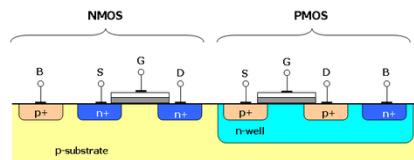
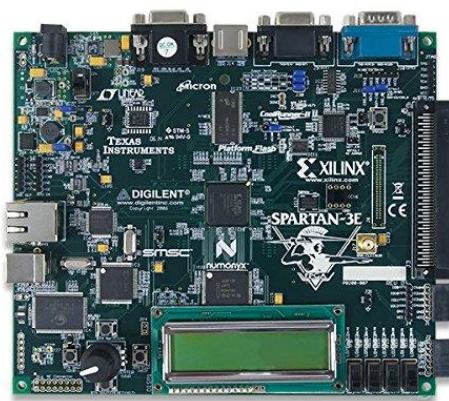


THỰC HÀNH THIẾT KẾ HỆ THÔNG SỐ VÀ VI MẠCH TÍCH HỢP



Trương Ngọc Sơn

Bộ Môn Kỹ thuật Máy tính – Viễn Thông
Đại học Sư phạm Kỹ thuật TP.HCM

HCMUTE - 2019

LỜI NÓI ĐẦU

Tài liệu thực hành môn học Thiết kế hệ thống số và vi mạch tích hợp được biên soạn phục vụ cho môn học thực hành Thiết kế hệ thống số và vi mạch tích hợp, ngành Công nghệ Kỹ thuật Máy tính, Công nghệ Kỹ thuật Điện tử Viễn Thông, Hệ thống nhúng và IoT. Thời lượng thực hành là 45 tiết. Nội dung môn học bao gồm 2 phần chính. Phần 1: Thiết kế hệ thống số bao gồm các bài thực hành về thiết kế các mạch tích hợp, mạch tuần tự đồng bộ, giao tiếp và điều khiển các thiết bị ngoại vi cơ bản sử dụng ngôn ngữ Verilog, mô phỏng trên phần mềm ISim và thực thi trên hệ thống FPGA Xilinx Spartan-3E. Phần 2: Thiết kế mạch thích hợp số bao gồm các bài thực hành về thiết kế vi mạch sử dụng CMOS công nghệ 0.13μ của Samsung, mô phỏng, phân tích các thông số của mạch tích hợp. Thiết kế mạch tích hợp, phân tích các thông số mạch tích hợp được thực hiện trên phần mềm thiết kế Candence Spectre và công nghệ CMOS $0.13\mu\text{m}$ của Samsung.

MỤC LỤC

Contents

PHẦN I: THIẾT KẾ HỆ THỐNG SỐ	5
CHƯƠNG 1. GIỚI THIỆU	5
1. Quy trình thiết kế hệ thống số với FPGA	5
2. Sử dụng phần mềm thiết kế ISE	6
2.1 Cài đặt và kích hoạt bản quyền	6
2.2 Thiết kế mạch cộng 4 bit.....	7
CHƯƠNG 2. THIẾT KẾ MẠCH TỔ HỢP	31
2.1. Thiết kế mạch giải mã	31
2.2. Thiết kế mạch mã hóa 4 đường sang 2 đường	32
2.3. Thiết kế mạch đa hợp 4 đường sang 1 đường.....	32
2.4. Thiết kế mạch giải đa hợp 1 đường sang 8 đường	33
CHƯƠNG 3. THIẾT KẾ MẠCH TUẦN TỰ ĐỒNG BỘ	34
3.1. Giới thiệu.....	34
3.2. Mạch đếm (Counter)	34
3.3. Thiết kế mạch chia xung, sử dụng mạch đếm lên	36
3.3.1. Thiết kế mạch chia xung với ngõ vào 50Mhz, 4 xung ngõ ra với tần số f, 2f, 4f, 8f, trong đó lựa chọn f ~ 1Hz.....	37
3.3.2. Thiết kế mạch tạo xung 1Hz	38
3.3.3. Thiết kế mạch tạo 4 xung ngõ ra với tần số lần lượt là 0.1Hz, 1Hz, 10Hz, 100Hz	39
3.3.4. Thiết kế mạch đếm đồng bộ, sử dụng phương pháp cài đặt các Flip – Flop. Xung đếm 1Hz được lấy từ mạch chia xung.....	40
3.3.5. Thiết kế mạch đếm lên 4 bit như bafi 3.3.4, sử dụng phương pháp thiết kế đồng bộ.....	40
3.3.6. Thiết kế mạch đếm lên 8 bit, lựa chọn tần số đếm, lựa chọn đếm lên hoặc đếm xuống	40
3.3.7. Thiết kế mạch đếm lên 8 bit, lựa chọn 8 tần số đếm khác nhau, lựa chọn đếm lên hoặc đếm xuống, có tín hiệu cho phép dừng đếm (Pause), có tín hiệu đảo trạng thái ngõ ra.....	43
3.4. Thanh ghi dịch (shift register).....	43
3.4.1. Thiết kế thanh ghi dịch 4 bit vào nối tiếp ra nối tiếp như hình 3. Sử dụng cài đặt các module FF-D.....	43
3.4.2. Thiết kế thanh ghi dịch vào nối tiếp ra nối tiếp. Sử dụng phương pháp thiết kế đồng bộ.....	43

3.4.3. Thiết kế mạch ghi dịch vào nối tiếp ra song song bằng cách cài đặt các Flip flop D	45
3.4.4. Thiết kế mạch ghi dịch vào nối tiếp ra song song bằng phương pháp thiết kế đồng bộ	45
3.4.5. Thiết kế mạch điều khiển LED sáng dần từ trái qua phải, tắt dần từ trái qua phải	46
3.4.6. Thiết kế mạch điều khiển LED sáng dần, tắt dần từ trái sang phải hoặc từ phải sang trái được lựa chọn bởi một switch	46
3.4.7. Thiết kế mạch điều khiển 1 led chạy từ trái sang phải, từ phải sang trái	47
3.4.8. Thiết kế mạch điều khiển 1 Led chạy từ trái sang phải rồi tự động chạy từ phải sang trái, có một switch cho phép đảo trạng thái ngược ra	47
3.4.9. Thiết kế mạch gồm 8 led đơn, 4 switch S1, S2, S3, S4	47
3.5. Máy trạng thái (Finite state machine)	48
3.5.1. Thiết kế mô hình máy trạng thái 1	48
3.5.2. Thiết kế mô hình máy trạng thái 2	49
3.5.3. Chống dội phím nhấn (debouncing circuit).....	50
3.6. Công tắc xoay (Rotary switch)	54
3.6.1. Thiết kế mạch đếm lên, đếm xuống được điều khiển bởi công tắc xoay, tần số đếm 1hz	55
3.6.2. Thiết kế mạch đếm lên, đếm xuống, được điều khiển bởi 1 nút nhấn, tần số đếm tăng hay giảm được điều khiển bởi công tắc xoay	56
3.7. LCD	56
3.7.1. Giới thiệu	56
3.7.2. Điều khiển LCD hiển thị chuỗi ký tự trên 2 hàng.....	61
3.7.3. Điều khiển LCD hiển thị chuỗi và số.....	65
3.7.4. Thiết kế mạch đếm và hiển thị giá trị đếm lên LCD.....	69
3.7.5. Thiết kế mạch điều khiển LCD, hiển thị giá trị giờ, phút, giây trên hàng thứ 2	
71	
3.7.6. Thiết kế mạch điều khiển đèn giao thông	71
3.7.7. Thiết kế mạch điều khiển đèn giao thông, có bộ đếm thời gian, đếm xuống ..	73
3.7.8. Thiết kế mạch điều khiển đèn giao thông, có bộ đếm thời gian, đếm xuống hiển thị trên LCD	74
PHẦN II. THIẾT KẾ MẠCH TÍCH HỢP SỐ	75
1. Giới thiệu	75
2. Thiết kế mạch cổng đảo (inverter) sử dụng CMOS công nghệ Samsung 0.13μm	75
2.1. Thiết kế và phân tích đặc tính cổng đảo	75
2.2. Ảnh hưởng các thông số CMOS đến điểm làm việc của cổng đảo	89

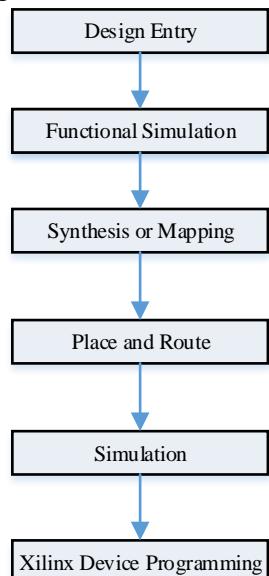
3. Thiết kế mạch cổng NAND sử dụng CMOS công nghệ Samsung 0.13μm.....	90
3.1. Giải thích hoạt động của cổng NAND sử dụng CMOS dựa trên bảng trạng thái	90
3.2. Thiết kế mạch cổng NAND sử dụng CMOS.....	90
4. Thiết kế mạch cổng NOR sử dụng CMOS công nghệ Samsung 0.13μm	90
4.1. Giải thích hoạt động của cổng NOR sử dụng CMOS.....	90
4.2. Thiết kế mạch cổng NOR sử dụng CMOS.....	91
5. Thiết kế mạch FLIP-FLIP sử dụng CMOS công nghệ Samsung 0.13μm.....	91
5.1. Giải thích hoạt động của mạch Flip-Flop D	91
5.2. Thiết kế mạch Flip-Flop sử dụng CMOS	92

PHẦN I: THIẾT KẾ HỆ THỐNG SỐ

CHƯƠNG 1. GIỚI THIỆU

1. Quy trình thiết kế hệ thống số với FPGA

Quy trình thiết kế một hệ thống số sử dụng vi mạch FPGA bao gồm các bước được mô tả trong hình 1.1. Các bước được thực hiện bởi phần mềm hỗ trợ thiết kế. Các họ vi mạch FPGA khác nhau sử dụng các phần mềm hỗ trợ thiết kế khác nhau. Ví dụ các vi mạch FPGA của Xilinx sử dụng phần mềm Xilinx ISE Design Suite hoặc Vavido trong khi đó các vi mạch FPGA của Altera sử dụng phần mềm hỗ trợ thiết kế là Quatus. Các phần mềm thiết kế tích hợp các phương pháp và công nghệ để hỗ trợ thiết kế cho các vi mạch FPGA của từng nhà sản xuất. Các FPGA có cấu trúc khác nhau, tuy nhiên, việc thiết kế bằng ngôn ngữ mô tả phân cứng (Verilog, VHDL) thì hầu như giống nhau. Trong tài liệu này, sinh viên được hướng dẫn thực hành thiết kế hệ thống số trên vi mạch FPGA của Xilinx, sử dụng phần mềm Xilinx ISE Design Suite. Quy trình thiết kế một hệ thống số với FPGA được tóm tắt trong hình 1.1



Hình 1. Quy trình thiết kế FPGA

Ý nghĩa các bước trong quy trình thiết kế

Desgin entry	
Functional simulation	
Synthesis or Mapping	
Place and Route	
Simulation (Static timing analysis,	

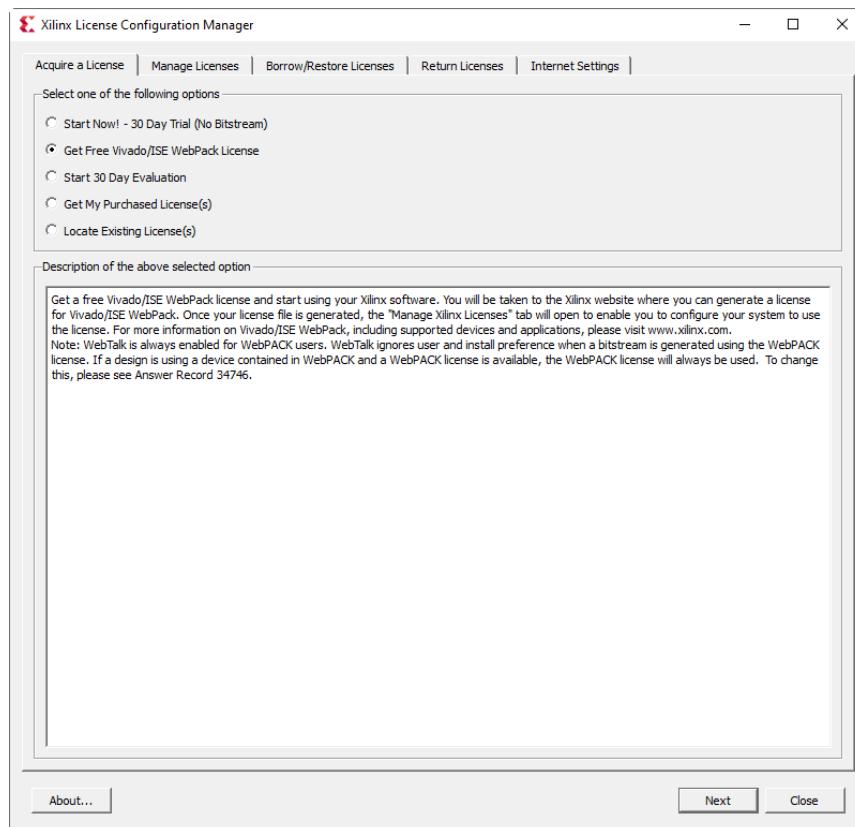
Timing simulation, Power estimation)	
Xilinx Device Programing	

2. Sử dụng phần mềm thiết kế ISE

Phần mềm thiết kế ISE Design Suite hỗ trợ thiết kế cho các vi mạch FPGA của nhà sản xuất Xilinx. Phần mềm hỗ trợ các bước trong quy trình thiết kế FPGA. Trong tài liệu này sẽ giới thiệu các bước sử dụng phần mềm cho phép thực hiện các bước trong quy trình thiết kế từ nhập thiết kế, mô phỏng thiết kế, tổng hợp thiết kế cho đến lập trình vi mạch FPGA của Xilinx

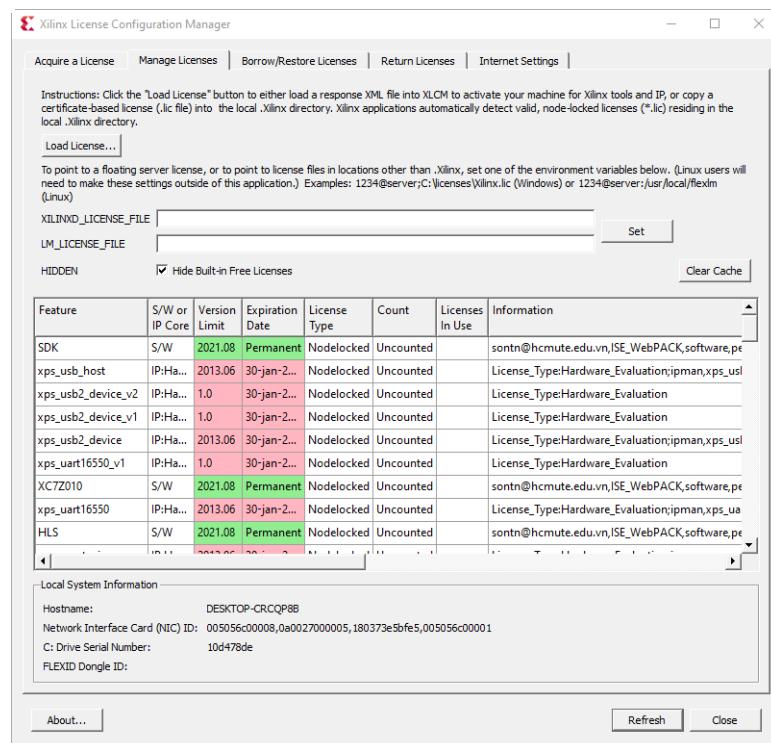
2.1 Cài đặt và kích hoạt bản quyền

Sau khi cài đặt phần mềm ISE Design Suite, trình quản lý bản quyền tự động được kích hoạt, yêu cầu người dùng lựa chọn các hình thức kích hoạt bản quyền như sau



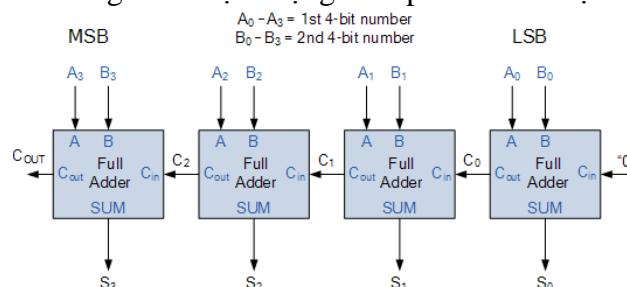
Nếu đã có bản quyền khi mua phần mềm, chúng ta có thể lựa chọn “Get My Purchased Licence”. Trong trường hợp chúng ta không mua bản quyền, Xilinx hỗ trợ bản quyền miễn phí, trong đó đã kích hoạt một số tính năng cơ bản cho phép thực hiện các thiết kế trên vi mạch FPGA. Để sử dụng phần mềm với bản quyền miễn phí, lựa chọn “Get Free Vavido/ISE Webpack License.” Sau khi chọn Next, phần mềm sẽ liên kết đến trang chủ của Xilinx. Chúng ta tiến hành tạo một tài khoản và lựa chọn bản quyền phần mềm cần tải để tải về máy. Bản quyền được cung cấp miễn phí dưới dạng tập tin “Xilinx.lic”. Sau khi tải bản Xilinx.lic về máy, chọn Load licence trong tab Manage Licence để tải bản quyền vào trong phần mềm. lúc này chúng ta sẽ thấy một số chức năng trong phần mềm đã được cấp bản quyền. Một chú ý là khi tải bản quyền miễn phí (Xilinx.lic), phần mềm tự động lấy địa chỉ và các thông số của máy tính (Các thông số của card mạng), cho nên khi

tải bản quyền bằng máy tính nào thì chỉ có thể dùng bản quyền để kích hoạt phần mềm trên chính máy tính đó mà thôi.

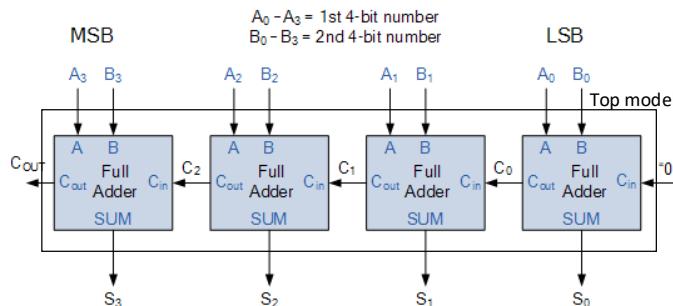


2.2 Thiết kế mạch cộng 4 bit

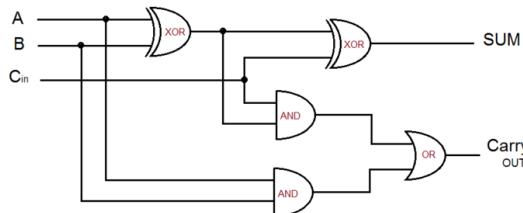
Mạch cộng 4 bit được thiết kế theo mô hình cấu trúc hoặc mô hình hành vi. Trong ví dụ này, chúng ta đi thiết kế mô hình mạch cộng 4 bit sử dụng mô hình cấu trúc (structural model). Mô hình cấu trúc của mạch cộng 4 bit bao gồm 4 mạch toàn phần 1 bit được kết nối như sau



Trong mô hình cấu trúc, mạch cộng 4 bit có 4 ngõ vào A₀-A₃, 4 ngõ vào B₀-B₃, một ngõ vào giá trị cờ nháy, một ngõ ra giá trị cờ nháy, 4 ngõ ra giá trị tổng. Mô đun này được gọi là mô đun chính (Top module).



Bên trong mô đun chính là các mô đun mạch cộng toàn phần 1 bit được thể hiện như hình bên dưới

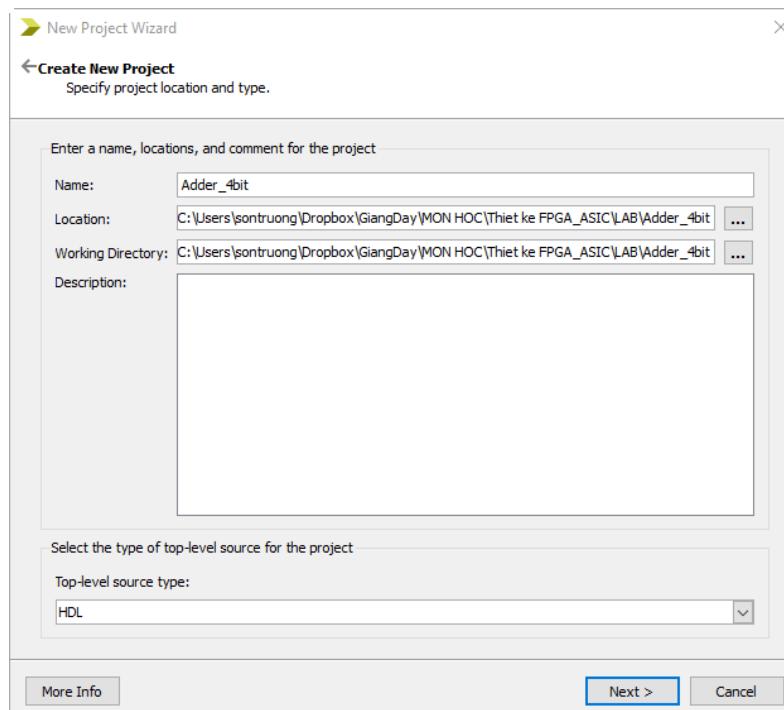


Thiết kế mạch cộng 4 bit được tiến hành với các bước như sau:

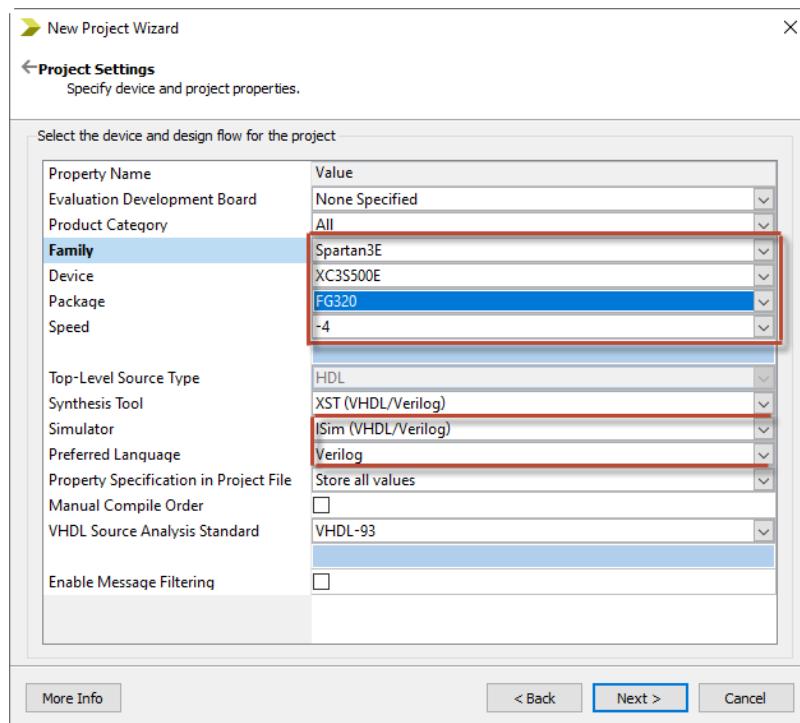
- Thiết kế mô đun chính (top module)
- Thiết mô đun mạch cộng 1 bit
- Thiết kế mô đun tạo tín hiệu mô phỏng
- Thực hiện gán tín hiệu ra các I/O
- Lập trình thiết bị FPGA

➤ Tạo Project mới cho thiết kế

Chọn File – New Project, đặt tên mô đun top là Adder_4bits

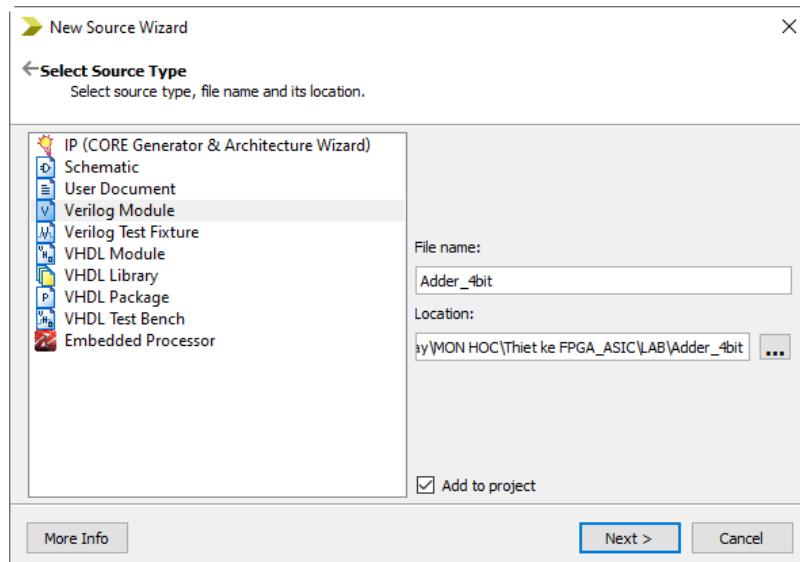


Sau khi chọn Next, xác định các thông số cho mô đun. Chú ý lựa chọn các thông số chính xác cho phần cứng, ngôn ngữ sử dụng và trình mô phỏng như sau

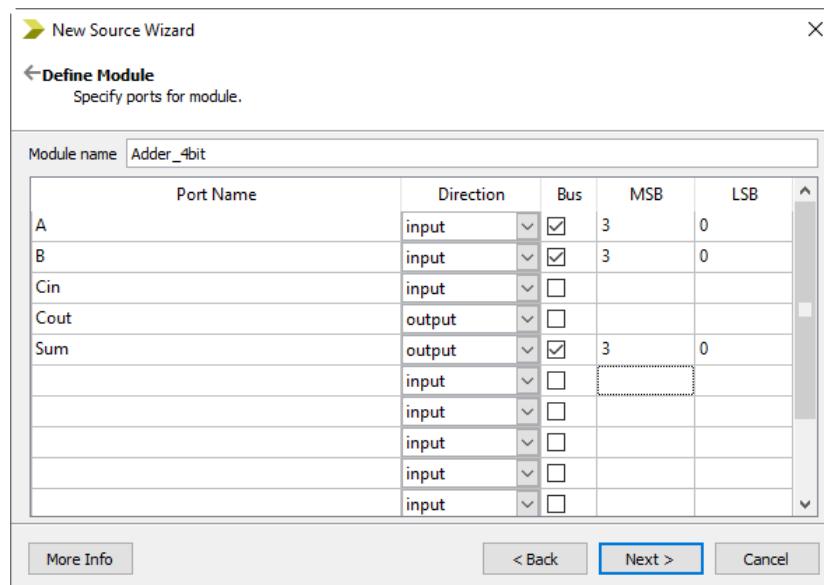


- Thiết kế mô đun chính cho mạch cộng 4 bit.

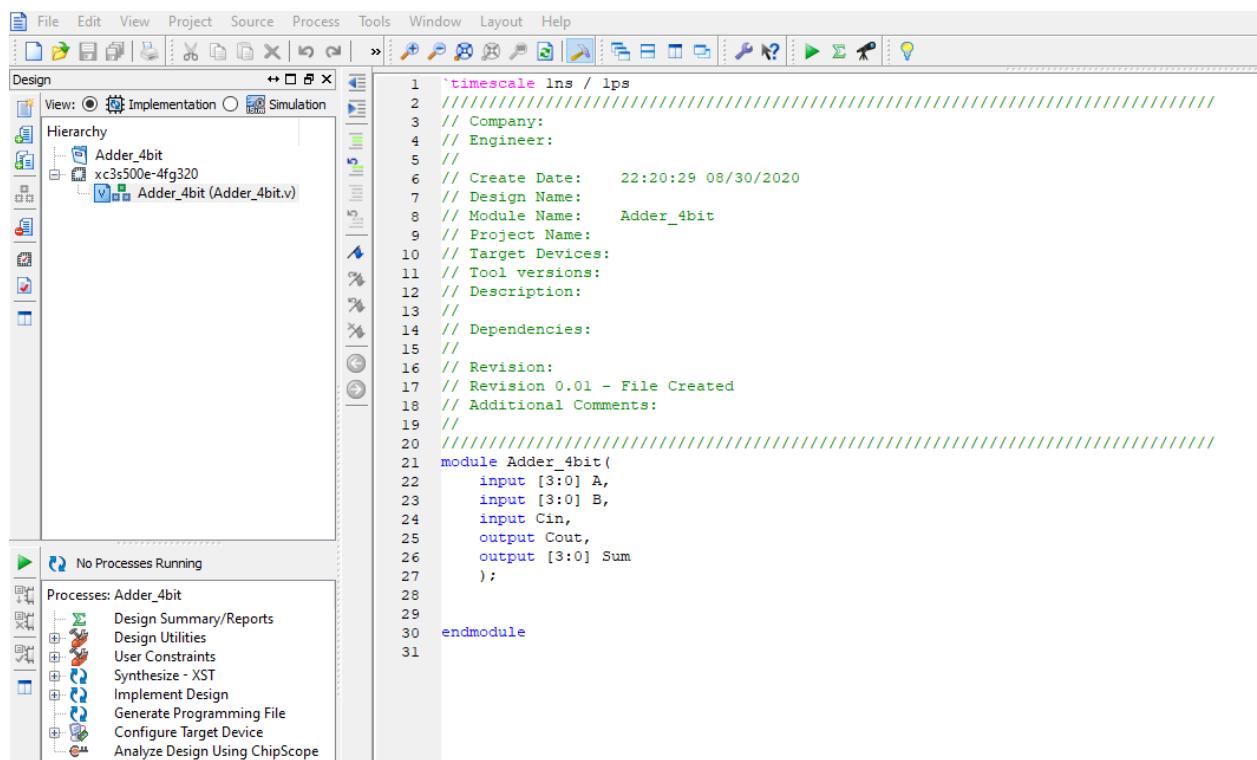
Từ Menu Project, chọn New Source, chọn Verilog Module, đặt tên mô đun như hình bên dưới



Chọn Next, bước tiếp theo cho phép xác định các tín hiệu vào ra của mô đun, chúng ta có thể khai báo các tín hiệu hoặc có thể thực hiện sau đó. Ví dụ, khai báo các tín hiệu cho mạch cộng 4 bit như sau



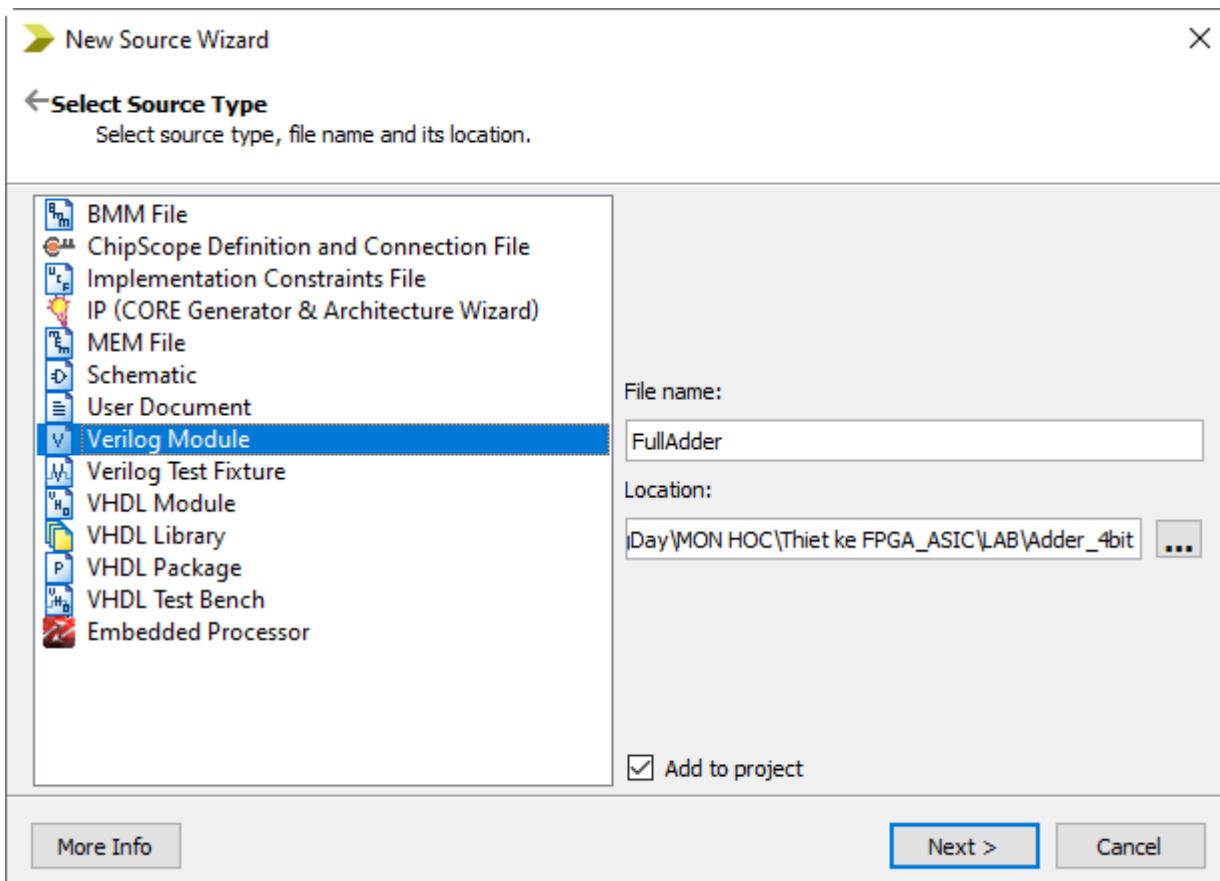
Kết quả chương trình tạo ra tập tin Adder_bit.v, cho phép thực hiện nhập thiết kế cho mạch đếm 4 bit như sau



Trước khi cài đặt cho mạch cộng 4 bit, chúng ta thiết kế mạch cộng toàn phần 1 bit với cấu trúc đã được đề cập trước đó. Quá trình thiết kế mạch cộng 1 bit được thực hiện tương tự.

