BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐÀO ĐẠO ĐIỆN-ĐIỆN TỬ

BÁO CÁO GIỮA KÌ

THIẾT KẾ VI MẠCH TÍCH HỢP VLSI

ĐỀ TÀI : MẠCH ĐÈN GIAO THÔNG

GVHD: PGS. TS. VÕ MINH HUÂN

MÃ HP: ICSD336764_23_2_11

NHÓM SINH VIÊN THỰC HIỆN:

1. Đinh Quang Dũng 22161230
2. Đoàn Lê Nhân 22161298
3. Huỳnh Bảo Ngọc 22161295
4. Nguyễn Vân Anh 22161219

TP. HỒ CHÍ MINH – THÁNG 4/2024

NHẬN XÉT CỦA GIẢNG VIÊN

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

KÝ TÊN

BẢNG PHÂN CÔNG VÀ ĐÁNH GIÁ CÁC CÔNG VIỆC TỪNG THÀNH VIÊN

22161298	Đoàn Lê nhân	Module MUX 2-1	100%
22161230	Đinh Quang Dũng	Module hiển thị Module TOP	100%
22161219	Nguyễn Vân Anh	Module led chớp tắt	100%
22161295	Huỳnh Bảo Ngọc	Máy trạng thái cho đèn giao thông	100%

Mục Lục

CHƯƠNG 1: TỔNG QUAN.....5

1.1 5

1.2 5

1.3 Lập trình mạch điều khiển đèn giao thông sử dụng Verilog.....5

1.4. . Lời cảm ơn.....5

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT.....6

2.1 Ngôn ngữ lập trình Verilog.....6

2.1.2.Các kiểu dữ liệu trong Verilog.....6

2.1.3 Toán tử và các phát biểu điều khiển.....8

2.1.4 Phép gán.....9

2.2 Quy trình thiết kế hệ thống.....10

2.2.1 Quy trình thiết kế một ASIC.....10

2.2.2 Quy trình thiết kế dựa trên FPGA.....10

CHƯƠNG 3: NỘI DUNG CHÍNH MẠCH ĐÈN GIAO THÔNG.....12

3.1 Sơ đồ khối thiết kế.....12

3.2 Các module.....13

3.2.1 Module traficlight.....13

3.2.2 Module ledchoptat.....16

3.2.3 Module Mux 2_1.....21

3.2.4 Module ledhienthi.....25

3.2.6 Module top.....27

CHƯƠNG 4: KẾT LUẬN.....29

CHƯƠNG 1: TỔNG QUAN

1.1 Đặt vấn đề

Giao thông là một vấn đề nhức nhối ở các thành phố lớn hiện nay. Việc ùn tắc giao thông, tai nạn giao thông ngày càng gia tăng gây ảnh hưởng nghiêm trọng đến đời sống và kinh tế xã hội. Đây là vấn đề nhức nhối ở hầu hết các thành phố lớn, đặc biệt là Hà Nội và Thành phố Hồ Chí Minh. Nguyên nhân chủ yếu là do mật độ phương tiện giao thông tăng cao, hạ tầng giao thông chưa đáp ứng kịp, ý thức của người tham gia giao thông còn hạn chế. Đèn giao thông đóng vai trò quan trọng trong việc giải quyết các vấn đề giao thông nêu trên, giảm thiểu ùn tắc giao thông, đèn giao thông giúp điều tiết giao thông một cách khoa học, hiệu quả, góp phần giảm thiểu ùn tắc giao thông, đảm bảo an toàn giao thông, đèn giao thông giúp phân luồng giao thông, hạn chế nguy cơ va chạm giữa các phương tiện, góp phần đảm bảo an toàn giao thông. Giảm thiểu ô nhiễm môi trường khi lưu thông thông suốt, các phương tiện sẽ giảm thiểu thời gian chờ đợi, từ đó giảm thiểu lượng khí thải ra môi trường. đèn giao thông là một thiết bị quan trọng và không thể thiếu trong hệ thống giao thông hiện đại. Việc sử dụng đèn giao thông một cách hợp lý sẽ góp phần giải quyết các vấn đề giao thông, đảm bảo an toàn cho người tham gia giao thông và bảo vệ môi trường.

1.2 Hiệu quả và an toàn của hệ thống đèn giao thông trong quản lý giao thông

Với sự ra đời của hệ thống đèn giao thông, quản lý giao thông trở nên hiệu quả và an toàn hơn đáng kể. Hệ thống đèn giao thông giúp tự động hóa việc điều chỉnh luồng giao thông, tạo ra sự phân luồng mạnh mẽ và đồng đều giữa các phương tiện di chuyển. Điều này không chỉ giảm thiểu nguy cơ tai nạn giao thông mà còn tối ưu hóa thời gian di chuyển của người tham gia giao thông

1.3 Lập trình mạch điều khiển đèn giao thông sử dụng Verilog

Trong bối cảnh trên, tụi em đã tiến hành lập trình một mạch điều khiển đèn giao thông sử dụng ngôn ngữ Verilog. Việc này sẽ giúp chúng ta tạo ra một hệ thống điều khiển đèn giao thông đơn giản và dễ dàng để thí nghiệm. Mục tiêu của bài tập này là hiểu hơn về cách thức hoạt động của mạch điều khiển đèn giao thông và áp dụng kiến thức lý thuyết vào thực tế.

1.4. . Lời cảm ơn

Chúng em xin gửi lời tri ân sâu sắc đến Thầy Võ Minh Huân đã tận tình hướng dẫn và hỗ trợ chúng em trong suốt quá trình thực hiện đề tài. Tuy còn nhiều hạn chế, chúng em mong muốn nhận được những góp ý quý báu từ Thầy và các bạn để bài báo cáo được hoàn thiện hơn. Xin chân thành cảm ơn thầy.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Ngôn ngữ lập trình Verilog

2.1.1 Giới thiệu khái quát về Verilog

Verilog là một ngôn ngữ mô tả phần cứng (HDL) được sử dụng để mô tả hành vi và cấu trúc của các hệ thống điện tử. Nó được sử dụng rộng rãi trong thiết kế mạch tích hợp (IC), hệ thống nhúng và các ứng dụng khác liên quan đến phần cứng. Verilog là một ngôn ngữ dạng text thuần túy, dễ học và sử dụng. Nó có nhiều tính năng mạnh mẽ cho phép mô tả các hệ thống điện tử một cách chính xác và hiệu quả.

2.1.2.Các kiểu dữ liệu trong Verilog

Verilog chỉ hỗ trợ những loại dữ liệu đã được định nghĩa trước. Những loại dữ liệu này bao gồm dữ liệu bit, mảng bit, vùng nhớ, số nguyên, số thực, sự kiện, và độ mạnh của dữ liệu. Những loại này định nghĩa trong phần lớn mô tả của Verilog.

Verilog chủ yếu xử lí trên bit và byte khi mô tả mạch điện tử. Loại số thực thì hữu dụng trong việc mô tả độ trì hoãn và định thời, nó cũng rất hữu dụng trong việc mô hình hóa ở mức cao như là phân tích xác suất kết nối mạch trong hệ thống và những giải thuật xử lí tín hiệu số. Loại dữ liệu phần cứng bao gồm net và reg. Thông thường những loại này có thể được xem như là dây kết nối và thanh ghi. Dữ liệu net có thể được khai báo chi tiết hơn để tạo ra những loại dữ liệu khác như tri -stated hay non-tri -stated và phụ thuộc vào các xử lí nhiều kết nối sẽ tạo ra những phép and, or hoặc dùng giá trị trước đó.

NET Mô tả kết nối dây dẫn trong một mạch điện và được dùng để kết nối các cổng hay các module. Giá trị của Net có thể đọc, nhưng không được gán trong hàm (function) hoặc khối (block). Một net sẽ không lưu giữ giá trị của nó. Nó phải được điều khiển bởi một trong hai cách sau:

+ Bằng việc kết nối net đến ngõ ra của một cổng hay một module.

+ Bằng việc gán một giá trị đến net trong một phép gán nối tiếp.

WIRE là một kiểu dữ liệu sử dụng phổ biến thuộc kiểu Net. wire là một kiểu dữ liệu đơn giản để kết nối giữa liệu hai cổng,module hay trong phép gán nối tiếp….Ngoài ra còn có nhiều kiểu dữ liễu khác như wand, wor, tri….

Cú Pháp: Wire [msb:lsb] tên biến wire.

Wand [msb:lsb] tên biến wand.

Wor [msb:lsb] tên biến wor.

Tri [msb:lsb] tên biến tri.

REG Khai báo reg được thực hiện cho tất cả những tín hiệu mà được điều khiển từ những mô tả hành vi. Loại dữ liệu reg lưu giữ một giá trị được cho đến khi nó được gán một giá trị mới trong một mô tả tuần tự (khối initial hoặc always). Có thể hiểu nôm na Reg là một kiểu net có khả năng lưu trữ giá trị và có thể xem là một register trong phần cứng.

Cú pháp: Reg [msb:lsb] tên biến reg.

Khai báo PORT: Những từ khoá Input, Output, Inout biểu thì đầu vào, đầu ra, và port hai chiều của một module hoặc task. Một port đầu ra có thể được cấu hình từ các dạng: wire, reg, wand, wor, hoặc tri. Mặc định là wire. Kiểu reg chỉ xuất hiện ở cổng Output hay phét gán nối tiếp có nhớ.

Cú pháp: Input [msb:lsb] port đầu vào.

Output [msb:lsb] port đầu ra.

Inout [msb:lsb] port đầu vào,ra hai chiều.

PARAMATER Trong Verilog HDL, loại dữ liệu tham số (parameter) không thuộc loại dữ liệu biến (variables: reg, integer, time, real, realtime) cũng như loại dữ liệu net. Dữ liệu tham số không phải là biến mà chúng là hằng số. Có hai loại tham số trong Verilog đó là:

++ Tham số module (module parameter): parameter và localparam.

++ Tham số đặc tả (specify parameter): specparam.

Cú pháp: Parameter par_1= giá trị, par_2= giá trị, …;

Parameter [msb:lsb] par_3 = giá trị;

- Cả hai loại tham số trên đều được phép khai báo độ rộng. Mặc định, parameter và

specparam sẽ có độ rộng đủ để chứa giá trị của hằng số, ngoại trừ khi tham số đó có khai báo độ rộng. Việc khai báo trùng tên giữa net , biến hay tham số là không được phép.

2.1.3 Toán tử và các phát biểu điều khiển

> : lớn hơn.

>= : lớn hơn hoặc bằng

< : nhỏ hơn

<= : nhỏ hơn hoặc bằng

== : bằng logic

!= : khác

Các toán tử điều kiện

! : đảo logic

&& : AND logic

|| : OR logic

Các toán tử set bit

~ : đảo bit

& : AND

| : OR

^ : XOR

~& : NAND

~| : NOR

Các toán tử khác

{,} : ghép thanh ghi hoặc dây

<< : dịch trái thanh ghi

>> : dịch phải thanh ghi

?: : điều kiện

2.1.4 Phép gán

Phép gán liên tục

Phép gán liên tục là phép gán cơ bản nhất trong Verilog.Phép gán liên tục dùng trong cách mạch tổ hợp logic. Phép gán liên tục được dùng để gán các giá trị cho wire của module, hay là cập nhật giá trị của các port(wire). Nó được đặt bên ngoài khối always@ hoặc initial. Lưu ý rằng câu lệnh phép gán liên tục xảy ra đồng thời và liên tục được thực hiện trong suốt quá trình mô phỏng.Thứ tự các lệnh gán không quan trọng.Bất kì thay đổi nào của bên phải phép gán sẽ làm cho giá trị bên trái thay đổi ngay lập tức.

Cú pháp:

net_data_type [delay] [size]= expression(biểu thức); // khai báo kiểu dữ liệu cho biến.

assign [#(delay)] net_name = expression; // explicit

Giải thích cú pháp:

- Net_data_type : kiểu dữ liệu của đầu vào hoặc đầu ra như: wire, reg

- Delay: khoảng thời gian trễ trong các phép gán.

- Size: độ rộng bit của các biến trong phép gán.

- Expression: biểu thức hoặc một biến nào đó.

Phép gán liên tục sẽ ngầm khai báo kiểu dữ liệu net cho từng câu lệnh. Một phép gán liên tục đầy đủ thường sử dụng một trong 2 cách: một là khai báo kiểu net, hai là câu lệnh gán liên tục assign.

