TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM

KHOA ĐIỆN – ĐIỆN TỬ

MÔN HỌC: THỰC TẬP THIẾT KẾ HỆ THỐNG VÀ VI MẠCH TÍCH HỢP

KIỂM TRA CUỐI KỲ

GVHD: Thầy Lê Minh Thành

Lớp sáng thứ 6. Tiết 1-5

Mã học phần: ICSL316764

Sinh viên thực hiện: Phan Công Danh

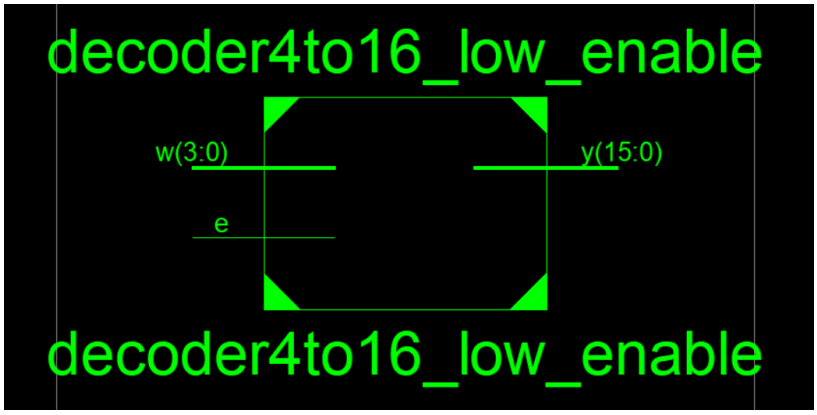
MSSV: 19119160

Nhóm 1

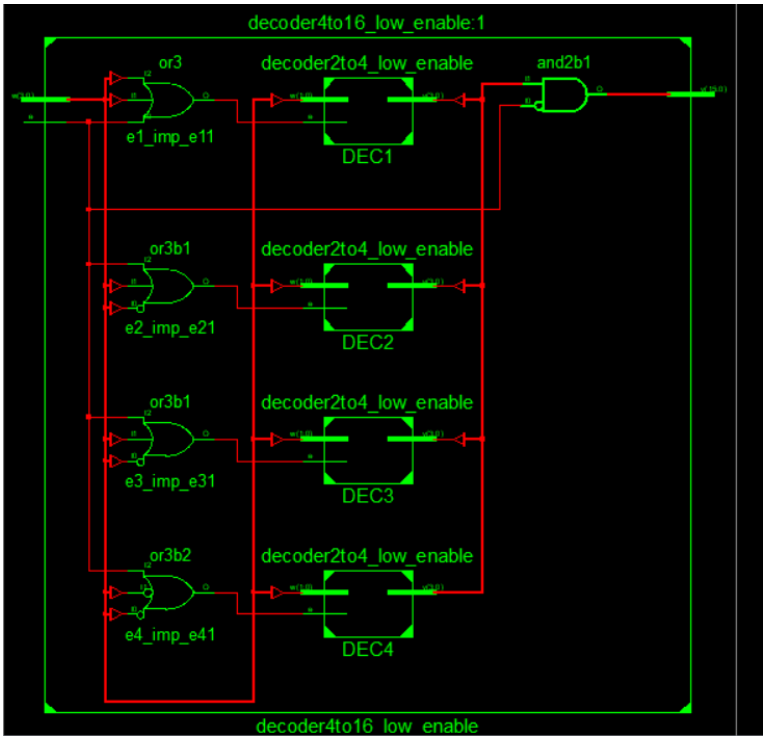
Tp. Hồ Chí Minh, tháng 12 năm 2021

1. Thiết kế mạch giải mã từ 4 ngõ vào sang 16 ngõ ra mức thấp, có ngõ vào cho phép E từ các bộ giải mã từ 2 ngõ vào sang 4 ngõ ra mức thấp có ngõ vào cho phép tích cực mức thấp. Giả sử khi ngõ cho phép không tích cực thì ngõ ra là X.

