به نام خداوند بخشنده مهربان

طراحی سیستم های دیجیتال

دانشکده مهندسی کامپیوتر

دکتر فصحتی
بهار ۱۴۰۳

مهدی علی نژاد، ۴۰۱۱۰۶۲۶۶

تمرین سوم

سوال ۱

بخش اول:

۱) یک حافظه همزمان به عمق ۵۱۲ و عرض ۱۶ طراحی کنید که دارای دو پورت دوطرفه A و B با قابلیت خواندن/ نوشتن در هر دو پورت باشد. در حافظه‌های دو پورت در یک پالس ساعت امکان دارد یک پورت خانه‌ای از حافظه که پورت دیگر می‌خواهد بر روی آن بنویسد را بخواند. در این حالت باید ابتدا داده خوانده شود و سپس بر روی آن خانه از حافظه نوشته شود. اگر هر دو پورت در یک پالس ساعت بخواهند در یک خانه از حافظه بنویسند باید داده پورت B در خانه حافظه نوشته شود و سیگنال w_race یک گردد. در این سوال پالس‌ساعت پورت A و B با هم یکسان است.

برای طراحی خود testbench نوشته تا از صحت طراحی خود مطمئن شوید.

```
module memory #(
    parameter integer width = 8,
    parameter integer depth = 16,
    parameter integer log_depth = 4
)(
    input clk,
    input reset,
    input write_on_a,
    input write_on_b,
    input [log_depth:0] address_a,
    input [log_depth:0] address_b,
    inout [width-1:0] a,
    inout [width-1:0] b,
    output reg we_race
);

reg [width-1:0] storage[depth-1:0];
reg [width-1:0] a_internal;
reg [width-1:0] b_internal;
integer i;

```

برای پیاده سازی این حافظه، ابتدا از یک سری پارامتر ها برای عمق و عرض کمک گرفته ایم، سپس ورودی های لازمه را گرفته، دو سیم a و b سیم های دو طرفه ی ما هستند و سیگنال we race نیز خروجی است. برای پیاده سازی این حافظه از یک آرایه به عمق depth و عرض width استفاده کرده ایم و دو رجیستر نیز داریم که در صورت خروجی دادن از خطوط دو طرفه، آنها را مقدار دهی می کنیم.

```
always @(posedge clk or posedge reset) begin
    we_race = 0;
    if (reset) begin
        for (i = 0; i < depth; i = i + 1) begin
            storage[i] <= {width{1'b0}};
        end
        a_internal <= 0;
        b_internal <= 0;
    end else begin
        case ({write_on_a, write_on_b})
            2'b11: begin
                b_internal = b;
                storage[address_b] <= b_internal;
                if (address_a == address_b)
                    we_race = 1;
                else begin
                    a_internal = a;
                    storage[address_a] = a_internal;
                end
            end
            2'b10: begin
                b_internal = storage[address_b];
                a_internal = a;
                storage[address_a] = a_internal;
            end
            2'b01: begin
                a_internal = storage[address_a];
                b_internal = b;
                storage[address_b] = b_internal;
            end
            2'b00: begin
                b_internal = storage[address_b];
                a_internal = storage[address_a];
            end
        endcase
    end
end

assign a = write_on_a ? {width{1'bz}} : a_internal;
assign b = write_on_b ? {width{1'bz}} : b_internal;
```

این حلقه ی اصلی برنامه است که حالات مختلف بین نوشتن و خواندن داده را مدل کرده است، ابتدا ریست را داریم که تمام خانه های حافظه را با یک حلقه صفر می کند. و در صورتی که ریست فعال نبود، بر روی حالات مختلف write on a، write on b case می زنیم و با توجه به خواسته های سوال کار های مورد نظر را انجام می دهیم.

```
module memory_tb;

    localparam integer WIDTH = 8;
    localparam integer DEPTH = 16;
    localparam integer LOG_DEPTH = $clog2(DEPTH);

    reg clk;
    reg reset;
    reg write_on_a;
    reg write_on_b;
    reg [LOG_DEPTH:0] address_a;
    reg [LOG_DEPTH:0] address_b;
    wire [WIDTH-1:0] a;
    wire [WIDTH-1:0] b;
    reg [WIDTH-1:0] a_reg;
    reg [WIDTH-1:0] b_reg;
    wire we_race;

    assign a = write_on_a ?  a_reg : {WIDTH{1'bz}};
    assign b = write_on_b ?  b_reg : {WIDTH{1'bz}};

    memory #(
        .width(WIDTH),
        .depth(DEPTH),
        .log_depth(LOG_DEPTH)
    ) uut (
        .clk(clk),
        .reset(reset),
        .write_on_a(write_on_a),
        .write_on_b(write_on_b),
        .address_a(address_a),
        .address_b(address_b),
        .a(a),
        .b(b),
        .we_race(we_race)
    );
```