➤ Thiết kế mô đun mạch cộng 1 bit

Từ menu Project, chọn New Source, chọn Verilog Module, đặt tên mô đun là FullAdder



Bỏ qua phần xác định các port cho mô đun, chúng ta được tập tin FullAdder.v như sau

```

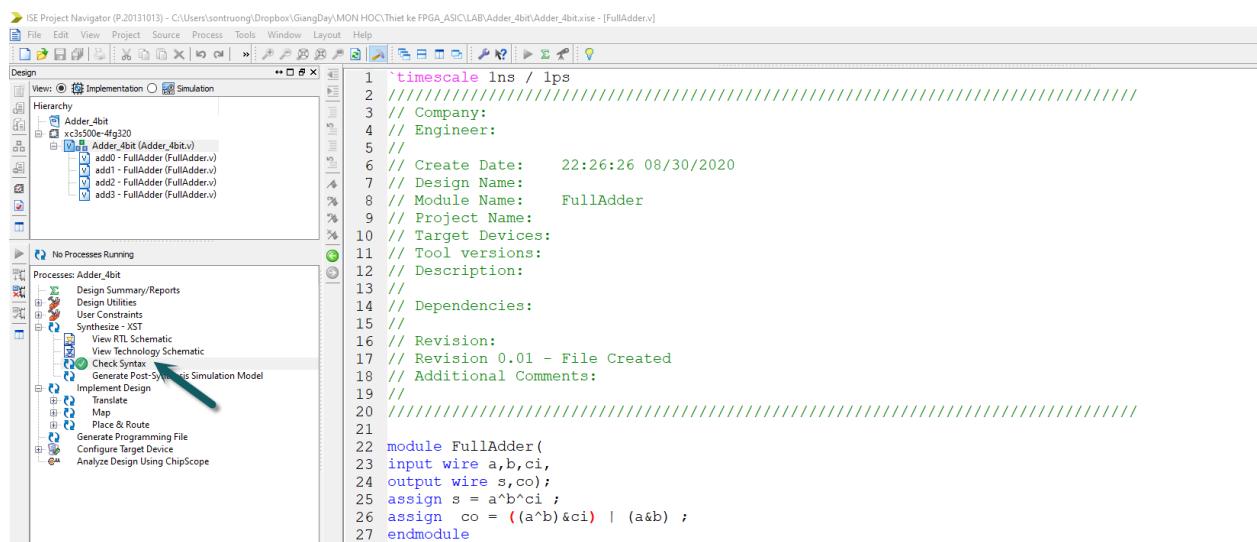
File Edit View Project Source Process Tools Window Layout Help
Design View: Implementation Simulation
Hierarchy
Adder_4bit
  xc3s500e-4fg320
    Adder_4bit (Adder_4bit.v)
    FullAdder (FullAdder.v)
Processes: Adder_4bit
  Design Summary/Reports
  Design Utilities
  User Constraints
  Synthesize - XST
  Implement Design
  Generate Programming File
  Configure Target Device
  Analyze Design Using ChipScope
1 `timescale 1ns / 1ps
2 // Company:
3 // Engineer:
4 //
5 // Create Date: 22:26:26 08/30/2020
6 // Design Name:
7 // Module Name: FullAdder
8 // Project Name:
9 // Target Devices:
10 // Tool versions:
11 // Description:
12 //
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 module FullAdder(
21   );
22
23
24
25 endmodule
26

```

Mô tả mạch cộng toàn phần sử dụng mô hình cấu trúc như sau:

```
module FullAdder(
    input wire a,b,ci,
    output wire s,co);
    assign s = a^b^ci ;
    assign co = ((a^b)&ci) | (a&b) ;
endmodule
```

Kiểm tra lỗi thiết kế: Chọn chức năng Check Syntax bên cửa sổ Design



➤ Cài đặt cho mô đun chính

Quay lại mô đun chính (Adder_bit), thực hiện cài đặt cho mô đun chính như sau

```
module Adder_4bit(
    input  wire [3:0] A,
    input  wire [3:0] B,
    input  wire Cin,
    output wire Cout,
    output wire [3:0] Sum
);

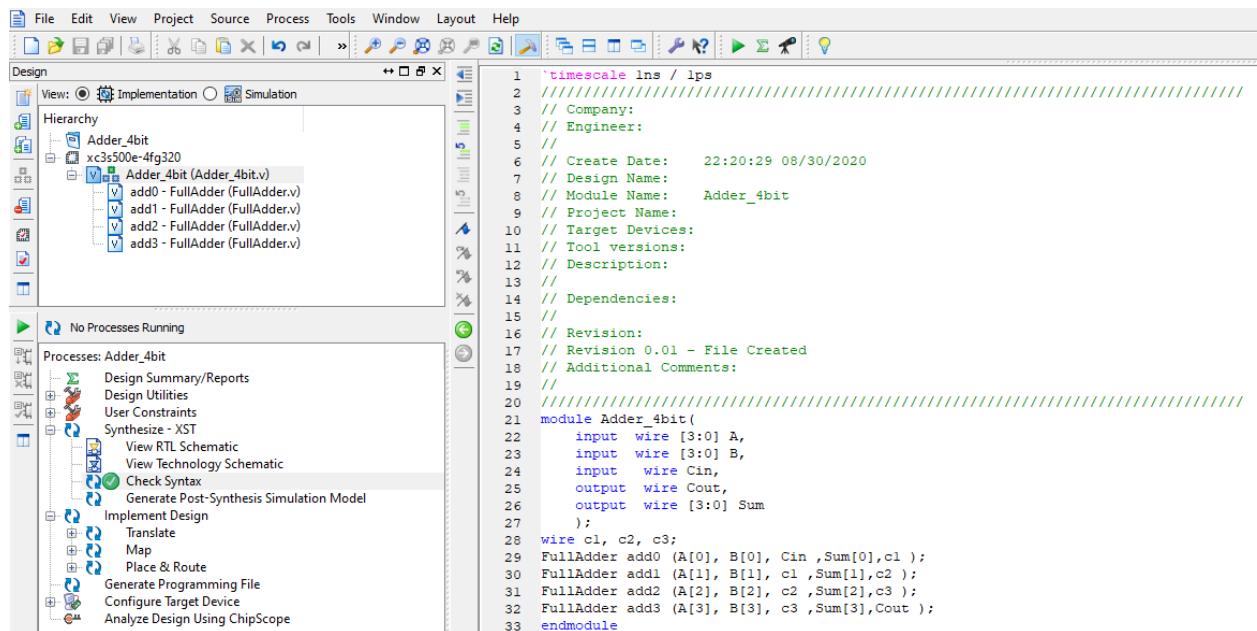
wire c1, c2, c3;
FullAdder add0 (A[0], B[0], Cin ,Sum[0],c1 );
FullAdder add1 (A[1], B[1], c1 ,Sum[1],c2 );
```

```

FullAdder add2 (A[2], B[2], c2 ,Sum[2],c3 );
FullAdder add3 (A[3], B[3], c3 ,Sum[3],Cout );
endmodule

```

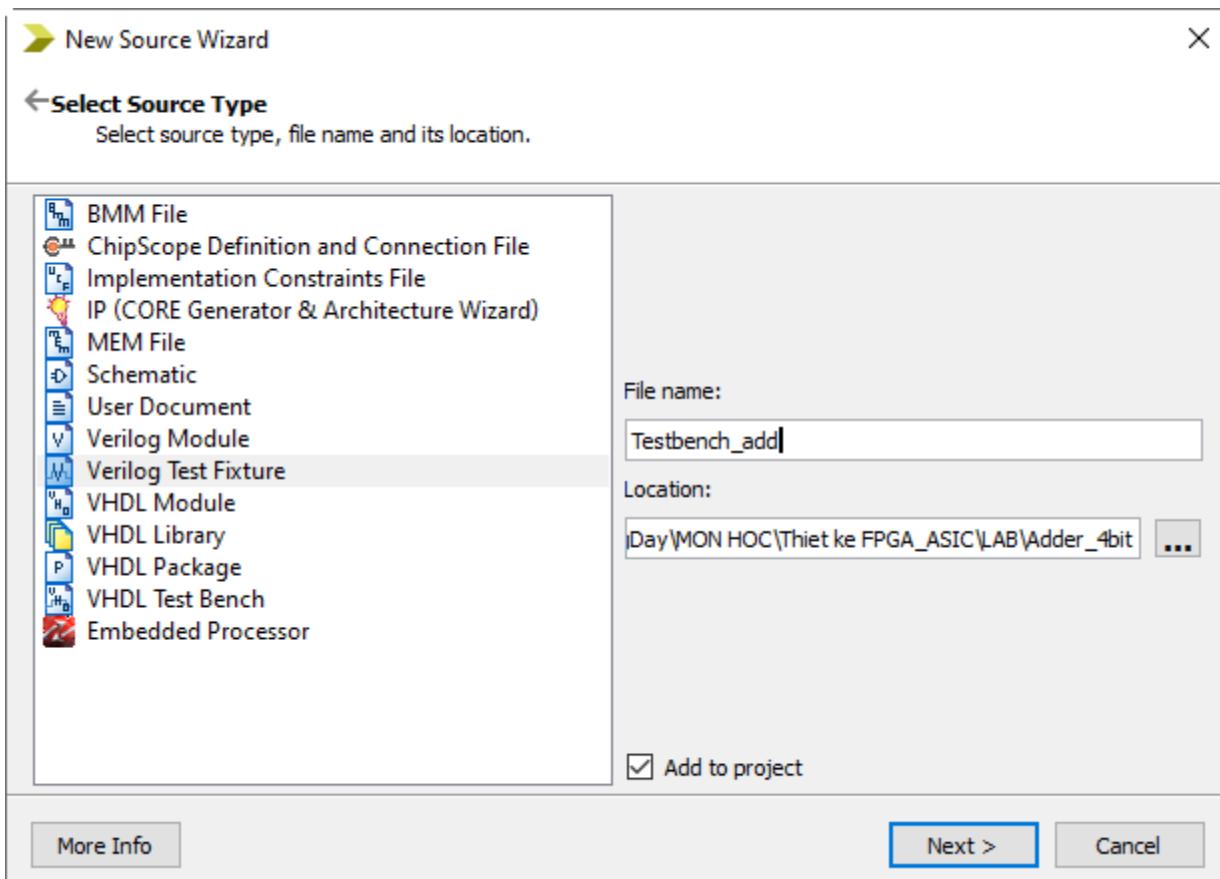
Kết quả sau khi thiết kế ta được như sau



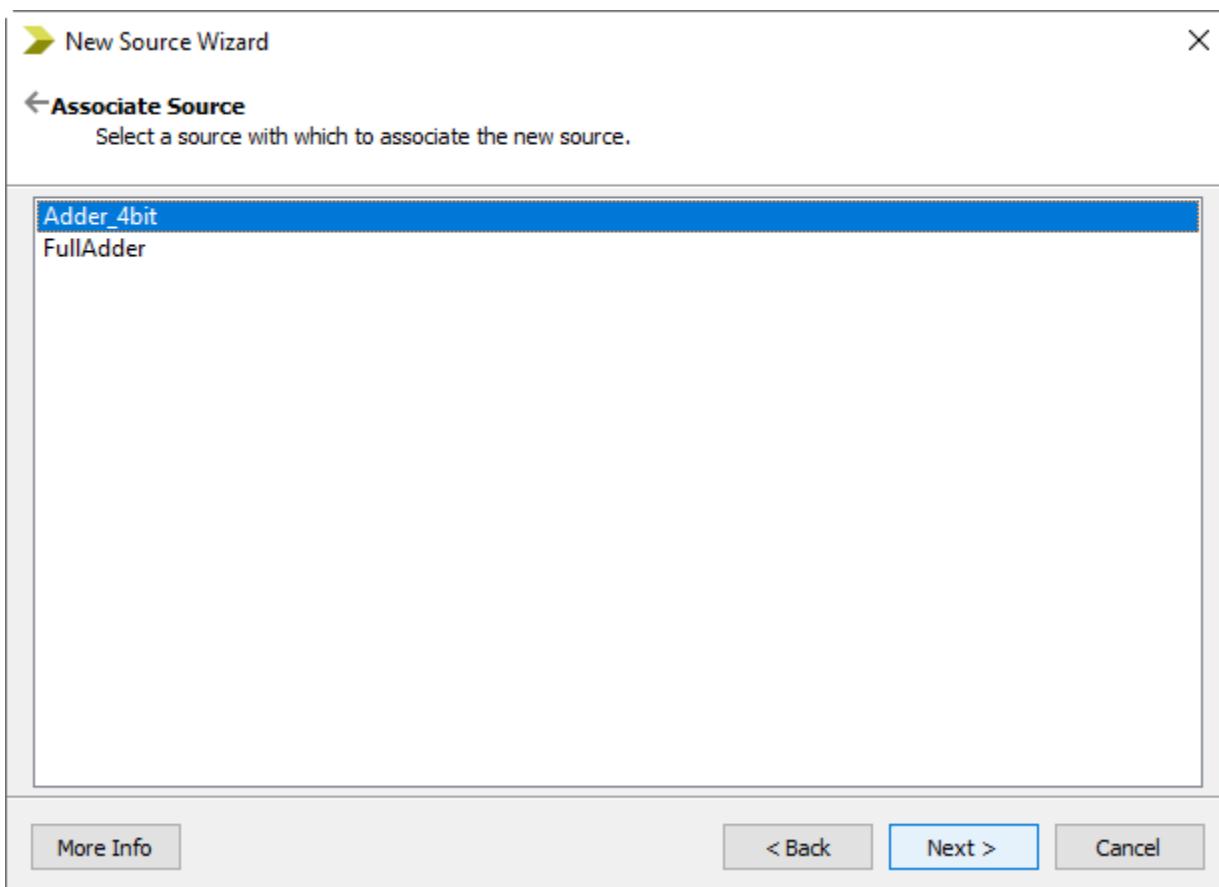
➤ Thiết kế mô phỏng hệ thống số

Để mô phỏng mạch cộng 4 bit, ta tạo mới một mô đun có 2 ngõ ra 4 bit, 1 ngõ ra 1 bit làm ngõ vào cho mạch cộng, mô đun này được kết nối trực tiếp với mô đun mạch cộng. Đồng thời, chúng ta tạo ra giá trị 2 ngõ ra 4 bit cho mô đun này để kiểm tra chức năng của mạch cộng.

Tương tự như quá trình thiết kế, chọn Project → New Source, chọn Verilog Test Fixture



Chọn mô đun để liên kết với mô đun tạo tín hiệu kiểm tra. Trong trường hợp này chúng ta chọn mô đun chính (Top module)



Tạo các tín hiệu để kiểm tra mạch cộng, giả sử trọng trường hợp này chúng ta tạo A =2, B=3, Cin =0.

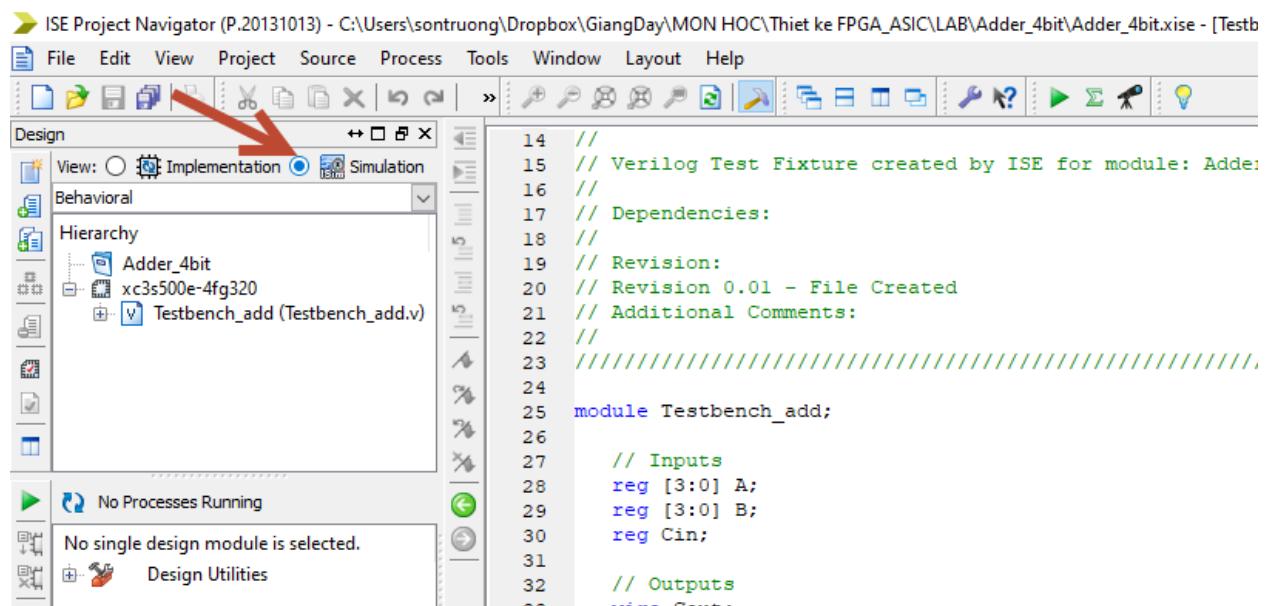
```
module Testbench_add;
    // Inputs
    reg [3:0] A;
    reg [3:0] B;
    reg Cin;
    // Outputs
    wire Cout;
    wire [3:0] Sum;
    // Instantiate the Unit Under Test (UUT)
    Adder_4bit uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Cout(Cout),
        .Sum(Sum)
    );
    initial begin
        // Initialize Inputs
    end
endmodule
```

```

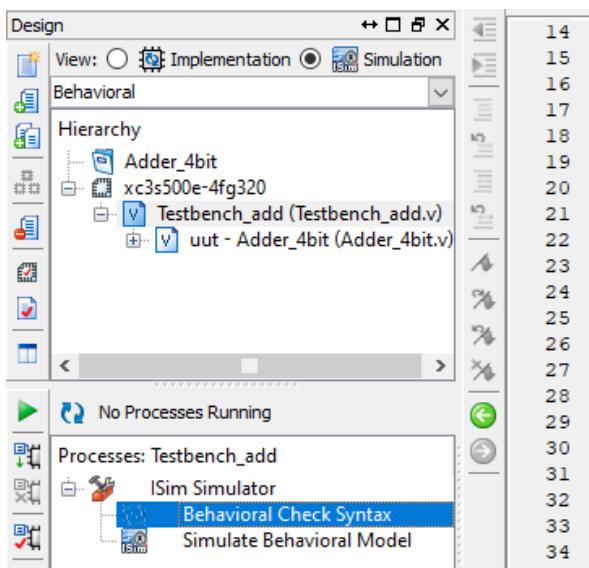
A = 2;
B = 3;
Cin = 0;
// Wait 100 ns for global reset to finish
#100;
// Add stimulus here
end
endmodule

```

- Mô phỏng mạch cộng
 - Chọn simulation



Chọn Behavioral Check syntax để kiểm tra lỗi. Nếu quá trình kiểm tra không báo lỗi, tiến hành mô phỏng mạch. Để khởi động chương trình mô phỏng, Double click vào Simulation Behavioral Model.

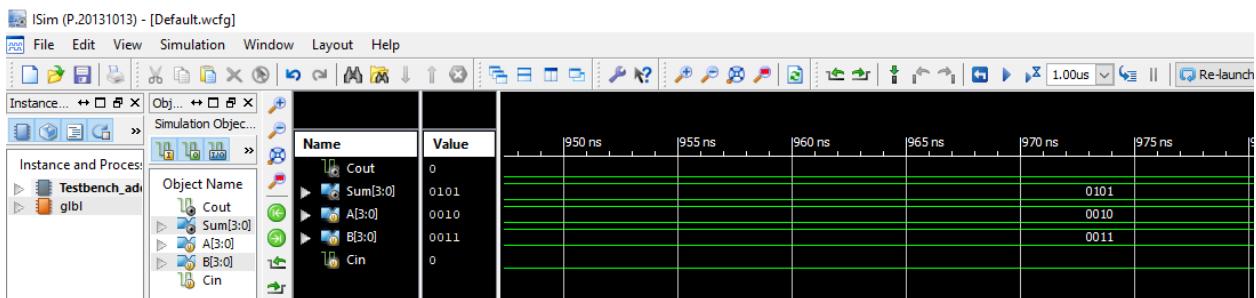


```

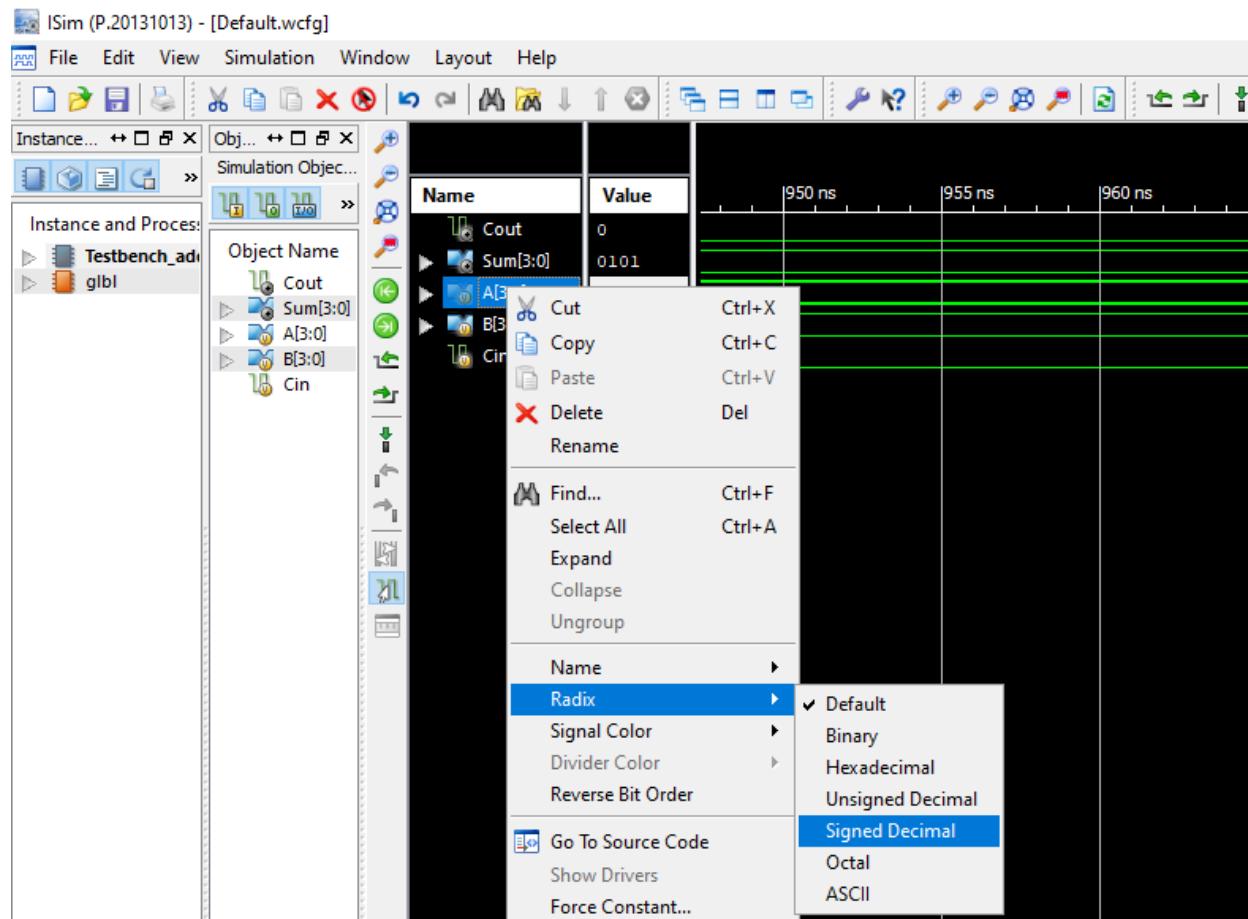
14 // Verilog Test Fixture created by ISE for module:
15 //
16 //
17 // Dependencies:
18 //
19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 /////////////////////////////////////////////////
24
25 module Testbench_add;
26
27     // Inputs
28     reg [3:0] A;
29     reg [3:0] B;
30     reg Cin;
31
32     // Outputs
33     wire Cout;
34     wire [3:0] Sum;
35
36 endmodule

```

Phần mềm Isim cho phép mô phỏng chức năng mạch (functional simulation). Các tín hiệu mặc định là các tín hiệu vào ra của mô đun chính và mô đun test. Các tín hiệu bên trong vi mạch có thể được lựa chọn để hiển thị. Kết quả mô phỏng mạch cộng như sau:

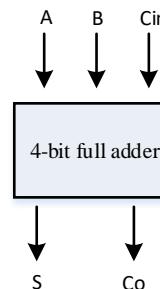


Mặc định chương trình mô phỏng sẽ tiến hành tính toán tín hiệu vào ra trong khoảng thời gian 1 μs. Thời gian thực hiện mô phỏng có thể được điều chỉnh trên thanh công cụ. Giá trị các tín hiệu có thể được hiển thị dưới dạng nhị phân, thập phân có dấu hoặc không dấu, hoặc thập lục phân bằng cách thay đổi định dạng dữ liệu.