1.1. RTL Schematic



Sơ đồ khối tổng quát của bộ giải mã có 5 chân ngõ vào là w[3:0], chân enable và 16 ngõ ra y[15:0]



Bao gồm bộ giải mã từ 2 ngõ vào sang 4 ngõ ra mức thấp. Ngõ vào cho phép của 4 bộ này chính là ngõ ra của lần lượt 4 cổng OR như sơ đồ kết nối. Cuối cùng là cổng AND với ngõ vào là chân Enable mức thấp và các ngõ ra của các bộ giải mã

1.2. Instance Code

```
module decoder2to4_low_enable(
  input e,
        input [1:0] w,
        output reg [3:0] y
  );
always @(e,w)
        if (~e)
                case (w)
                        0: y = 4'b1110;
                        1: y = 4'b1101;
                        2: y = 4'b1011;
                        3: y = 4'b0111;
                        default: y = 4'bx;
                endcase
        else
                y = 4'b1111;
endmodule
```

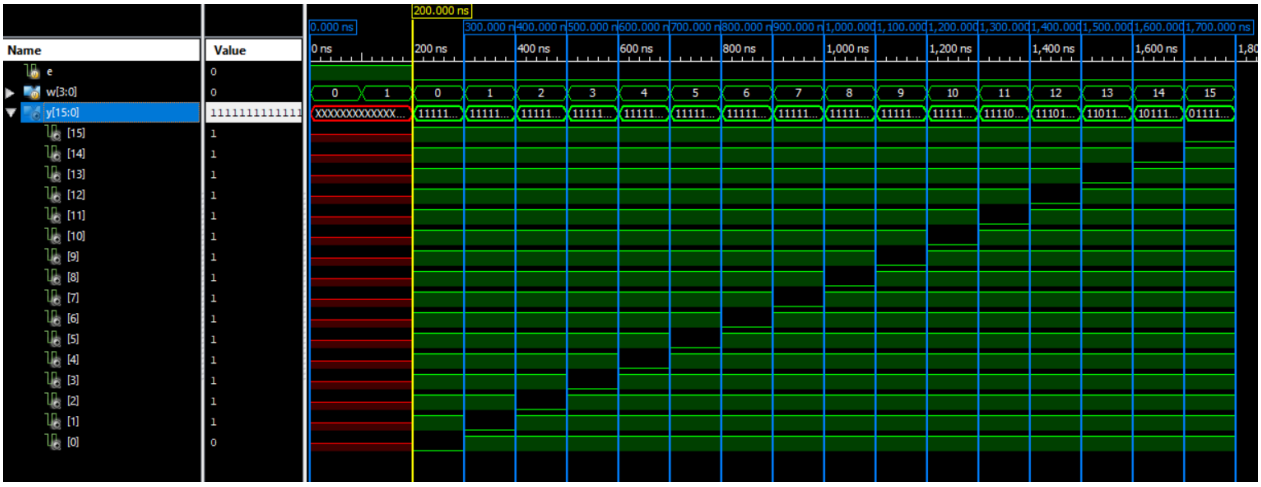
1.3. Top Module Code

```
module decoder4to16_low_enable(
        input e,
        input [3:0] w,
        output [15:0] y
   );
         wire e1, e2, e3, e4;
         wire [15:0] a, b;
assign e1 = e | w[3] | w[2];
assign e2 = e | w[3] | (~w[2]);
assign e3 = e | (~w[3]) | w[2];
assign e4 = e | (~w[3]) | (~w[2]);
decoder2to4_low_enable DEC1 (e1, w[1:0], a[3:0]);
decoder2to4_low_enable DEC2 (e2, w[1:0], a[7:4]);
decoder2to4_low_enable DEC3 (e3, w[1:0], a[11:8]);
decoder2to4_low_enable DEC4 (e4, w[1:0], a[15:12]);
assign b = (e == 0) ? 16'b1111111111111111 : 16'bx;
assign y = a & b;
endmodule
```

1.4. Testbench Code

```
module tb;

	// Inputs
	reg e;
	reg [3:0] w;

	// Outputs
	wire [15:0] y;

	// Instantiate the Unit Under Test (UUT)
	decoder4to16_low_enable uut (
		.e(e),
		.w(w),
		.y(y)
	);
	integer i;
	initial begin
		// Initialize Inputs
		// khong cho phep giai ma
		e = 1;
		w = 0;
		#100;
		w = 1;
		#100;
    // cho phep giai ma
		e = 0;
		for (i=0;i<16;i=i+1)
		begin
			w = i;
			#100;
		end
		$stop;
	end

endmodule
```

1.5. Waveform



Nhận xét:

Trong khoảng [0,200] ns, e = 1 tức ngõ cho phép không tích cực mức thấp, nên ngõ ra sẽ là 16'bX.

