vật lý để cập đến sự hình thành các màng mỏng bằng các phương tiện vật lý. Phún xạ là một phương pháp lăng đọng hơi vật lý. Nguyên tắc của nó là các nguyên tử của vật liệu đích bị bắn ra do bắn phá plasma của argon và lăng đọng trên bề mặt wafer để tạo

thành một màng mỏng.

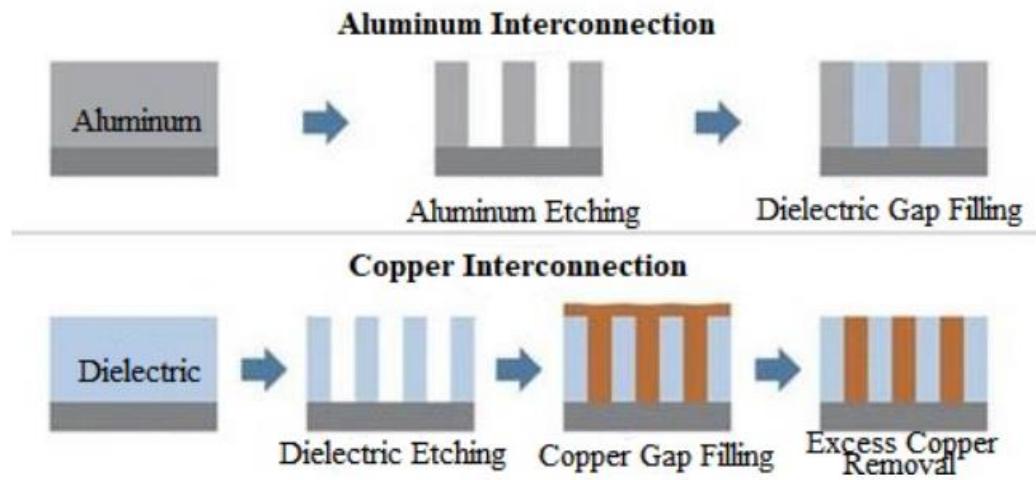


*Physical Vapor Deposition*

#### 6.4.1.6 Kết nối (Interconnection)

Sự dẫn điện của chất bán dẫn là nằm giữa chất dẫn điện và chất không dẫn điện (tức là chất cách điện). Đặc tính này cho phép chúng ta kiểm soát hoàn toàn dòng điện. Thông qua quá trình quang khắc, khắc và lăng đọng dựa trên tám wafer, các bóng bán dẫn và các thành phần khác có thể được tạo ra, nhưng chúng cũng cần được kết nối để đạt được năng lượng và tín hiệu truyền và nhận. Kim loại được sử dụng để nối mạch cần đáp ứng các điều kiện sau:

- + Điện trở thấp: Vì cần cho dòng điện đi qua nên kim loại phải có điện trở thấp.
- + Tính bền nhiệt hóa: Tính chất của vật liệu kim loại phải không thay đổi trong quá trình liên kết kim loại.
- + Độ tin cậy cao: Với sự phát triển của công nghệ mạch tích hợp, ngay cả một lượng nhỏ vật liệu kết nối bằng kim loại cũng phải có đủ độ bền.
- + Chi phí sản xuất: Ngay cả khi đáp ứng được ba điều kiện trước đó, giá thành cao không phù hợp để sản xuất hàng loạt.
- Quá trình liên kết chủ yếu sử dụng hai chất là nhôm (Al) và đồng (Co).