➤ Cấu hình chân và lập trình thiết bị

Mạch cộng 4 bit có 2 ngõ vào 4 bit, a và b, 1 ngõ vào nhớ Cin, 1 ngõ ra 4 bit và một ngõ ra nhớ, được minh họa như sau:

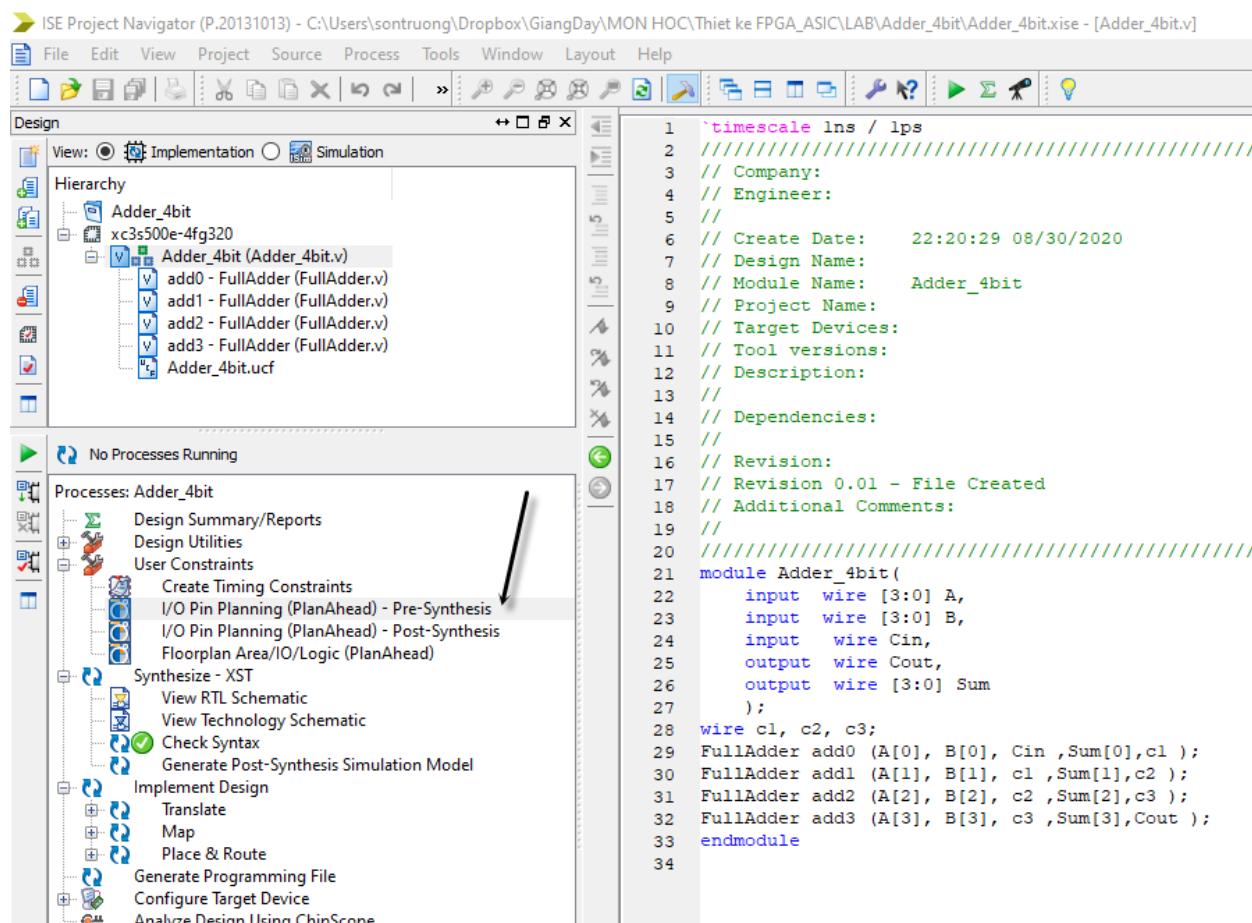


Các tín hiệu vào ra của mạch cộng được thiết lập kết nối đến các chân của vi mạch FPGA. Các chân giao tiếp của FPGA có thể được sử dụng là các tín hiệu vào, ra, hoặc cả tín hiệu vào và tín hiệu ra. Đồng thời có thể thiết lập giá trị điện trở treo cho các chân giao tiếp của FPGA. Việc kết nối các tín hiệu từ các mô đun đến chân nào của FPGA là tùy vào từng ứng dụng. Trong trường hợp sử dụng các board phát triển, các chân của vi mạch FPGA đã được kết nối đến một số thiết bị ngoại vi cho phép tạo tín hiệu ngõ vào như các nút nhấn (button), các công tắc (switch), các tín hiệu ngõ ra như LED, LCD. Như vậy để kiểm tra hoạt động của mạch cộng, có thể lựa chọn kết nối các tín hiệu A, B, Cin của mô đun cộng 4 bit đến các chân FPGA đã kết nối với các ngõ vào nút nhất hoặc công tắc. Ngõ ra của mô đun cộng có thể được kết nối đến các chân FPGA đã kết

nối đến LED để có thể kiểm tra kết quả phép cộng một cách trực quan. Để thiết lập kết nối chân, có thể sử dụng việc kết nối trực tiếp trên phần mềm hoặc sử dụng tập tin cho phép thiết lập kết nối tín hiệu. Cụ thể trong trường hợp này, tín hiệu A sẽ được kết nối với 4 nút nhất, tín hiệu B sẽ được kết nối đến 4 công tắt, Cin sẽ được kết nối đến nút nhấn. Ngõ ra được kết nối đến các LED

➤ Kết nối chân các mô đun đến các chân của FPGA

Chọn User Constraints – I/O Planning (PlanAhead) – Pre-synthesis để khởi động chương trình hỗ trợ kết gán tín hiệu đến các chân FPGA



Việc kết gán tín hiệu đến các chân của vi mạch FPGA có thể được thực hiện trực tiếp trên giao diện hoặc thông qua tập tin cấu hình. Việc thực hiện kết gán trực tiếp trên ứng dụng PlanAhead như sau

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Stre...	Slew Type	Pull Type
All ports (14)											
A (4)	Input					1 LVTTL*	3.300				PULLUP*
A[3]	Input		N17	<input checked="" type="checkbox"/>		1 LVTTL*	3.300				PULLUP*
A[2]	Input		H18	<input checked="" type="checkbox"/>		1 LVTTL*	3.300				PULLUP*
A[1]	Input		L14	<input checked="" type="checkbox"/>		1 LVTTL*	3.300				PULLUP*
A[0]	Input		L13	<input checked="" type="checkbox"/>		1 LVTTL*	3.300				PULLUP*
B (4)	Input					LVTTL*	3.300				PULLDOWN*
B[3]	Input		V4	<input checked="" type="checkbox"/>		2 LVTTL*	3.300				PULLDOWN*
B[2]	Input		D18	<input checked="" type="checkbox"/>		1 LVTTL*	3.300				PULLDOWN*
B[1]	Input		K17	<input checked="" type="checkbox"/>		1 LVTTL*	3.300				PULLDOWN*
B[0]	Input		H13	<input checked="" type="checkbox"/>		1 LVTTL*	3.300				PULLDOWN*
Sum (4)	Output					0 LVTTL*	3.300		12 SLOW		NONE
Sum[3]	Output		F11	<input checked="" type="checkbox"/>		0 LVTTL*	3.300		12 SLOW		NONE
Sum[2]	Output		E11	<input checked="" type="checkbox"/>		0 LVTTL*	3.300		12 SLOW		NONE
Sum[1]	Output		E12	<input checked="" type="checkbox"/>		0 LVTTL*	3.300		12 SLOW		NONE
Sum[0]	Output		F12	<input checked="" type="checkbox"/>		0 LVTTL*	3.300		12 SLOW		NONE
Scalar ports (2)											
Cin	Input		V16	<input checked="" type="checkbox"/>		2 LVTTL*	3.300				PULLDOWN*
Cout	Output		C11	<input checked="" type="checkbox"/>		0 LVTTL*	3.300				NONE

FPGA hỗ trợ nhiều mức điện áp cho các chân giao tiếp để tương thích các chuẩn giao tiếp khác nhau. Trong ví dụ này chúng ta sử dụng điện áp thấp TTL (LVTTL). Ngõ vào ra có thể được thiết lập điện trở kéo lên hoặc kéo xuống. Đồi với các nút nhất và Switch được thiết kế không có điện trở treo bên ngoài nhằm tận dụng được đặc tính này của FPGA. Sinh viên đọc sơ đồ nguyên lý của board phát triển FPGA Spartan 3E – Starter để hiểu thêm cách gán các chân FPGA. Sau khi lưu lại, chương trình tự tạo tập tin cấu hình (.ucf) có nội dung như bên dưới. Trong các ứng dụng tiếp theo, có thể thực hiện phương pháp cấu hình như đã trình bày, hoặc có thể sử dụng việc cấu hình bằng cách thêm các nội dung vào tập tin cấu hình (.ucf).

```
# PlanAhead Generated IO constraints
NET "A[3]" PULLUP;
NET "A[2]" PULLUP;
NET "A[1]" PULLUP;
NET "A[0]" PULLUP;
NET "B[3]" PULLDOWN;
NET "B[2]" PULLDOWN;
NET "B[1]" PULLDOWN;
NET "B[0]" PULLDOWN;
NET "A[3]" IOSTANDARD = LVTTL;
NET "A[2]" IOSTANDARD = LVTTL;
NET "A[1]" IOSTANDARD = LVTTL;
NET "A[0]" IOSTANDARD = LVTTL;
NET "B[3]" IOSTANDARD = LVTTL;
NET "B[2]" IOSTANDARD = LVTTL;
NET "B[1]" IOSTANDARD = LVTTL;
NET "B[0]" IOSTANDARD = LVTTL;
NET "Sum[3]" IOSTANDARD = LVTTL;
NET "Sum[2]" IOSTANDARD = LVTTL;
NET "Sum[1]" IOSTANDARD = LVTTL;
NET "Sum[0]" IOSTANDARD = LVTTL;
NET "Cin" IOSTANDARD = LVTTL;
NET "Cout" IOSTANDARD = LVTTL;

# PlanAhead Generated physical constraints
```

```

NET "A[0]" LOC = L13;
NET "A[1]" LOC = L14;
NET "A[2]" LOC = H18;
NET "A[3]" LOC = N17;
NET "B[3]" LOC = V4;
NET "B[2]" LOC = D18;
NET "B[1]" LOC = K17;
NET "B[0]" LOC = H13;
NET "Cin" LOC = V16;
NET "Sum[3]" LOC = F11;
NET "Sum[0]" LOC = F12;
NET "Sum[1]" LOC = E12;
NET "Sum[2]" LOC = E11;
NET "Cout" LOC = C11;
NET "Cin" PULLDOWN;

```

Các cấu hình trong tập tin .ucf có thể được viết kết hợp lại như sau

```

# PlanAhead Generated IO constraints
NET "A<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "A<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "A<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;
NET "A<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;

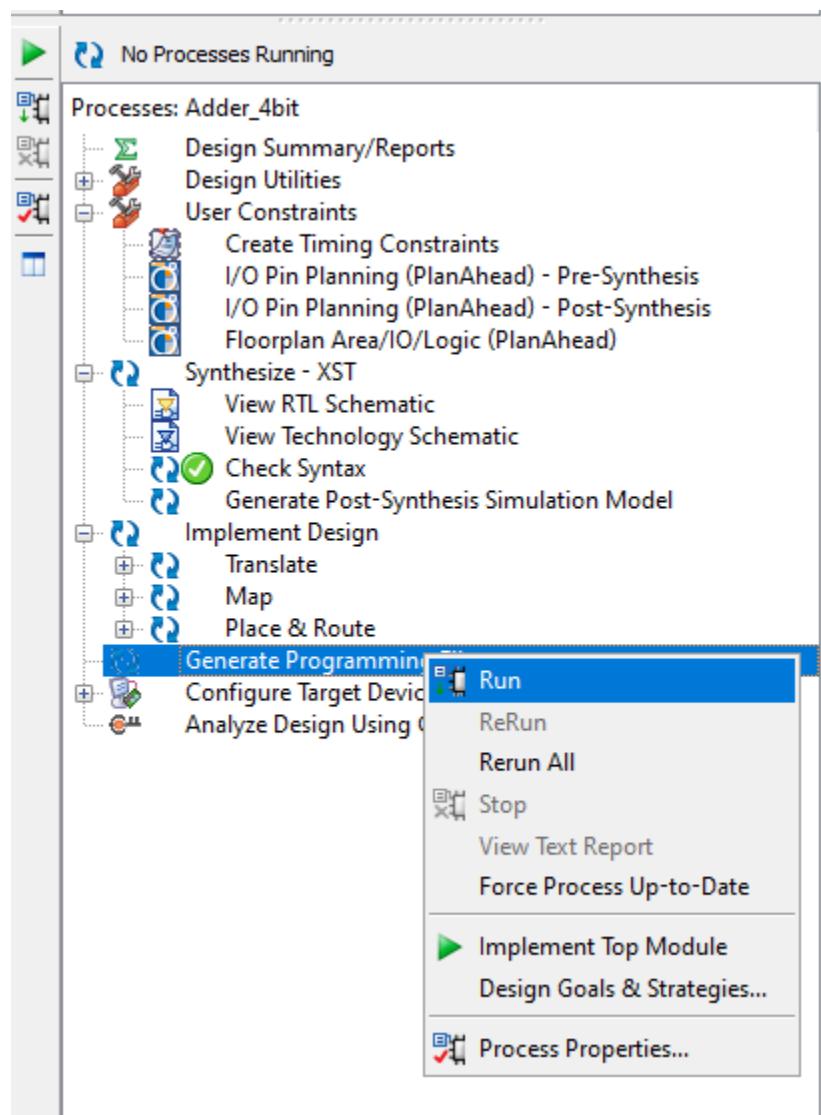
NET "B<3>" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN;
NET "B<2>" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN;
NET "B<1>" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN;
NET "B<0>" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN;

NET "Sum<3>" LOC = "F11" | IOSTANDARD = LVTTL;
NET "Sum<0>" LOC = "F12" | IOSTANDARD = LVTTL;
NET "Sum<1>" LOC = "E12" | IOSTANDARD = LVTTL;
NET "Sum<2>" LOC = "E11" | IOSTANDARD = LVTTL ;

NET "Cin" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN;
NET "Cout" LOC = "C11" | IOSTANDARD = LVTTL;

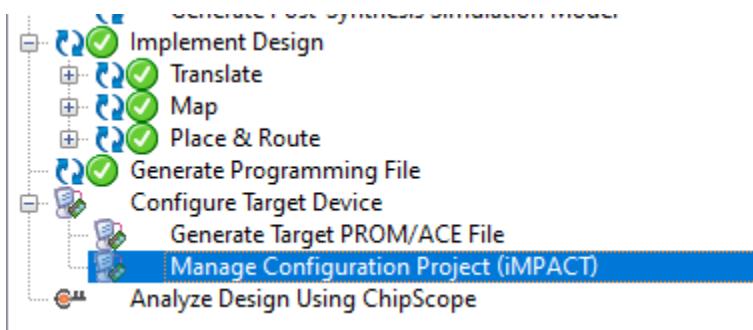
```

Sau khi gán chân, thực hiện các bước synthesis, mapping, place and route. Tuy nhiên các bước này có thể được thực hiện một cách liên tục theo thứ tự bằng phần mềm ISE. Chúng ta chọn Generate Programming File, phần mềm sẽ thực hiện các bước và tạo ra tập tin nhị phân lập trình cho FPGA

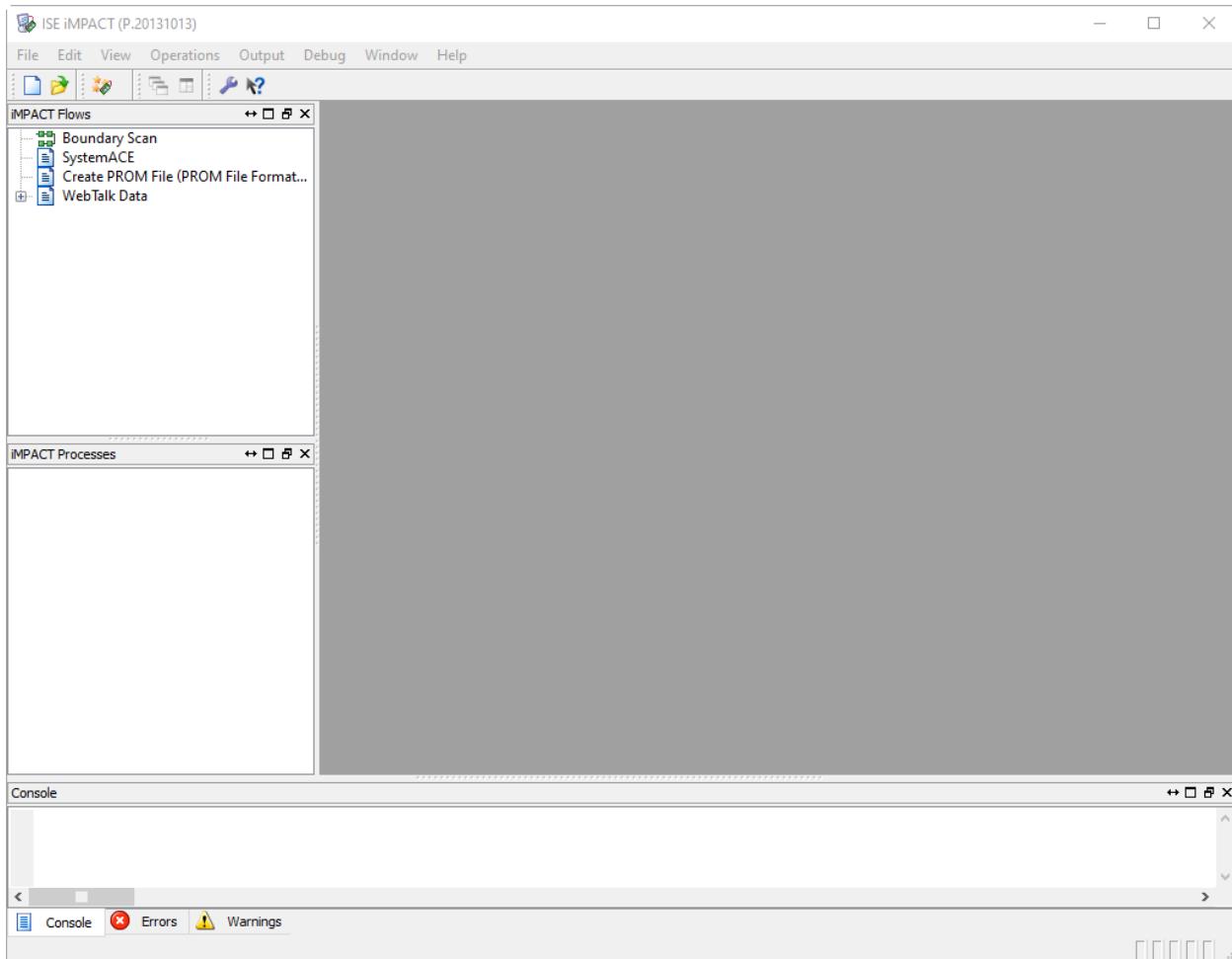


Quá trình tạo ra tập tin cấu hình cho FPGA thành công, chương trình sẽ tạo ra tập tin nhị phân (.bit). Tập tin này có thể được sử dụng để lập trình cho FPGA. Các vi mạch FPGA của xilinx có thể được lập trình thông qua ứng dụng iMPACT.

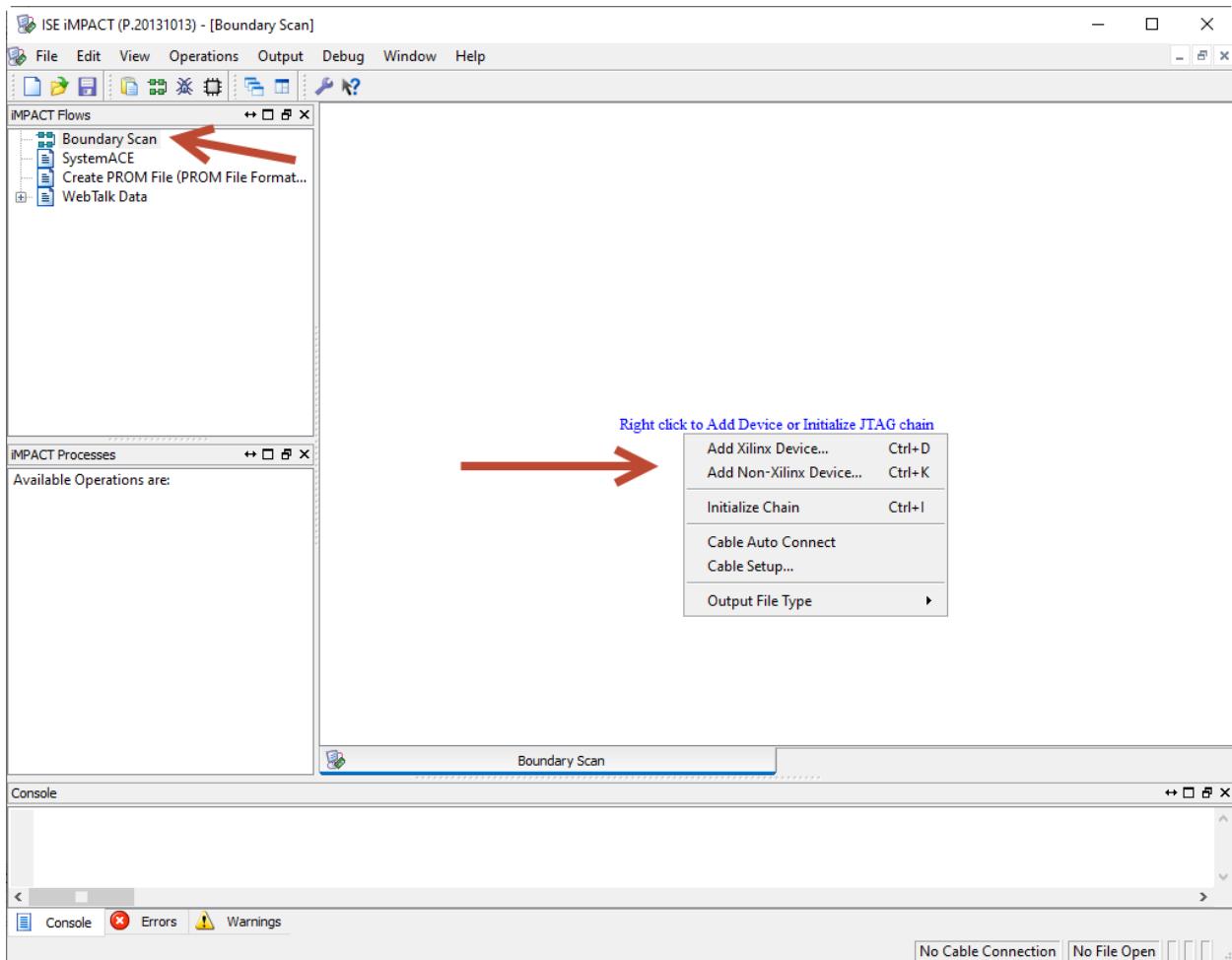
Kết nối board phát triển với máy tính thông qua USB, đảm bảo trong quá trình cài đặt ISE Design Suite chúng ta đã chọn cài đặt luôn các driver hỗ trợ cho kết nối máy tính và board phát triển. Khởi động ứng dụng iMPACT.



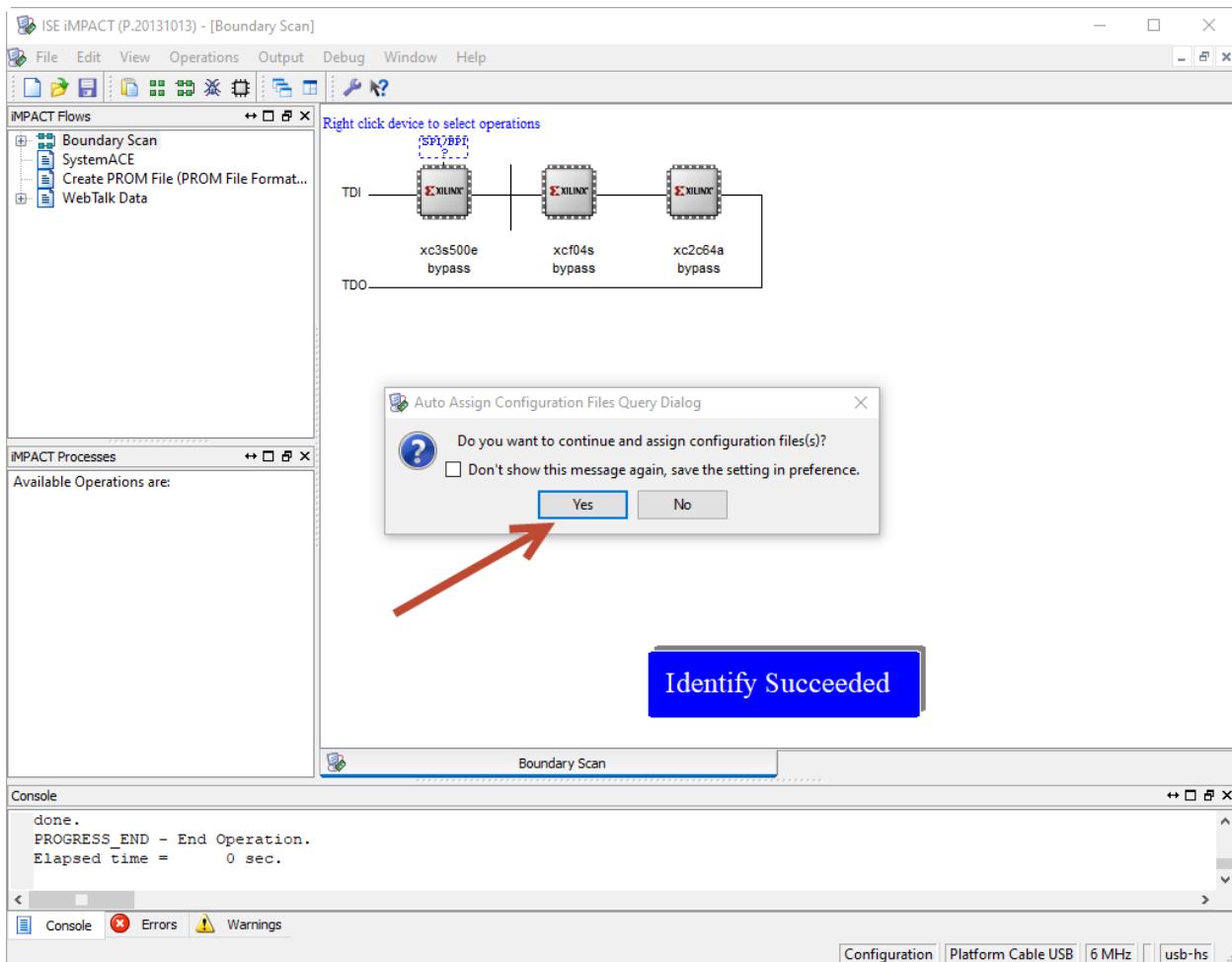
Ứng dụng ISE iMPACT được khởi động. Việc lập trình FPGA có thể được thực hiện bằng nhiều cách. Lập trình trực tiếp vào bộ nhớ bên trong FPGA, phương pháp này cho phép lập trình nhanh, tuy nhiên, chương trình sẽ bị mất khi reset FPGA hay khi tắt và mở nguồn lại. Phương pháp này được sử dụng cho phép kiểm tra nhanh chương trình và đặc biệt trong quá trình thiết kế chương trình. Chương trình có thể được lưu vào một bộ nhớ ROM bên ngoài, giao tiếp chuẩn SPI. Trong trường hợp này chương trình sẽ không bị mất nội dung khi mất điện. Sau khi cấp nguồn, chương trình sẽ được tải vào bộ nhớ của FPGA để cấu hình cho FPGA. Trong các bài tập thực hành, chúng ta sử dụng phương pháp thứ nhất để cấu hình cho FPGA nhằm tiết kiệm được thời gian. Quá trình lập trình cho FPGA được tiến hành các bước như sau



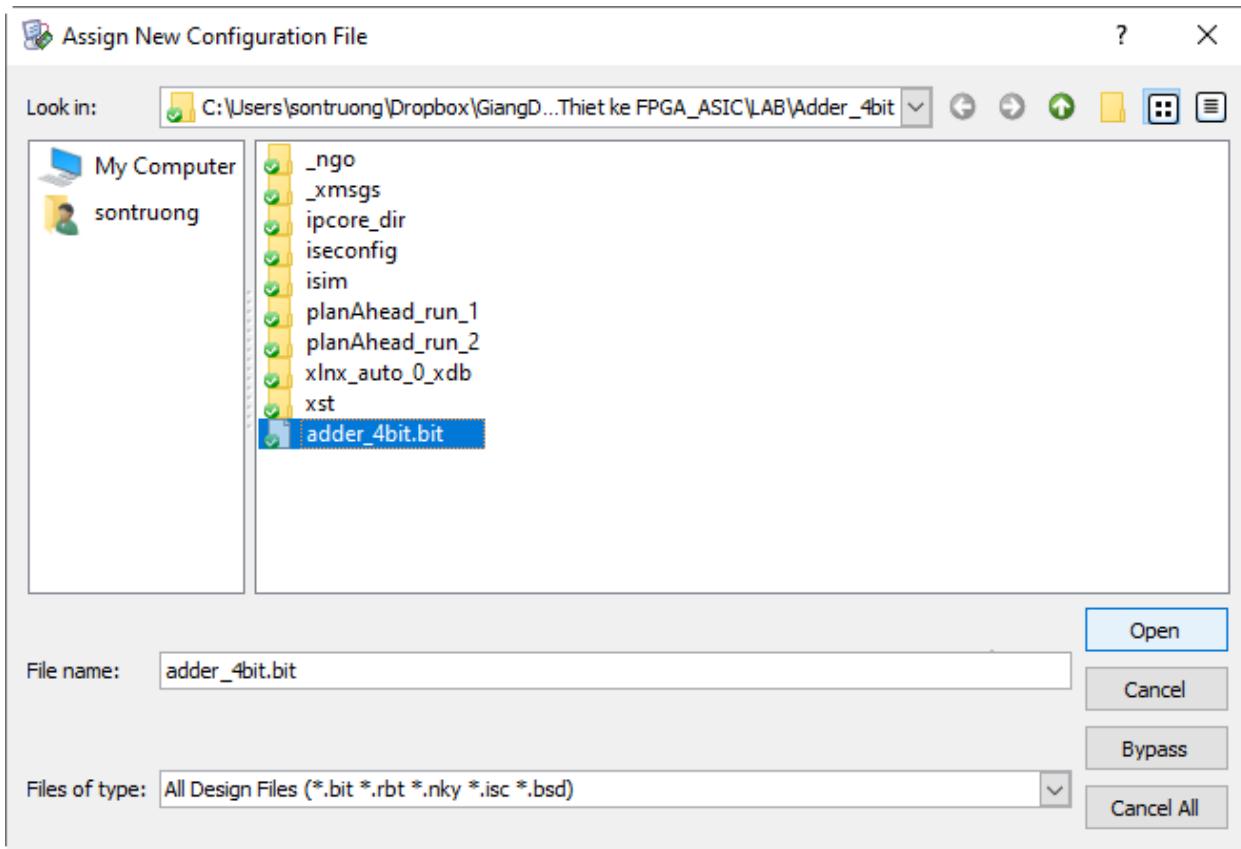
Double click vào Boundary scan, sau đó click chuột phải vào khu vực trung tâm chọn initialize chain



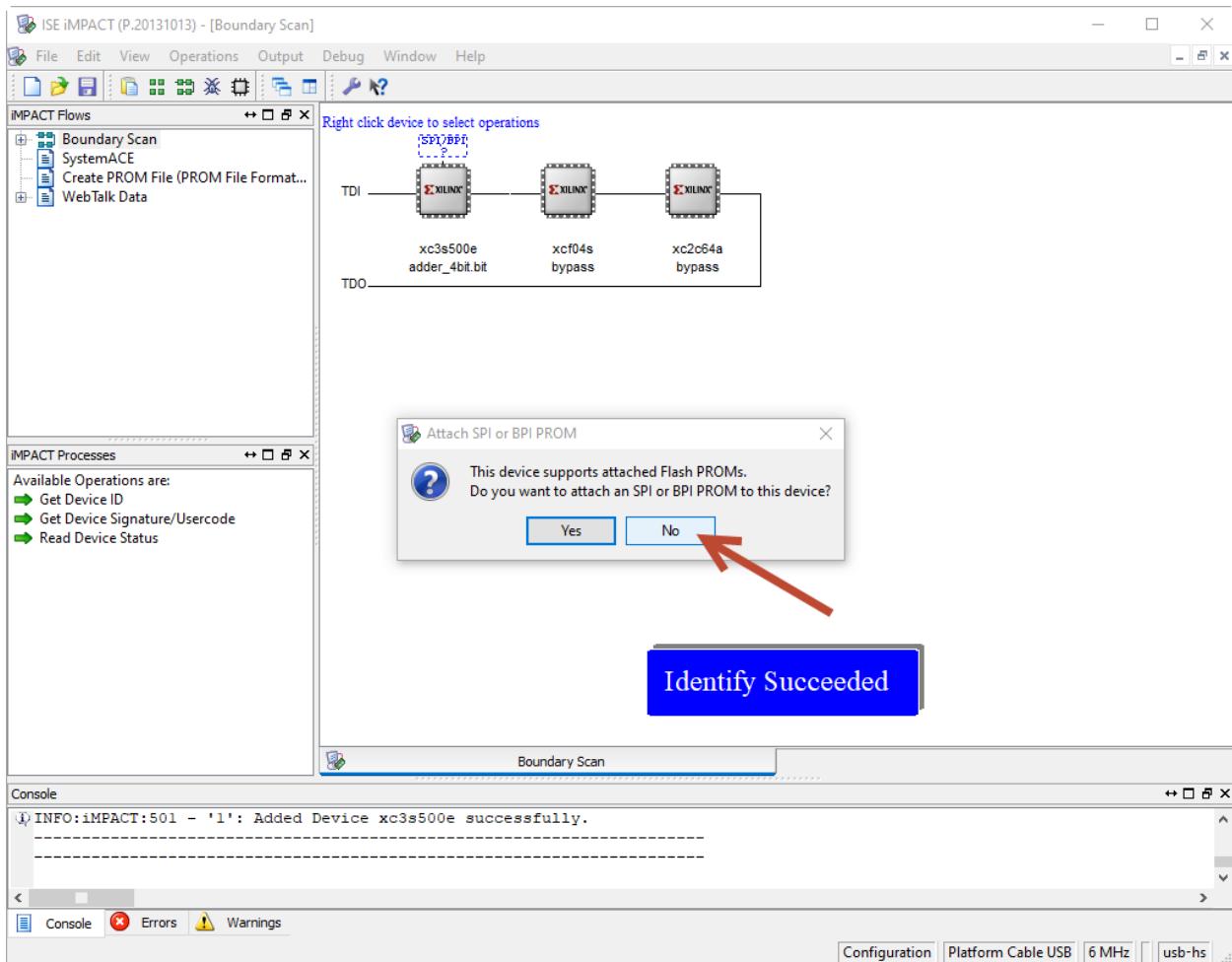
Nếu quá trình kết nối máy tính với board FPGA thành công, chúng ta có thể chọn tập tin nhị phân để lập trình cho FPGA