Trong khoảng [200,1700] ns, e = 0 tức cho phép mạch giải mã. Cứ sau 100ns, ngõ vào w sẽ tăng lên 1 (từ 0 đến 15), đầu ra y tương ứng sẽ giải mã ở mức logic "0", các đầu ra còn lại sẽ ở mức logic "1".

2. Thiết kế thanh ghi dịch 8 bit theo phương pháp cài đặt các D-FF có xung CLK tác động cạnh xuống, có CLR để xóa các ngõ ra Q và PRE để đặt trước các ngõ Q, theo yêu cầu sau:

- Xung CLK được lựa chọn bởi SW0 tương ứng hai tần số 4 Hz và 10 Hz. Giả sử xung clock nội là 2 MHz.

- Hai công tắc SW2, SW1 chọn mode: Mode 1: sáng dần lên từ phải sang trái; mode 2: điểm sáng điểm tối dịch xen kẽ từ phải qua trái; mode 3: hai điểm sáng và hai điểm tối dịch xen kẽ từ phải sang trái; mode 4: sáng dần lên từ trong ra ngoài và tắt dần từ trong ra ngoài.

- Công tắc SW3 có chức năng dừng PAUSE.

2.1. RTL Schematic

2.2. Instance Code

```
module clock_div
  #(parameter M1=500000, N1=19, M2=200000, N2=18) // 4Hz va 10Hz, xung
noi 2Mhz
        (input clk,reset,
        output [1:0] q
   );
        reg [N1-1:0] r_reg4H;
        reg [N2-1:0] r_reg10H;
        wire [N1-1:0] r_next4H;
        wire [N2-1:0] r_next10H;

always @ (posedge clk, posedge reset)
        if (reset)
        begin
                r_reg4H <= 0;
                r_reg10H <= 0;
        end
        else
        begin
                r_reg4H <= r_next4H;
                r_reg10H <= r_next10H;
        end
assign r_next4H = (r_reg4H == M1-1) ? 0 : r_reg4H+1 ;
assign r_next10H = (r_reg10H == M2-1) ? 0 : r_reg10H+1 ;
assign q[1] = (r_reg4H < M1/2) ? 0 : 1;
assign q[0] = (r_reg10H < M2/2) ? 0 : 1;
endmodule
```

```
module mux2to1(
	input [1:0] i,
	input s,
	output reg y
	);

always @ (i, s)
	case (s)
		0: y = i[0];
		1: y = i[1];
		default: y = 1'bx;
	endcase
endmodule
```

```
module mux4to1(
	input [3:0] i,
	input [1:0] s,
	output reg y
	);
always @ (i, s)
	case (s)
		0: y = i[0];
		1: y = i[1];
		2: y = i[2];
		3: y = i[3];
		default: y = 1'bx;
	endcase
endmodule
```

```
module FFD(
        input clk, pre, reset, D,
        output reg Q
   );
always @(negedge clk, negedge pre, negedge reset)
        if (~pre)
                Q <= 1;
        else if (~reset)
                Q <= 0;
        else
                Q <= D;
endmodule
```

```
module shift_register_8b_SIPO(
        input clk, pre, reset, s_in,
        output [7:0] p_out
   );
FFD D7 (clk,pre,reset,s_in,p_out[7]);
FFD D6 (clk,pre,reset,p_out[7],p_out[6]);
FFD D5 (clk,pre,reset,p_out[6],p_out[5]);
FFD D4 (clk,pre,reset,p_out[5],p_out[4]);
FFD D3 (clk,pre,reset,p_out[4],p_out[3]);
FFD D2 (clk,pre,reset,p_out[3],p_out[2]);
FFD D1 (clk,pre,reset,p_out[2],p_out[1]);
FFD D0 (clk,pre,reset,p_out[1],p_out[0]);
endmodule
```

```
module shift_register_8b_SIPO_1 (
        input clk, pre, reset, s_in,
        output [7:0] p_out
   );
        wire [7:0] out;
FFD D0 (clk,pre,reset,s_in,out[0]);
FFD D1 (clk,pre,reset,out[0],out[1]);
FFD D2 (clk,pre,reset,out[1],out[2]);
FFD D3 (clk,pre,reset,out[2],out[3]);
FFD D4 (clk,pre,reset,out[3],out[4]);
FFD D5 (clk,pre,reset,out[4],out[5]);
FFD D6 (clk,pre,reset,out[5],out[6]);
FFD D7 (clk,pre,reset,out[6],out[7]);
assign p_out = {out[6:0],s_in};
endmodule
```

```
module LedOnGra_RL(
        input clk, pre, reset,
        output [7:0] q
        );
        wire s_in;
shift_register_8b_SIPO SIPO1 (clk_out, reset, s_in, q);
assign s_in = ~q[7] ;
endmodule
```