*Qui trình kết nối Al và Co*

\* *Quy trình kết nối nhôm:*

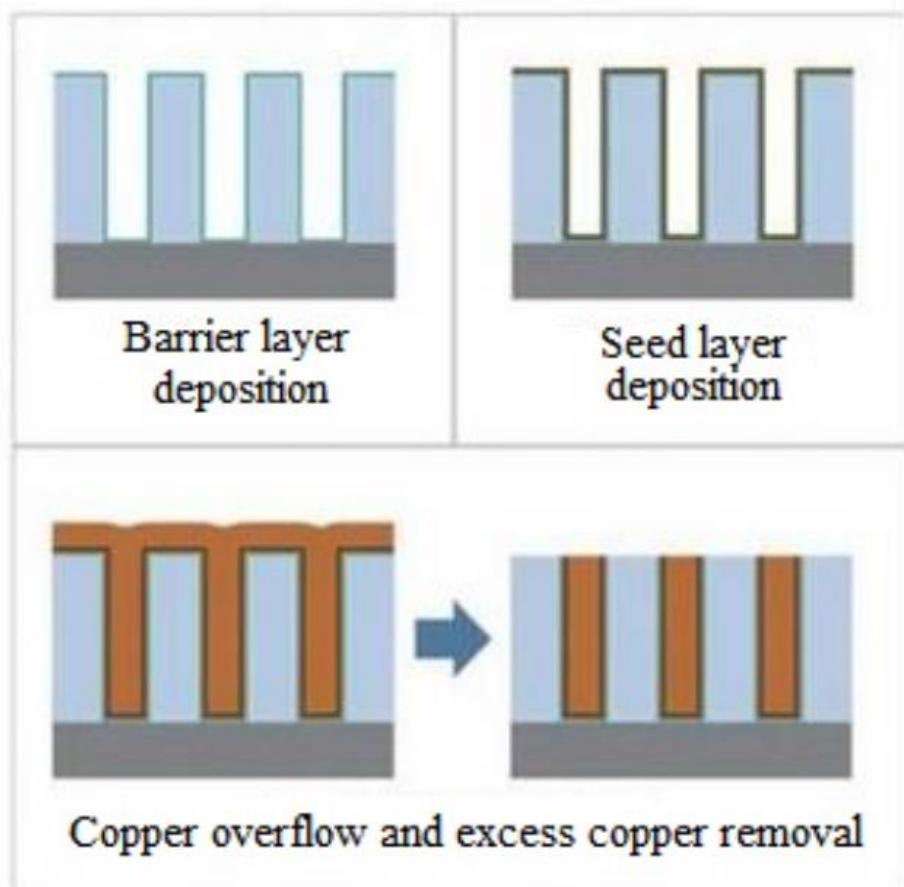
Quá trình này bắt đầu với quá trình lăng đọng nhôm, ứng dụng chất cản quang, phơi sáng và phát triển, loại bỏ bất kỳ nhôm và chất cản quang dư thừa nào trước khi bước vào quá trình ôxy hóa thông qua công nghệ khắc. Sau khi hoàn thành các bước trên, hãy lặp lại chúng cho đến khi hoàn thành kết nối.

Với khả năng dẫn điện tuyệt vời, nhôm cũng dễ quang khắc, khắc và lăng đọng. Ngoài ra, nó có giá thành thấp hơn và khả năng bám dính tốt hơn với màng oxit. Nhược điểm là dễ bị ăn mòn và có nhiệt độ nóng chảy thấp. Ngoài ra, để ngăn phản ứng của nhôm và silicon gây ra sự cố kết nối, người ta cũng cần thêm một chất lăng kim loại để tách nhôm ra khỏi tấm wafer, nó được gọi là "kim loại cản (barrier metal)".

\* *Quy trình kết nối đồng:*

Với việc cải thiện độ chính xác của quy trình bán dẫn và thu nhỏ kích thước thiết bị, tốc độ kết nối và đặc tính điện của mạch nhôm dần không thể đáp ứng được yêu cầu. Do đó với điện trở thấp hơn, đồng có thể đạt được tốc độ kết nối nhanh hơn. Hơn nữa, đồng đáng tin cậy hơn vì nó có khả năng chống chuyển động điện tốt hơn nhôm.

Tuy nhiên, đồng không dễ dàng tạo thành các hợp chất, vì vậy rất khó để bay hơi và loại bỏ nó khỏi bề mặt tấm wafer. Để giải quyết vấn đề này, chúng ta không còn khắc đồng nữa, mà là các vật liệu điện môi, để các mảnh mạch kim loại bao gồm các rãnh và lỗ via thể được hình thành, và sau đó đồng được điền vào như đã nói ở trên để giúp kết nối với nhau, được gọi là "quá trình dát (inlaid process)".



Khi các nguyên tử đồng tiếp tục khuếch tán vào chất điện môi, độ cách điện của chất điện môi sẽ giảm và tạo ra một lớp rào cản ngăn các nguyên tử đồng tiếp tục khuếch tán. Sau đó một lớp hạt đồng rất mỏng sẽ được hình thành trên lớp rào cản. Sau bước này, có thể tiến hành mạ điện. Sau khi lấp đầy, đồng dư thừa có thể được loại bỏ bằng phương pháp đánh bóng cơ hóa kim loại (chemical mechanical polishing- CMP). Sau khi hoàn thành, một lớp màng oxit có thể được lắng đọng và lớp màng thừa có thể được loại bỏ bằng quá trình khắc và quang khắc. Toàn bộ quá trình đầy đủ cần được lặp lại liên tục cho đến khi hoàn thành kết nối đồng.