Phép gán assign: phía bên trái của phép gán là một biến còn phía bên phải là một biểu thức. Kiểu dữ liệu bên trái bắt buộc phải là net và bên phải có thể là kiểu net, reg... Phía bên trái tách biệt với phía bên phải bằng kí tự "=". Gía trị của phía bên trái thay đổi khi giá trị biểu thức bên phải thay đổi.

2.2 Quy trình thiết kế hệ thống

2.2.1 Quy trình thiết kế một ASIC

(Application Specific Integrated Circuit): Mạch tích hợp ứng dụng cụ thể ASIC là linh kiện được sản xuất chưa hoàn chỉnh (hay một phần) bởi nhà cung cấp ASIC ở dạng tổng quát.

Quá trình chế tạo ban đầu này rất phức tạp, mất nhiều thời gian và là phần đắc tiền nhất trong toàn bộ quá trình sản xuất.

Kết quả của quá trình chế tạo ban đầu này sẽ là những chip silicon có các dải transistor chưa nối với nhau. Quá trình chế tạo sau cùng là quá trình kết nối các transistor với nhau, sẽ được hoàn tất khi người thiết kế chip có một thiết kế cụ thể và người này muốn thực hiện lên trên ASIC.

Nhà cung cấp ASIC thường có thể thực hiện điều này trong vài tuần và gọi đây là thời gian làm thay đổi hoàn toàn.

Có hai loại linh kiện ASIC, đó là dải cổng (gate array) và cell chuẩn (standard cell).

2.2.2 Quy trình thiết kế dựa trên FPGA

(Field Programmable Gate Array): Dải cổng lập trình được dạng trường là linh kiện được sản xuất hoàn chỉnh nhưng vẫn duy trì được tính độc lập với thiết kế.

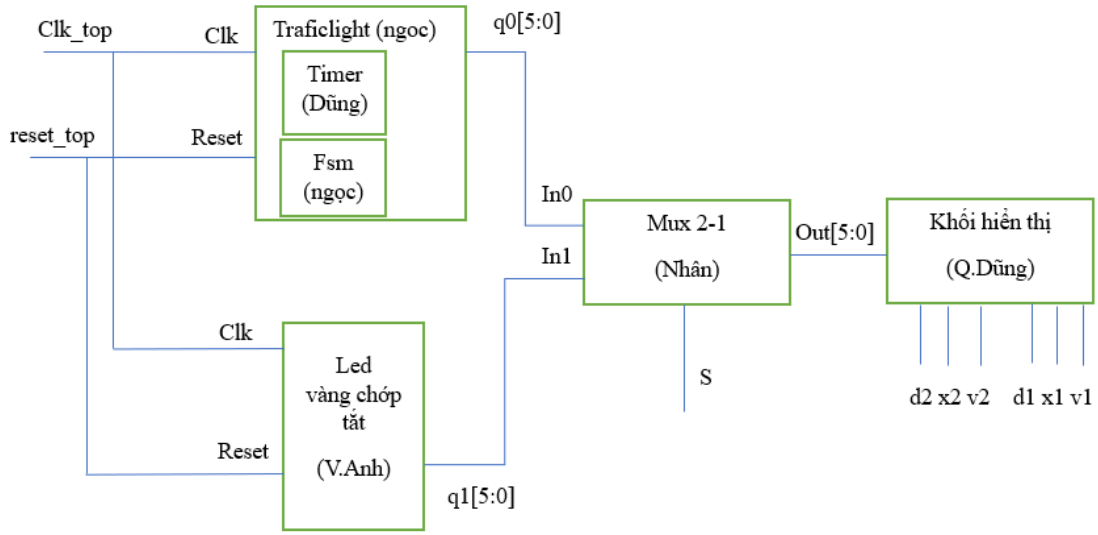
Mỗi nhà sản xuất FPGA đều đăng ký độc quyền các kiến trúc FPGA của mình. Tuy nhiên, những kiến trúc này sẽ bao gồm một số khối logic lập trình được và những khối này được nối với nhau bằng các ma trận chuyển mạch lập trình được.

Để cấu hình cho một linh kiện cho một chức năng cụ thể, những ma trận chuyển mạch này được lập trình để định tuyến các tín hiệu giữa nhiều khối logic riêng rẽ. Như vậy với các tính năng của ASIC và FPGA sẽ được chọn lựa tùy vào giá thành của sản phẩm.

Tuy nhiên với FPGA, việc lập trình thường dễ dàng và nhanh chóng, chức năng tùy thuộc khách hàng. Hơn nữa các FPGA cho phép việc bố trí bo mạch in bằng công cụ CAD được tiến hành, trong khi thiết kế bên trong FPGA vẫn đang hoàn tất. Thủ tục này cho phép ta kiểm tra sự tích hợp phần cứng và phần mền. Nếu việc kiểm tra hệ thống thất bại, thiết kế này có thể được sửa đổi và linh kiện khác FPGA được lập trình ngay lập tức với chi phí tương đối thấp. Với các chip FPGA và CPLD dạng SSP (lập trình ngay trên hệ thống) hiện nay, việc lập trình lại sẽ hoàn toàn dễ dàng với chi phí không đáng kể. Với những lý do trên, các thiết kế thường trước tiên hướng đến FPGA để kiểm tra hệ thống và có thời gian sản xuất nhỏ. Kế đến, thiết kế được định hướng lại đến một ASIC để sản xuất ở quy mô lớn hơn. Các thỏa hiệp thiết kế phải được xem xét khi định hướng lại từ FPGA sang ASIC. Thí dụ như thời gian giữ dài có thể không bao giờ xuất hiện trong ASIC do tốc độ thực hiện chức năng được cải tiến.

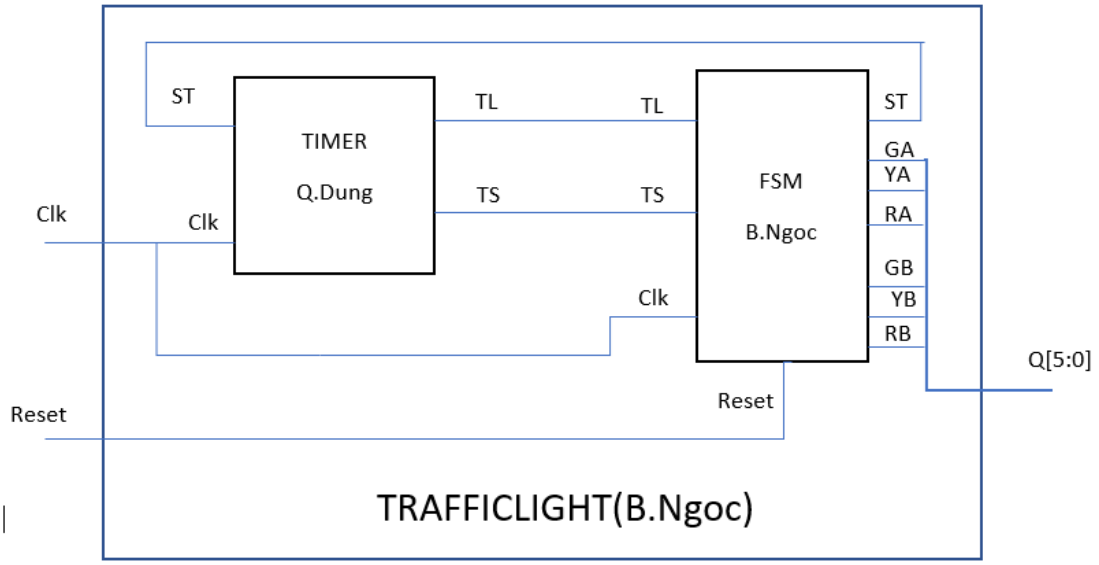
CHƯƠNG 3: NỘI DUNG CHÍNH MẠCH ĐÈN GIAO THÔNG

3.1 Sơ đồ khối thiết kế



3.2 Các module

3.2.1 Module traficlight



Bao gồm 2 khối submodule timer và fsm

Kết nối

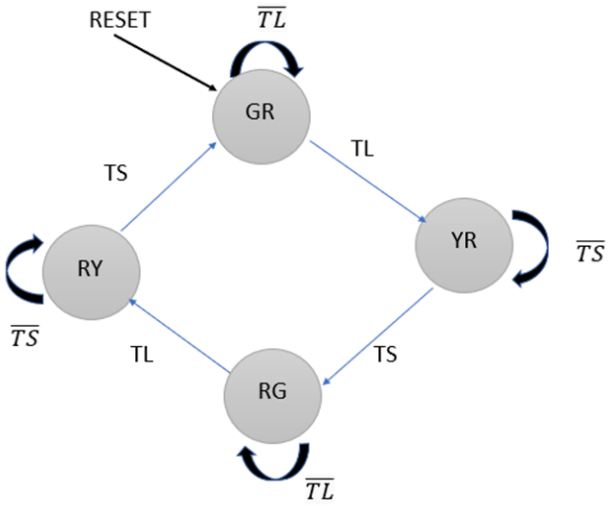
input

- clk: Cổng vào 1 bit xung clk.
- reset: Cổng vào reset 1 bit.

output Q[5:0]: Ngõ ra 6 bit (Đỏ2,Vàng2, Xanh2,Đỏ1,Vàng1,Xanh1).

Nguyên lý hoạt động:

- GR (Xanh-Đỏ): Trạng thái ban đầu, đèn xanh cho một hướng, đèn đỏ cho hướng còn lại.



Nếu TL true (đếm 14xung clk) ST=1 trong 1 chu kỳ xung clock đặt lại bộ đếm.

-YR (Vàng-Đỏ): Trạng thái chuyển từ xanh sang đỏ.

Nếu TS true (đếm 4xung clk) ST=1 trong 1 chu kỳ xung clock đặt lại bộ đếm.

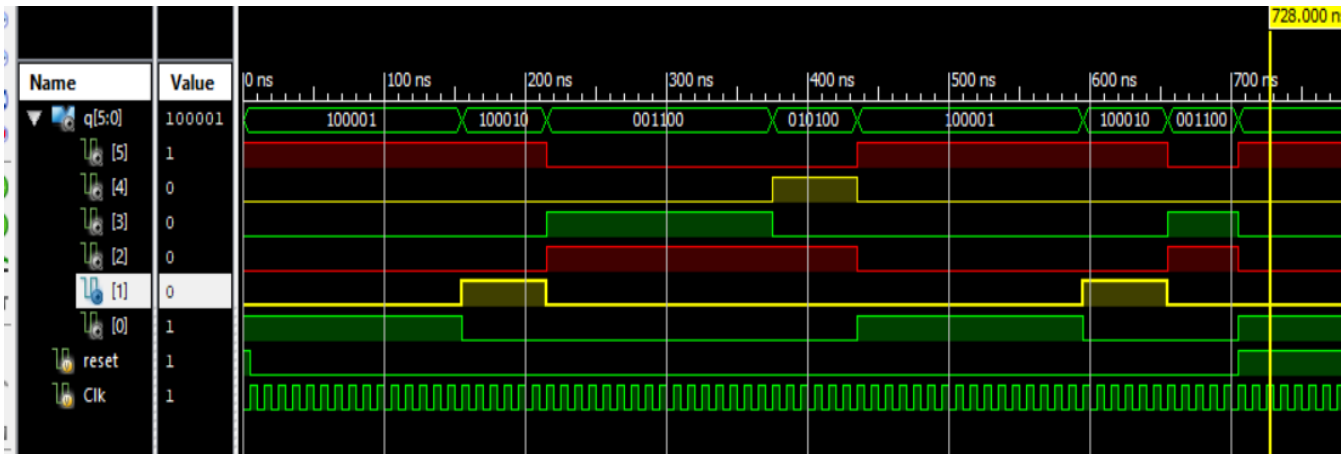
-RG (Đỏ-Xanh): Đèn xanh cho hướng ngược lại.

Nếu TL true (đếm 14xung clk) ST=1 trong 1 chu kỳ xung clock đặt lại bộ đếm.

-RY (Đỏ-Vàng): Trạng thái chuyển từ đỏ sang xanh.

Nếu TS true (đếm 4xung clk) ST=1 trong 1 chu kỳ xung clock đặt lại bộ đếm.