2.3. Top Module Code

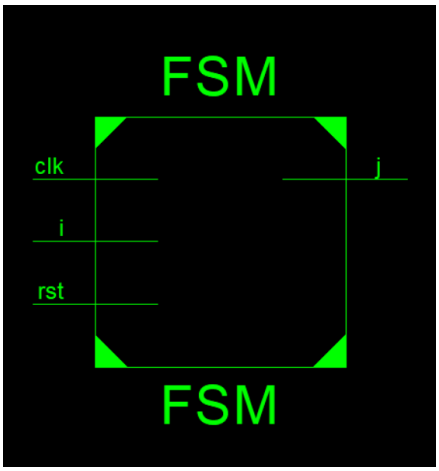
```
module Shift_Register(
  input clk, reset, SW0, SW1, SW2, SW3,
        output [7:0] q_out
   );
         wire [3:0] clk_out;
         wire [7:0] q1, q2, q3, q4;
         wire f, w;
clock_div clockdivider (clk, reset, clk_out);
mux2to1 MUX1 (clk_out, SW0, f);
assign w = SW3 & f;
//
mux4to1 MUX2 ({q4,q3,q2,q1}, {SW1,SW0}, q_out);
endmodule
```

2.4. Testbench Code

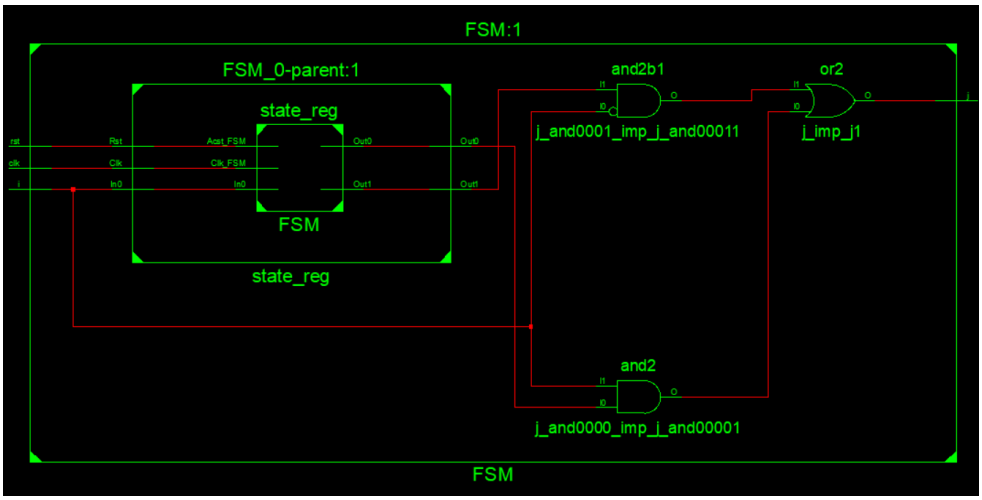
2.5. Waveform

3. Thiết kế máy trạng thái sau: Cho biết ký hiệu i/j trên các mũi tên tương ứng giá trị vào i và giá trị ra j. Trạng thái *RST* là q0.

3.1. RTL Schematic



Sơ đồ khối tổng quát của máy trạng thái có 3 chân ngõ vào là clk, i, rst và ngõ ra j



Bên trong gồm 1 khối state_reg và một số cổng logic tương ứng với các phép gán. Cổng AND có 1 ngõ vào mức thấp tương ứng với phép gán khi (state_reg == 2)& ~i . Cổng AND còn lại tương ứng với phép gán (state_reg == 1)& i . Khi state_reg

= 0 thì ngõ ra j ở mức 0.Cổng OR để tổng hợp các trường hợp lại và đưa kết quả tới ngõ ra