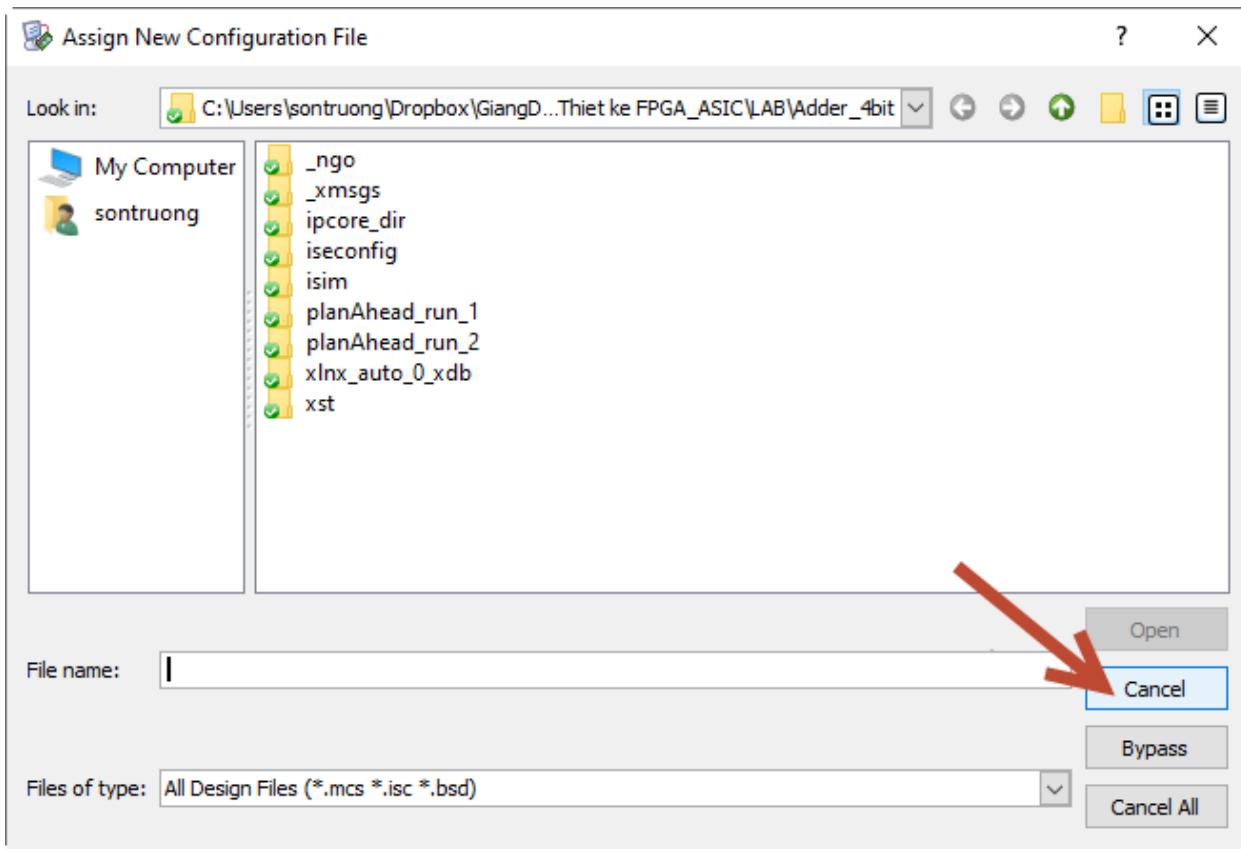
Chọn tập tin lập trình (.bit)



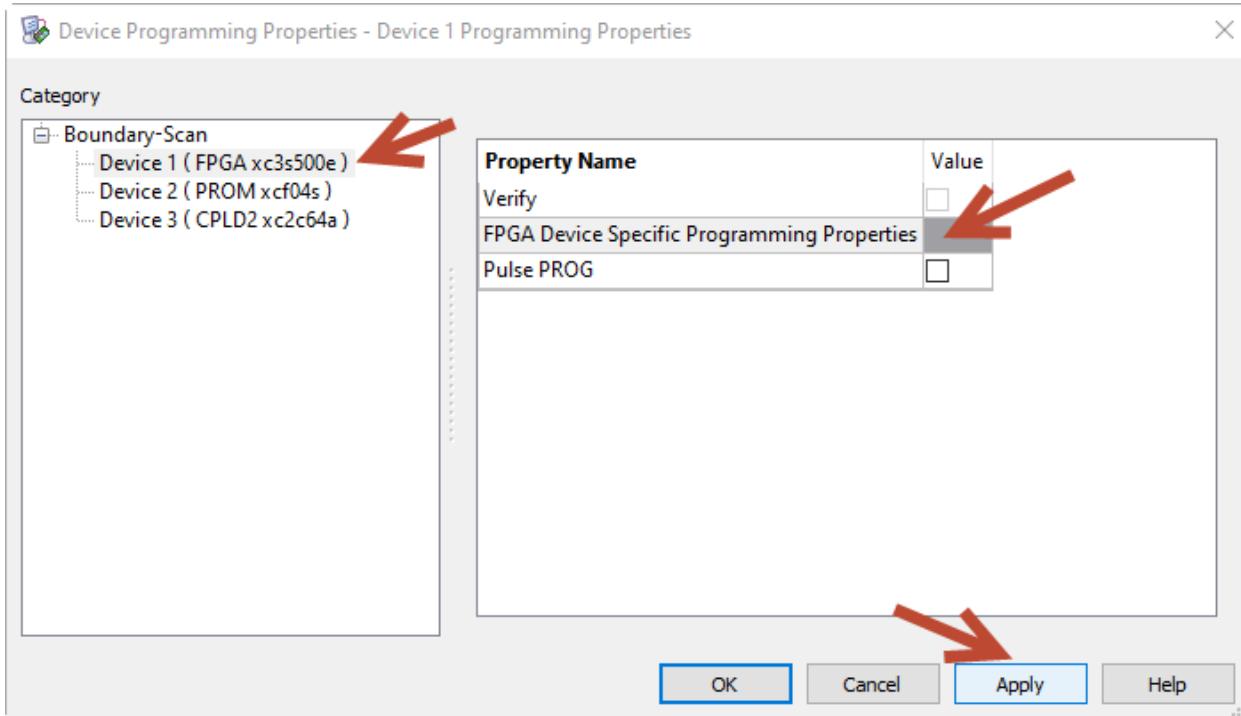
Trường hợp chúng ta không sử dụng PROM để lưu chương trình, chọn NO để bỏ qua chức năng lập trình PROM, sau đó chọn Cancel nếu chương trình hiển thị hộp thoại yêu cầu chọn tập tin cấu hình nạp vào PROM



Chọn Cancel (2 lần) để bỏ qua bước tiếp theo

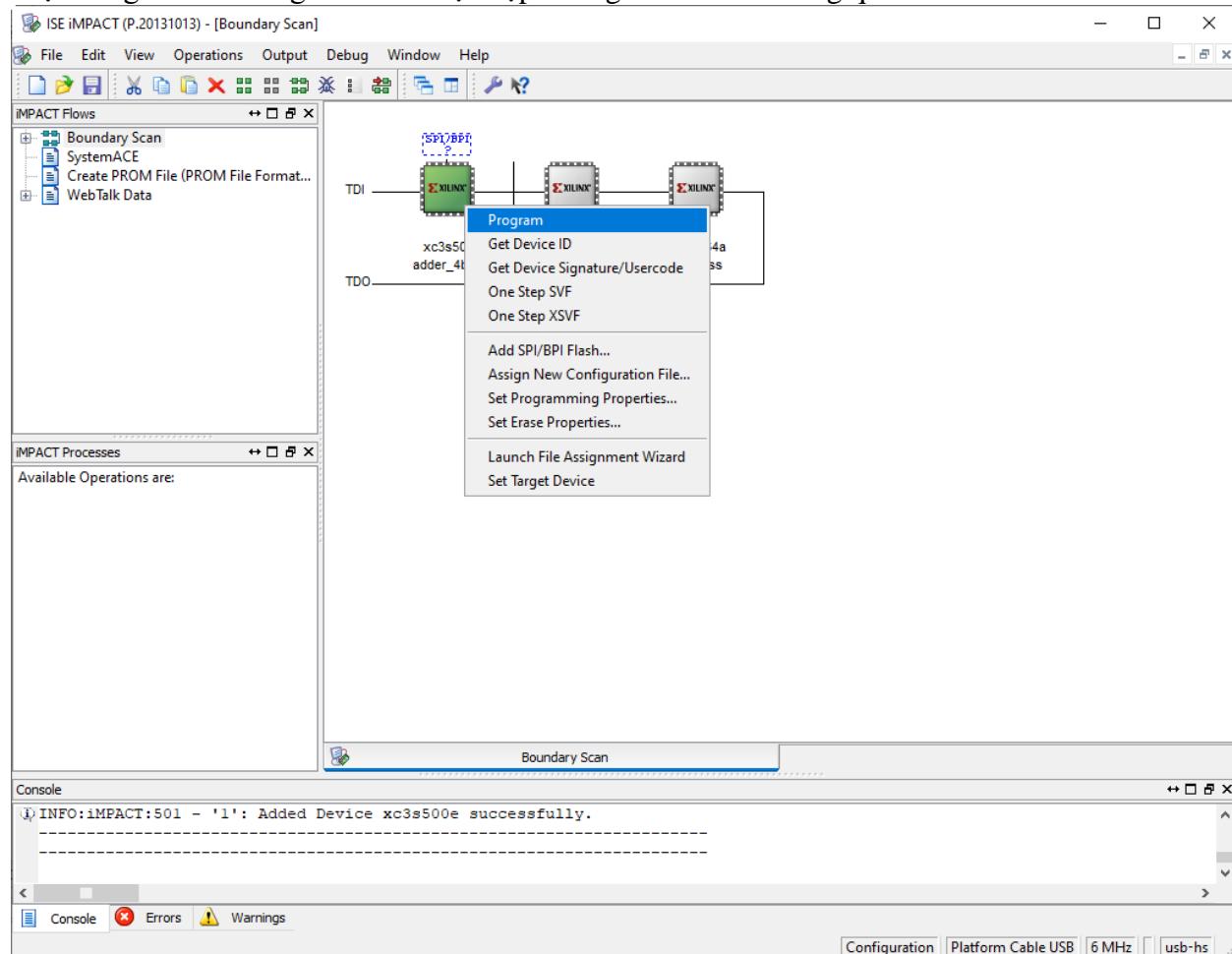


Tại bước này chúng ta lựa chọn lập trình trực tiếp vào bộ nhớ bên trong FPGA

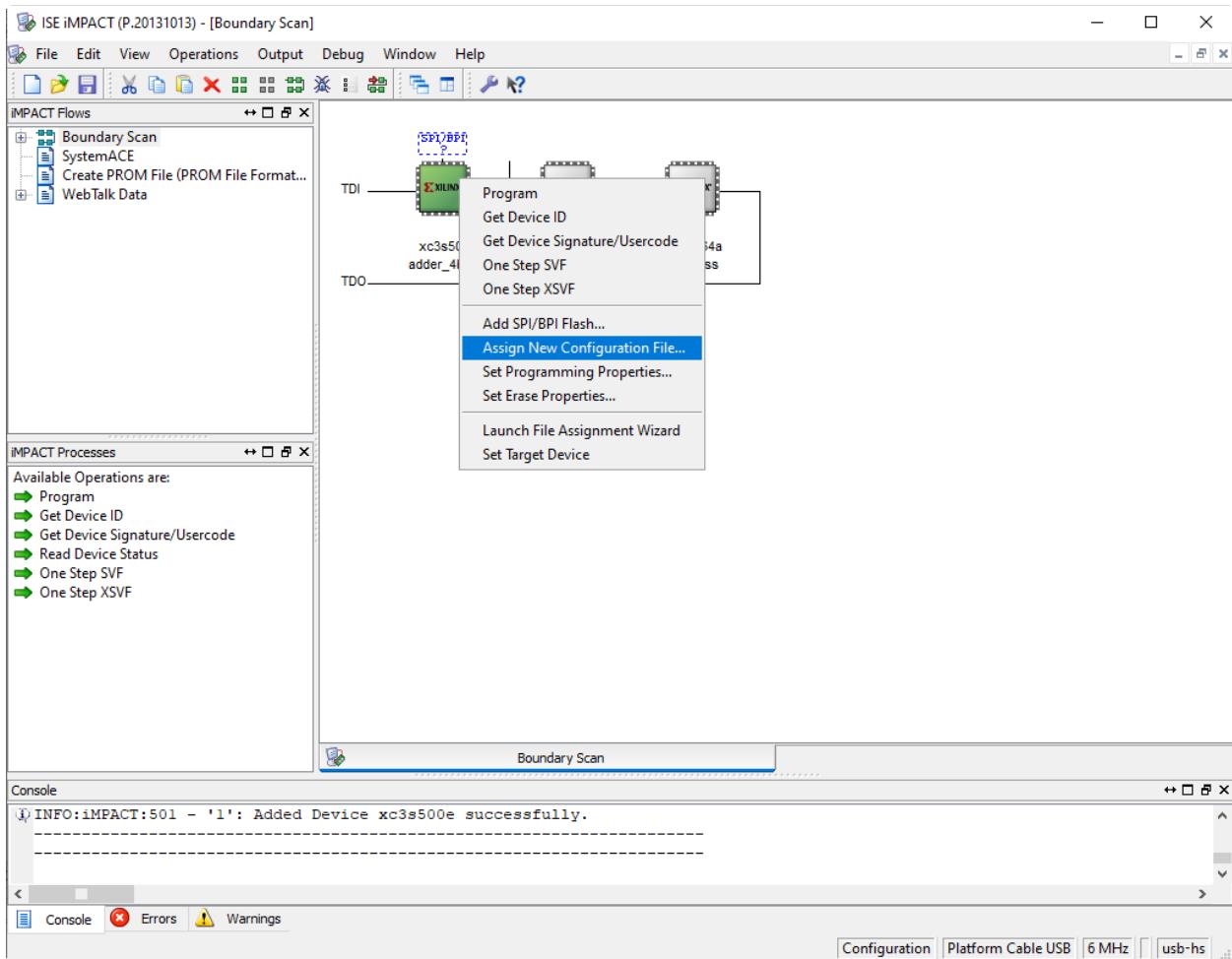


Chọn Apply, sau đó chọn OK.

Bước cuối cùng là lập trình tập tin đã chọn xuống cho FPGA. Click phải vào biểu tượng FPGA, chọn Program. Chương trình đã được nạp xuống cho FPGA thông qua USB.



Thay đổi các ngõ vào và quan sát ngõ ra để kiểm tra tính logic của mạch cộng. Khi thay đổi chương trình, tạo ra tập tin mới, thực hiện nạp lại chương trình chỉ cần click phải vào biểu tượng FPGA, chọn Assign New Configuration File.



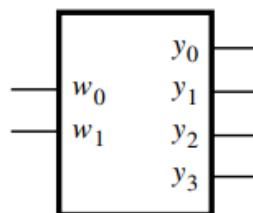
CHƯƠNG 2. THIẾT KẾ MẠCH TỐ HỢP

2.1.Thiết kế mạch giải mã

Thiết kế mạch giải mã 2 đường sang 4 đường, ngõ ra tích cực mức cao.

- Thiết kế mạch giải mã bằng ngôn ngữ Verilog.
- Mô phỏng chức năng mạch trên phần mềm Isim.
- Cấu hình chân và lập trình FPGA.

w_1	w_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



(a) Truth table

(b) Graphical symbol

Mô đun mạch giải mã

```
module decoder24(
  input wire [1:0] w,
  output reg [3:0] y
);
  always @(w)
    case (w)
      0: y = 4'b1000;
      1: y = 4'b0100;
      2: y = 4'b0010;
      3: y = 4'b0001;
    endcase
endmodule
```

Mô đun mạch kiểm tra (test Fixture)

```
module test;

  // Inputs
  reg [1:0] w;
  // Outputs
  wire [3:0] y;
  // Instantiate the Unit Under Test (UUT)
  decoder24 uut (
    .w(w),
    .y(y)
  );
endmodule
```

```

initial begin
    // Initialize Inputs
    w = 0;
    #100;
    w = 1;
    #100;
    w = 2;
    #100;
    w = 3;
    #100;
    // Add stimulus here
end

endmodule

```

Cấu hình chân cho FPGA

```

NET "w<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "w<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;

NET "y<0>" LOC = "F12" | IOSTANDARD = LVTTL;
NET "y<1>" LOC = "E12" | IOSTANDARD = LVTTL;
NET "y<2>" LOC = "E11" | IOSTANDARD = LVTTL ;
NET "y<3>" LOC = "F11" | IOSTANDARD = LVTTL;

```

1. Giải thích sự khác nhau giữa wire và reg sử dụng trong mô đun giải mã
2. Vẽ lại dạng sóng tín hiệu mô phỏng mạch giải mã
3. Mô tả lại mạch giải mã 2 sang 4, sử dụng phát biểu if.

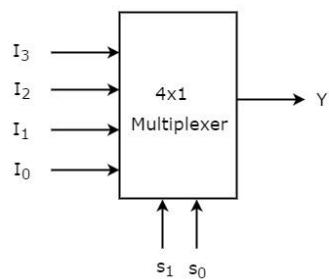
2.2.Thiết kế mạch mã hóa 4 đường sang 2 đường

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(a) Truth table

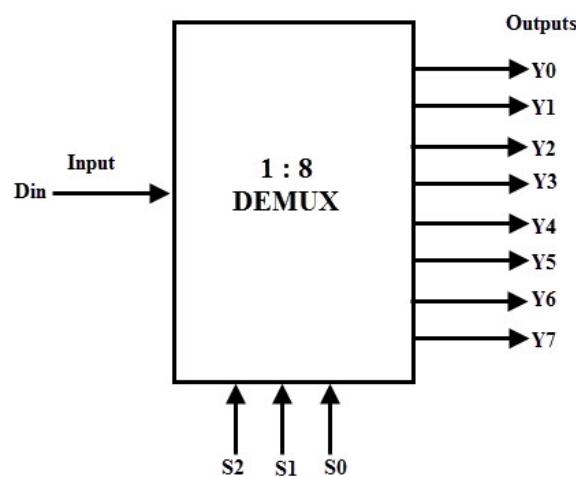
- Thiết kế mạch mã hóa 4 đường sang 2 đường sử dụng phát biểu if, và phát biểu case
- Thiết kế mô mạch tạo tín hiệu kiểm tra mạch mã hóa (Test fixture)
- Mô phỏng mạch mã hóa 4 sang 2 sử dụng Isim
- Cấu hình chân và lập trình FPGA, kiểm tra chức năng mạch mã hóa

2.3.Thiết kế mạch đa hợp 4 đường sang 1 đường



- Thiết kế mạch đa hợp 4 đường sang 1 đường
- Thiết kế mạch tạo tín hiệu mô phỏng mạch đa hợp 4 đường sang 1 đường (Test fixture)
- Kiểm tra chức năng mạch trên board FPGA

2.4.Thiết kế mạch giải đa hợp 1 đường sang 8 đường



- Thiết kế mạch giải đa hợp 1 đường sang 8 đường
- Thiết kế mạch tạo tín hiệu mô phỏng mạch giải đa hợp 1 đường sang 8 đường (test fixture)
- Kiểm tra chức năng mạch trên board FPGA

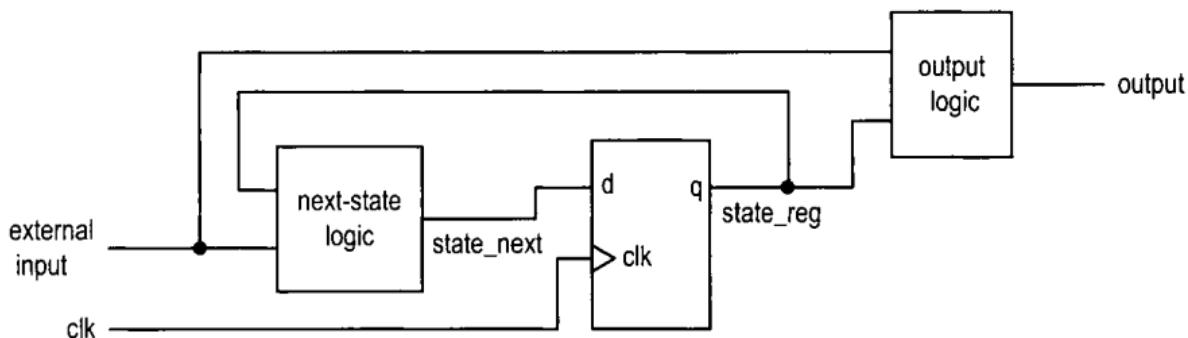
CHƯƠNG 3. THIẾT KẾ MẠCH TUẦN TỰ ĐỒNG BỘ

3.1. Giới thiệu

Mạch tuần tự là các mạch logic trong đó ngõ ra phụ thuộc vào trạng thái hiện tại của mạch và ngõ vào. Không giống như các mạch tổ hợp, trong đó ngõ ra là hàm của các ngõ vào, mạch tuần tự tạo ra ngõ ra là hàm của ngõ vào và các trạng thái bên trong của nó.

Mạch tuần tự bao gồm mạch tuần tự đồng bộ (được điều khiển bởi xung clock) và mạch tuần tự không đồng bộ.

Phương pháp thiết kế đồng bộ được sử dụng khá phổ biến khi thiết kế các mạch tuần tự. Trong phương pháp này, các phần tử lưu trữ được điều khiển bởi một xung clock chung, dữ liệu được lấy mẫu, hoặc lưu trữ ở thời điểm xung cạnh lên hoặc xuống của clock.



Hình 3.1. Mô hình thiết kế mạch tuần tự đồng bộ

Hình 3.1 mô tả sơ đồ khối của một mạch tuần tự. Trong đó

- + State register: là tập hợp các flip flop D được điều khiển bởi cùng một xung clock
- + Next state logic: mạch tổ hợp, sử dụng các ngõ vào bên trong và trạng thái bên trong
- + Output logic: mạch tổ hợp tạo ra tín hiệu ngõ ra

Sơ đồ khối một mạch tuần tự được mô tả như sau

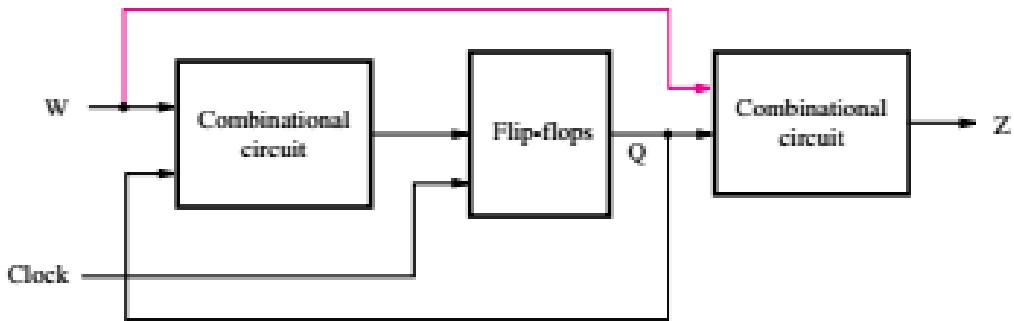


Figure 6.1 The general form of a sequential circuit.

Hình 3.2. Sơ đồ khái quát của mạch tuần tự

3.2. Mạch đếm (Counter)

Thiết kế mạch đếm 4 bit, ngõ vào xung đếm lấy từ switch, ngõ ra hiển thị trên 4 LED đơn.

- Thiết kế mạch đếm 4 bit bằng phương pháp thiết kế mạch tuần tự đồng bộ

- Thiết kế mô mạch tạo tín hiệu kiểm tra mạch đếm (Test fixture)
- Mô phỏng mạch đếm sử dụng ISim
- Cấu hình chân và lập trình FPGA, kiểm tra chức năng mạch đếm

Mạch đếm nhị phân 4 bit

```
module SynCounter4bit(
    input wire clk, reset,
    output wire [3:0] q
);
// signal declaration
    reg [3:0] r_reg;
    wire [3:0] r_next;
// body, register
    always @(posedge clk, posedge reset)
        if (reset)
            r_reg <= 0;
        else
            r_reg<=r_next;      // <= is non-blocking statement
// next state logic
    assign r_next = r_reg + 1;
// output logic
    assign q=r_reg;
endmodule
```

Mô đun kiểm tra mạch đếm 4 bit. (**Sinh viên giải thích chương trình tạo xung**)

```
module TestCounter;
    // Inputs
    reg clk;
    reg reset;
    // Outputs
    wire [3:0] q;
    // Instantiate the Unit Under Test (UUT)
    SynCounter4bit uut (
        .clk(clk),
        .reset(reset),
        .q(q)
    );
    integer i;

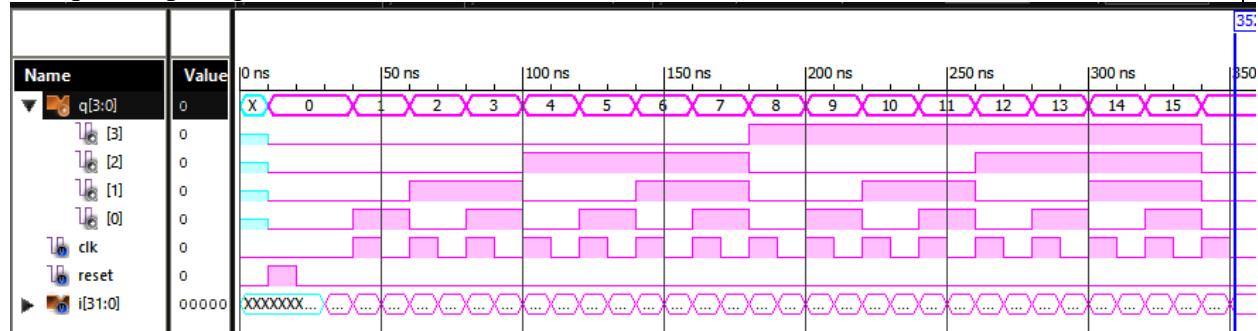
    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;
        #10;
        reset = 1;
        #10;
        reset = 0;
        // Wait 10 ns for global reset to finish
    end
endmodule
```

```

#10;
for (i=0;i<32;i=i+1)
#10 clk = ~clk;
    // Add stimulus here
end
endmodule

```

Kết quả mô phỏng.



Cấu hình chân cho FPGA

```

NET "clk" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
NET "reset" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;

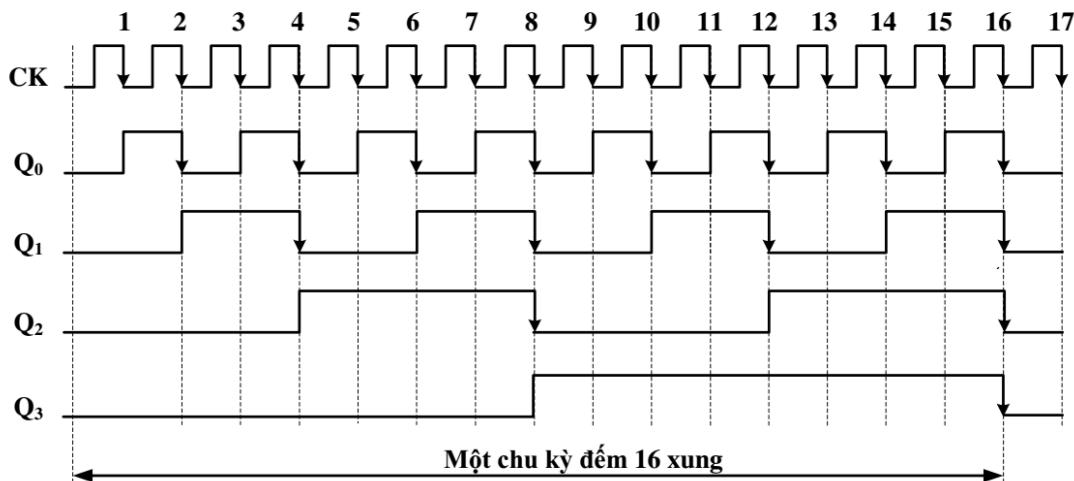
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL;

```

3.3. Thiết kế mạch chia xung, sử dụng mạch đếm lên

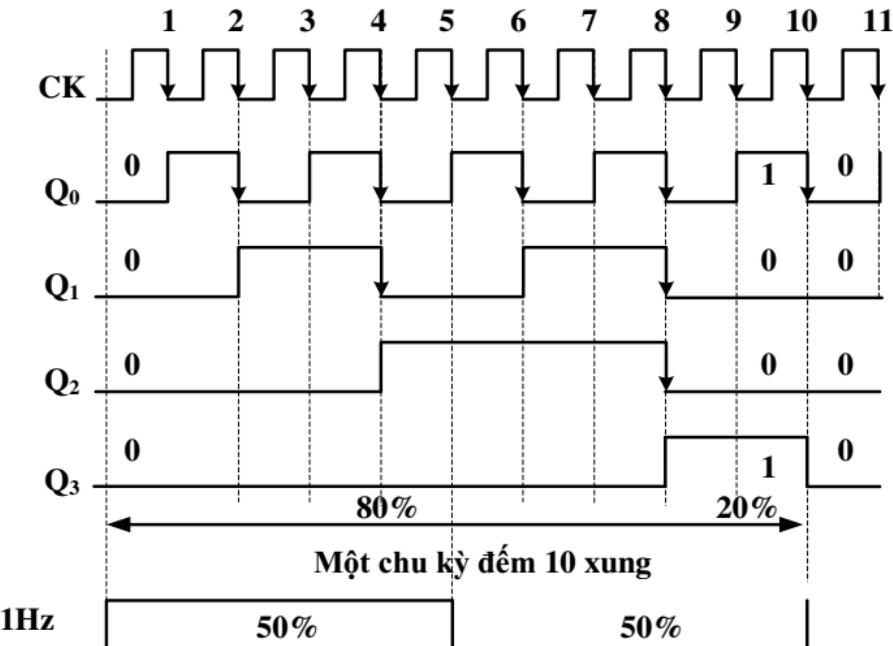
FPGA prototype kit cung cấp mạch dao động tần số cao (từ 50Mhz đến 100MHz). Trong một số ứng dụng cần tần số thấp hơn, ví dụ mạch đếm, mạch ghi dịch, hoặc điều khiển ngoại vi cần các xung clock có tần số khác nhau và thấp hơn để có thể quan sát được. Module chia xung được thiết kế nhằm tạo ra một hoặc nhiều ngõ ra xung clock có tần số thấp hơn.