Hình ảnh mô phỏng

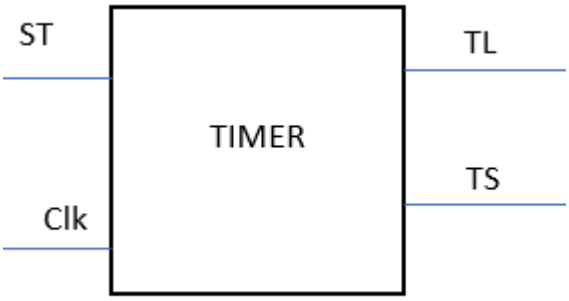


khi reset tác động mạch quay về trạng thái ban đầu GR

a.Khối timer: đếm thời gian TS (thời gian ngắn) và TL (thời gian dài).

input

- ST: Đầu vào ST được sử dụng để đặt lại bộ đếm khi ST=1.



ouput

- TS: clk tác động lên >=4 thì TS thỏa điều kiện true.

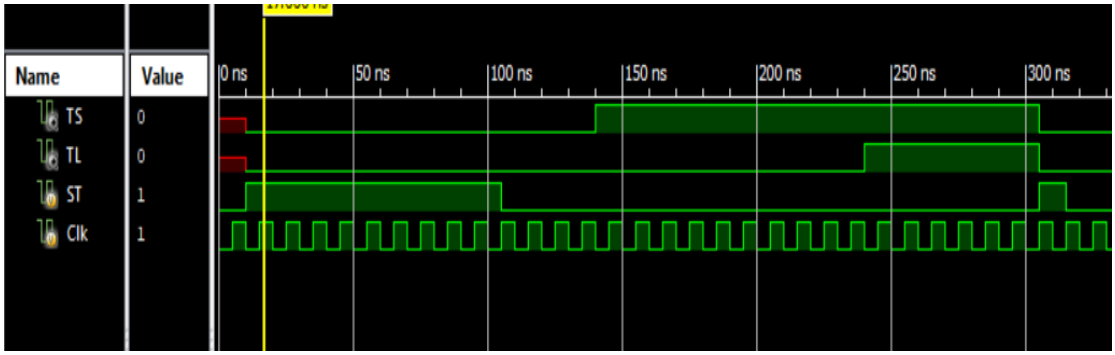
- TL: clk tác động lên >=14 được thỏa điều kiện true.

Khối hoạt động:

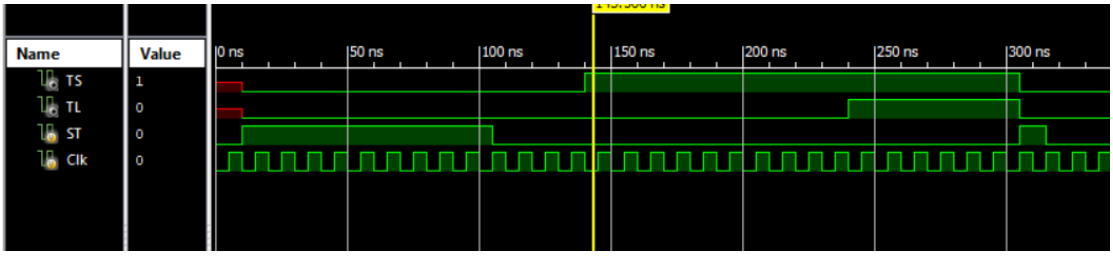
- Bộ đếm thời gian đếm chu kỳ đồng hồ và tăng biến giá trị.
- Khi ST tác động mức cao hoặc mức 1 thì module timer không đếm giá trị TL và TS luôn mức 0.
- khi ST mức 0 và module timer bắt đầu đếm khi đếm lớn hơn hoặc bằng 4 xung thì TS lên mức 1 và TL mức 0
- khi đếm lớn hơn hoặc bằng 14 xung thì TS mức 1 và TL lên mức 1

testbench khối timer

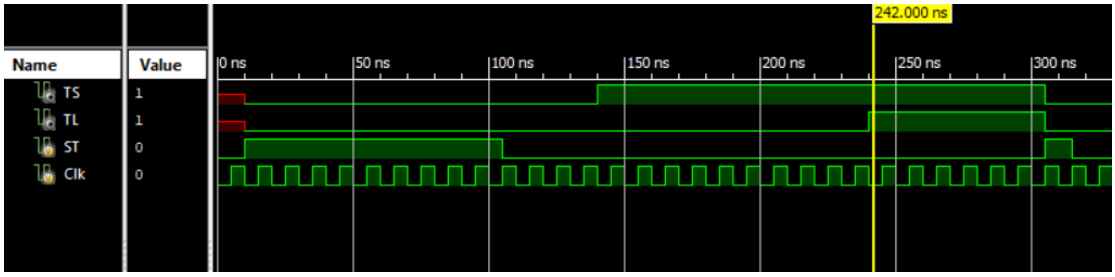
TH1: Khi ST tác động cạnh lên hoặc mức 1 thì timer không đếm đầu ra out mong đợi TL =0, TS=0



TH2: Khi ST mức 0 timer bắt đầu đếm khi xung clk tác động lên 4 lần thì mong đợi TS=1 và TL=0



TH3 Khi ST mức 0 và xung clk tác động lên 14 lần thì mong đợi TS=1 và TL=1



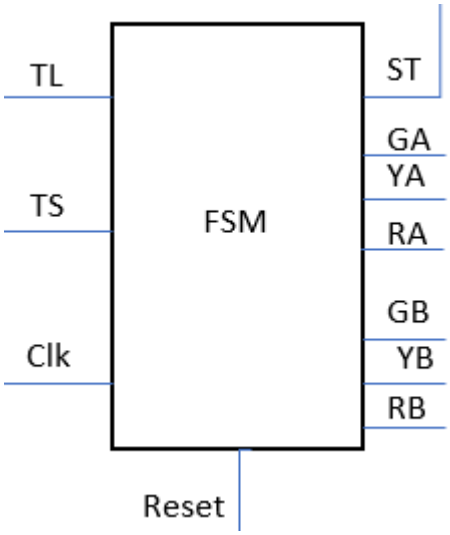
b.Khối fsm:

input

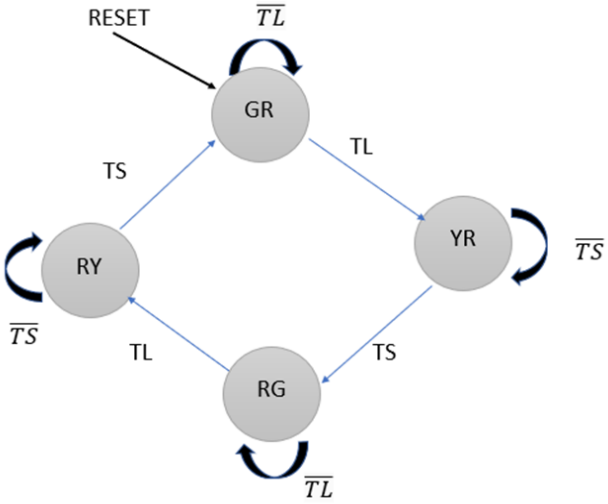
- TS và TL: những tín hiệu đầu vào làm thay đổi case
- reset: Tín hiệu reset để khởi tạo FSM.
- Clk: xung clock

output

- RA, YA, GA, RB, YB và GB (đại diện cho từng đèn riêng lẻ).
- ST chân đưa về reset bộ đếm Module timer



Khối hoạt động



-GR (Xanh-Đỏ): Trạng thái ban đầu khi reset.

Nếu TL true chuyển sang case YR

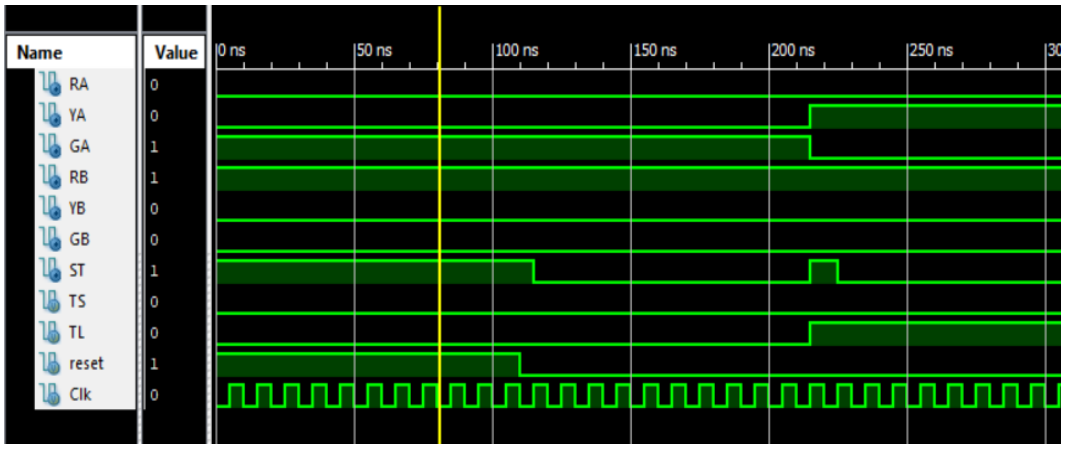
-YR (Vàng-Đỏ): Nếu TS true chuyển sang case RG

-RG (Đỏ-Xanh): Nếu TL true chuyển sang case RY

-RY (Đỏ-Vàng):Nếu TS true quay về case GR

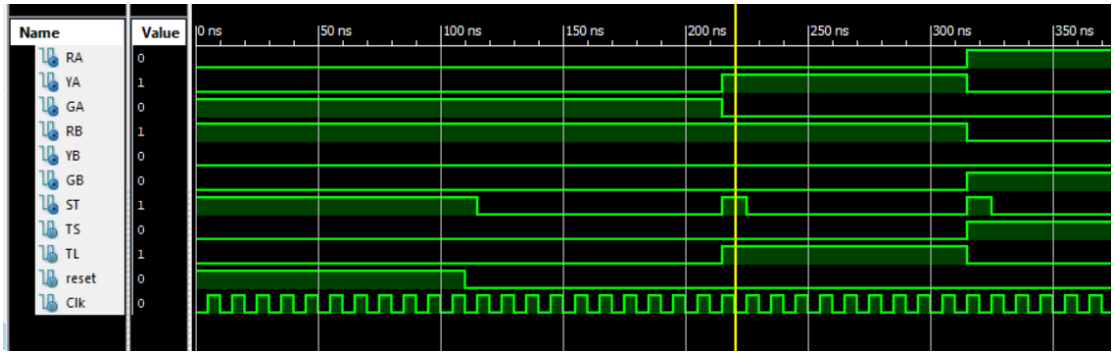
testbench khối fsm

a.TH1 Khi reset tác động mong đợi GA =1, RB =1 và ST=1 thuộc case GR



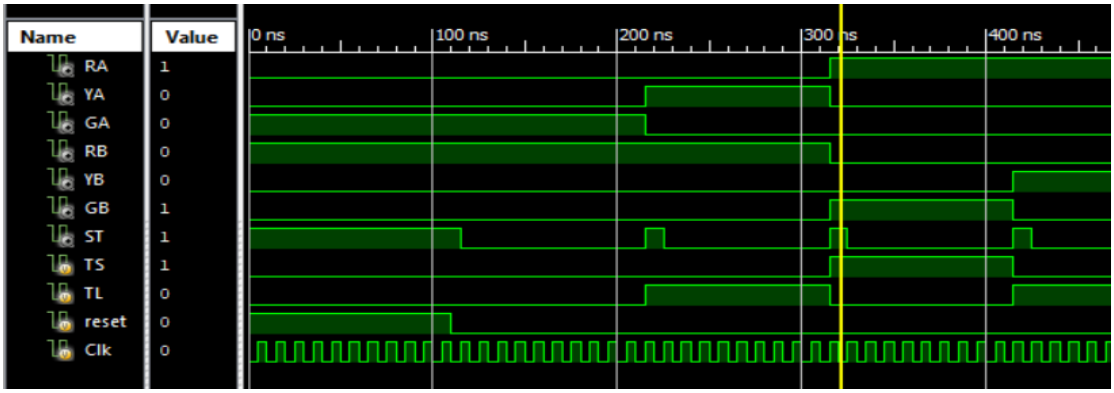
b.TH2 Khi TS=0,TL=1 mong đợi YA =1, RB =1 thuộc case YG

ST=1 tồn tại trong 1 chu kỳ xung đầu khi TL thay đổi(để đưa vào chân reset timer)