#### 6.4.1.7 Kiểm tra (Test)

Mục tiêu chính của bước kiểm tra là kiểm tra xem chất lượng của chip bán dẫn có đạt một tiêu chuẩn nhất định hay không, từ đó loại bỏ các sản phẩm lỗi và nâng cao độ tin cậy của chip. Phân loại khuôn điện tử (Electronic die shorting- EDS) là một phương pháp đối với wafer. EDS là một quá trình để kiểm tra các đặc tính điện của mỗi chip ở trạng thái wafer và do đó cải thiện năng suất chất bán dẫn. EDS được chia thành năm bước:

## **EPM**

EPM là bước đầu tiên trong quá trình thử nghiệm chip bán dẫn. Bước này sẽ kiểm tra mọi thiết bị (bao gồm bóng bán dẫn, tụ điện và điốt) mà mạch tích hợp bán dẫn cần sử dụng để đảm bảo rằng các thông số điện của nó đáp ứng tiêu chuẩn.

### **Kiểm tra tuổi thọ của Wafer (Wafer Aging Test)**

Tỷ lệ khuyết tật bán dẫn xuất phát từ hai khía cạnh, đó là tỷ lệ lỗi sản xuất (cao hơn trong giai đoạn đầu) và tỷ lệ lỗi xảy ra trong suốt vòng đời sau đó. Kiểm tra tuổi thọ Wafer để cập đến việc kiểm tra wafer dưới nhiệt độ nhất định và điện áp AC/DC để tìm ra sản phẩm nào có thể có khuyết tật trong giai đoạn đầu, nghĩa là để cải thiện độ tin cậy của sản phẩm cuối cùng bằng cách phát hiện ra các khuyết tật tiềm ẩn.

### **Kiểm tra các thông số (Parameters Test)**

#### *\* Temp test:*

- High Temperature: Xác minh chip có thể hoạt động ở nhiệt độ lớn khi mà tăng nhiệt độ hơn 10% và cao hơn nữa.
- Low Temperature: Xác minh chip có thể hoạt động ở nhiệt độ thấp khi mà giảm nhiệt độ hơn 10% và thấp hơn nữa.
- Room Temperature: Kiểm tra chip có thể hoạt động ở nhiệt độ phòng (25 độ C).

#### *\* Speed test:*

- Core: Kiểm tra các chức năng của Core có hợp lệ hay không.
- Speed: Đánh giá tốc độ truyền dữ liệu.

#### *\* Motion Test:*

- DC: Đặt vào dòng điện 1 chiều để kiểm tra dòng và điện áp có hoạt động bình thường.
- AC: Đặt vào dòng điện xoay chiều để kiểm tra đặc tính truyền dữ liệu.
- Function: Kiểm tra tất cả các chức năng có hoạt động bình thường.

### **Sửa chữa (Repair):**

Sửa chữa là bước kiểm tra quan trọng nhất, vì một số chip bị lỗi có thể được sửa chữa và bạn chỉ cần thay thế các thành phần bị lỗi.

### **Mực (Ink)**

Trước đây, cần đánh dấu các con chip bị lỗi bằng loại mực đặc biệt để đảm bảo rằng chúng có thể được xác định bằng mắt thường. Ngày nay, hệ thống tự động sắp xếp chúng dựa trên các giá trị dữ liệu thử nghiệm.

#### 6.4.1.8 Đóng gói (Package)

Các chip vuông (còn được gọi là single wafers) có kích thước bằng nhau được hình thành trên các tấm xốp được xử lý bởi một số quy trình trước đó. Việc tiếp theo cần làm là lấy các phần riêng lẻ bằng cách cắt. Con chip vừa bị cắt rất mỏng manh, không trao đổi được tín hiệu điện nên cần phải xử lý riêng. Quá trình này là đóng gói, bao gồm hình thành lớp vỏ bảo vệ bên ngoài chip bán dẫn và cho phép chúng trao đổi tín hiệu điện với bên ngoài. Toàn bộ quy trình đóng gói được chia thành năm bước, cụ thể là cắt tấm wafer, gắn tấm wafer đơn lẻ, kết nối với nhau, đúc và kiểm tra bao bì.

##### *Cắt wafer (Wafer Sawing)*

Để nhiều con chip được sắp xếp dày đặc từ wafer, trước tiên chúng ta phải mài mặt sau của wafer cho đến khi độ dày của nó có thể đáp ứng nhu cầu của quá trình đóng gói. Sau khi mài, chúng ta có thể cắt dọc theo đường vẽ nguệch ngoạc trên tấm wafer cho đến khi tách được chip bán dẫn.