Mạch chia xung có thể được thiết kế bằng nhiều cách khác nhau, trong đó, có thể sử dụng mạch đếm nhị phân n-bit để thiết kế mạch chia xung. Ý tưởng thiết kế mạch chia xung sử dụng mạch đếm được mô tả như sau: giả sử một bộ đếm nhị phân 4 bit, ngõ vào CK, các ngõ ra đếm sẽ có tần số khác nhau được biểu diễn như hình bên dưới.

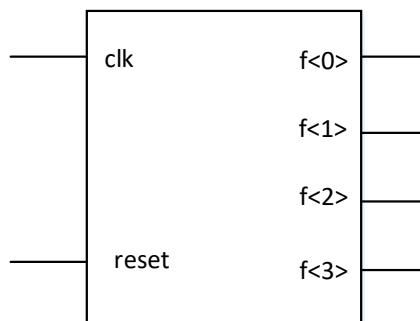


Ý tưởng thiết kế bộ chia xung dựa vào mạch đếm nhị phân rất đơn giản. Tần số ngõ ra thấp hơn bao nhiêu tùy thuộc vào việc lựa chọn ngõ ra nào làm xung clock. Ví dụ cụ thể trong hình trên, tần số tại ngõ ra Q_0 sẽ là $\frac{1}{2}$ tần số CK, tương tự tần số ngõ ra Q_1 sẽ là $\frac{1}{4}$ tần số xung CK. Một cách tổng quát, tần số ngõ ra Q_n sẽ là $\frac{1}{2^{n+1}}$ của tần số CK. Một hạn chế của phương pháp này là không thể đạt được tần số chính xác mong muốn, ví dụ tạo tần số ngõ ra 1Hz từ tần số xung clock 50Mhz

Để tạo ra tần số chính xác, phương pháp đếm số xung ngõ vào để chuyển trạng thái ngõ ra được áp dụng thay cho việc đếm nhị phân như trên. Phương pháp tạo ra xung có tần số mong muốn được trình bày trong hình bên dưới.



3.3.1. Thiết kế mạch chia xung với ngõ vào 50Mhz, 4 xung ngõ ra với tần số f, 2f, 4f, 8f, trong đó lựa chọn f ~ 1Hz



Mạch đếm mode N

```

module Counter
  #(parameter N= 26)
  (input wire clk, reset,
   output wire [3:0] q
  );
  // signal declaration
  reg [N-1:0] r_reg;
  wire [N-1:0] r_next;

  // body, register
  always @(posedge clk, posedge reset)
    if (reset)
      r_reg <= 0;
    else
      r_reg<=r_next; // <= is non-blocking statement
  // next state logic
  assign r_next = r_reg + 1;
  // output logic
  assign q=r_reg[25:22];
endmodule

```

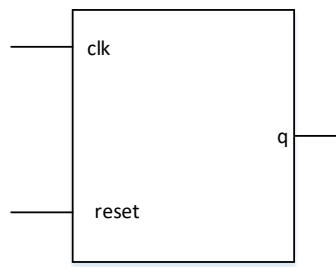
Cấu hình chân FPGA

```

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

```

3.3.2. Thiết kế mạch tạo xung 1Hz



```

module Counter
#(parameter N= 26, M = 50000000)
( input wire clk, reset,
  output wire q
);
// signal declaration

reg [N-1:0] r_reg;
wire [N-1:0] r_next;

// body, register
always @(posedge clk, posedge reset)
if (reset)
  r_reg <= 0;
else
  r_reg=r_next;
// next state logic
assign r_next = (r_reg==M)?0:r_reg + 1;
// output logic
assign q=(r_reg<M/2)?0:1;
endmodule

```

Cáu hình chân FPGA

```

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "q" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

```

3.3.3. Thiết kế mạch tạo 4 xung ngõ ra với tần số lần lượt là 0.1Hz, 1Hz, 10Hz, 100Hz

```

module Counter
#(parameter N= 30, M = 500000000) // 500,000,000 for
0.1Hz
( input wire clk,
  output wire [3:0]q
);
// signal declaration

reg [N-1:0] r_reg01H,r_reg1H,r_reg10H,r_reg100H;

```

```

wire [N-1:0]
r_next01H,r_next1H,r_next10H,r_next100H;

// body, register
always @(posedge clk) begin
    r_reg01H<=r_next01H;
    r_reg1H<=r_next1H;
    r_reg10H<=r_next10H;
    r_reg100H<=r_next100H;
end
// next state logic
assign r_next01H = (r_reg01H==M)?0:r_reg01H + 1;
assign r_next1H = (r_reg1H==M/10)?0:r_reg1H + 1;
assign r_next10H = (r_reg10H==M/100)?0:r_reg10H + 1;
assign r_next100H = (r_reg100H==M/1000)?0:r_reg100H +
1;
// output logic
assign q[0]=(r_reg01H<M/2)?0:1;
assign q[1]=(r_reg1H<M/20)?0:1;
assign q[2]=(r_reg10H<M/200)?0:1;
assign q[3]=(r_reg100H<M/2000)?0:1;
endmodule

```

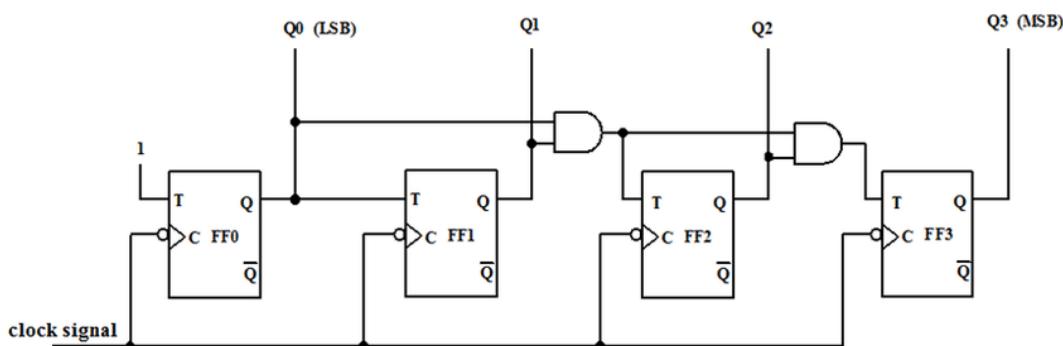
Cáu hình chân FPGA

```

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

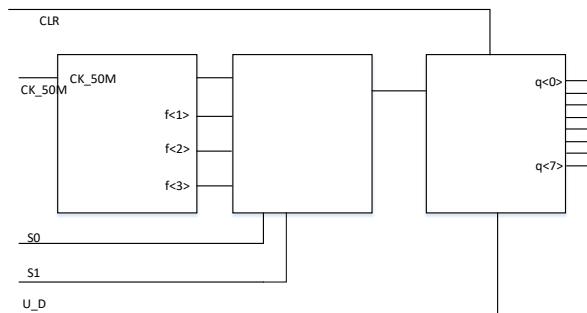
```

3.3.4. Thiết kế mạch đếm đồng bộ, sử dụng phương pháp cài đặt các Flip – Flop. Xung đếm 1Hz được lấy từ mạch chia xung



3.3.5. Thiết kế mạch đếm lên 4 bit như bafi 3.3.4, sử dụng phương pháp thiết kế đồng bộ
3.3.6. Thiết kế mạch đếm lên 8 bit, lựa chọn tần số đếm, lựa chọn đếm lên hoặc đếm xuống

Thiết kế mạch đếm như hình vẽ, mỗi mô đun được thiết kế với 1 tập mã nguồn (.v) khác nhau.



```

module LED_COUNTER(
    input wire clk, reset,
    input wire [1:0] SW,
    input wire UD,
    output wire [7:0] LED
);

// wire declaration
wire [3:0] f;
wire clk_o ;
// module instance
Clock_div clockdivider (clk, f) ;
Mux41 mux4to1 (f,SW,clk_o);
CounterUD counter (clk_o, reset,UD,LED);

endmodule

module Clock_div
    #(parameter N= 30, M = 500000000) // 500,000,000 for
0.1Hz
    ( input wire clk,
      output wire      [3:0]q
    );
    // signal declaration

    reg [N-1:0] r_reg01H,r_reg1H,r_reg10H,r_reg100H;
    wire [N-1:0]
r_next01H,r_next1H,r_next10H,r_next100H;

    // body, register
    always @(posedge clk) begin
        r_reg01H<=r_next01H;
        r_reg1H<=r_next1H;
        r_reg10H<=r_next10H;
        r_reg100H<=r_next100H;
    end
endmodule

```

```

        end
    // next state logic
    assign r_next01H = (r_reg01H==M)?0:r_reg01H + 1;
    assign r_next1H = (r_reg1H==M/10)?0:r_reg1H + 1;
    assign r_next10H = (r_reg10H==M/100)?0:r_reg10H + 1;
    assign r_next100H = (r_reg100H==M/1000)?0:r_reg100H +
1;
    // output logic
    assign q[0]=(r_reg01H<M/2)?0:1;
    assign q[1]=(r_reg1H<M/20)?0:1;
    assign q[2]=(r_reg10H<M/200)?0:1;
    assign q[3]=(r_reg100H<M/2000)?0:1;

endmodule
module Mux41
(   input wire [3:0] clk,
    input wire [1:0] sw,
    output reg      clk_o
);
// signal declaration

// clk_o ;
always @(clk,sw)
case (sw)
0: clk_o = clk[0];
1: clk_o = clk[1];
2: clk_o = clk[2];
3: clk_o = clk[3];
endcase
endmodule
module CounterUD
#(parameter N= 8) // 500,000,000 for 0.1Hz
( input wire clk,reset,ud,
  output wire      [7:0]q
);
// signal declaration
reg [N-1:0] r_reg;
wire [N-1:0] r_next;
// body, register
always @(posedge clk, posedge reset)
if (reset)
r_reg<=0;
else
r_reg<=r_next;

```

```

// next state logic
assign r_next = (ud==1)?r_reg + 1:r_reg - 1;
// output logic
assign q=r_reg;
endmodule

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "Sw<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "Sw<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "UD" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "reset" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;

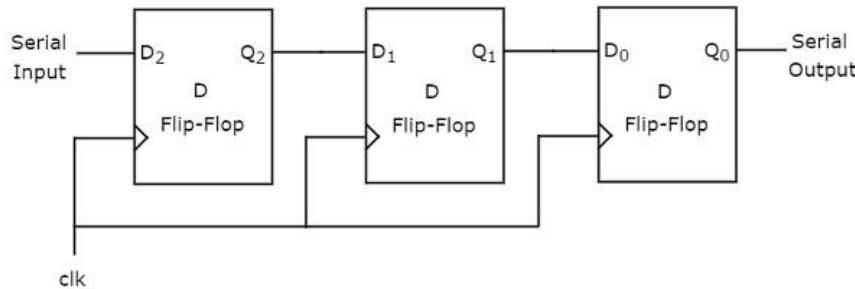
```

3.3.7. Thiết kế mạch đếm lên 8 bit, lựa chọn 8 tần số đếm khác nhau, lựa chọn đếm lên hoặc đếm xuống, có tín hiệu cho phép dừng đếm (Pause), có tín hiệu đảo trạng thái ngõ ra.

3.4. Thanh ghi dịch (shift register)

Thanh ghi là tập hợp các Flip Flop D được điều khiển bởi cùng một xung clock. Thanh ghi dịch (shift register) là một mạch tuần tự mà nội dung bên trong của nó dịch trái, hoặc phải một vị trí mỗi khi có một xung clock.

Thanh ghi dịch vào nối tiếp, ra nối tiếp (SISO) có sơ đồ nguyên lý như sau.



3.4.1. Thiết kế thanh ghi dịch 4 bit vào nối tiếp ra nối tiếp như hình 3. Sử dụng cài đặt các module FF-D

3.4.2. Thiết kế thanh ghi dịch vào nối tiếp ra nối tiếp. Sử dụng phương pháp thiết kế dòng bộ

```

module ShiftRegister(
    input wire clk,
    input wire s_in,
    output wire s_out
);

```

```

wire clk_o ;
// module instance
Clock_1Hz clockdivider (clk, clk_o) ;
Shift_SISO SISO (clk_o,s_in,s_out);
endmodule

module Clock_1Hz
#(parameter N= 26, M = 50000000) // for 50Mhz
  ( input wire clk,
    output wire f
  );
  // signal declaration
  reg [N-1:0] r_reg;
  wire [N-1:0] r_next;
  // body, register
  always @(posedge clk)
    r_reg<=r_next;
  // next state logic
  assign r_next = (r_reg==M)?0:r_reg + 1;
  // output logic
  assign f=(r_reg<M/2)?0:1;

endmodule

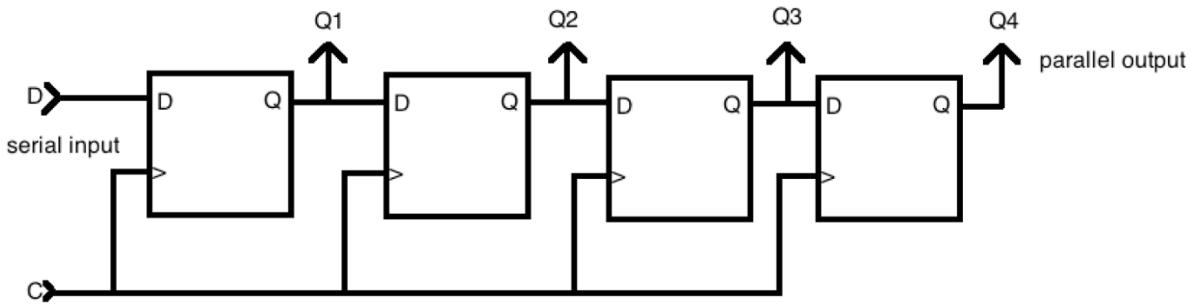
module Shift_SISO
#(parameter N= 4) // 500,000,000 for 0.1Hz
  ( input wire clk,reset,s_in,
    output wire s_out
  );
  // signal declaration
  reg [N-1:0] r_reg;
  wire [N-1:0] r_next;
  // body, register
  always @(posedge clk, posedge reset)
if (reset)
  r_reg<=0;
else
  r_reg<=r_next;
// next state logic
assign r_next = {s_in,r_reg[N-1: 1]};

// output logic
assign s_out= r_reg[0];
endmodule

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "s_out" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "s_in" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ; // SW2

```

3.4.3. Thiết kế mạch ghi dịch vào nối tiếp ra song song bằng cách cài đặt các Flip flop D



3.4.4. Thiết kế mạch ghi dịch vào nối tiếp ra song song bằng phương pháp thiết kế đồng bộ

```

module ShiftRegister(
    input wire clk,
    input wire s_in,
    output wire [7:0] q
);

wire clk_o ;
// module instance
Clock_1Hz clockdivider (clk,clk_o) ;
Shift_SIPO SIPO (clk_o,s_in, q);
endmodule

module Clock_1Hz
#(parameter N= 30, M = 50000000) // for 50Mhz
    ( input wire clk,
      output wire f
    );
    // signal declaration
    reg [N-1:0] r_reg;
    wire [N-1:0] r_next;
    // body, register
    always @(posedge clk)
        r_reg<=r_next;
    // next state logic
    assign r_next = (r_reg>=M)?0:r_reg + 1;
    // output logic
    //assign f=(r_reg>M/2)?1:0;
    assign f=r_reg[25] ;
endmodule

module Shift_SIPO
(

```

```

    input wire clk,s_in,
    output wire      [7:0] q_out
);
// signal declaration
reg [7:0] r_reg;
wire [7:0] r_next;
// body, register
always@(negedge clk)
r_reg<=r_next;
// next state logic
assign r_next = {s_in,r_reg[7:1]};
// output logic
assign q_out= r_reg;
endmodule
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "s_in" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ; // SW2
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

```

3.4.5. Thiết kế mạch điều khiển LED sáng dần từ trái qua phải, tắt dần từ trái qua phải

```

module ShiftRegiter(
    input wire clk,
    output wire [7:0] q );
wire clk_o ;
wire s_in;
// module instance
Clock_1Hz clockdivider (clk,clk_o) ;
Shift_SIPO SIPO (clk_o,s_in, q);
assign s_in = ~q[0] ;
endmodule

```

3.4.6. Thiết kế mạch điều khiển LED sáng dần, tắt dần từ trái sang phải hoặc từ phải sang trái được lựa chọn bởi một switch

```

module ShiftRegiter(
    input wire clk, lr,
    output wire [7:0] q);
wire clk_o ;
wire s_in;
// module instance

```

```

Clock_1Hz clockdivider (clk,clk_o) ;
Shift_SIPO SIPO (clk_o,s_in,lr, q);
assign s_in =(lr == 1)?~q[0]: ~q[7] ;
endmodule

module Shift_SIPO
(
    input wire clk,s_in,lr,
    output wire      [7:0] q_out
);
// signal declaration
reg [7:0] r_reg;
wire [7:0] r_next;
// body, register
always@(negedge clk)
r_reg<=r_next;
// next state logic
assign r_next =(lr==1) ? {s_in,r_reg[7:1]}:{r_reg[6:0],s_in};
// output logic
assign q_out= r_reg;
endmodule

```

3.4.7. Thiết kế mạch điều khiển 1 led chạy từ trái sang phải, từ phải sang trái

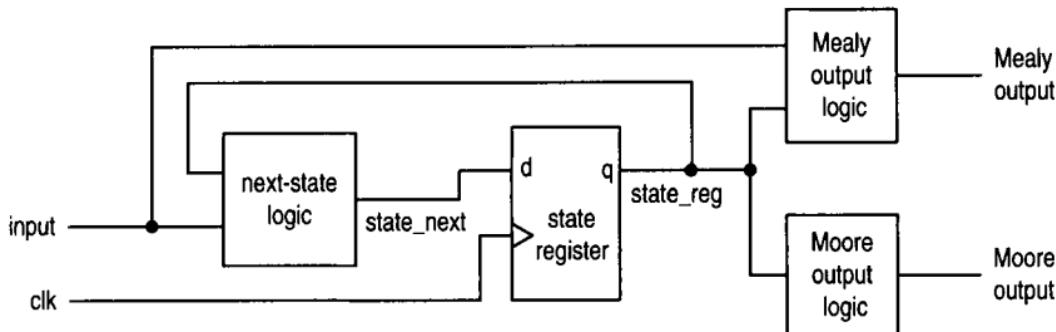
3.4.8. Thiết kế mạch điều khiển 1 Led chạy từ trái sang phải rồi tự động chạy từ phải sang trái, có một switch cho phép đảo trạng thái ngõ ra

3.4.9. Thiết kế mạch gồm 8 led đơn, 4 switch S1, S2, S3, S4

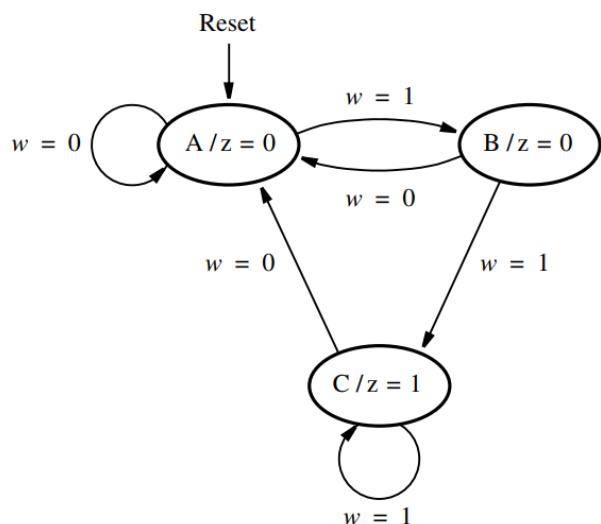
- S1, S2, lựa chọn 1 trong 4 tần số
- S3, S4, lựa chọn mode cho 8 led.
- Mode =1: 8 led chớp tắt,
- Mode =2: 8 led sáng dần, tắt dần,
- Mode = 3: 1 led sáng chạy từ trái sang phải, rồi từ phải sang trái.
- Mode = 4: 8 led sáng dần

3.5. Máy trạng thái (Finite state machine)

Máy trạng thái được sử dụng để thiết kế các hệ thống số mà trong đó trạng thái của hệ thống có thể chuyển đổi giữa nhiều trạng thái được định nghĩa trước. Mô hình máy trạng thái là một mạch tuần tự đồng bộ, trong đó ngõ ra có thể phụ thuộc vào ngõ vào (mô hình Mealy) hoặc ngõ ra chỉ phụ thuộc vào các trạng thái bên trong của hệ thống (Mô hình Moore)



3.5.1. Thiết kế mô hình máy trạng thái 1



Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	C	1

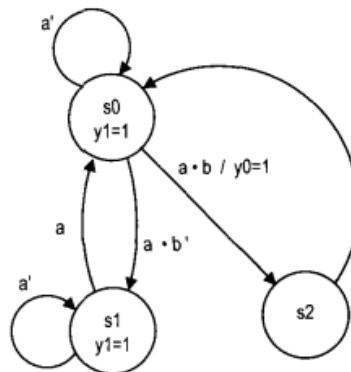
```
module simple (Clock, Resetn, w, z);
input Clock, Resetn, w; output z;
reg [2:1] y, Y;
parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;
// Define the next state combinational circuit
always @(w, y)
case (y)
A: if (w) Y = B;
else Y = A;
B: if (w) Y = C;
```

```

else Y = A;
C: if (w) Y = C;
else Y = A;
default: Y = 2'bxx;
endcase
// Define the sequential block
always @(negedge Resetn, posedge Clock)
if (Resetn == 0) y <= A;
else y <= Y;
// Define output
assign z = (y == C);
Endmodule

```

3.5.2. Thiết kế mô hình máy trạng thái 2



```

module fsm_eg_mult_seg
(
    input wire clk , reset ,
    input wire a , b ,
    output wire yo, y1 );
//symbolic state declaration
localparam [1:0] S0 = 2'b00, S1 = 2'b01 , S2=2'b10;
// signal declaration
reg [1 : 0] state_reg,state_next ;
    // state register
always @ (posedge clk ,posedge reset)
if (reset)
state_reg<=S0;
else
state_reg<=state_next;
//next_state logic

```

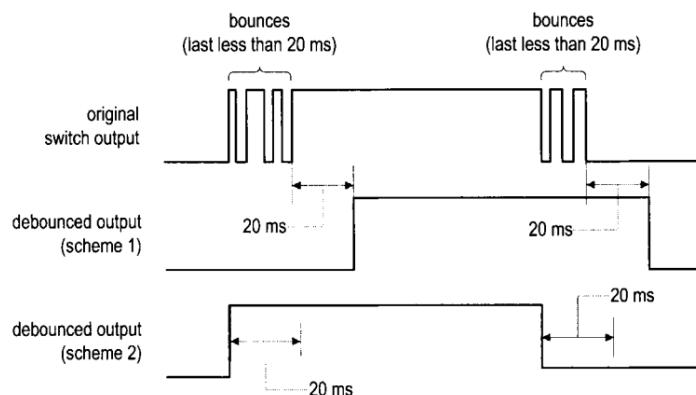
```

always @*
case (state_reg)
S0: if(a)
    if(b)
        state_next=S2;
    else
        state_next=S1;
else
    state_next=S0;
S1: if(a)
    state_next=S0;
    else
        state_next=S1;
S2: state_next=S0;
default: state_next=S0;
endcase
//Moore outputlogic
assign y1=(state_reg==S0) || (state_reg==S1);
//Mealy outputlogic
assign y0=(state_reg==S0)&a&b;
endmodule

```

3.5.3. Chống dội phím nhấn (debouncing circuit)

Các nút nhấn do cấu tạo là các tiếp xúc lá kim loại hoặc tương tự nên khi nhấn tín hiệu chuyển trạng thái một vài lần trước khi ổn định. Điều này làm cho việc xử lý bị sai, giá trị ngõ vào có thể không như mong muốn. Trong bài thực hành này, dao động tại thời điểm nhấn (press) và thả (release) được bỏ qua bằng cách sử dụng mô hình máy trạng thái.



3.5.3.1. Thiết kế mạch đếm xung, hiển thị trên 8 LED, xung ngõ vào được tạo ra từ một nút nhấn

Thiết kế mô hình đọc phím nhấn có chống dội sử dụng mô hình máy trạng thái như sau:



Figure 5.9 State diagram of a debouncing circuit.

```

module FSM(
    input wire reset,clk,btn,
    output wire [7:0] q
);

wire tick;
button btn1(reset, clk,btn,tick);
Counter8bs counter(tick, reset,q);

endmodule
module button(
    input wire reset,clk,btn,
    output reg db
);
localparam [2:0]
    zero = 3'b000,
    wait1_1 = 3'b001,
    wait1_2 = 3'b010,
    wait1_3= 3'b011,
    one = 3'b100,
    wait0_1= 3'b101,
    wait0_2 = 3'b110,
    wait0_3 = 3'b111;
localparam N = 13;

```

```
//signal declaration
reg [N-1:0] q_reg;
wire [N-1:0] q_next;
wire m_tick;
reg [2:0] state_reg, state_next;
// counter to generate 10ms tick
always @(posedge clk)
q_reg <=q_next;
// next state logic
assign q_next = q_reg +1 ;
// output tick
assign m_tick = (q_reg==0)?1'b1:1'b0;
//debouncing FSM
//state register
always @(posedge clk, posedge reset)
if(reset)
    state_reg <= zero;
else
    state_reg<= state_next;
// next state logic and output logic
always @*
begin state_next = state_reg;// default state
db = 1'b0;
case (state_reg)
    zero:
        if(btn)
            state_next = wait1_1;
    wait1_1:
        if (~btn)
            state_next = zero;
        else
            if (m_tick)
                state_next = wait1_2;
    wait1_2:
        if (~btn)
            state_next = zero;
        else
            if (m_tick)
                state_next = wait1_3;
    wait1_3:
        if (~btn)
            state_next = zero;
        else
            if (m_tick)
                state_next = one;
one:
```

```

begin
  db = 1'b1;
  if(~btn)
    state_next = wait0_1;
  end
wait0_1:
begin
  db = 1'b1;
  if (btn)
    state_next = one;
  else
    if (m_tick)
      state_next = wait0_2;
  end
wait0_2:
begin
  db = 1'b1;
  if (btn)
    state_next = one;
  else
    if (m_tick)
      state_next = wait0_3;
  end
wait0_3:
begin
  db = 1'b1;
  if(btn)
    state_next = one;
  else
    if (m_tick)
      state_next = zero;
  end
default: state_next = zero;
endcase
end
endmodule

module Counter8bs
#(parameter N= 8) // 500,000,000 for 0.1Hz
(
  input wire clk,reset,
  output wire [7:0]q
);
// signal declaration

reg [N-1:0] r_reg;
wire [N-1:0] r_next;
// body, register

```

```

    always @(posedge clk, posedge reset)
        if (reset)
            r_reg<=0;
        else
            r_reg<=r_next;
    // next state logic
    assign r_next = r_reg + 1;
    // output logic
    assign q=r_reg;
endmodule

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
NET "btn" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;

NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

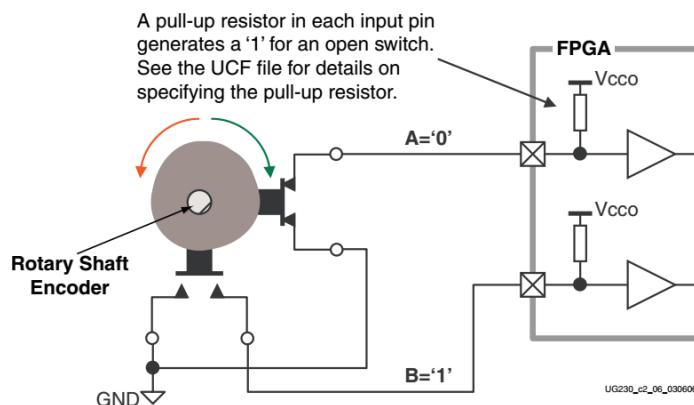
```

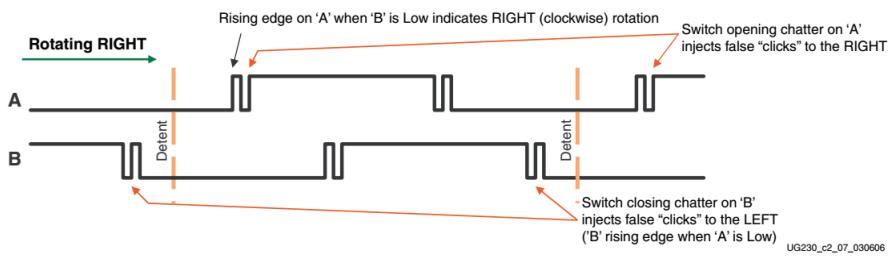
3.5.3.2. Thiết kế mạch đếm lên, đếm xuống với tần số 1Hz, được điều khiển bởi một nút nhấn.

3.5.3.3. Thiết kế mạch đếm xung 1Hz, đếm lên / xuống, điều khiển bằng một nút nhấn, có nút PAUSE, có nút SPEED để thay đổi tốc độ đếm (4 tốc độ khác nhau)

3.6. Công tắc xoay (Rotary switch)

Các công tắc xoay cho phép điều khiển hệ thống linh hoạt hơn. Các công tắc xoay cơ bản được cấu tạo từ 2 switch như hình bên dưới. Xác định chiều xoay dựa vào dạng tín hiệu ở 2 ngõ ra.





3.6.1.Thiết kế mạch đếm lên, đếm xuống được điều khiển bởi công tắc xoay, tần số đếm 1hz

```

module FSM(
    input wire reset,clk,ROT_A,ROT_B,
    output wire [7:0] q
);
wire ticka,tickb,pulse;
reg dir;
always @(posedge ticka)
begin
    if (tickb==0)
        dir =1;
    else
        dir =0;
end
button btn1(clk,reset,ROT_A,ticka);
button btn2(clk,reset,ROT_B,tickb);
Counter8bs counter(pulse, reset,dir,q);

assign pulse = ticka&tickb ;
endmodule

module Counter8bs
#(parameter N= 8) // 500,000,000 for 0.1Hz
( input wire clk,reset,dir,
  output wire [7:0]q
);
// signal declaration
reg [N-1:0] r_reg;
wire [N-1:0] r_next;
// body, register
always @(posedge clk, posedge reset)
if (reset)
r_reg<=0;
else
r_reg<=r_next;
// next state logic
assign r_next =(dir==1)?r_reg + 1:r_reg - 1;

```

```

// output logic
assign q=r_reg;
endmodule
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
//NET "btn" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "ROT_A" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP ;
NET "ROT_B" LOC = "G18" | IOSTANDARD = LVTTL | PULLUP ;

```

3.6.2. Thiết kế mạch đếm lên, đếm xuống, được điều khiển bởi 1 nút nhấn, tần số đếm tăng hay giảm được điều khiển bởi công tắc xoay

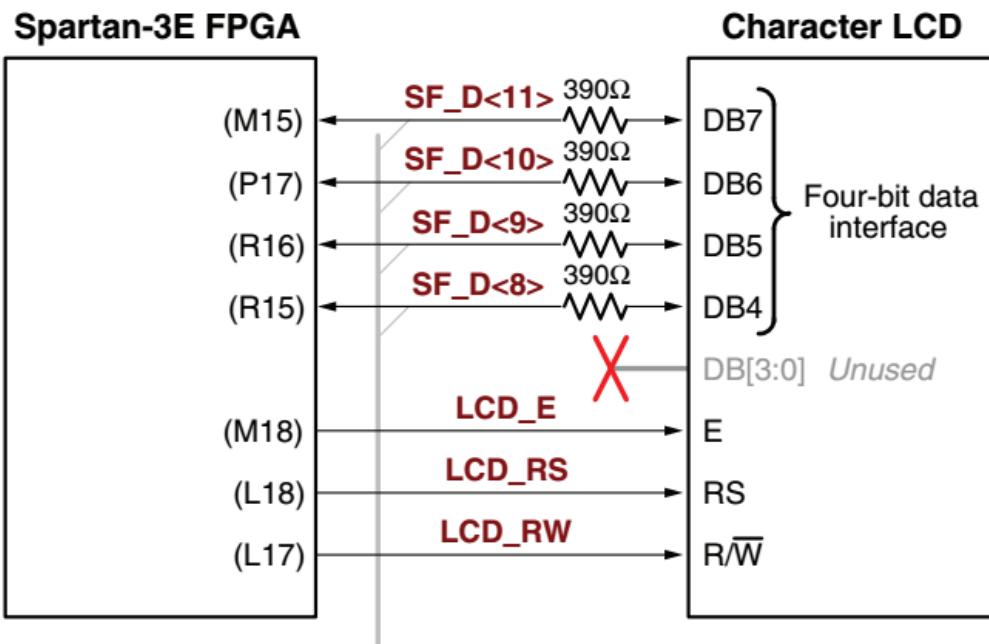
3.7. LCD

3.7.1. Giới thiệu

LCD là thiết bị hiển thị phổ biến trong các hệ thống số. LCD cho phép hiển thị các thông tin dạng văn bản phong phú hơn. Trong phần thực hành này, chúng ta thiết kế các mô đun điều khiển các LCD cho phép hiển thị các thông tin dạng chuỗi và số. LCD sử dụng trong phần thực hành này có 2 dòng, mỗi dòng có 20 ký tự.