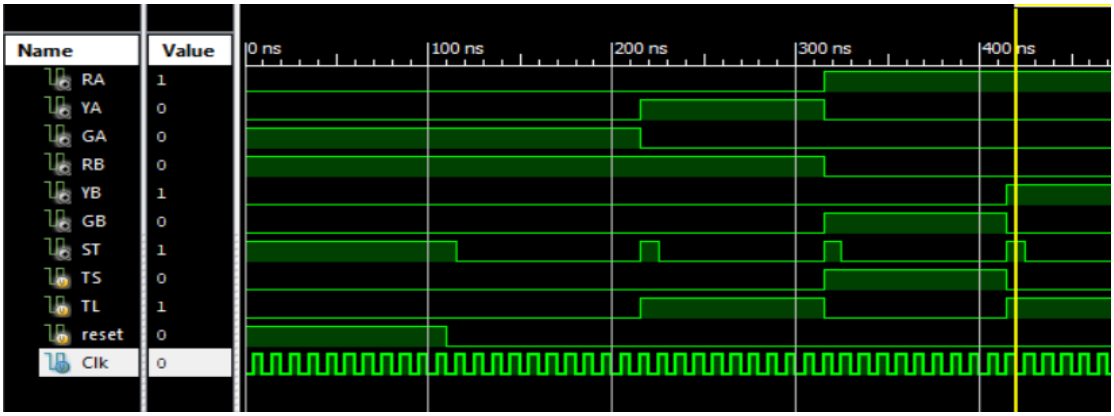
c.TH3 Khi TS=1,TL=0 mong đợi RA =1, GB =1 và thuộc case RG

ST=1 tồn tại trong 1 chu kỳ xung clock đầu khi TS thay đổi(để đưa vào chân reset timer)



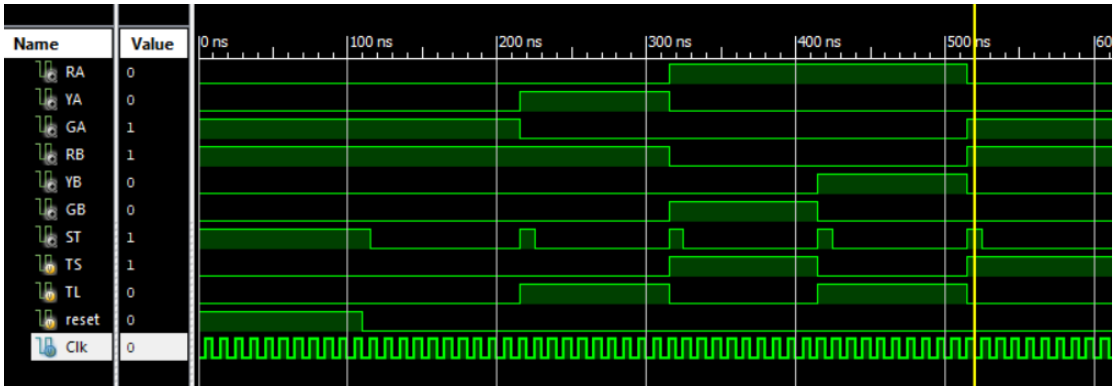
d.TH4 Khi TS=0,TL=1 mong đợi RA =1, GB =1 và thuộc case RY

ST=1 tồn tại trong 1 chu kỳ xungclock đầu khi TL thay đổi(để đưa vào chân reset timer)



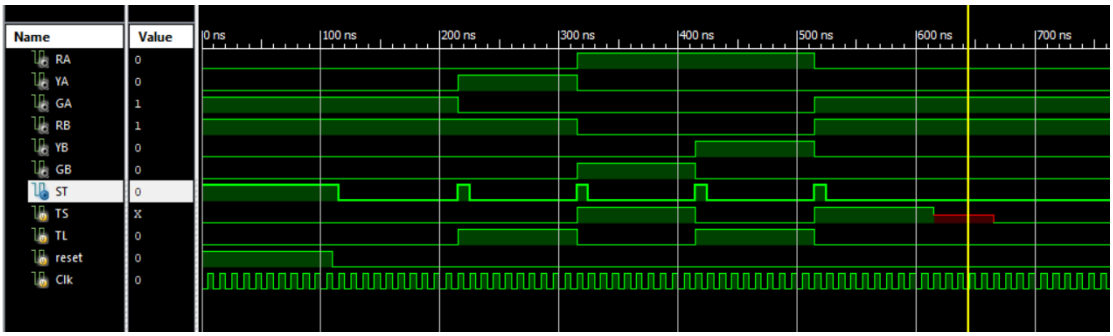
e.TH5 Khi TS=1,TL=0 mong đợi RA =1, GB =1 và thuộc case GR quay về trạng thái ban đầu

ST=1 tồn tại trong 1 chu kỳ xung clock đầu khi TS thay đổi(để đưa vào chân reset timer)



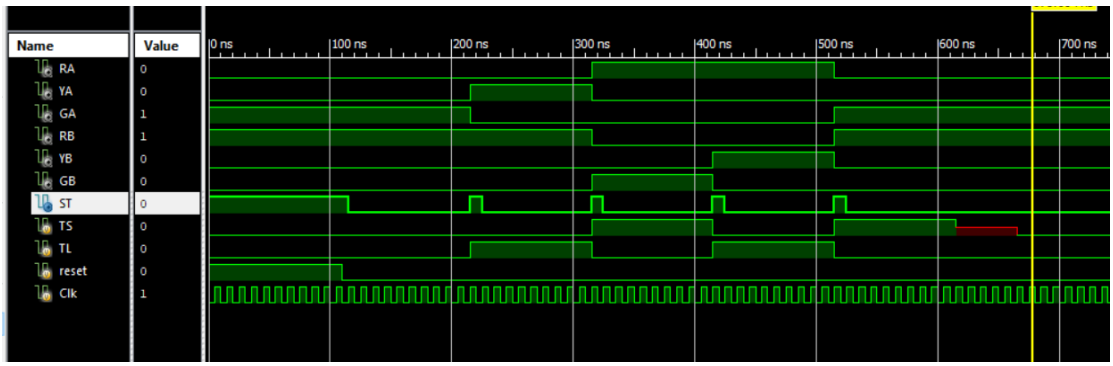
f.TH6 Khi TS=X,TL=0 mong đợi RA =1, GB =1 và thuộc case GR

ST=0 chưa chuyển trường hợp case do TS chưa lên mức 1

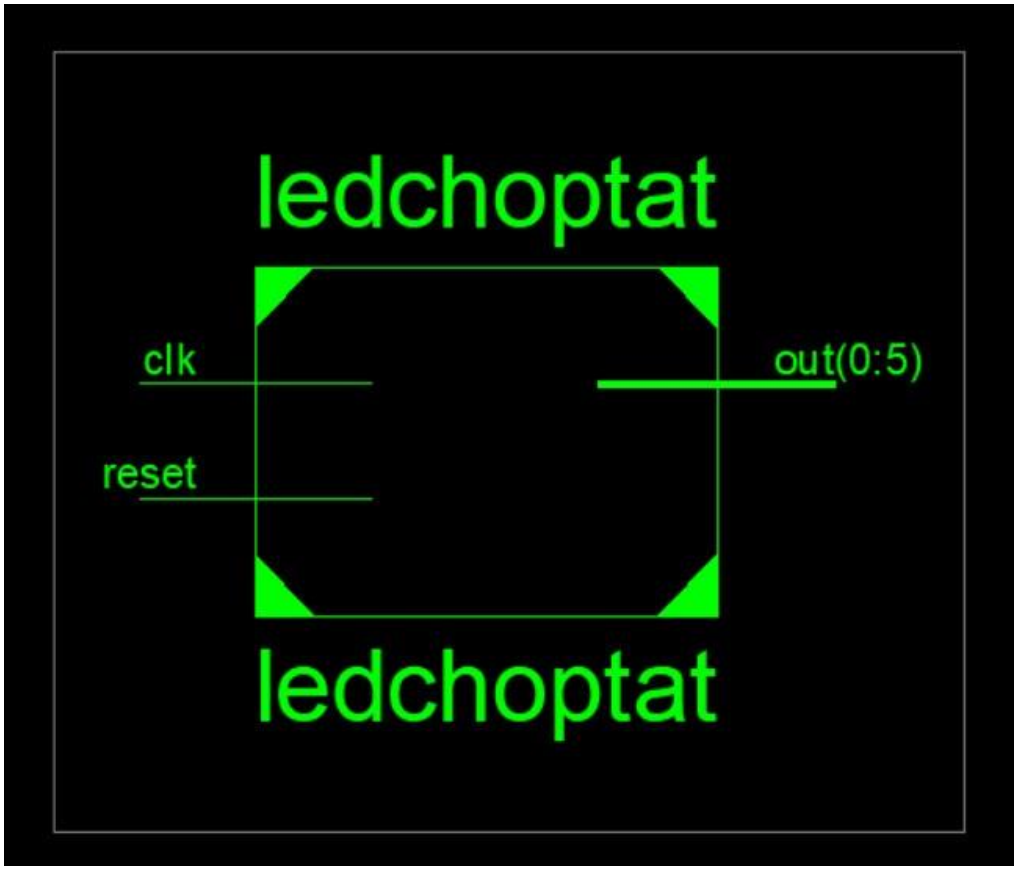


f.TH7 Khi TS=0,TL=0 mong đợi RA =1, GB =1 và thuộc case GR

ST= 0 chưa chuyển trường hợp case do TS chưa lên mức 1



3.2.2 Module ledchoptat



Kết nối

input clk: Cổng vào 1 bit xung clk.

input reset: Cổng vào reset 1 bit.

output [5:0]: Ngõ ra 6 bit (Đỏ2, Vàng2, Xanh2, Đỏ1, Vàng1, Xanh1).

Nguyên lí hoạt động: Chỉ có đèn vàng chớp tắt liên tục theo chu kì gấp đôi chu kì của xung clk, các đèn còn lại sẽ tắt. Khi reset được tác động thì trong bất kì trường hợp nào, 6 đèn đều sẽ tắt.

Bảng sự thật

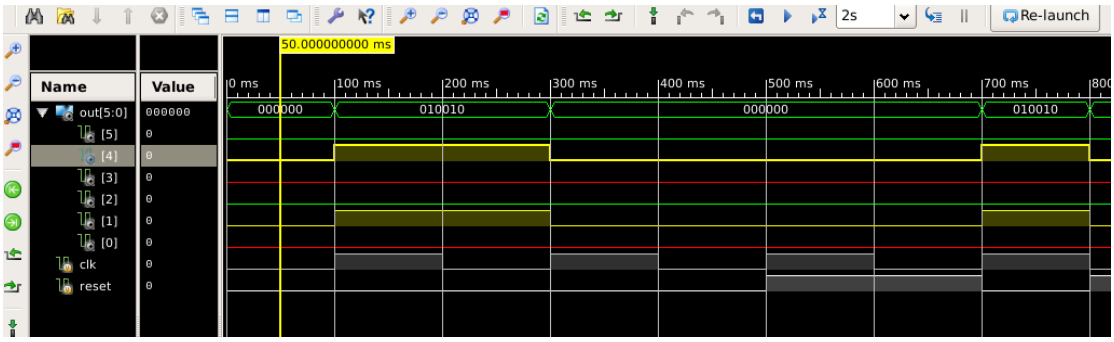
clk	reset	out[5:0]
X	1	0
↑	0	out[0] = 0 out[1] = toggle out[2] = 0 out[3] = 0

		out[4] = toggle out[5] = 0

Kết quả dạng sóng

Ban đầu

Khi xung clk = 0, reset = 0 (không có tín hiệu reset) thì out[5:0] = 0 nghĩa là 6 đèn đều tắt.