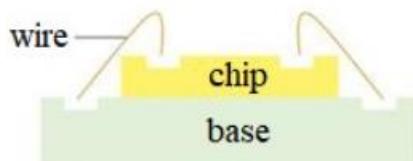
Có ba loại kỹ thuật cắt wafer: cắt bằng lưỡi, cắt bằng laser và cắt plasma. Cắt bằng lưỡi đề cập đến việc cắt các tấm wafer bằng các lưỡi kim cương, dễ sinh ra nhiệt ma sát và các mảnh vụn và do đó làm hỏng các tấm wafer. Cắt bằng tia laser có độ chính xác cao hơn và có thể dễ dàng xử lý các tấm wafer có độ dày mỏng hoặc đường nét vẽ nhỏ. Cắt plasma sử dụng nguyên lý khắc plasma, vì vậy ngay cả khi bước nét vẽ rất nhỏ, công nghệ này cũng có thể được áp dụng.

##### *Đính kèm Wafer đơn lẻ (Single Wafer Attachment)*

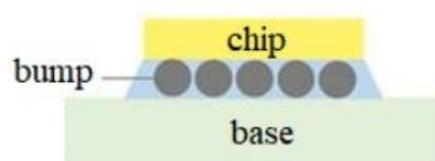
Sau khi tất cả các chip được tách ra khỏi wafer, chúng ta cần gắn các chip riêng lẻ (chip đơn) vào đế (lead frame). Vai trò của đế là bảo vệ các chip bán dẫn và cho phép chúng trao đổi tín hiệu điện với mạch bên ngoài.

##### *Bond*

Sau khi gắn chip vào đế, chúng ta cũng cần kết nối các điểm tiếp xúc của cả hai đế đạt được sự trao đổi tín hiệu điện. Có hai phương pháp kết nối có thể được sử dụng trong bước này: liên kết dây bằng dây kim loại mỏng (metal wire) và liên kết chip lật (flip chip bonding) bằng vàng hoặc khói thiếc hình cầu.



**Wire Bonding**

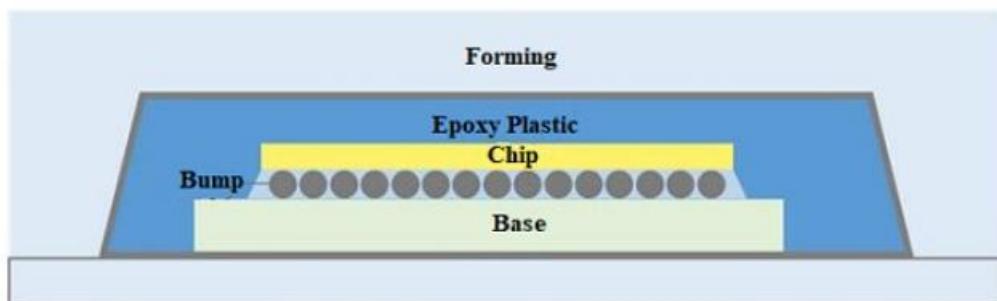


**Flip Chip Bonding**

### *Bonding*

#### **Đúc (Molding)**

Sau khi hoàn thành việc kết nối chip bán dẫn, người ta sử dụng quy trình đúc thêm một gói vào bên ngoài chip để bảo vệ mạch tích hợp bán dẫn khỏi các điều kiện bên ngoài như nhiệt độ và độ ẩm. Sau khi khuôn đóng gói được tạo ra theo yêu cầu, chúng ta đưa chip bán dẫn và hợp chất đúc epoxy (EMC) vào khuôn và hàn kín. Con chip được niêm phong nằm trong sản phẩm cuối cùng của nó.



### *Molding*

#### **Kiểm tra đóng gói (Package Test)**

Chip có dạng cuối cùng phải vượt qua bài kiểm tra các khuyết tật cuối cùng. Tất cả những gì bước vào bài kiểm tra cuối cùng là chip bán dẫn đã hoàn thiện. Chúng sẽ được đưa vào thiết bị thử nghiệm, thiết lập các điều kiện khác nhau như điện áp, nhiệt độ và độ ẩm... để kiểm tra dẫn điện, chức năng và tốc độ. Kết quả của các thử nghiệm này có thể được sử dụng để tìm ra các khuyết tật, nâng cao chất lượng sản phẩm và hiệu quả sản xuất.