LCD được điều khiển thông qua các tín hiệu điều khiển (E, RS, R/W) và các đường dữ liệu. Có thể sử dụng 8 đường dữ liệu hoặc chỉ sử dụng 4 đường dữ liệu. Kết nối FPGA và LCD được trình bày như hình bên dưới



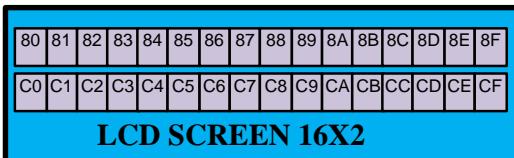
Signal Name	FPGA Pin	Function	
SF_D<11>	M15	Data bit DB7	Shared with StrataFlash pins SF_D<11:8>
SF_D<10>	P17	Data bit DB6	
SF_D<9>	R16	Data bit DB5	
SF_D<8>	R15	Data bit DB4	
LCD_E	M18	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	
LCD_RS	L18	Register Select 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations	
LCD_RW	L17	Read/Write Control 0: WRITE, LCD accepts data 1: READ, LCD presents data	

- Địa chỉ vùng nhớ DD RAM của LCD.
Vùng nhớ DD RAM chứa dữ liệu sẽ được hiển thị trên LCD

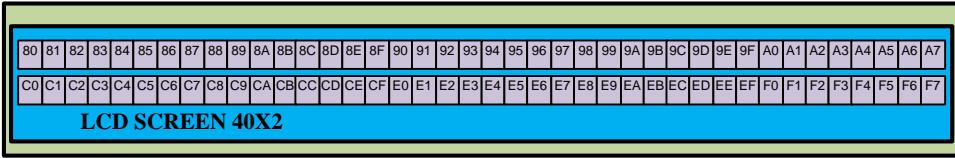
Character Display Addresses																Undisplayed Addresses			
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	...	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	...	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40

Figure 5-3: DD RAM Hexadecimal Addresses (No Display Shifting)

- Địa chỉ các hàng trên LCD



DDRAM MEMORY



- Tập lên cơ bản điều khiển LCD

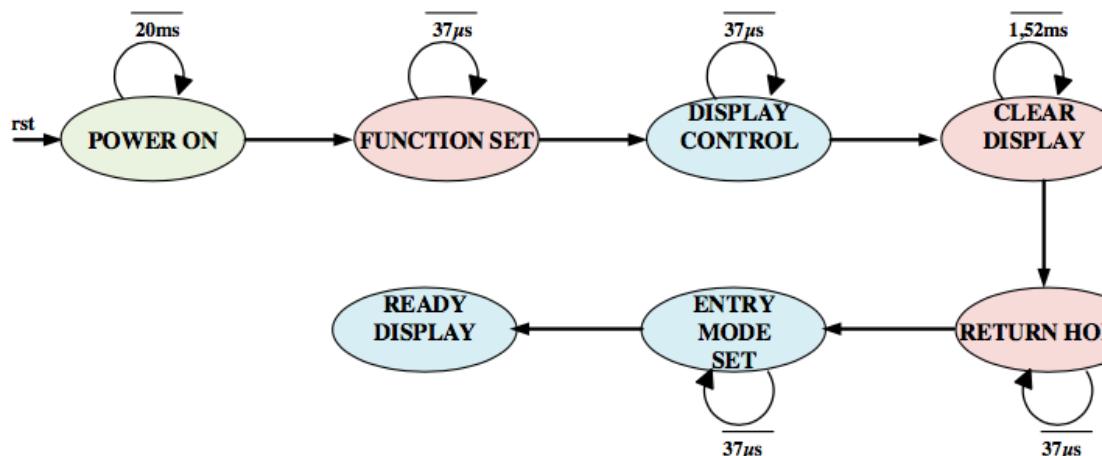
LỆNH	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Mô tả	clock
(1) NOP	0	0	0	0	0	0	0	0	0	0	No operation	0
(2) Clear display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM and set DDRAM address counter to "00H".	1.52ms
(3) Cursor home	0	0	0	0	0	0	0	0	1	0	Sets address counter to zero, returns shifted display to original position. DDRAM contents remain unchanged.	39μs
(4) Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction, and specifies automatic shift.	39μs
(5) Display control	0	0	0	0	0	0	1	D	C	B	Turns display (D), cursor on/off (C) or cursor blinking (B).	39μs
(6) Cursor display shift	0	0	0	0	0	1	S/C	R/L	0	0	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	39μs

(7) Function set	0	0	0	0	1	DL	N	F	0	0	Set interface data length (DL : 4-bit/8-bit), numbers of display line (N : 1-line/2-line), display font type(F : 5 X 8 dots/ 5 X 11 dots)	39μs
(8) Set CGRAM addr	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter	39μs
(9) Set DDRAM addr	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter	39μs
(10) Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0μs
(11) Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data to CGRAM or DDRAM	43μs
(12) Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from CGRAM or DDRAM	43μs

➤ Bảng mã ASCII

	Upper 4 Bits Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			Ø Ø P ^ P						- - T E x p							
xxxx0001	(2)			! 1 A Q a q						¤ ¤ T C ä q							
xxxx0010	(3)			" 2 B R b r						‘ ‘ I T X e 0							
xxxx0011	(4)			# 3 C S c s						♪ ♪ U T e e w							
xxxx0100	(5)			\$ 4 D T d t						~ ~ I T b p o							
xxxx0101	(6)			% 5 E U e u						• • O N U s 0							
xxxx0110	(7)			& 6 F U f v						ヲ カ ニ ベ p z							
xxxx0111	(8)			' 7 G W g w						ア キ マ ラ g n							
xxxx1000	(1)			< 8 H X h x						イ ウ ネ リ j x							
xxxx1001	(2)) 9 I Y i y						ウ ケ ノ ル “ y							
xxxx1010	(3)			* ; J Z j z						エ コ ハ レ j k							
xxxx1011	(4)			+ ; K C k {						オ サ ハ ロ * n							
xxxx1100	(5)			, < L F l l						フ シ フ ワ f n							
xxxx1101	(6)			- = M J m }						ユ ズ ヘ ニ k +							
xxxx1110	(7)			. > N ^ n *						ミ セ ホ ^ n							
xxxx1111	(8)			/ ? O _ o *						シ リ ブ ^ o							

- Các bước cấu hình và điều khiển LCD



3.7.2. Điều khiển LCD hiển thị chuỗi ký tự trên 2 hàng

```

module LCD(
    clk,
    chars,
    Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7);
    input          clk;
    input [256:0]   chars;
    output         Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6,
    Lcd_7;

    wire [256:0]   chars;
    reg           Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7;
    reg [5:0]      Lcd_code;
    reg [1:0]      write = 2'b10; // write code has 10 for rs rw
    // delays
    reg [1:0]      before_delay = 3;      // time before on
    reg [3:0]      on_delay = 13;        // time on
    reg [23:0]     off_delay = 750_001; // time off
    // states and counters
    reg [6:0]      Cs = 0;
    reg [19:0]     count = 0;
    reg [1:0]      delay_state = 0;
    // character data
    reg [256:0]    chars_hold = "";
    wire [3:0]     chars_data [63:0]; // array of characters
    // redirects characters data to an array
    generate
        genvar i;
        for (i = 64; i > 0; i = i-1)
            begin : for_name
                assign chars_data[64-i] = chars_hold[i*4-
1:i*4-4];
            end
        endgenerate

        always @ (posedge clk) begin

            // store character data
            if (Cs == 10 && count == 0) begin
                chars_hold <= chars;
            end

            // set time when enable is off
            if (Cs < 3) begin

```

```

        case (Cs)
            0: off_delay <= 750_001;      // 15ms delay
            1: off_delay <= 250_001;      // 5ms delay
            2: off_delay <= 5_001;          // 0.1ms
delay
        endcase
    end else begin
        if (Cs > 12) begin
            off_delay <= 2_001; // 40us delay
        end else begin
            off_delay <= 250_001; // 5ms delay
        end
    end

// delays during each state
if (Cs < 80) begin
    case (delay_state)
        0: begin
            // enable is off
            lcd_e <= 0;

{lcd_rs,lcd_rw,lcd_7,lcd_6,lcd_5,lcd_4} <= lcd_code;
            if (count == off_delay) begin
                count <= 0;
                delay_state <= delay_state + 1;
            end else begin
                count <= count + 1;
            end
        end
        1: begin
            // data set before enable is on
            lcd_e <= 0;
            if (count == before_delay) begin
                count <= 0;
                delay_state <= delay_state + 1;
            end else begin
                count <= count + 1;
            end
        end
        2: begin
            // enable on
            lcd_e <= 1;
            if (count == on_delay) begin
                count <= 0;
                delay_state <= delay_state + 1;
            end else begin

```

```

        count <= count + 1;
    end
end
3: begin
    // enable off with data set
    Lcd_e <= 0;
    if (count == before_delay) begin
        count <= 0;
        delay_state <= 0;
        Cs <= Cs + 1;           //      next
    case
        end else begin
            count <= count + 1;
        end
    end
endcase
end
//-----Cs = 0 - 11 -----
-----
// set Lcd_code
if (Cs < 12) begin
    // initialize LCD
    case (Cs)
        0: Lcd_code <= 6'h03;          // power-on
initialization
        1: Lcd_code <= 6'h03;
        2: Lcd_code <= 6'h03;
        3: Lcd_code <= 6'h02;
        4: Lcd_code <= 6'h02;          // function set
        5: Lcd_code <= 6'h08;
        6: Lcd_code <= 6'h00;          // entry mode
set
        7: Lcd_code <= 6'h06;
        8: Lcd_code <= 6'h00;          // display on/off
control
        9: Lcd_code <= 6'h0C;
        10:Lcd_code <= 6'h00;          // display clear
        11:Lcd_code <= 6'h01;
        default: Lcd_code <= 6'h10;
    endcase
end else begin
//-----Cs = 44-----
-----
// set character data to Lcd_code
if (Cs == 44) begin                // change address
at end of first line

```

```

        Lcd_code <= {2'b00, 4'b1100}; // 0100
0000 address change

    end else if (Cs == 45) begin
        Lcd_code <= {2'b00, 4'b0000};
    end else begin
        if (Cs < 44) begin
            Lcd_code <= {write, chars_data[Cs-12]};
        end else begin
            Lcd_code <= {write, chars_data[Cs-14]};
        end
    end

end
// hold and Loop back
if (Cs == 80) begin
    Lcd_e <= 0;
    if (count == off_delay) begin
        Cs <= 10;
        count <= 0;
    end else begin
        count <= count + 1;
    end
end
end
endmodule
module LCDtest(
    input wire clk,
    output wire Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7);
    wire [256:0] chars;
    // module installation
    LCD Lcd( clk,
        chars,
        Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7);
    // define data memory
    assign chars[255:128] = " HCMUTE ";
    assign chars[127:0] = " 05 - 05 2019 ";
    //assign chars[7:0] = {4'b0011, I3, I2, I1, I0};

endmodule

# clock
NET "clk" LOC = C9 | IOSTANDARD = LVCMOS33 ;
NET "clk" PERIOD = 20.0ns HIGH 50%;
```

```

# LCD control inputs
NET "lcd_e"      LOC = M18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_rs"     LOC = L18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_rw"     LOC = L17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;

# The LCD four-bit data interface
NET "lcd_4"       LOC = R15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_5"       LOC = R16 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_6"       LOC = P17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_7"       LOC = M15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
= SLOW ;

```

3.7.3. Điều khiển LCD hiển thị chuỗi và số

```

module LCD(
    clk,
    chars,
    Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7);

    input          clk;
    input [256:0]   chars;
    output         Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6,
    Lcd_7;

    wire [256:0]   chars;
    reg           Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7;
    reg [5:0]     Lcd_code;
    reg [1:0]     write = 2'b10; // write code has 10 for rs rw
    // delays
    reg [1:0]     before_delay = 3;      // time before on
    reg [3:0]     on_delay = 13;        // time on
    reg [23:0]    off_delay = 750_001; // time off
    // states and counters
    reg [6:0]     Cs = 0;
    reg [19:0]    count = 0;
    reg [1:0]     delay_state = 0;
    // character data
    reg [256:0]   chars_hold = "";
    wire [3:0]    chars_data [63:0];   // array of characters
                                         ";

    // redirects characters data to an array
    generate
        genvar i;
        for (i = 64; i > 0; i = i-1)
            begin : for_name

```

```

        assign chars_data[64-i] = chars_hold[i*4-1:i*4-
4];
      end
endgenerate

always @ (posedge clk) begin

  // store character data
  if (Cs == 10 && count == 0) begin
    chars_hold <= chars;
  end

  // set time when enable is off
  if (Cs < 3) begin
    case (Cs)
      0: off_delay <= 750_001; // 15ms delay
      1: off_delay <= 250_001; // 5ms delay
      2: off_delay <= 5_001; // 0.1ms delay
    endcase
  end else begin
    if (Cs > 12) begin
      off_delay <= 2_001; // 40us delay
    end else begin
      off_delay <= 250_001; // 5ms delay
    end
  end
end

// delays during each state
if (Cs < 80) begin
  case (delay_state)
    0: begin
      // enable is off
      lcd_e <= 0;
      {lcd_rs,lcd_rw,lcd_7,lcd_6,lcd_5,lcd_4} <=
LCD_code;
      if (count == off_delay) begin
        count <= 0;
        delay_state <= delay_state + 1;
      end else begin
        count <= count + 1;
      end
    end
    1: begin
      // data set before enable is on
      lcd_e <= 0;
      if (count == before_delay) begin

```

```

        count <= 0;
        delay_state <= delay_state + 1;
    end else begin
        count <= count + 1;
    end
end
2: begin
    // enable on
    Lcd_e <= 1;
    if (count == on_delay) begin
        count <= 0;
        delay_state <= delay_state + 1;
    end else begin
        count <= count + 1;
    end
end
3: begin
    // enable off with data set
    Lcd_e <= 0;
    if (count == before_delay) begin
        count <= 0;
        delay_state <= 0;
        Cs <= Cs + 1;           // next case
    end else begin
        count <= count + 1;
    end
end
endcase
end

// set Lcd_code
if (Cs < 12) begin
    // initialize LCD
    case (Cs)
        0: Lcd_code <= 6'h03;          // power-on
initialization
        1: Lcd_code <= 6'h03;
        2: Lcd_code <= 6'h03;
        3: Lcd_code <= 6'h02;
        4: Lcd_code <= 6'h02;          // function set
        5: Lcd_code <= 6'h08;
        6: Lcd_code <= 6'h00;          // entry mode set
        7: Lcd_code <= 6'h06;
        8: Lcd_code <= 6'h00;          // display on/off
control
        9: Lcd_code <= 6'h0C;

```

```

    10:Lcd_code <= 6'h00;           // display clear
    11:Lcd_code <= 6'h01;
    default: Lcd_code <= 6'h10;
  endcase
end else begin
  // set character data to Lcd_code
  if (Cs == 44) begin           // change address at
end of first line
  Lcd_code <= {2'b00, 4'b1100}; // 0100 0000
address change
end else if (Cs == 45) begin
  Lcd_code <= {2'b00, 4'b0000};
end else begin
  if (Cs < 44) begin
    Lcd_code <= {write, chars_data[Cs-12]};
  end else begin
    Lcd_code <= {write, chars_data[Cs-14]};
  end
end
end

// hold and Loop back
if (Cs == 78) begin
  Lcd_e <= 0;
  if (count == off_delay) begin
    Cs          <= 10;
    count       <= 0;
  end else begin
    count <= count + 1;
  end
end
end
endmodule
module LCDtest(
  clk,
  I3, I2, I1, I0,
  Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7
);
  input      clk; // for 50Mhz Clock
  input I3, I2, I1, I0;
  output     Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7;

  wire Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7;
  wire I3, I2, I1, I0;
  wire [256:0] chars;

```

```

// module installation
LCD  Lcd( clk,
           chars,
           Lcd_rs, Lcd_rw, Lcd_e, Lcd_4, Lcd_5, Lcd_6, Lcd_7);

// define data memory
assign chars[255:8] = "Hello World!!!!!"      Value:   "";
assign chars[7:0] = {4'b0011, I3, I2, I1, I0};
endmodule

# clock
NET "clk" LOC = C9 | IOSTANDARD = LVCMOS33 ;
NET "clk" PERIOD = 20.0ns HIGH 50%;

# LCD control inputs
NET "lcd_e"      LOC = M18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW
;
NET "lcd_rs"     LOC = L18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW
;
NET "lcd_rw"     LOC = L17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW
;

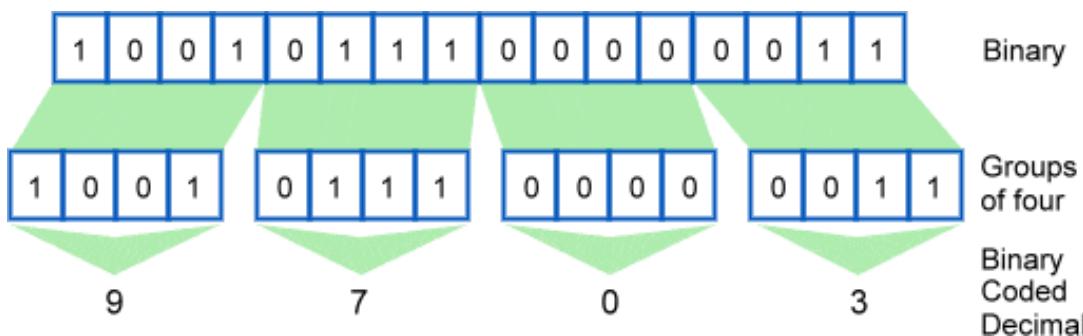
# The LCD four-bit data interface
NET "lcd_4"       LOC = R15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW
;
NET "lcd_5"       LOC = R16 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW
;
NET "lcd_6"       LOC = P17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW
;
NET "lcd_7"       LOC = M15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW
;

# inputs
NET "I0"        LOC = L13 | IOSTANDARD = LVTTL;
NET "I1"        LOC = L14 | IOSTANDARD = LVTTL;
NET "I2"        LOC = H18 | IOSTANDARD = LVTTL;
NET "I3"        LOC = N17 | IOSTANDARD = LVTTL;

```

3.7.4. Thiết kế mạch đếm và hiển thị giá trị đếm lên LCD

Giá trị đếm nhị phân hoặc hệ thập lục phân được giải mã sang giá trị BCD tương ứng cho đơn vị chục và trăm. Phương pháp chuyển đổi số nhị phân sang số BCD được minh họa như sau



- Giải thuật chuyển đổi từ số nhị phân sang số BCD

Operation	Tens	Units	Binary
HEX			E
Start			1 1 1 0
Shift 1		1	1 1 0
Shift 2		1 1	1 0
Shift 3		1 1 1	0
Add 3		1 0 1 0	0
Shift 4	1	0 1 0 0	
BCD	1	4	

- Dịch số nhị phân sang trái 1 bit
- Nếu số nhị phân là 8 bit, số BCD tương ứng bao gồm hàng trăm, chục, đơn vị
- Nếu số nhị phân trong nhóm số BCD nào hơn hơn 5 thì cộng nó với 3.
- Lặp lại quá trình dịch

- Mạch chuyển đổi từ HEX sang BCD

Module chuyển đổi từ HEX sang BCD

```
module add3(in,out);
input [3:0] in;
output [3:0] out;
reg [3:0] out;
always @ (in)
case (in)
4'b0000: out <= 4'b0000;
4'b0001: out <= 4'b0001;
```

```

4'b0010: out <= 4'b0010;
4'b0011: out <= 4'b0011;
4'b0100: out <= 4'b0100;
4'b0101: out <= 4'b1000;
4'b0110: out <= 4'b1001;
4'b0111: out <= 4'b1010;
4'b1000: out <= 4'b1011;
4'b1001: out <= 4'b1100;
default: out <= 4'b0000;
endcase
endmodule
module HEX2BCD(A, ONES, TENS, HUNDREDS);
input [7:0] A;
output [3:0] ONES, TENS;
output [1:0] HUNDREDS;
wire [3:0] c1,c2,c3,c4,c5,c6,c7;
wire [3:0] d1,d2,d3,d4,d5,d6,d7;
assign d1 = {1'b0,A[7:5]};
assign d2 = {c1[2:0],A[4]};
assign d3 = {c2[2:0],A[3]};
assign d4 = {c3[2:0],A[2]};
assign d5 = {c4[2:0],A[1]};
assign d6 = {1'b0,c1[3],c2[3],c3[3]};
assign d7 = {c6[2:0],c4[3]};
add3 m1(d1,c1);
add3 m2(d2,c2);
add3 m3(d3,c3);
add3 m4(d4,c4);
add3 m5(d5,c5);
add3 m6(d6,c6);
add3 m7(d7,c7);
assign ONES = {c5[2:0],A[0]};
assign TENS = {c7[2:0],c5[3]};
assign HUNDREDS = {c6[3],c7[3]};
endmodule

```

3.7.5. Thiết kế mạch điều khiển LCD, hiển thị giá trị giờ, phút, giây trên hàng thứ 2

3.7.6. Thiết kế mạch điều khiển đèn giao thông

Thiết kế mạch điều khiển đèn giao thông, ngõ vào tần số 50Mhz, ngõ ra bao gồm 6 led đơn (X1 V1 DD1 – X1 V2 DD2)

```

module TraficLight(
input wire clk, rs,
output wire X1, V1, D1, X2, V2, D2
);

```

```
/*
S0: 15
S1 : 5
S2: 20

*/
localparam [1:0] S0 = 2'b00 , S1 = 2'b01 , S2 = 2'b10, S3 =
2'b11;
reg [1 : 0] state_reg,state_next ;
reg [7:0] count =15;
// next state generating
always @ (posedge clk, posedge rs)
begin
if (rs)
state_reg <=S0;
else
state_reg <= state_next ;
//count =count+1 ;

end
always @(posedge clk)
case (state_reg)
S0: if (count == 0)
begin state_next = S1;count = 5;end
else count = count -1 ;

S1: if (count==0)
begin state_next = S2;count = 15;end
else count = count -1 ;

S2: if (count==0)
begin state_next = S3; count = 5 ; end
else count = count -1 ;
S3: if (count==0)
begin
state_next = S0; count = 15;
end
else count = count -1 ;
default:state_next = S0;
endcase
// output logic
assign X1 =(state_reg == S0);
assign V1 =(state_reg == S1);
assign D1 =(state_reg == S2)|| (state_reg == S3);
```

```

assign X2 =(state_reg == S2);
assign V2 =(state_reg == S3);
assign D2 =(state_reg == S0)|| (state_reg == S1);
endmodule

```

3.7.7. Thiết kế mạch điều khiển đèn giao thông, có bộ đếm thời gian, đếm xuống

```

module TraficLight(
input wire clk, rs,
output wire X1, V1, D1, X2, V2, D2,
output wire [7:0] counter1, counter2
);

/*
S0: 15
S1 : 5
S2: 20

*/
localparam [1:0] S0 = 2'b00 , S1 = 2'b01 , S2 = 2'b10, S3 = 2'b11;
reg [1 : 0] state_reg,state_next ;
reg [7:0] count =15;
// next state generating
always @*
begin
if (rs)
state_reg <=S0;
else
state_reg <= state_next ;
//count =count+1 ;

end
always @(posedge clk)
case (state_reg)
S0: if (count == 0)
begin state_next = S1;count = 5;end
else count = count -1 ;

S1: if (count==0)
begin state_next = S2;count = 15;end
else count = count -1 ;

S2: if (count==0)
begin state_next = S3; count = 5 ; end
else count = count -1 ;

```

```
S3: if (count==0)
begin
    state_next = S0; count = 15;
end
else count = count -1 ;
default:state_next = S0;
endcase
// output logic
assign X1 =(state_reg == S0);
assign V1 =(state_reg == S1);
assign D1 =(state_reg == S2)|| (state_reg == S3);

assign X2 =(state_reg == S2);
assign V2 =(state_reg == S3);
assign D2 =(state_reg == S0)|| (state_reg == S1);

assign counter1 = (state_reg == S2)?count+5:count ;
assign counter2 = (state_reg == S0)?count+5:count;

endmodule
```

3.7.8. Thiết kế mạch điều khiển đèn giao thông, có bộ đếm thời gian, đếm xuống hiển thị trên LCD

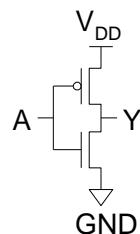
PHẦN II. THIẾT KẾ MẠCH TÍCH HỢP SỐ

1. Giới thiệu

Trong những năm gần đây, ngành thiết kế mạch tích hợp ở Việt Nam phát triển mạnh và ngày càng thu hút nguồn nhân lực có trình độ về thiết kế vi mạch tích hợp. Các công thiết thiết kế vi mạch dần dần hình thành và phát triển ở Việt Nam. Đây là một lĩnh vực đặc thù và mang tính chất quan trọng đóng góp cho sự phát triển của kỹ thuật công nghệ. Thiết kế mạch tích hợp bao gồm thiết kế mạch tích hợp số (Digital circuit design), thiết kế mạch tích hợp tương tự (Analog circuit design) và thiết kế mạch thích hợp bao gồm mạch số và mạch tương tự kết hợp (Mixed signal Integrated circuit design). Trong thiết kế mạch số thông thường, chúng ta thường bắt đầu với bảng trạng thái của hệ thống và sau đó sử dụng các phương pháp tối ưu để đưa ra phương trình cuối cùng biểu diễn cho hệ thống. Từ phương trình thể hiện mối liên hệ giữa các ngõ ra và các ngõ vào, người thiết kế lựa chọn các công logic, các flip-flop tương ứng. Phương pháp thiết kế này thường được gọi là thiết kế mạch số hoặc thiết kế hệ thống số. Trong thiết kế mạch tích hợp, chúng ta không sử dụng các vi mạch được chế tạo như các công logic, các flip-flop mà chúng ta thiết kế từ đơn vị nhỏ nhất là transistor. Trong thiết kế mạch tích hợp, transistor là linh kiện chính để tạo nên các mạch tích hợp số và mạch tích hợp tương tự. Transistor được sử dụng trong thiết kế mạch thích hợp là CMOS transistor (Complementary-Metal-Oxide Semiconductor). Các CMOS phụ thuộc vào công nghệ bán dẫn cụ thể của từng nhà sản xuất. Do đó, khi thiết kế mạch tích hợp, chúng ta phải xác định trước công nghệ mà chúng ta sử dụng cho thiết kế. Trong tài liệu này chúng ta thực hành thiết kế mạch tích hợp số cơ bản sử dụng công nghệ CMOS 0.13 µm của Samsung. Phần mềm thiết kế được sử dụng là Cadence, trong đó chủ yếu sử dụng Candence Spectre để thiết kế và thực hiện mô phỏng thiết kế, cũng như tính toán các thông số cho thiết kế. Tài liệu hướng dẫn thực hành thiết kế mạch cổng đảo (inverter), cổng NAND, NOR, Flip-Flop D, sử dụng CMOS công nghệ Samsung 0.13 µm. Sử dụng phần mềm để phân tích các thông số của hệ thống như công suất tiêu thụ, độ trễ, chức năng logic của mạch tích hợp.

2. Thiết kế mạch cổng đảo (inverter) sử dụng CMOS công nghệ Samsung 0.13µm

Mạch đảo (Inverter) hay còn gọi là cổng đảo (NOT) là thiết kế cơ bản nhất sử dụng 2 transistor CMOS bao gồm 1 pMOS và 1 nMOS như hình 2.1



Hình 2.1. Sơ đồ nguyên lý cổng đảo sử dụng CMOS

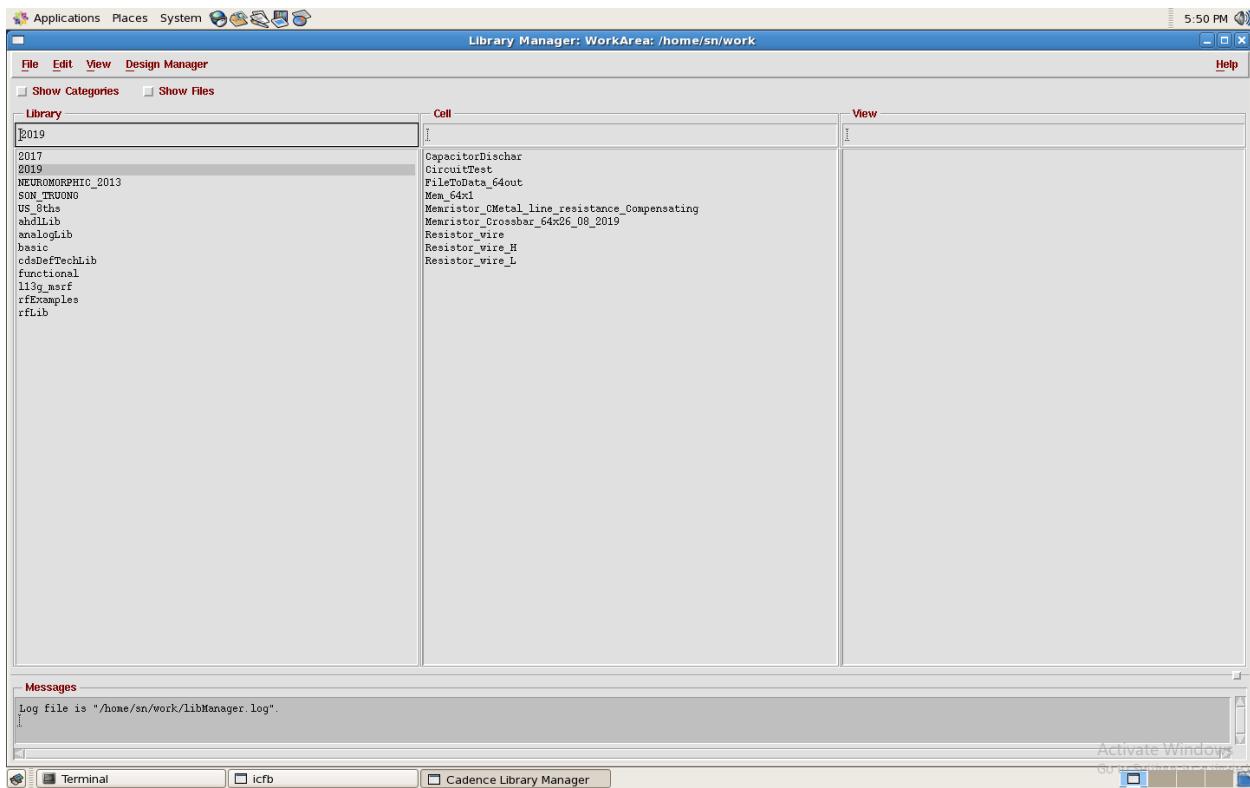
2.1.Thiết kế và phân tích đặc tính cổng đảo

Trong hình 2.1 nMOS có chức năng kéo ngõ ra xuống Vss và pMOS có chức năng kéo ngõ ra lên Vdd. Trong thiết kế mạch tích hợp số, Vdd là điện áp dương, điện áp cung cấp, Vss là điện áp âm, hoặc có thể là 0V (GND). Công nghệ CMOS 0.13 µm của Samsung có thể sử dụng điện áp cung cấp từ 1 đến 1.2V.