CASE 1

Khi reset = 0 (không có tín hiệu reset) và xung clk tích cực cạnh lên thì tín hiệu out[1], out[4] (VÀNG 1, VÀNG 2) đảo trạng thái so với trạng thái trước đó nên sáng đèn, còn các tín hiệu out còn lại bằng 0 nên vẫn tắt:

out[0] = 0

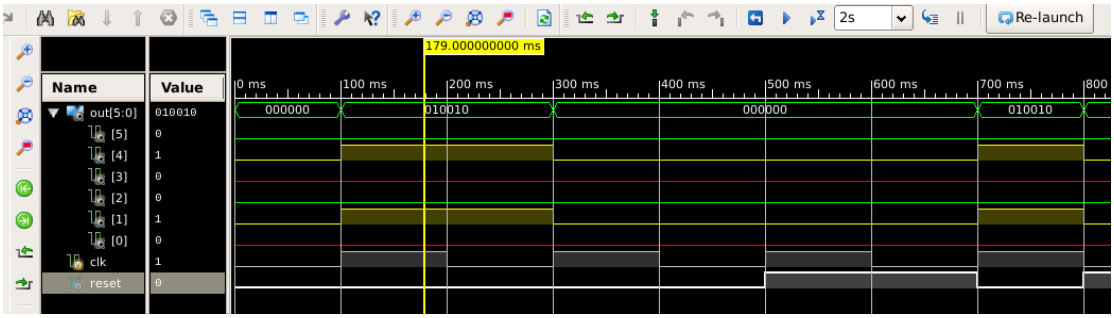
out[1] = 1

out[2] = 0

out[3] = 0

out[4] = 1

out[5] = 0



Khi reset = 0 (không có tín hiệu reset) và xung clk tích cực cạnh xuống hoặc clk=0 thì các tín hiệu out giữ nguyên trạng thái trước đó:

out[0] = 0

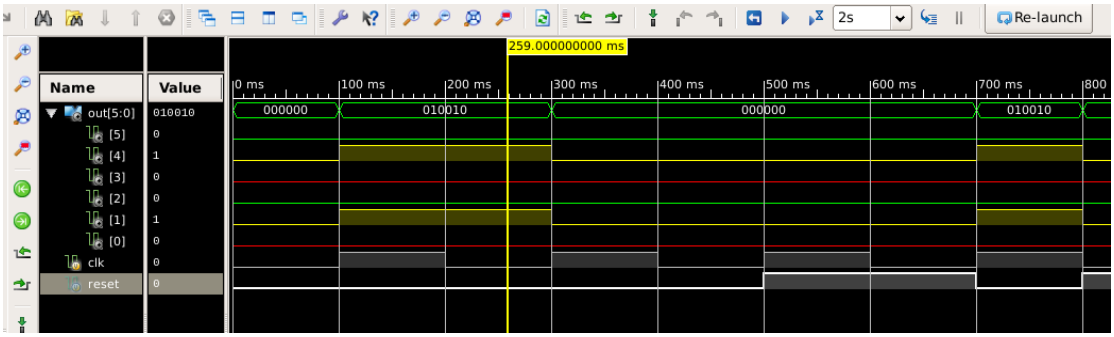
out[1] = 1

out[2] = 0

out[3] = 0

out[4] = 1

out[5] = 0



Khi reset = 0 (không có tín hiệu reset) và xung clk tích cực cạnh lên lần tiếp theo thì chỉ có tín hiệu out[1], out[4] (VÀNG 1, VÀNG 2) đảo trạng thái so với trạng thái trước đó nên tắt, còn các tín hiệu out còn lại bằng 0 nên vẫn tắt:

out[0] = 0

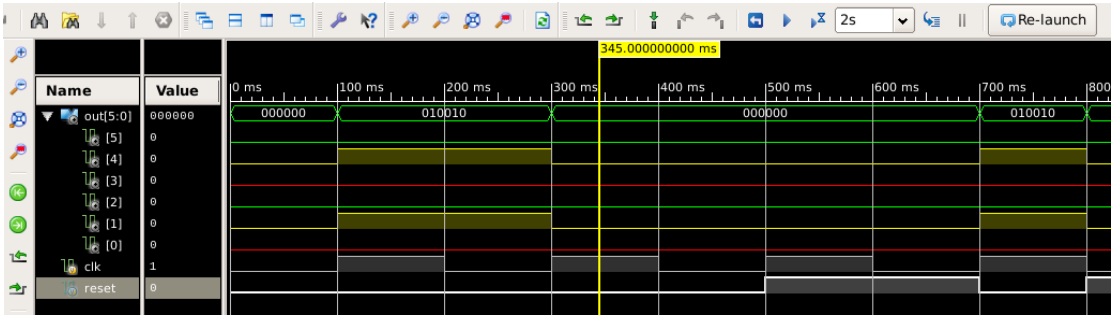
out[1] = 0

out[2] = 0

out[3] = 0

out[4] = 0

out[5] = 0



CASE 2

Khi tín hiệu reset = 1 (có tín hiệu reset), thì tín hiệu reset có sự thay đổi từ 0 nhảy lên 1, nên mặc dù xung clk tích cực cạnh lên thì tín hiệu out[1], out[4] (VÀNG 1, VÀNG 2) bằng 0 và các tín hiệu out còn lại bằng 0, nên 6 đèn đều tắt:

out[0] = 0

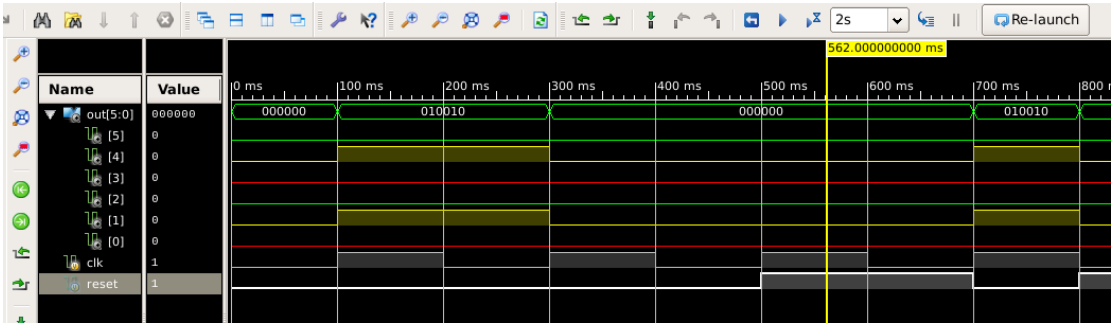
out[1] = 0

out[2] = 0

out[3] = 0

out[4] = 0

out[5] = 0



Khi tín hiệu reset = 1 (có tín hiệu reset), thì tín hiệu reset có sự thay đổi từ 0 nhảy lên 1, nên mặc dù tín hiệu out[1], out[4] (VÀNG 1, VÀNG 2) trước đó đang bằng 1 thì ngay lập tức bằng 0 và các tín hiệu out còn lại bằng 0, nên 6 đèn đều tắt:

out[0] = 0

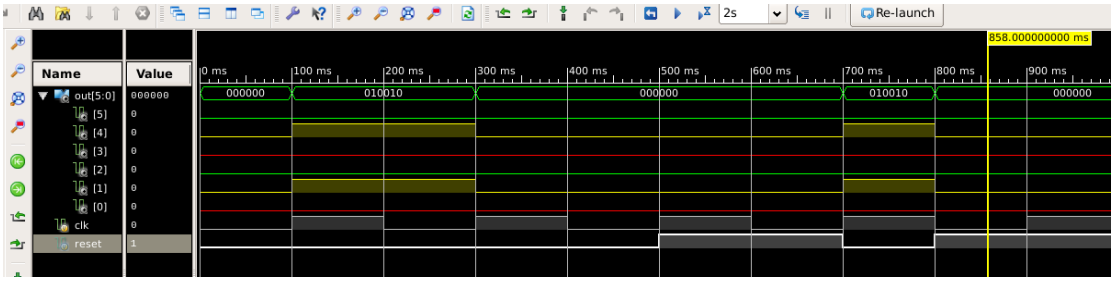
out[1] = 0

out[2] = 0

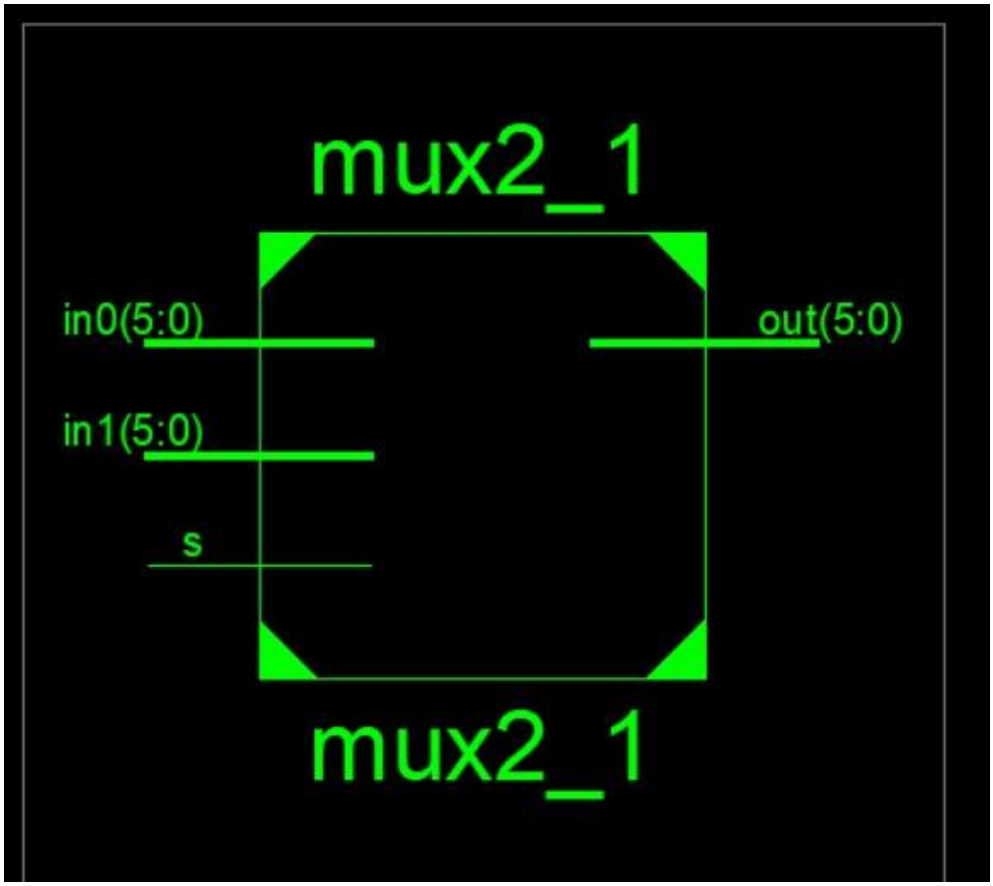
out[3] = 0

out[4] = 0

out[5] = 0



3.2.3 Module Mux 2_1



- input [5:0] in0 Cổng vào 6 bit đầu tiên.
- input [5:0] in1 Cổng vào 6 bit thứ hai.
- input s Cổng điều khiển 1 bit.
- output [5:0] out Cổng ra 6 bit.