این نیز ماژول تست مان است.

```
1
2       initial begin
3           clk = 0;
4           reset = 1;
5           write_on_a = 0;
6           write_on_b = 0;
7           address_a = 0;
8           address_b = 0;
9           a_reg = 0;
10          b_reg = 0;
11          #5;
12          reset = 0;
13          write_on_a = 1;
14          write_on_b = 1;
15          address_a = 5;
16          address_b = 3;
17          a_reg = 25;
18          b_reg = 9;
19          #10;
20          write_on_a = 0;
21          write_on_b = 0;
22          address_a = 3;
23          address_b = 5;
24          #10;
25          write_on_a = 1;
26          write_on_b = 0;
27          address_a = 3;
28          address_b = 3;
29          a_reg = 20;
30          #10;
31          write_on_a = 0;
32          write_on_b = 1;
33          address_a = 5;
34          address_b = 5;
35          b_reg = 70;
36          #10;
37          write_on_a = 0;
38          write_on_b = 0;
39          address_a = 3;
40          address_b = 5;
41          #10;
42          write_on_a = 1;
43          write_on_b = 1;
44          address_a = 3;
45          address_b = 3;
46          a_reg = 25;
47          b_reg = 35;
48          #10;
49          write_on_a = 0;
50          write_on_b = 0;
51          address_a = 3;
52          address_b = 0;
53          #10;
54          $stop();
55      end
```

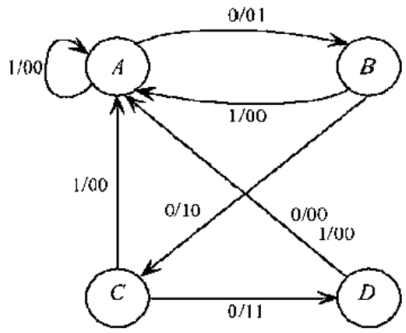
سناریو نیز به این صورت است که ابتدا در دو خانه ی متفاوت نوشتن اتفاق می افتد، سپس این دو خانه را می خوانیم تا مطمئن شویم درست نوشته شده اند، پس از آن در یک خوانه می نویسیم و می خوانیم و سپس در خانه ای دیگر همین کار را انجام می دهیم و در نهایت جفت این خانه ها را می خوانیم تا از صحت عملکرد مطمئن شویم، و در نهایت نیز در یک خانه همزمان دو مقدار مختلف می ریزیم و آن خانه را می خوانیم تا ببینیم چه مقداری به خود گرفته.

```
# Time:  0, reset: 1, write_on_a: 0, write_on_b: 0, address_a: 0, address_b: 0, a:  0, b:  0, we_race: 0
# Time:  5, reset: 0, write_on_a: 1, write_on_b: 1, address_a: 5, address_b: 3, a: 25, b:  9, we_race: 0
# Time: 15, reset: 0, write_on_a: 0, write_on_b: 0, address_a: 3, address_b: 5, a:  9, b: 25, we_race: 0
# Time: 25, reset: 0, write_on_a: 1, write_on_b: 0, address_a: 3, address_b: 3, a: 20, b:  9, we_race: 0
# Time: 35, reset: 0, write_on_a: 0, write_on_b: 1, address_a: 5, address_b: 5, a: 25, b: 70, we_race: 0
# Time: 45, reset: 0, write_on_a: 0, write_on_b: 0, address_a: 3, address_b: 5, a: 20, b: 70, we_race: 0
# Time: 55, reset: 0, write_on_a: 1, write_on_b: 1, address_a: 3, address_b: 3, a: 25, b: 35, we_race: 1
# Time: 65, reset: 0, write_on_a: 0, write_on_b: 0, address_a: 3, address_b: 0, a: 35, b:  0, we_race: 0
```

همانطور که مشاهده می کنید، نوشتن در آدرس های مختلف، خواندن از آدرس های مختلف، نوشتن و خواندن همزمان از یک خانه و نوشتن همزمان در یک خانه همگی در تست آمده و آنها طبق خواسته ی سوال به درستی عملیاتی می شوند.