Phần mềm Cadence được cài đặt trên hệ điều hành Centos. Đăng nhập hệ điều hành Centos, sử dụng tài khoản sn và mật khẩu sn. Mở cửa sổ lệnh (Terminal). Di chuyển đến thư mục làm việc mà khởi động phần mềm Candence.

```
[asic:/home/sn]#cd work/  
[asic:/home/sn]#icfb &
```

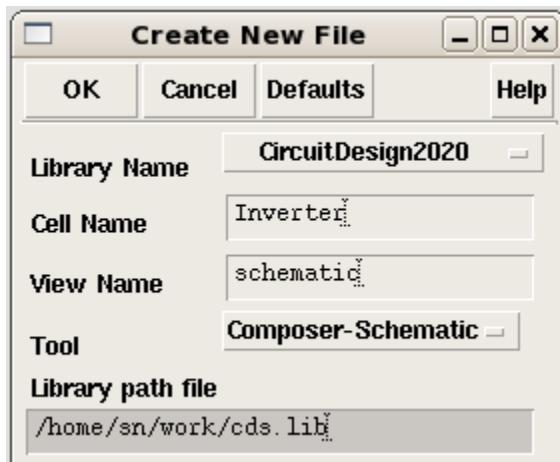
➤ Giao diện trình quản lý thư viện của Cadence



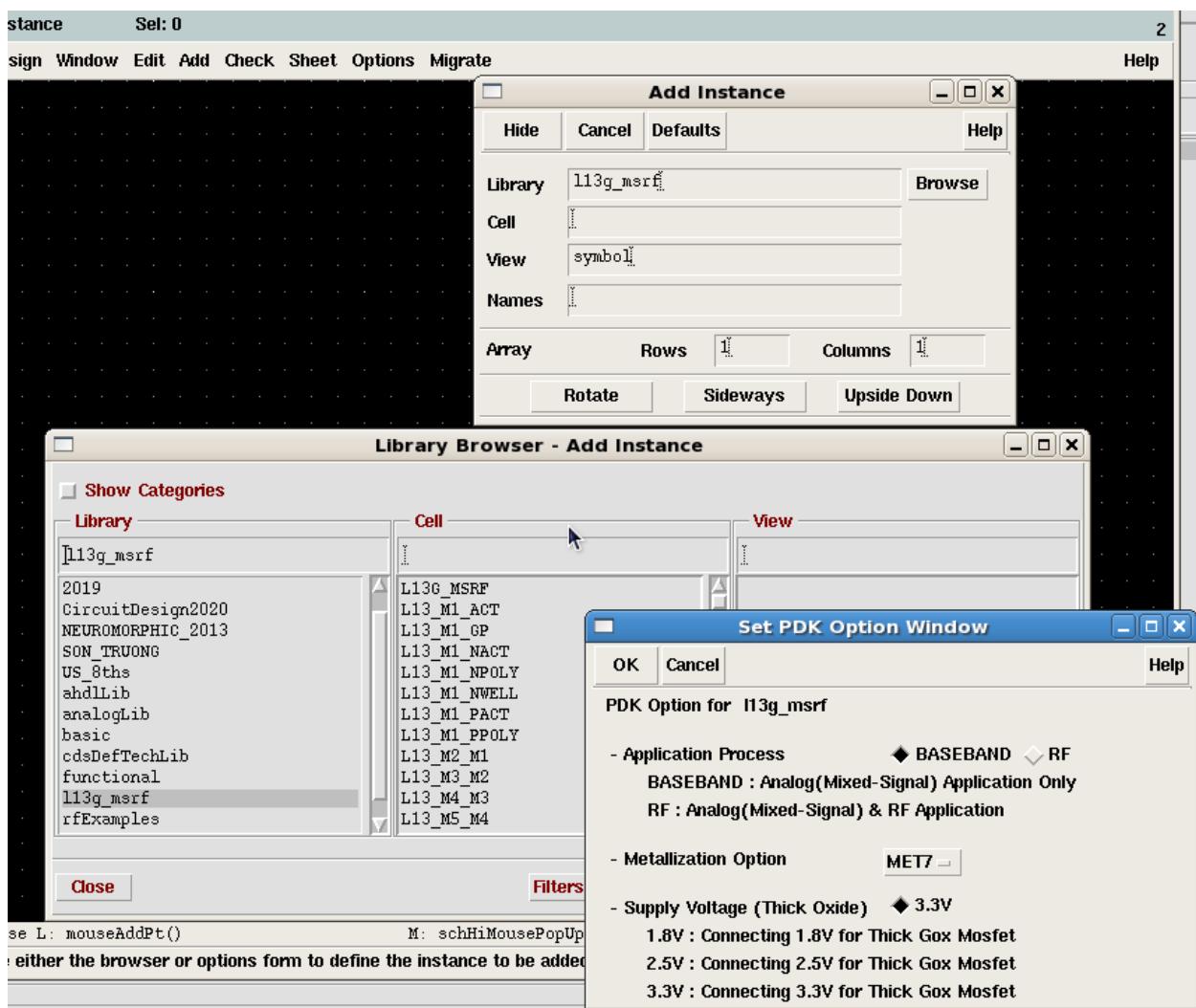
Tạo thư viện để quản lý các thiết kế. Chọn File → New → Library, đặt tên thư viện là CircuitDesign2020. Chú ý các tên dùng trong Cadence không có dấu và các ký tự đặc biệt, tuân thủ theo qui tắc đặt tên nhưng trong ngôn ngữ lập trình C hoặc trong hệ điều hành Dos, Linux



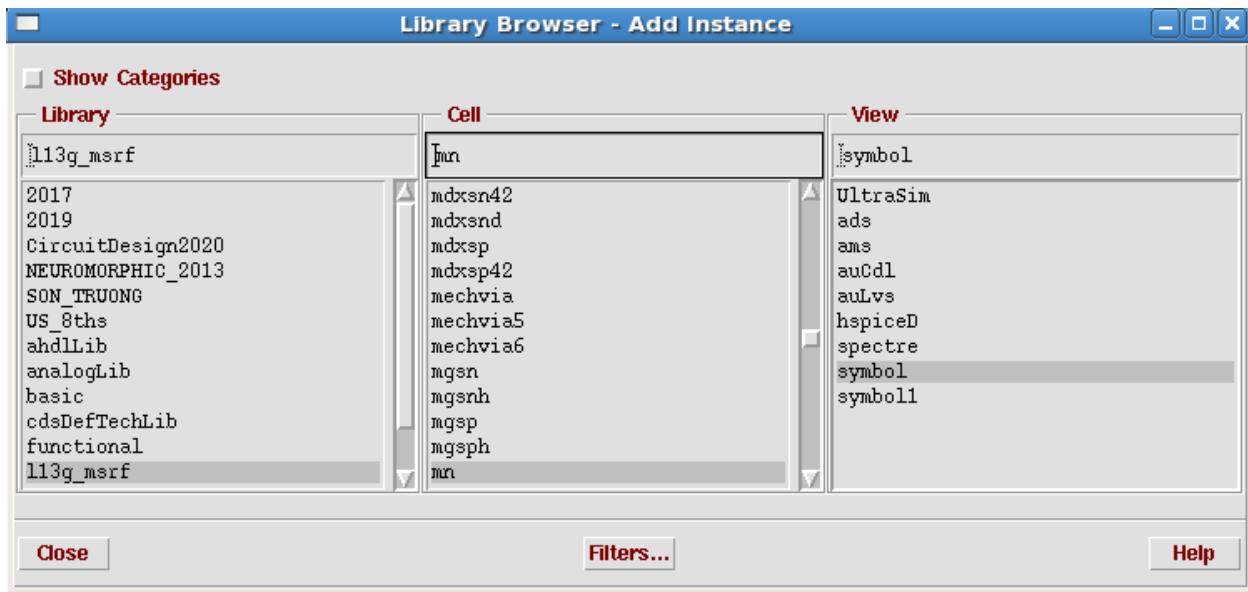
Sau khi thư viện được tạo, chọn thư viện, Chọn File → New → Cell View
Chọn tên Cell Name là Inverter, Tool chọn Composer-Schematic như hình



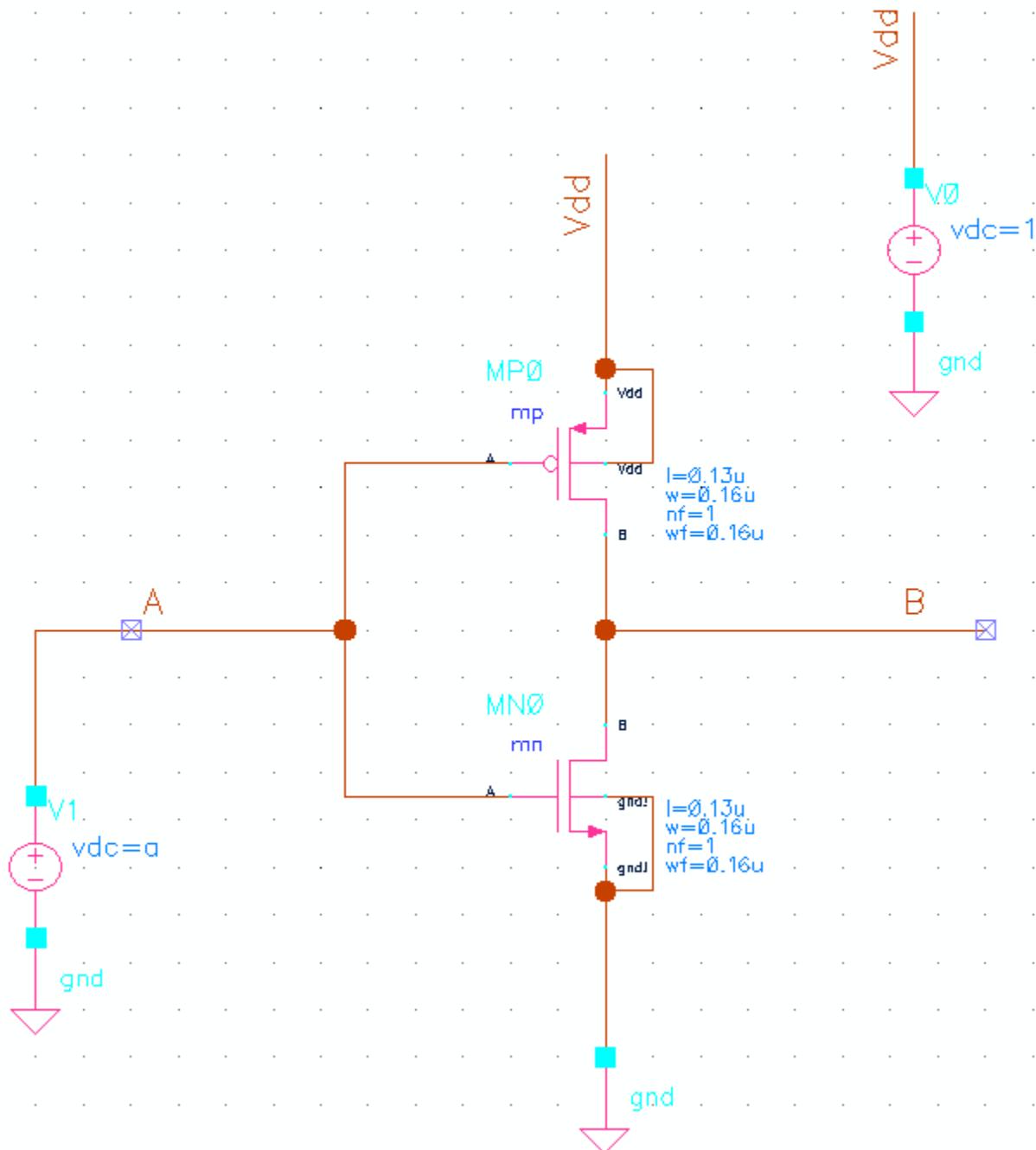
Thiết kế mạch cổng đảo sử dụng pMOS và nMOS, sử dụng các thông số mặc định của CMOS. Để thêm linh kiện vào mạch, chọn chức năng instance hoặc nhập phím tắt “i”. Để sử dụng các CMOS công nghệ 0.13 μ của Samsung, tại Library, nhấn Brown và chọn thư viện 113g_msrf, chon OK bên của sổ thiết lập PDK như hình



Thư viện 113g_msrf chứa các linh kiện công nghệ của Samsung, chọn nMOS và pMOS bằng cách tìm mp (pMOS) và mn (nMOS)



Đặt pMOS và nMOS vào schematic và thiết kế cổng đảo sử dụng pMOS và nMOS. Nguồn cung cấp được thiết kế như hình vẽ, các nguồn được lấy từ thư viện analogLib

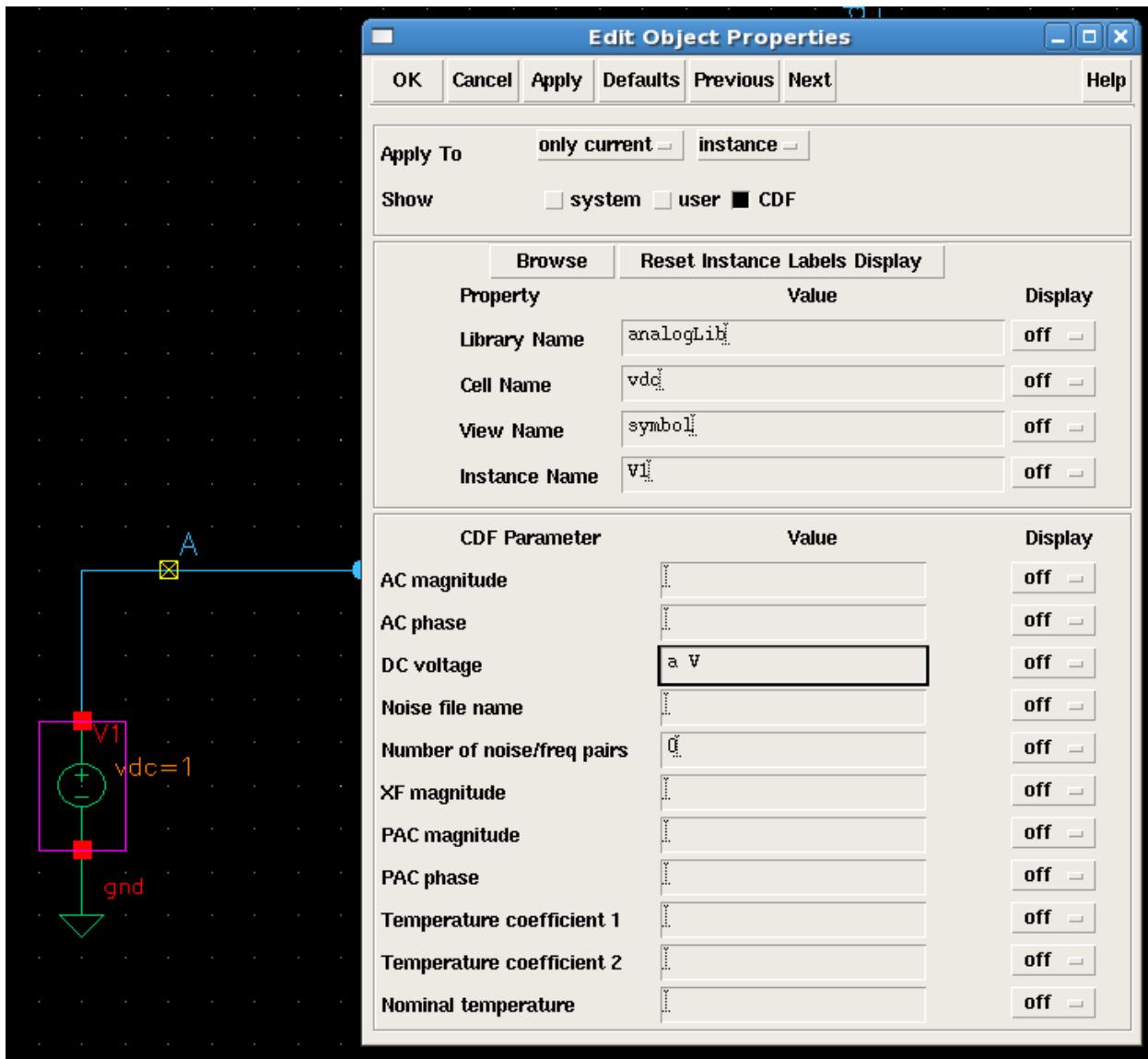


Chọn nguồn cung cấp $vdc = 1V$ hoặc $1.2 V$ cho các CMOS công nghệ 0.13 của Samsung. Mặc định các thông số cơ bản của nMOS và pMOS được thể hiện trong hình trên. Công nghệ 0.13μ , các nMOS và pMOS có kích thước chiều dài nhỏ nhất có thể là 0.13μ . Chúng ta không thể thay đổi chiều dài nhỏ hơn 0.13μ m. Chiều rộng có kích thước lớn hơn chiều dài. Tùy theo từng thiết kế, việc thay đổi kích thước chiều rộng và chiều dài sẽ dẫn đến những kết quả khác nhau. Thông thường, chiều rộng có kích thước lớn hơn chiều dài từ 10 đến 20 lần. Tuy nhiên, tùy vào mục đích cụ thể chúng ta mong muốn như điện trở nội của CMOS, điện dung, độ trễ, chúng ta có thể điều chỉnh kích thước của CMOS cho phù hợp. Trong các ứng dụng thông thường thì

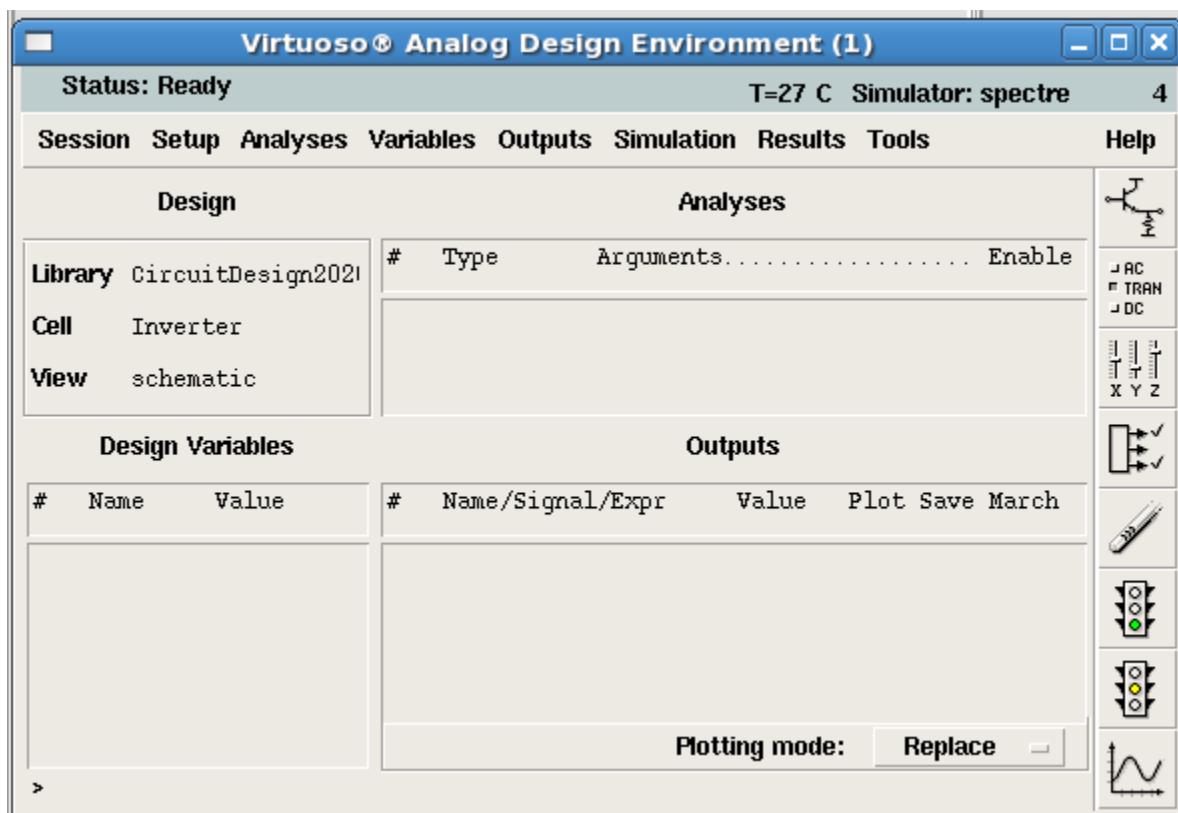
Body của nMOS được nối xuống GND và body của pMOS được nối lên Vdd. Trong một số trường hợp, body của pMOS và nMOS có thể được nối với mức điện áp khác Vdd và GND nhằm vào một số mục đích như giảm dòng rò.

Để mô phỏng đặc tính của Inverter, chúng ta cho điện áp ngõ vào tăng lên từ 0 đến 1V và ghi nhận điện áp ngõ ra. Có 2 cách thực hiện tăng điện áp ngõ vào: (1) sử dụng một nguồn điện áp DC với giá trị điện áp là một biến có thể thay đổi, (2) sử dụng một nguồn có điện áp tăng tuyến tính từ 0 đến 1V.

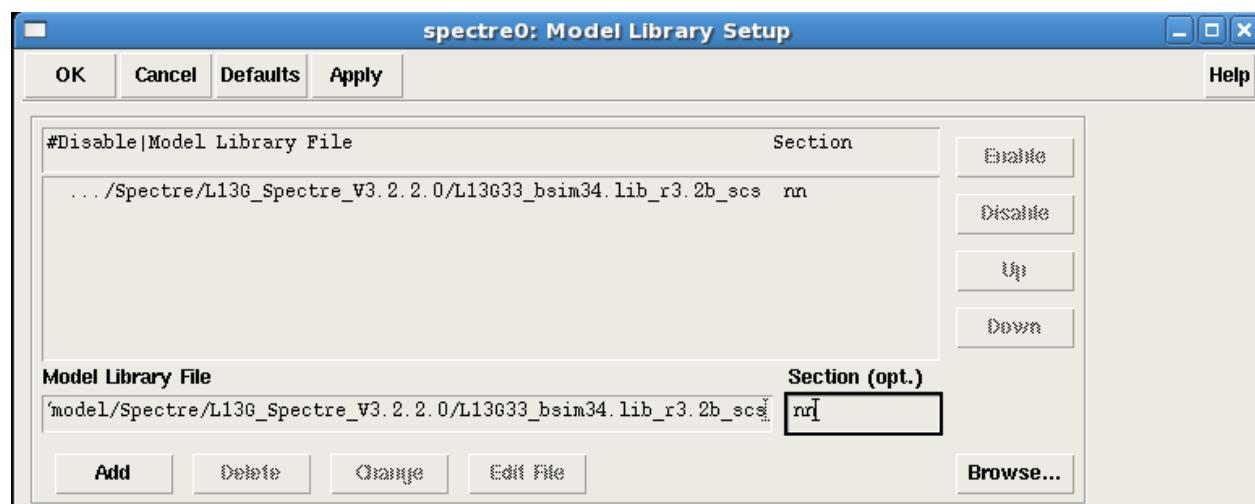
Trong bài thực hành này, chúng ta sử dụng một nguồn dc có điện áp được xem như một biến nhằm thay đổi từ 0 đến 1 V khi tiến hành phân tích DC. Thiết lập giá trị cho nguồn ngõ vào công Inverter như sau



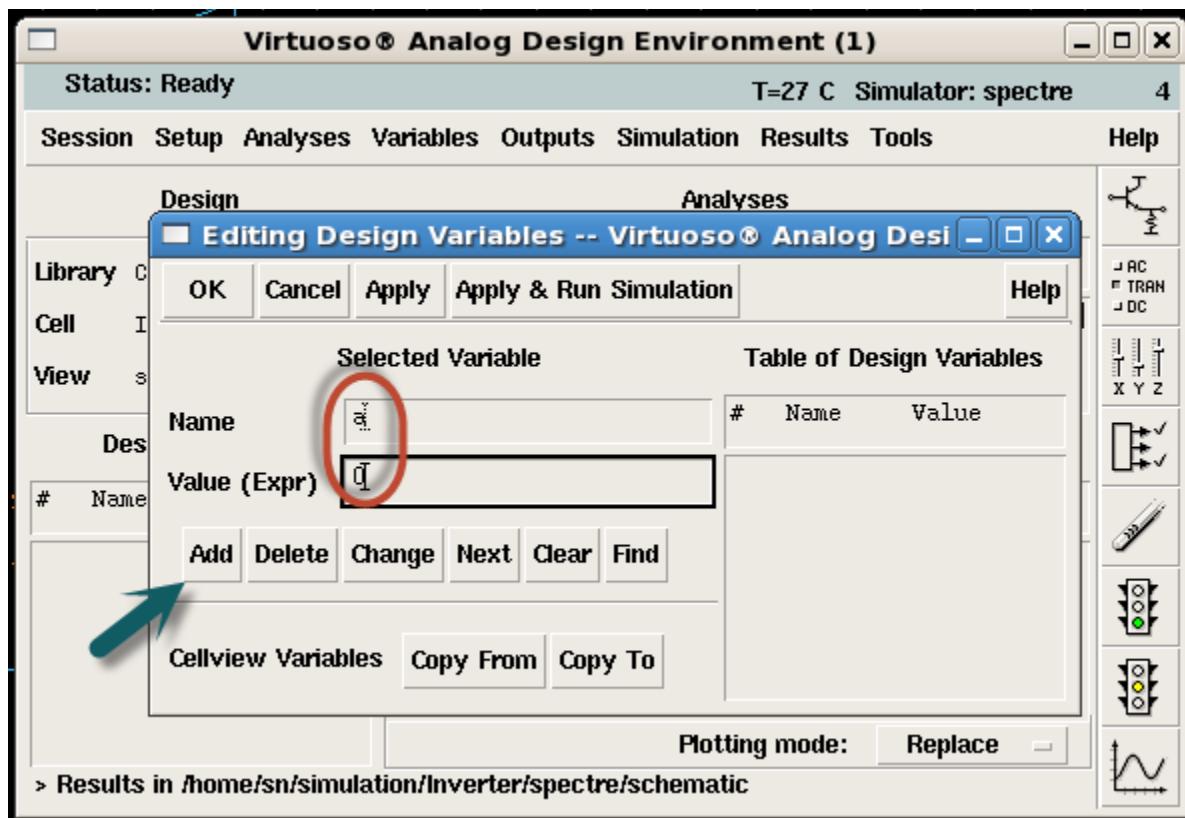
Chọn menu Tool – Analog Environment



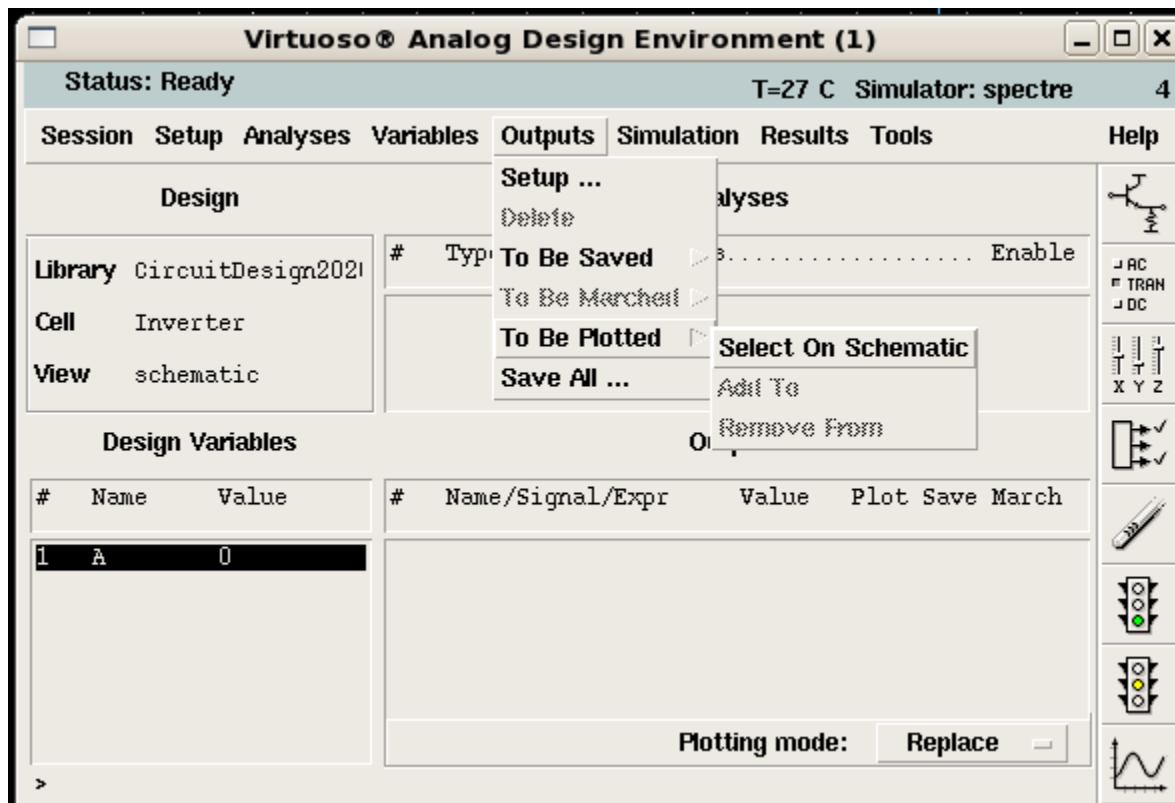
- Một số thiết lập trên giao diện Analog Design Environment
- Lựa chọn model cho CMOS: Setup Model Library, Nhấn Brown lựa chọn đường dẫn. Đường dẫn cho mô hình của các CMOS công nghệ 0.13 của Samsung như sau