Bảng sự thật

in0[5:0]	in1[5:0]	s	out[5:0]
in0	in1	0	in0
in0	in1	1	in1

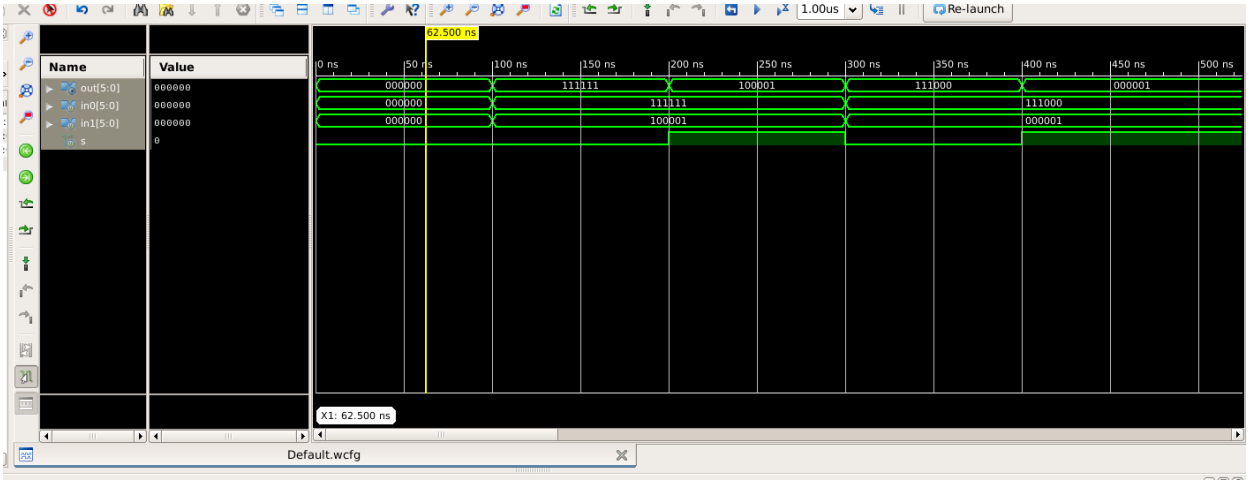
Kết quả dạng sóng

Ban đầu

Khi in0 = 0 và in1 = 0

Chân điều khiển s = 0

Ngõ ra là out = in0 = 0

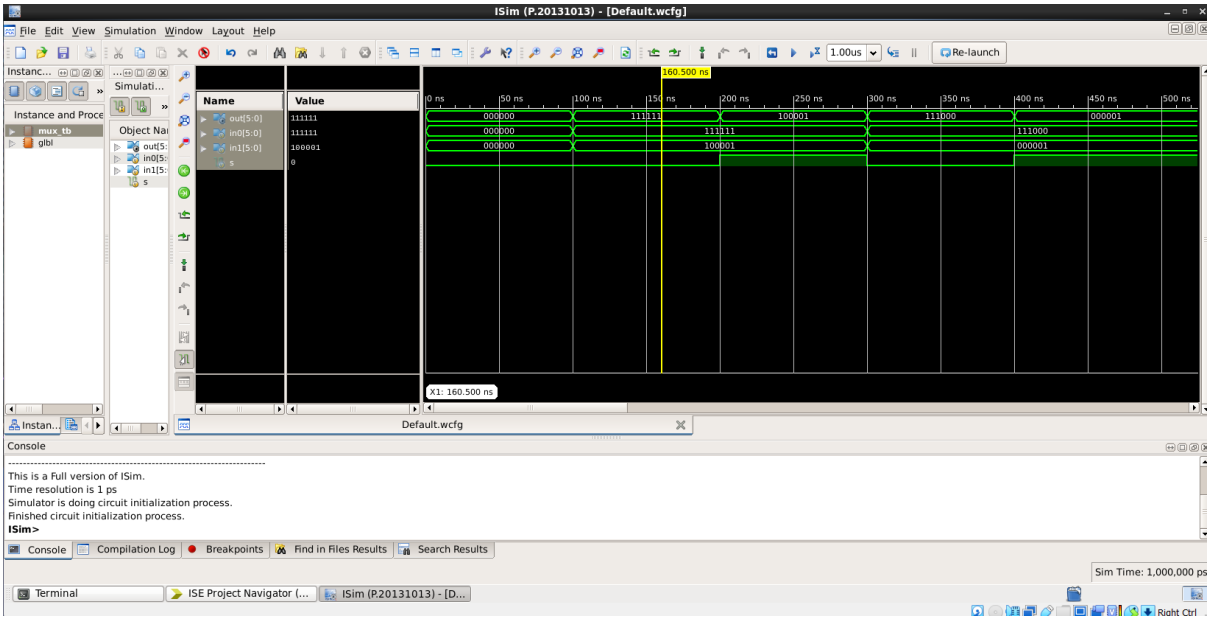


Case1

Khi in0 = 111111 và in1 = 100001

Chân điều khiển s = 0

Ngõ ra là out = in0 =111111

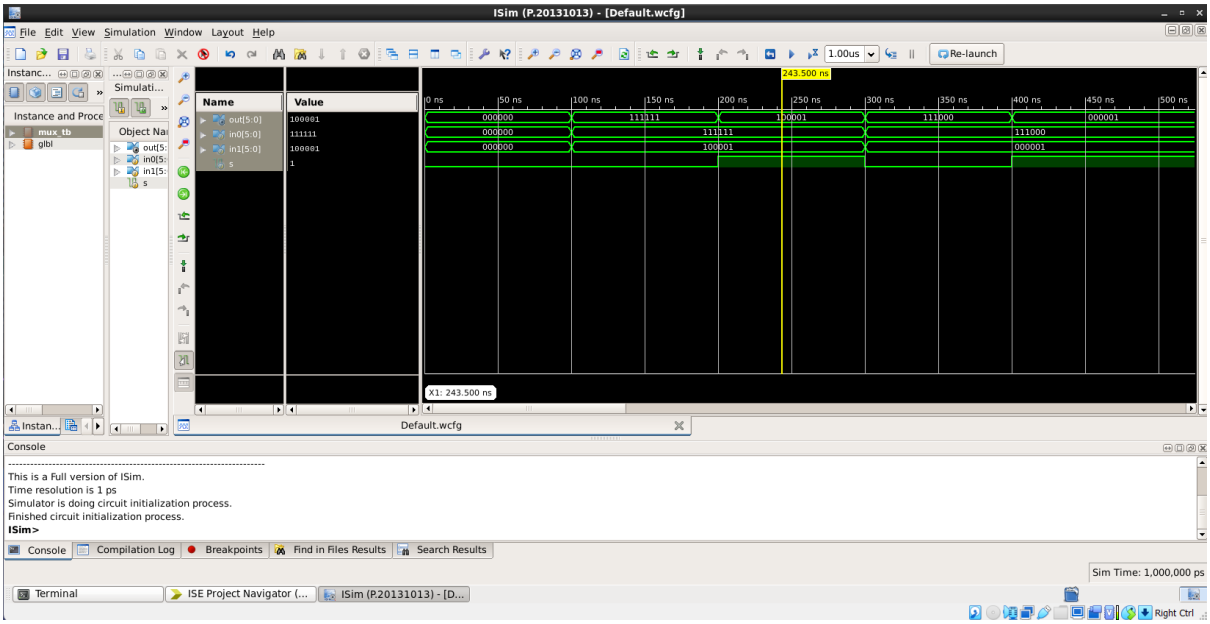


Case2

Khi in0 = 111111 và in1 = 100001

Chân điều khiển s = 1

Ngõ ra là out = in1 = 100001

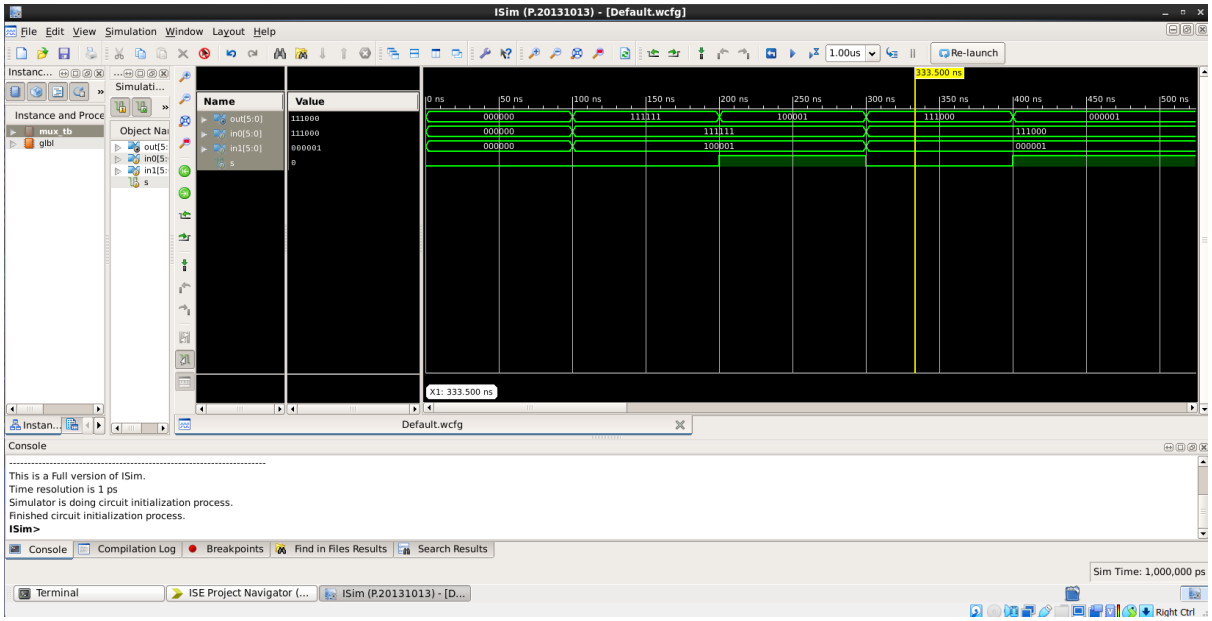


Case 3

Khi in0 = 111000 và in1 = 000001

Chân điều khiển s = 0

Ngõ ra là out = in0 = 111000

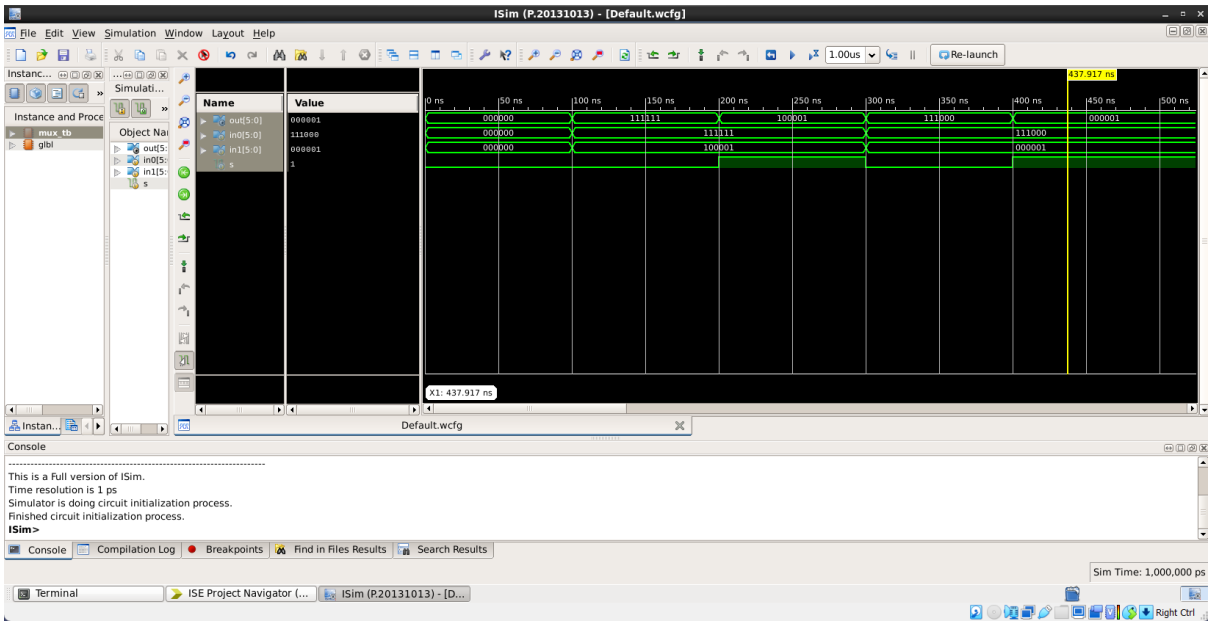


Case4

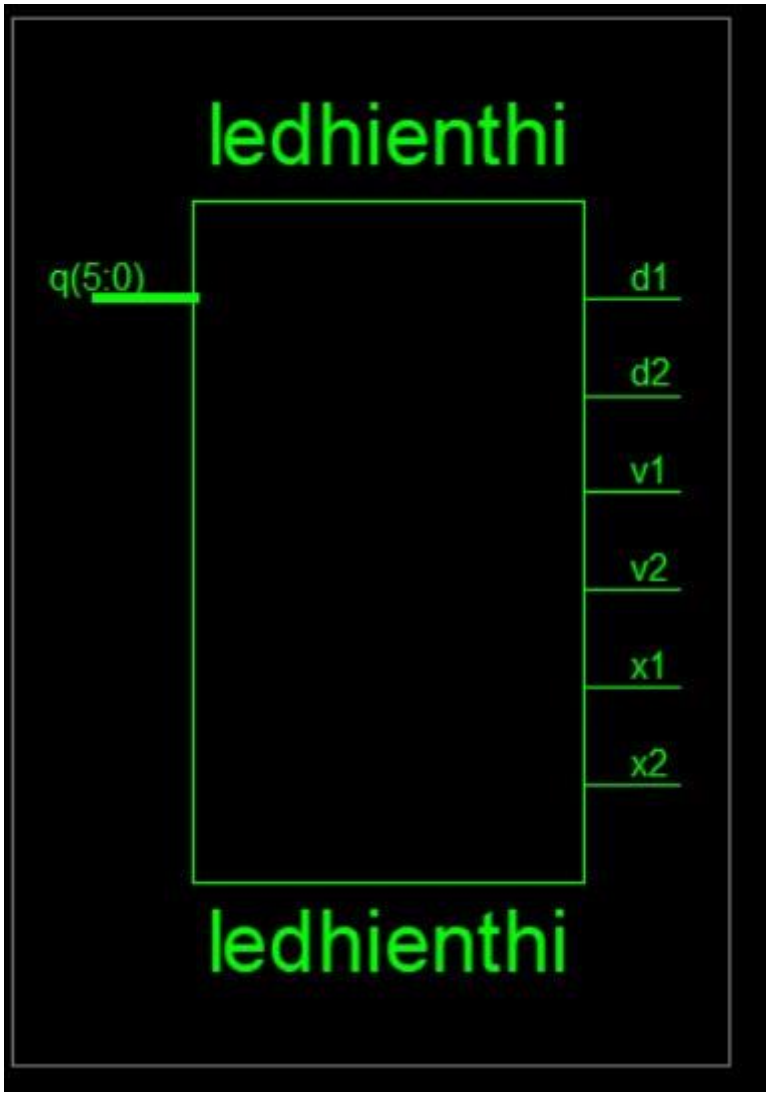
Khi in0 = 111000 và in1 = 000001

Chân điều khiển s = 1

Ngõ ra là out = in1 = 000001



3.2.4 Module ledhienthi



INPUT là ngõ ra out(5:0) của module mux2_1, từ ngõ ra vào q(5:0) hiển thị ra ngõ ra 6 bit