سوال ۲

۲) کدام توصیف وریلاگ زیر معادل ماشین حالت متناهی مقابل است؟ (i ورودی و o خروجی است)



الف)
```
reg i; reg [2:0] o;
always @(i) begin
   if (i == 1'b1)
      o = 2'b00;
   else
      o = o + 1'b1;
end
```

ب)
```
reg i; reg [1:0] o;
always @(posedge clock) begin
   if (i === 1'b1)
      o = 2'b00;
  else
      o = o + 1'b1;
end
```

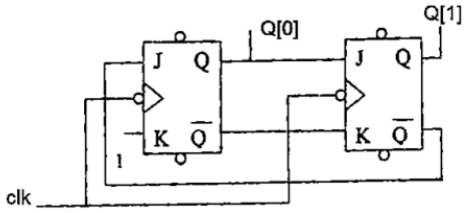
پ)
```
reg i; reg [1:0] o;
always @(posedge clock) begin
   case (o)
   2'b00  :  o = 2'b01;
   2'b01  :  o = i ? 2'b00 : 2'b10;
   2'b10  :  o = i ? 2'b00 : 2'b11;
   2'b11  :  o = 2'b00;
   endcase
end
```

ت)
```
reg i; reg [2:0] o;
always @(posedge clock, i) begin
   if (i == 1'b1)
      o = 2'b00;
   else
      o = o + 1'b1;
end
```

گزینه ی ب
زیرا ماشین ۴ حالت دارد پس نیاز به ۲ رجیستر است و تمامی حالت اگر ۱ == i می بود به استیت ۰۰ می روند.

سوال ۳

۳) کدام توصیف وریلاگ زیر معادل مدار مقابل است؟ (مقدار ابتدایی فیلیپ فلاپ‌ها صفر است)



الف)
```
reg [1:0]q = 2'b00;
always @(negedge clk) begin
    q[0] <= ~q[1];
    q[1] <= q[0];
end
```

ب)
```
reg [1:0]q = 2'b00;
always @(negedge clk) begin
    q[0] <= ~q[1];
    q[1] <= ~q[0];
end
```

پ)
```
reg [1:0]q = 2'b00;
always @(negedge clk) begin
   q[0] <= ~q[1];
   @(negedge clk) q[0] <= q[1];
    q[1] <= q[0];
   @(negedge clk) q[1] <= q[0];
end
```

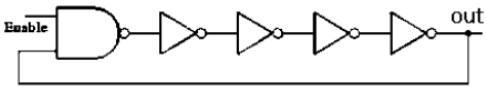
ت)
```
reg [1:0]q = 2'b00;
always @(negedge clk) begin
   q[0] <= ~q[1];
   @(negedge clk)
    q[0] <= q[1];
    q[1] <= q[0];
end
```

گزینه ی پ

اگر دقت کنیم متوجه می شویم که $Q[۱]$ در واقع همان مقدار $Q[۰]$ را به خود می گیرد و همچنین پس از دو لبه ی پایین رونده ی کلاک مقدار $Q[۰]$ برابر با ۰ است ولی در الف اینگونه نیست.

سوال ۴

۴) اگر هر گیت تأخیری برابر با ۲ داشته باشد؛ کدام توصیف وریلاگ زیر، معادل مدار مقابل است؟



الف)
```
reg out = 1'b0;
always @(enable, out) begin
    out = #10 (enable ? ~out : 1'b0);
end
```

ب)
```
reg out = 1'b0;
always @(enable, out) begin
    out <= #2 (enable ? ~out : 1'b0);
end
```

پ)
```
reg out = 1'b0;
always @(enable, out) begin
    out = #2 (enable ? ~out : 1'b1);
end
```

ت)
```
reg out = 1'b0;
always @(enable, out) begin
    out <= #10 (enable ? ~out : 1'b1);
end
```

گزینه ی ت

مجموع تاخیر کل برابر با ۱۰ است و همچنین در صورت غیر فعال بودن، مقدار یک در out نمایش داده می شود.

سوال ۵

۵) کدام توصیف زیر بدون خطا است؟

الف)
```
module a(output reg x, y);
always
    y <= 0;

endmodule
```

ب)
```
module a(x, y);
input x, y;
always @(x, y)
    x <= 0;
endmodule
```

پ)
```
module a(input reg x, y);
always @(x, y)
    #10 y <= 0;
endmodule
```

ت)
```
module a(x, y);
output x, y;
assign x = y;
endmodule
```

گزینه ی ت
در گزینه ی الف نمی توان یک پین خروجی را بی مقدار گذاشت.
در گزینه ی ب، x و y سیم هستند و نمی توان به آنها داخل always مقدار داد.
در گزینه ی پ نیز نمی توان ورودی از نوع رجیستر داشت.