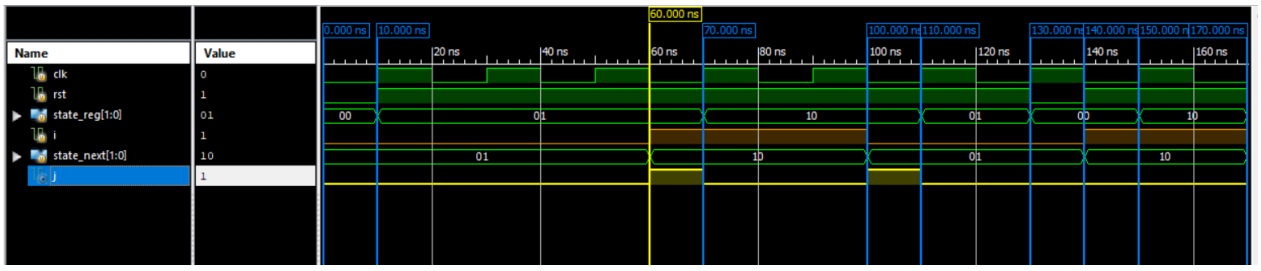
3.3. Top Module Code

```
module FSM(
	input clk, rst,
	input i,
	output j
	);
//symbolic state declaration
localparam [1:0] q0 = 2'b00, q1 = 2'b01, q2=2'b10;
// signal declaration
reg [1:0] state_reg, state_next;
// state register
always @ (posedge clk ,negedge rst)
	if (~rst)
		state_reg <= q0;
	else
		state_reg <= state_next;
//next_state logic
always @*
	case (state_reg)
		q0: if(~i)
				state_next = q1;
			else
				state_next = q2;
		q1: if(~i)
				state_next = q1;
			else
				state_next = q2;
		q2: if(~i)
				state_next = q1;
			else
				state_next = q2;
		default: state_next = q0;
	endcase
//Mealy outputlogic
assign j = (state_reg==q1)&i || (state_reg==q2)&~i || (state_reg==q0)&~i&i ;
endmodule
```

3.4. Testbench Code

```
module tb;
	// Inputs
	reg clk;
	reg rst;
	reg i;
	// Outputs
	wire j;
	// Instantiate the Unit Under Test (UUT)
	FSM uut (
		.clk(clk),
		.rst(rst),
		.i(i),
		.j(j)
	);
	initial begin
		// khoi tao
		clk = 0;
		rst = 0;
		i = 0;
		#10;
		rst = 1;
		i = 0;
		#50;
		i = 1;
		#40;
		i = 0;
		#30;
		// reset về q0
		rst = 0;
		#10;
		rst = 1;
		i = 1;
		#30;
		$stop;
	end
  always begin
		#10 clk = ~clk;
	end
endmodule
```

3.5. Waveform



Nhận xét: Cứ mỗi lần xung clk kích cạnh lên thì state_next mới được gán vào state_reg.

- Trong khoảng [0,10] ns, khi rst = 0 thì máy trạng thái sẽ ở trạng thái q0 (2’b00). Ngõ vào i trong thời điểm này ở mức 0 nên trạng thái kế tiếp là q1 (2’b01) và j = 0.
- Trong khoảng [10,60] ns, trạng thái q1 và i = 0 nên trạng thái kế tiếp vẫn là q1, đồng thời j = 0.
- Trong khoảng [60,70] ns, trạng thái q1 và i = 1 nên trạng thái kế tiếp là q2 (2’b10), đồng thời j = 1.
- Trong khoảng [70,100] ns, trạng thái q2 và i = 1 nên trạng thái kế tiếp vẫn là q2 (2’b10), đồng thời j = 0.
- Trong khoảng [100,110] ns, trạng thái q2 và i = 0 nên trạng thái kế tiếp là q1 (2’b01), đồng thời j = 1.
- Trong khoảng [110,130] ns, trạng thái q1 và i = 0 nên trạng thái kế tiếp vẫn là q1, đồng thời j = 0.
- Trong khoảng [130,140] ns, khi rst = 0 thì máy trạng thái sẽ về trạng thái q0 (2’b00), i = 0 nên trạng thái kế tiếp là q1, đồng thời j = 0.
- Trong khoảng [140,150] ns, do xung clk chưa kích cạnh lên nên trạng thái state_reg vẫn là q0, i = 1 nên trạng thái kế tiếp là q2 (2’b10), đồng thời j = 0.
- Trong khoảng [150,170] ns, trạng thái q2 và i = 1 nên trạng thái kế tiếp vẫn là q2, đồng thời j = 0.