Lấy tín hiệu từ ngõ ra Mux2_1

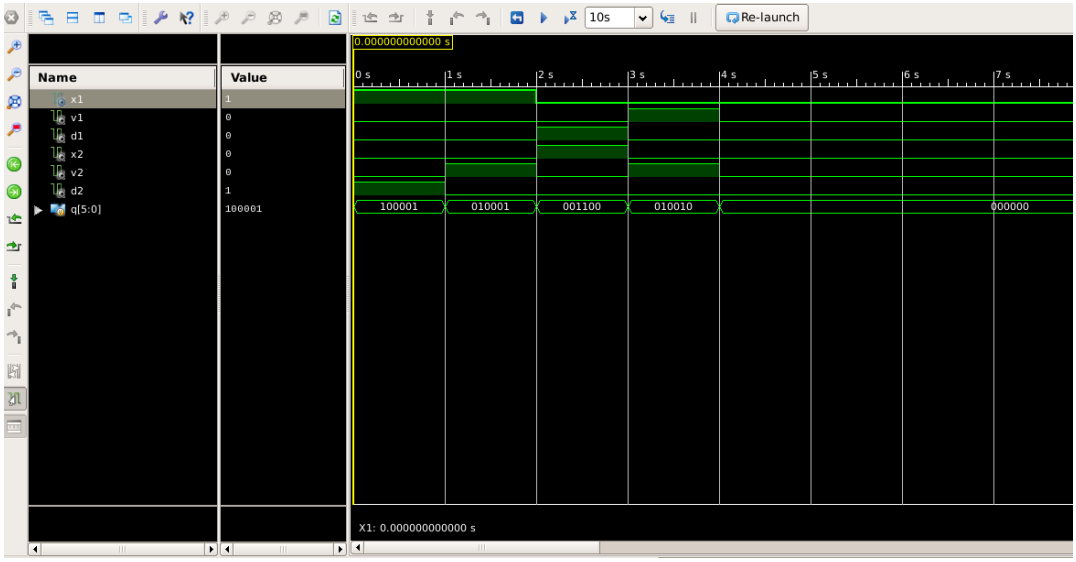
Case1: khi INPUT là q(5:0) = 100001, đèn 1 đèn xanh, đèn 2 đèn đỏ

Case2: Khi INPUT là q(5;0) = 010001, đèn 1 đèn vàng, đèn 2 đèn đỏ

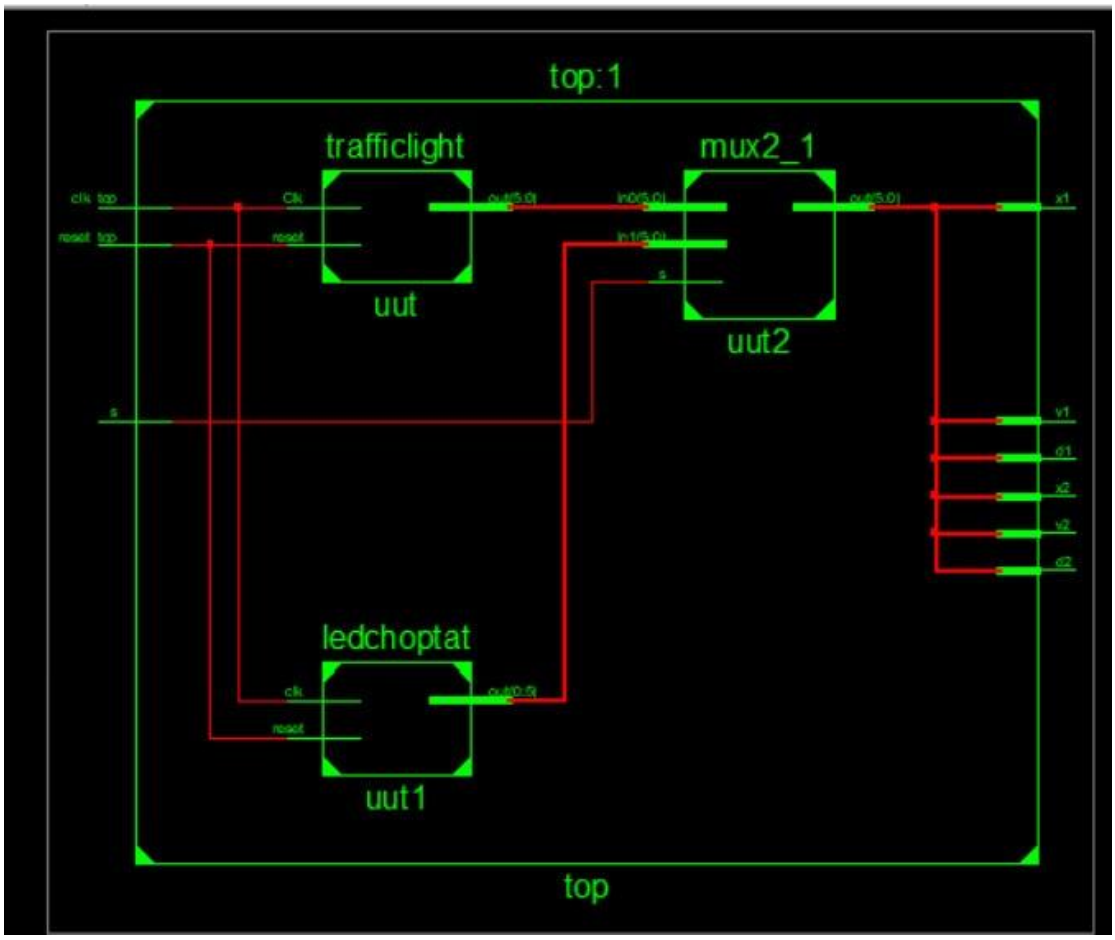
Case3: khi INPUT là q(5:0) = 001100, đèn 1 đèn đỏ, đèn 2 đèn xanh

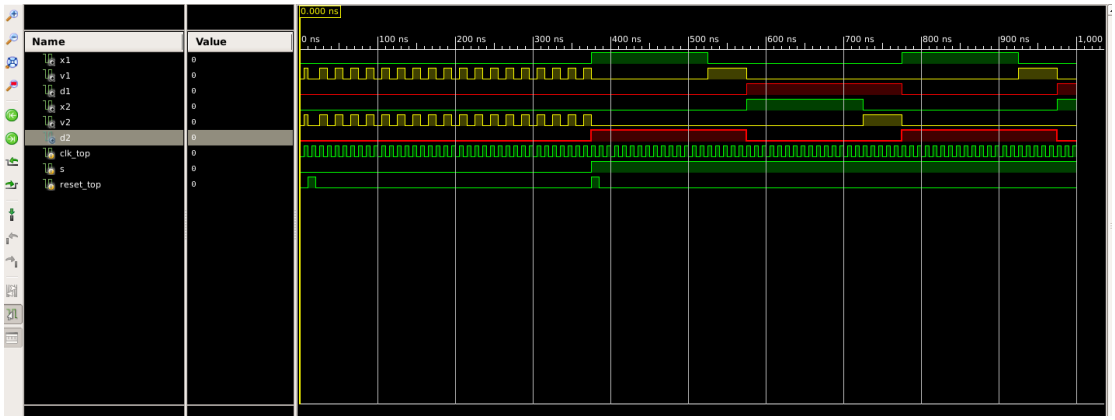
Case4: Khi INPUT là q(5;0) = 010010, Khi ngõ ra mux2_1 là 2 đèn vàng cùng sáng

Case5: Khi INPUT là q(5;0) = 000000, Khi ngõ ra mux2_1 là 2 đèn vàng cùng tắt


Re-launch
10s
Name
Value
x1
v1
d1
x2
v2
d2
q[5:0]
1
0
0
0
0
1
100001
0.000000000000 s
0 s
1 s
2 s
3 s
4 s
5 s
6 s
7 s
100001
010001
001100
010010
000000
X1: 0.000000000000 s

3.2.6 Module top





Các ngõ vào clk_top, s, reset_top

Ngõ ra x1, v1, d1, x2, v2, d2

Khởi tạo in1, in2, q và khai báo chúng ở kiểu dữ liệu wire

Chức năng các module gọi ra trong module top:

Module traficlight: tạo ra tín hiệu đèn giao thông các trạng thái của đèn giao thông

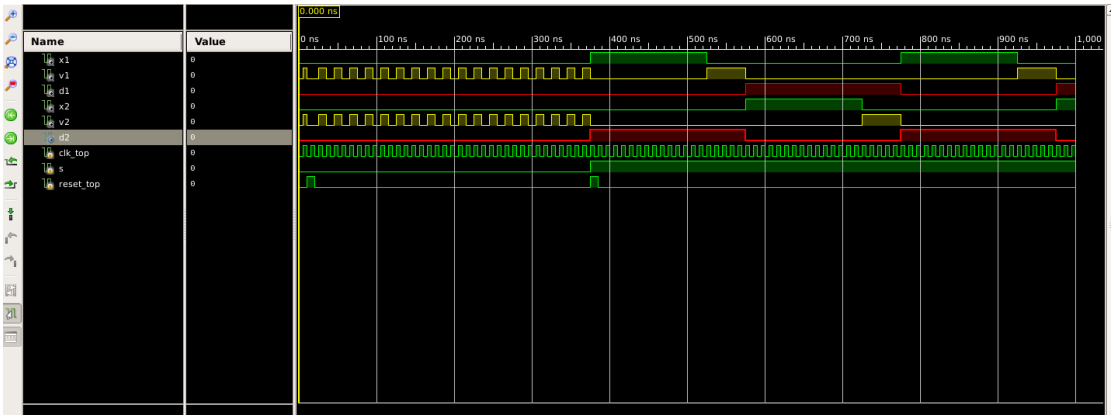
Module ledchoptat: tạo đèn vàng chớp tắt khi không sử dụng xanh đỏ vàng đèn giao thông

Module Mux2_1: Lựa chọn trạng thái khi s=0 thì chọn ngõ vào là trafficlight đèn giao thông sáng bình thường, còn khi s=1 thì chọn ngõ vào là led vàng chớp tắt

Module hiển thị hiển thị: Hiển thị ra trạng thái của đèn giao thông khi sáng bình thường và sáng chớp tắt

3.2.7 Module test

Kết quả



Kết nối các khối lại với nhau.

Ngõ vào là clk_top, reset_top, s

Các ngõ ra x1, v1, d1, x2, v2, d2

Ban đầu kích 1 xung reset_Top lên 1 rồi về 0 sau đó delay 355ns đặt trạng thái lúc này s=1 nên ngõ ra hiển thị led vàng chớp tắt, đèn vàng chớp tắt theo chu kỳ xung clk, sau mỗi 1 chu kỳ xung clk đèn vàng sáng, 1 chu kỳ clk tiếp theo đèn vàng tắt, rồi tiếp tục lặp lại như thế.