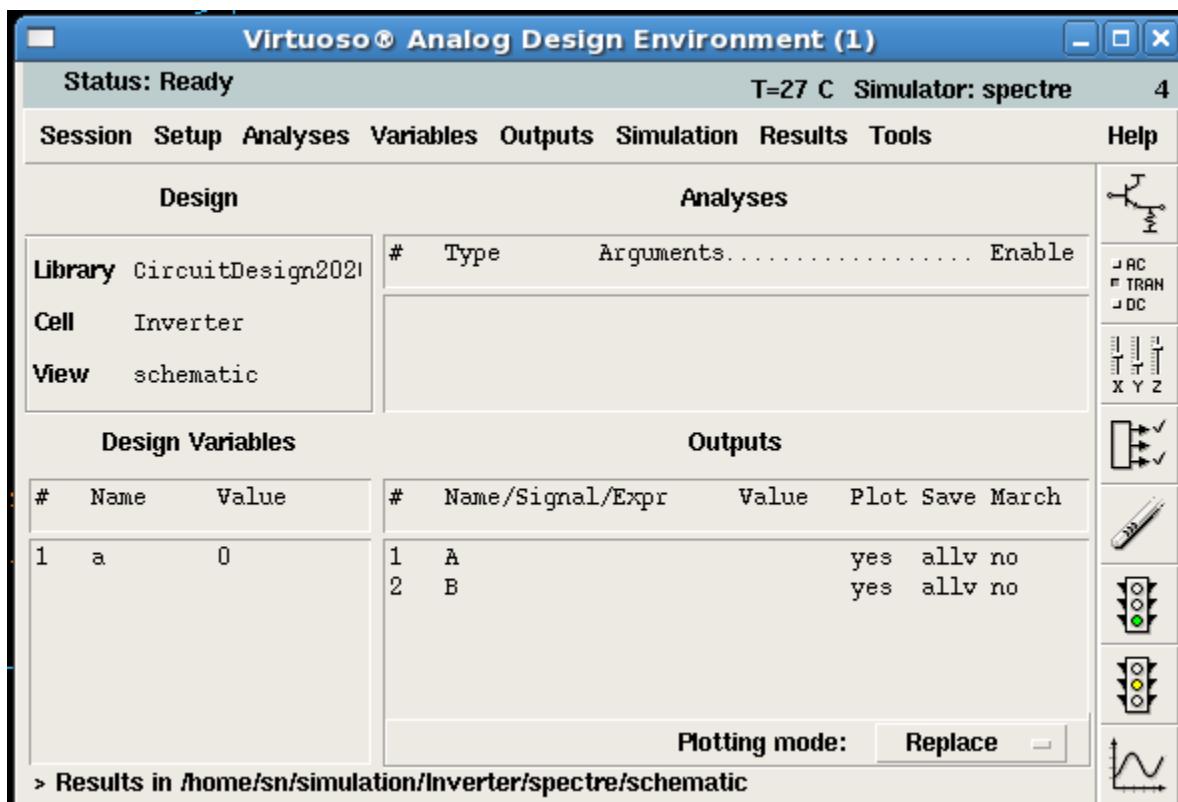
Thêm biến vào cho ngõ vào, Chọn Edit variable bên thanh công cụ bên phải, Chọn biến a và thiết lập giá trị đầu là 0



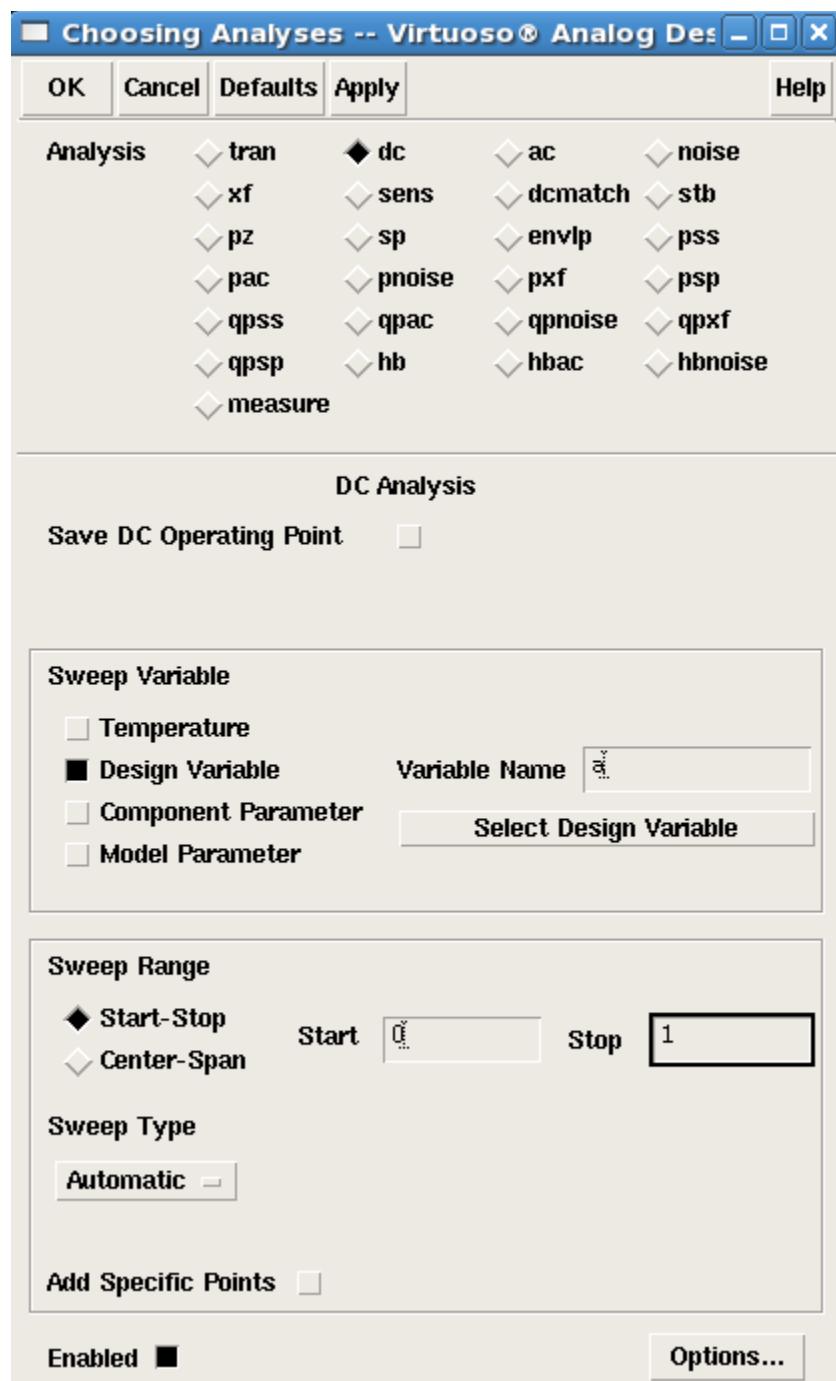
- Chọn ngõ ra hiển thị



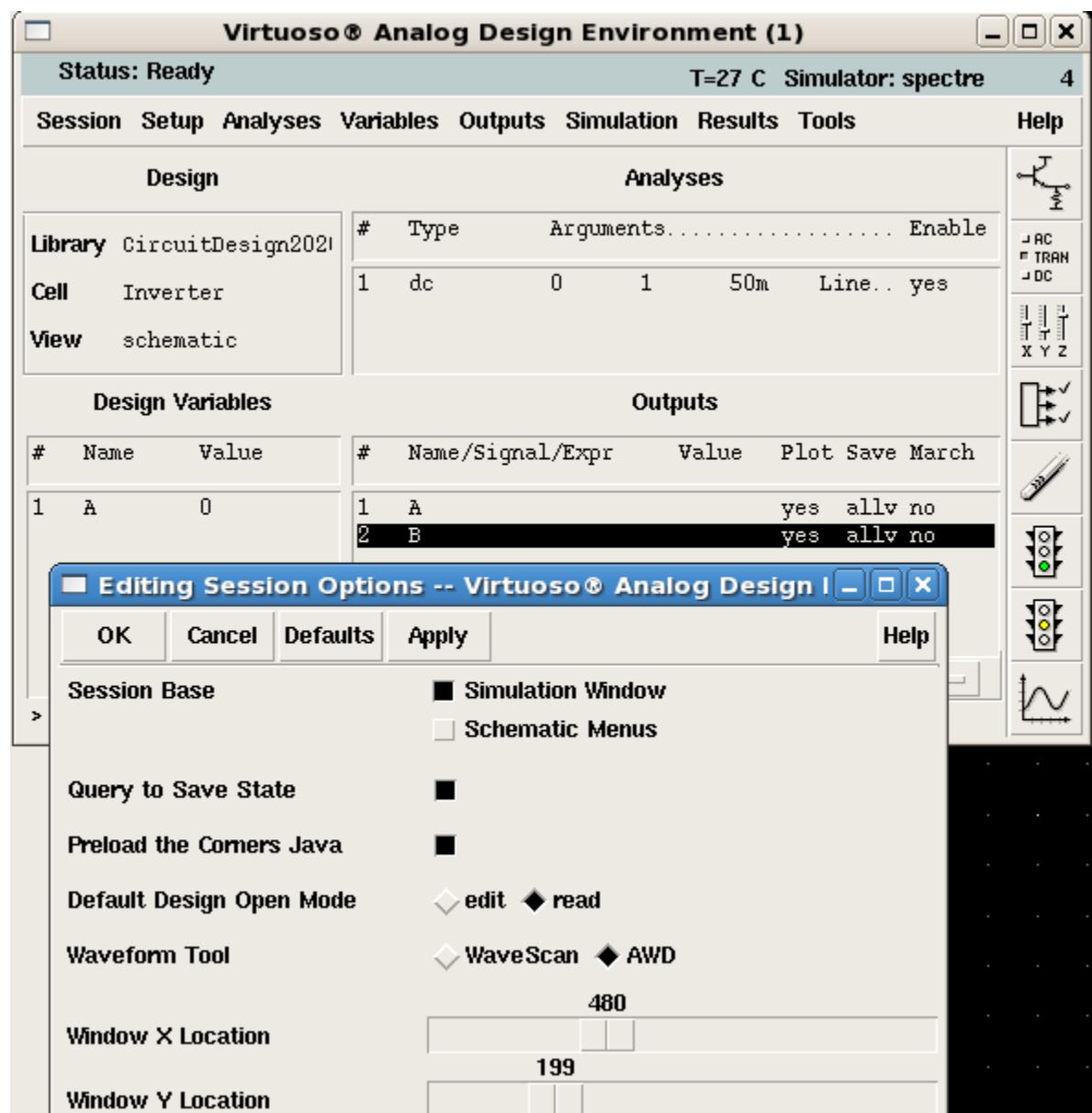
Click chọn trực tiếp trên sơ đồ nguyên lý, kết quả các tín hiệu cần được hiển thị trong mô phognr sẽ được liệt kê trong danh sách



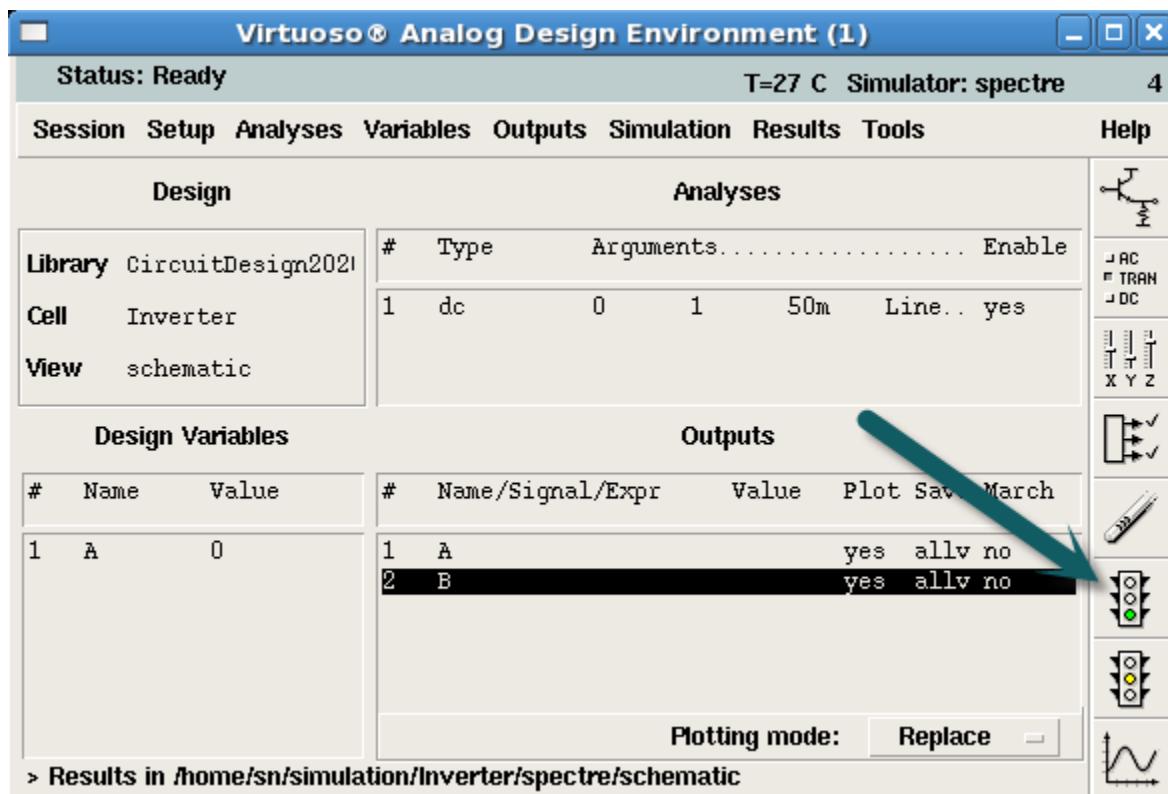
Thiết lập mô phỏng. Phần mềm Cadence Spectre hỗ trợ nhiều chế độ mô phỏng khác nhau, trong trường hợp mô phỏng đặc tính công đảo, chúng ta sử dụng chế độ mô phỏng dc. Thiết lập các thông số như hình bên dưới để mô phỏng dc các tín hiệu, trong đó tín hiệu A thay đổi từ 0 đến 1 với bước thay đổi tuyến tính là 0.05

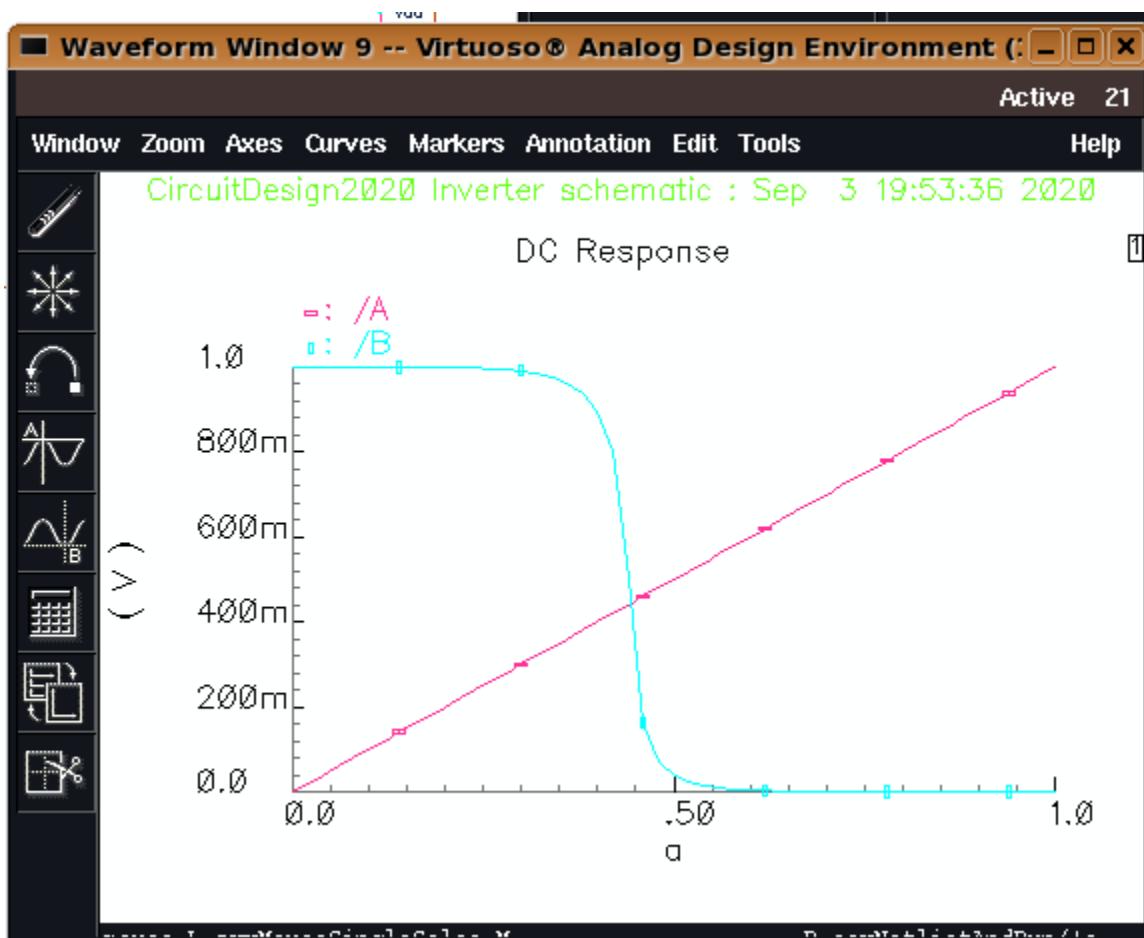


Thiết lập nghẽ ra tín hiệu phân tích. Chọn Session → Option, chọn AWD thay vì WaveScane

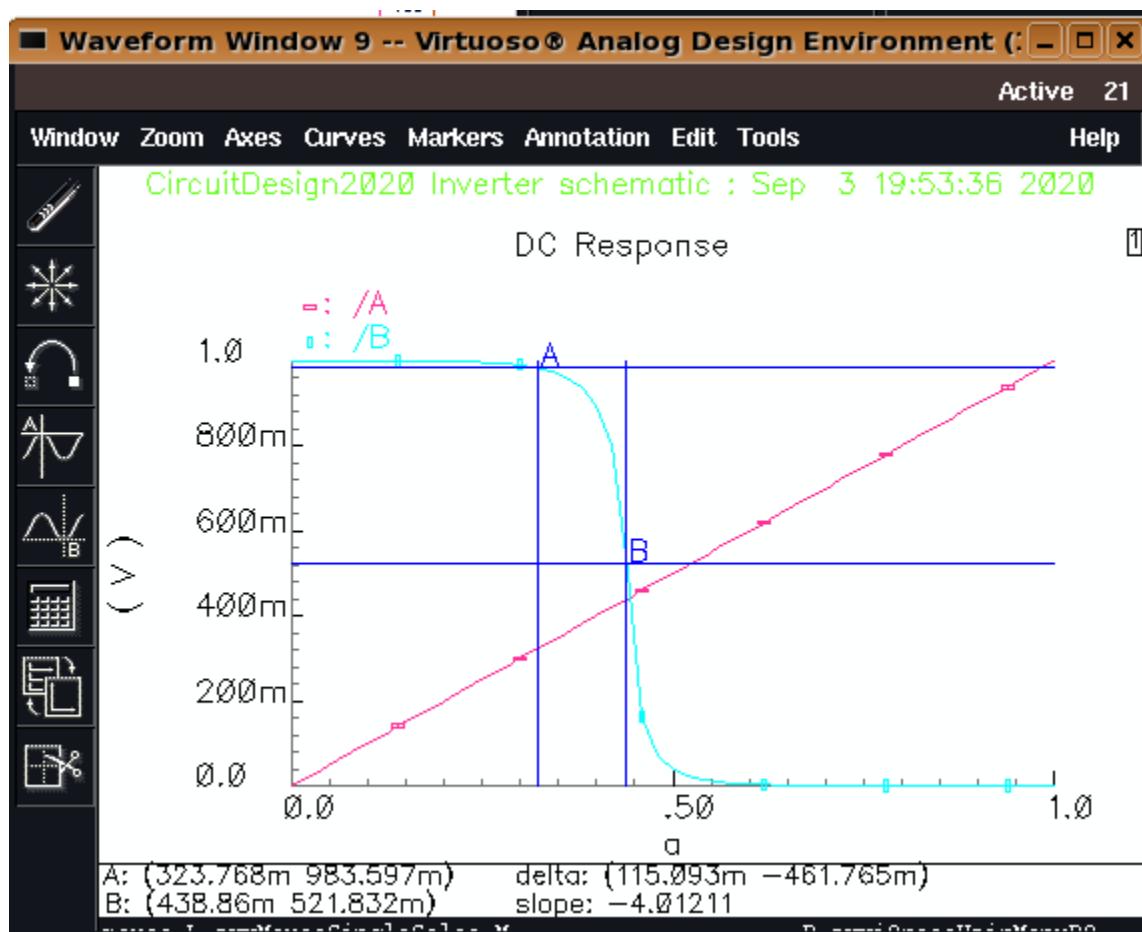


Sau khi thiết lập các thông số cho mô phỏng, xác định mô hình cho CMOS, có thể tính hành phân tích DC mạch cổng đảo, chọn netlist and run





Từ kết quả phân tích ta có thể thấy, khi ngõ vào tăng lên đến 0.33v, ngõ ra bắt đầu chuyển dần từ cao xuống thấp, tại vị trí ngõ vào 0.43V, ngõ ra giảm còn $\frac{1}{2}$ giá trị Vdd. Phương pháp phân tích DC cho phép phân tích hoạt động của mạch công đảo và tìm điểm chuyển mạch của công đảo trên đồ thị



2.2. Ảnh hưởng các thông số CMOS đến điểm làm việc của cổng đảo

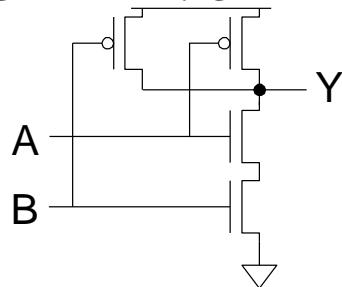
Thay đổi kích thước nMOS và pMOS, tìm điểm chuyển mạch (switching point) dựa vào phân tích DC

nMOS (L/W (μ))	pMOS (L/W (μ))	Switching point
0.13 / 0.16	0.13/0.5	
0.13 /0.5	0.13/0.16	
0.13/0.16	1/50	
1/50	0.13/0.16	

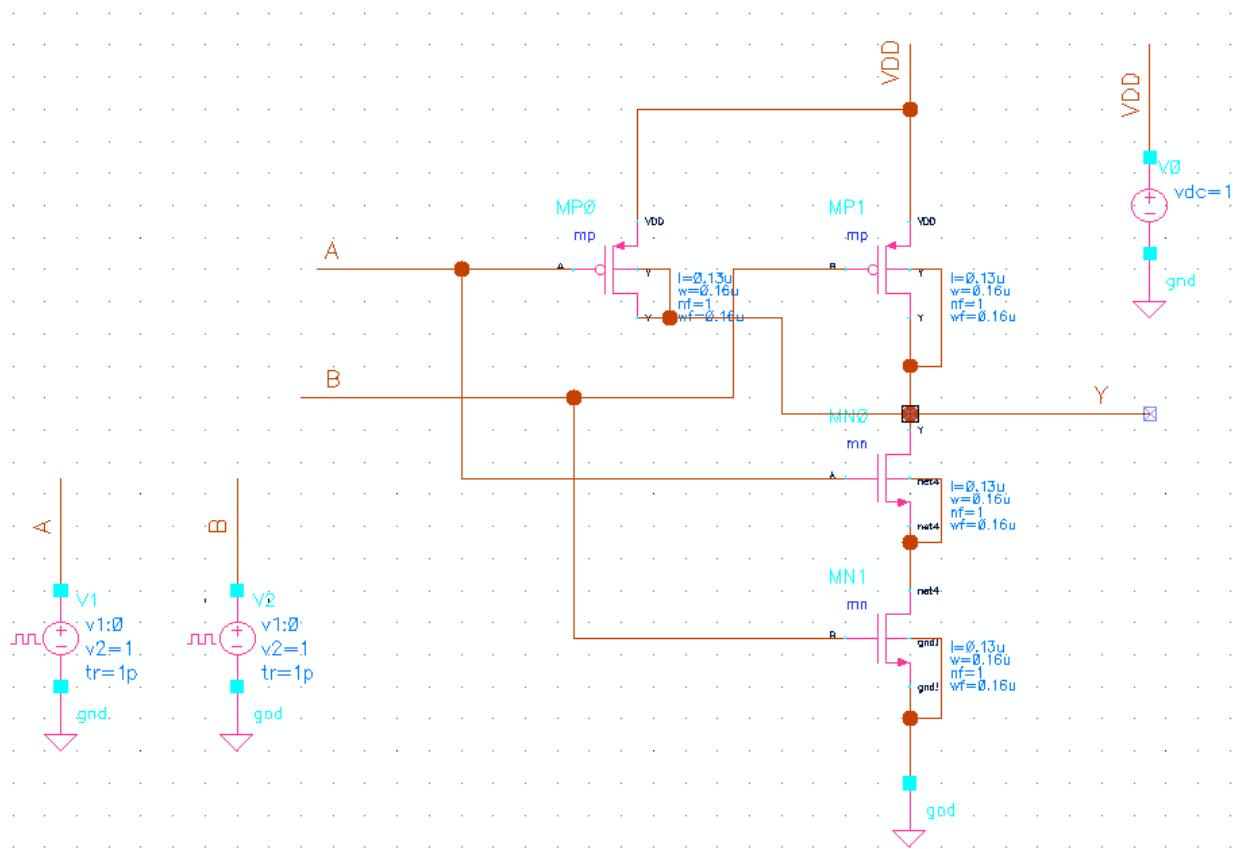
Sẵn sàng rút ra kết luận gì từ kết quả mô phỏng? Giải thích kết quả mô phỏng trong mối liên hệ với cơ sở lý thuyết?

3. Thiết kế mạch cỗng NAND sử dụng CMOS công nghệ Samsung 0.13μm

3.1. Giải thích hoạt động của cỗng NAND sử dụng CMOS dựa trên bảng trạng thái



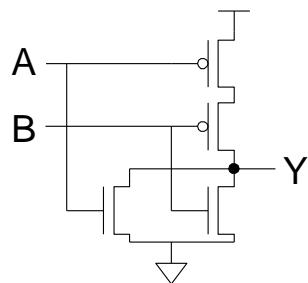
3.2. Thiết kế mạch cỗng NAND sử dụng CMOS



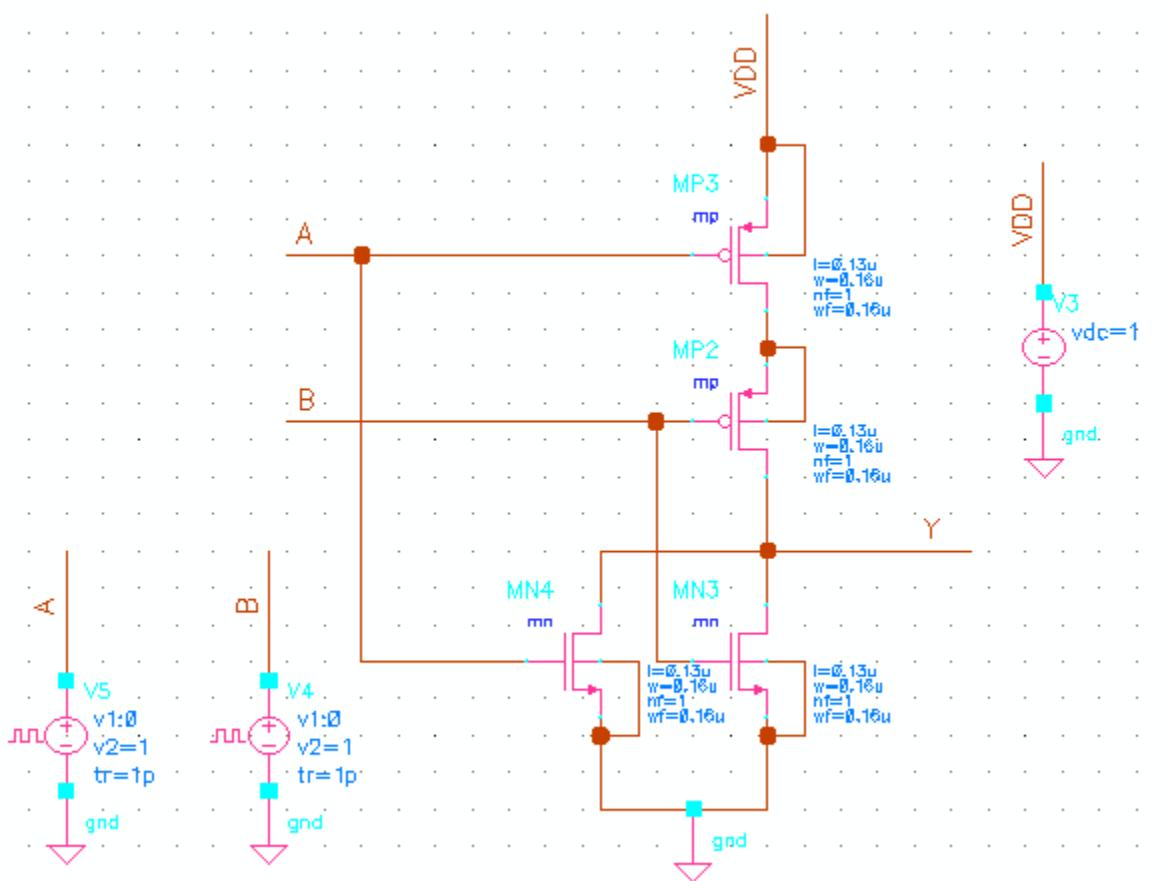
- Sử dụng nguồn vpulse để tạo tín hiệu logic cho ngõ vào
- Sử dụng chế độ phân tích “tran” để kiểm tra logic của mạch.
- Trình bày kết quả phân tích mạch

4. Thiết kế mạch cỗng NOR sử dụng CMOS công nghệ Samsung 0.13μm

4.1. Giải thích hoạt động của cỗng NOR sử dụng CMOS



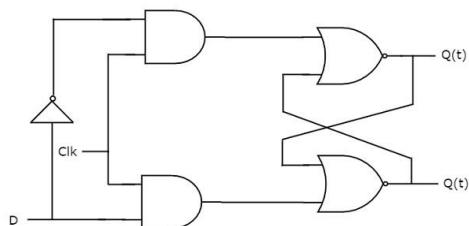
4.2.Thiết kế mạch công NOR sử dụng CMOS.



- Sử dụng nguồn vpulse để tạo tín hiệu logic cho ngõ vào, thực hiện kiểm tra bảng trạng thái,
- Sử dụng chế độ phân tích “tran” để kiểm tra logic của mạch.
- Trình bày kết quả phân tích.

5. Thiết kế mạch FLIP-FLIP sử dụng CMOS công nghệ Samsung 0.13μm

5.1.Giải thích hoạt động của mạch Flip-Flop D



5.2.Thiết kế mạch Flip-Flop sử dụng CMOS

Thiết kế mạch Flip-Flop sử dụng CMOS như sơ đồ nguyên lý. Sử dụng các công logic đã thiết kế trước đó

- Sử dụng nguồn vpulse để tạo tín hiệu logic cho ngõ vào, thực hiện kiểm tra bảng trạng thái,
- Sử dụng chế độ phân tích trans, hoặc ac để kiểm tra logic của mạch.
- Trình bày kết quả mô phỏng.