Sau đó kích tiếp tục 1 xung reset_Top. Đặt trạng thái s=0 ngõ ra hiển thị trạng thái sáng tắt của đèn giao thông, khi đèn xanh1 sáng trong 15 chu kỳ xung ckl

đèn đỏ2 cũng sáng, tiếp theo đèn vàng1 sáng trong 5 chu kỳ xung clk đèn xanh1 ở trạng thái tắt, đỏ2 sáng tổng cộng 20 chu kỳ xung clk. sau đó tiếp tục lặp lại như trên. Khi nào tác động xung reset_top và đặt trạng thái s=1 cả 2 đèn giao thông hiển thị đèn vàng chớp tắt.

clk	s	x1	v1	d1	x2	v2	d2
↑	1	0	1	0	0	1	0
↑	1	0	0	0	0	0	0
…	…	…	…	…	…	…	…
↑	0	1	0	0	0	0	1
↑	0	1	0	0	0	0	1
…(15 chu kỳ clk)	…	…	…	…	…	…	…
↑	0	0	1	0	0	0	1
↑	0	0	1	0	0	0	1
…(5 chu kỳ clk)	…	…	…	…	…	…	…
↑	0	0	0	1	1	0	0
↑	0	0	0	1	1	0	0
…(15 chu kỳ clk)	…	…	…	…	…	…	…
↑	0	0	0	1	0	1	0
↑	0	0	0	1	0	1	0
…(5 chu kỳ	…	…	…	…	…	…	…

clk)							
↑	1	0	0	0	0	0	1
↑	1	0	0	0	0	0	1

CHƯƠNG 4: KẾT LUẬN

Sau thời giam tìm hiểu và thực hiện đề tài “ Mạch đèn giao thông”, nhóm đã đạt được những kết quả như :

Tìm hiểu được cách sử dụng phần mềm xilinx ise, thực hiện được mạch đèn giao thông bằng Verilog

Việc thiết kế mạch đèn giao thông thông minh bằng Verilog là một giải pháp hiệu quả cho vấn đề giao thông.

Mạch đèn giao thông thông minh có nhiều ưu điểm so với mạch đèn giao thông truyền thống.

Kết quả nghiên cứu đã cho thấy mạch đèn giao thông thông minh được thiết kế bằng Verilog có thể hoạt động hiệu quả.

Việc ứng dụng mạch đèn giao thông thông minh vào thực tế sẽ góp phần cải thiện tình hình giao thông, nâng cao ý thức chấp hành luật giao thông và bảo vệ môi trường.

Với những ưu điểm và tiềm năng ứng dụng to lớn, việc nghiên cứu và phát triển mạch đèn giao thông thông minh bằng Verilog là một hướng đi đầy hứa hẹn cho việc giải quyết vấn đề giao thông hiệu quả.

Phụ Lục

Code:

Module ledhienthi

```
module ledhienthi(
    input [5:0] q,
    output x1,v1,d1,x2,v2,d2
    );
assign x1 = q[0];
assign v1 = q[1];
assign d1 = q[2];
assign x2 = q[3];
assign v2 = q[4];
assign d2 = q[5];
endmodule
```

Module mux2_1

```
module mux2_1(
    input [5:0] in0,
    input [5:0] in1,
    input s,
    output [5:0] out
    );

reg [5:0] y=0;
always @(*)
begin
y= s ? in1 : in0;
end
assign out = y;
```

```
endmodule
```

Module Vang_chop_tat

```
`timescale 100ms / 1ms
module vang_chop_tat(
   input clk,
        input reset,
   output reg [0:5] out
   );


        initial out = 0;
        always @(posedge clk, posedge reset)
        if (reset)
                begin
                        out[1] = 0;                     //VANG1
                        out[4] = 0;                     //VANG2
                end
        else
                begin
                        out[1] = ~out[1];       //VANG1
                        out[4] = ~out[4];       //VANG2
                end
endmodule
```

Module trafficlight

```
`timescale 1ns / 1ps

module trafficlight (
output [5:0] out,
input reset,
```

```
input Clk
);
timer part1 (TS, TL, ST, Clk);
fsm part2(RA, YA, GA, RB, YB, GB, ST, TS, TL, reset, Clk);
assign out[0]= GA;
assign out[1]= YA;
assign out[2]= RA ;
assign out[3]= GB ;
assign out[4]= YB ;
assign out[5]= RB ;
endmodule

module timer(
output TS,
output TL,
input ST,
input Clk
);
integer value;
assign TS  = (value>=4);
assign TL  = (value>=14);
always@(posedge ST or posedge Clk)
begin
if(ST==1)begin
value=0;
end
else begin
value=value+1;
end
```

```
end
endmodule

module fsm(
output RA,
output YA,
output GA,
output RB,
output YB,
output GB,
output reg ST,
input TS,
input TL,
input reset,
input Clk
);
reg [6:1] state;
parameter GR= 6'b001100;
parameter YR= 6'b010100;
parameter RG= 6'b100001;
parameter RY= 6'b100010;
assign RA = state[6];
assign YA = state[5];
assign GA = state[4];
assign RB = state[3];
assign YB = state[2];
assign GB = state[1];
initial begin state = GR; ST = 1; end
always @(posedge Clk)
```

```
begin
if (reset)
begin state = GR; ST = 1; end
else
begin
ST = 0;
case (state)
GR:
        if (TL) begin state = YR; ST = 1; end
YR:
        if (TS) begin state = RG; ST = 1; end
RG:
        if (TL) begin state = RY; ST = 1; end
RY:
        if (TS) begin state = GR; ST = 1; end

endcase
end
end
endmodule
```

Module TOP

```
module top(
   input clk_top,
   input s,
   input reset_top,
   output x1,v1,d1,x2,v2,d2
   );
        wire [5:0] in1,in2,q;
trafficlight b_ngoc (
```

```
                .out(in1),
                .Clk(clk_top),
                .reset(reset_top)
        );
ledchoptat v_anh (
                .clk(clk_top),
                .reset(reset_top),
                .out(in2)
        );
mux2_1 l_nhan (
                .in0(in1),
                .in1(in2),
                .s(s),
                .out(q)
        );
ledhienthi q_dung (
                .q(q),
                .x1(x1),
                .v1(v1),
                .d1(d1),
                .x2(x2),
                .v2(v2),
                .d2(d2)
        );

Endmodule
```

Testbench

Testbench trafficlight

```
`timescale 1ns / 1ps
module trafficlight_tb;

	// Inputs
	reg reset;
	reg Clk;

	// Outputs
	wire [5:0] q;
	// Instantiate the Unit Under Test (UUT)
	trafficlight uut (
		.out(q),
		.reset(reset),
		.Clk(Clk)
	);
always #5 Clk=~Clk;
	initial begin
		// Initialize Inputs
		Clk = 0;
		reset = 1;



		#5;
		reset = 0;

		#700;
		reset = 1;
```

```
            #100;
            reset = 0;


        end

endmodule
```

testbench timer

```
module timer_tb;

        // Inputs
        reg ST;
        reg Clk;

        // Outputs
        wire TS;
        wire TL;

        // Instantiate the Unit Under Test (UUT)
        timer uut (
            .TS(TS),
            .TL(TL),
            .ST(ST),
            .Clk(Clk)
        );
always #5 Clk=~Clk;
        initial begin
            // Initialize Inputs
```

```
        ST = 0;
        Clk = 0;

        #10;
        ST = 1;
        // th1 timer chua dem TL,TS muc 0
        #100;
        ST=0;
        // timer bat đau đem
        #200;
         ST = 1;
        // reset timer

        #10;
        ST=0;
        // timer bat đau đem
        #150;
        ST = 1;

        // reset timer

    end
```

testbench fsm

```
module fsm_tb;

    // Inputs
    reg TS;
    reg TL;
    reg reset;
```

```
        reg Clk;

        // Outputs
        wire RA;
        wire YA;
        wire GA;
        wire RB;
        wire YB;
        wire GB;
        wire ST;

        // Instantiate the Unit Under Test (UUT)
        fsm uut (
                .RA(RA),
                .YA(YA),
                .GA(GA),
                .RB(RB),
                .YB(YB),
                .GB(GB),
                .ST(ST),
                .TS(TS),
                .TL(TL),
                .reset(reset),
                .Clk(Clk)
        );
always #5 Clk=~Clk;
        initial begin
                // Initialize Inputs
                TS = 0;
```

```
            TL = 0;
            reset = 1;
            Clk = 0;
            #10;

            reset = 1;
            #100;

            reset = 0;
            #105;
            TS = 0;
            TL = 1;
    #100;
            TS = 1;
            TL = 0;
    #100;
            TS = 0;
            TL = 1;
    #100;
            TS = 1;
            TL = 0;
    #100;


        end

endmodule
```

TestBench top

```
module top_test;
```

```
        // Inputs
        reg clk_top;
        reg s;
        reg reset_top;

        // Outputs
        wire x1;
        wire v1;
        wire d1;
        wire x2;
        wire v2;
        wire d2;

        // Instantiate the Unit Under Test (UUT)
        top uut (
                .clk_top(clk_top),
                .s(s),
                .reset_top(reset_top),
                .x1(x1),
                .v1(v1),
                .d1(d1),
                .x2(x2),
                .v2(v2),
                .d2(d2)
        );
always #5 clk_top = ~clk_top;
        initial begin
                // Initialize Inputs
```

```
        clk_top = 0;
        s = 0;
        reset_top = 0;

        // Wait 100 ns for global reset to finish
        #10;
        reset_top = 1;

        // Wait 100 ns for global reset to finish
        #10;
        reset_top = 0;

        // Wait 100 ns for global reset to finish
        #355;
        s=1;
        reset_top = 1;
        #10;
        reset_top = 0;
#300;


        // Add stimulus here

    end

endmodule
```

testbench vang_chop_tat

```
`timescale 100ms / 1ps
module vang_chop_tat_tb;
```

```
// Inputs
reg clk;
reg reset;

// Outputs
wire [5:0] out;

//Tao xung clock
initial begin
	clk = 0;
	forever #1 clk=~clk;
end

// Instantiate the Unit Under Test (UUT)
vang_chop_tat uut (
	.clk(clk),
	.reset(reset),
	.out(out)
);

always begin

	//TESTCASE1
	reset = 0;
	#5;

	//TESTCASE2
	reset = 1'b1;
```

```
            #2;

            reset = 0;
            #1;

            reset = 1'b1;
            #2;

        end

endmodule
```

testbench mux21

```
module mux_tb;
        // Inputs
        reg [5:0] in0;
        reg [5:0] in1;
        reg s;

        // Outputs
        wire [5:0] out;
        // Instantiate the Unit Under Test (UUT) = tạo ra một instance của
module mux
        mux uut (
                .in0(in0),
                .in1(in1),
                .s(s),
                .out(out)
        );
        initial begin
```

```
        in0 = 0;
        in1 = 0;
        s = 0;
        #100;

        //case1
        in0 = 6'b111111;
        in1 = 6'b100001;
        s = 0;
        #100;  //mong muốn out = in0 = 6'b111111

        //case2
        in0 = 6'b111111;
        in1 = 6'b100001;
        s = 1;
        #100;  //mong muốn out = in1 = 6'b100001

        //case3
        in0 = 6'b111000;
        in1 = 6'b000001;
        s = 0;
        #100;  //mong muốn out = in0 = 6'b111000

        //case4
        in0 = 6'b111000;
        in1 = 6'b000001;
        s = 1;   //mong muốn out = in1 = 6'b000001;
end
    endmodule
```

testbench led hiển thị

```
module ledhienthi_test;

	// Inputs
	reg [5:0] q;

	// Outputs
	wire x1;
	wire v1;
	wire d1;
	wire x2;
	wire v2;
	wire d2;

	// Instantiate the Unit Under Test (UUT)
	ledhienthi uut (
		.q(q),
		.x1(x1),
		.v1(v1),
		.d1(d1),
		.x2(x2),
		.v2(v2),
		.d2(d2)
	);

	initial begin

		q=6'b100001;
		#1000000000;
```

```
            q=6'b010001;
            #1000000000;

            q=6'b001100;
            #1000000000;

            q=6'b010010;
        #1000000000;

            q=6'b000000;
            #1000000000;

    end

endmodule
```