«به نام خدا»

نام: فاطمه زهرا بخشنده استاد: دکتر میثم عبداللهی

گزارش تمرین دوم:

سوال اول:

در این سوال ابتدا ماژول های خواسته شده را به ترتیب طراحی می کنیم.

```
module SUM(S, A1, A2);

    input[31:0] A1, A2;
    output[31:0] S;
    assign S = A1 + A2;

endmodule
```

```
module SUB(S, A1, A2);

    input[31:0] A1, A2;
    output[31:0] S;
    assign S = A1 - A2;

endmodule
```

```
module MUL(M, A1, A2);

    input[31:0] A1, A2;
    output[31:0] M;
    assign M = A1 * A2;

endmodule
```

```
module POWER(P, A1, A2);

    input[31:0] A1, A2;
    output[31:0] P;
    assign P = A1 ** A2;

endmodule
```

```
module DIV(D, A1, A2);

    input [31:0] A1, A2;
    output [31:0] D;
    assign D = A1 / A2;

endmodule
```

```
module GCD(G, A1, A2);

    input [31:0] A1, A2;
    output integer G;
    integer N1, N2, temp;
    integer done;

    always @(*)
    begin
        N1 = A1;
        N2 = A2;
        while (N2 != 0)
        begin
            if (N1 < N2)
            begin
                temp = N1;
                N1 = N2;
                N2 = temp;
            end
            else if (N2 != 0)
                N1 = N1 - N2;
        end
        G = N1;
    end

endmodule
```

```
module MUX(O, S, SUM_OUT,
        SUB_OUT, MUL_OUT, DIV_OUT,
        GCD_OUT, POWER_OUT);

    input[31:0] SUM_OUT, SUB_OUT, MUL_OUT;
    input[31:0] DIV_OUT, GCD_OUT, POWER_OUT;
    input[2:0] S;
    output reg[31:0] O;

    always @(*)
    begin
        case(S)
            3'b000  : O <= SUM_OUT;
            3'b001  : O <= SUB_OUT;
            3'b010  : O <= MUL_OUT;
            3'b011  : O <= DIV_OUT;
            3'b100  : O <= GCD_OUT;
            3'b101  : O <= POWER_OUT;
            default : O <= 0;
        endcase
    end
```

در نهایت، ALU را طوری پیاده سازی می کنیم که با دریافت 2 عدد صحیح A و B و S عملیات مناسب را با توجه به سیگنال S انتخاب کرده و آن را روی دو عدد انجام دهد.

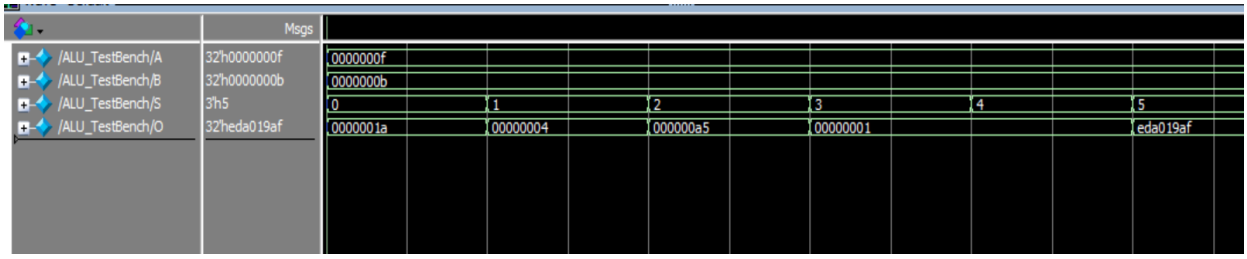
```
99
100  module ALU (O, A, B, S);
101      input[31:0] A, B;
102      input[2:0] S;
103      wire[31:0] SUM_OUT, SUB_OUT, MUL_OUT;
104      wire[31:0] DIV_OUT, GCD_OUT, POWER_OUT;
105      output reg[31:0] O;
106
107      SUM SM(SUM_OUT, A, B);
108      SUB SB(SUB_OUT, A, B);
109      MUL ML(MUL_OUT, A, B);
110      DIV DV(DIV_OUT, A, B);
111      GCD GD(GCD_OUT, A, B);
112      POWER PW(POWER_OUT, A, B);
113
114      always @(*)
115      begin
116          case(S)
117              3'b000  : O <= SUM_OUT;
118              3'b001  : O <= SUB_OUT;
119              3'b010  : O <= MUL_OUT;
120              3'b011  : O <= DIV_OUT;
121              3'b100  : O <= GCD_OUT;
122              3'b101  : O <= POWER_OUT;
123              default : O <= 0;
124          endcase
125      end
126
127  endmodule
```

برای تست کد خود، در فایل Q1_Test برای ماژول ALU یک testbench می نویسیم:

```
1
2   module ALU_TestBench;
3
4       reg[31:0] A, B;
5       reg[2:0] S;
6       wire[31:0] O;
7       ALU AU(O, A, B, S);
8
9       initial
10      begin
11          A = 32'd15;
12          B = 32'd11;
13          S = 0;
14          #10;
15          S = 1;
16          #10;
17          S = 2;
18          #10;
19          S = 3;
20          #10;
21          S = 4;
22          #10;
23          S = 5;
24          #10;
25      end
26
27  endmodule
```

اعداد ورودی 15 و 11 هستند و S را از 0 تا 5 هر 10 واحد، تغییر می دهیم تا ALU عملیات مورد نظر را روی دو عدد انجام دهد.

شبیه سازی waveform آن به صورت زیر است:



سوال دوم:

در این سوال یک cache تعریف می کنیم که 64 کیلوبایت data در آن قرار گیرد. همچنین حافظه cache ما باید دارای بیت valid و tag باشد. هر Block از cache، 4 word را ذخیره میکند و هرword 32 بیت را در خود جای میدهد.

Cache controller خود را به صورت زیر طراحی می کنیم:

دو سیگنال read و write داریم که مشخص می کند میخواهیم داده ای را از cache بخوانیم یا روی آن بنویسیم.

اگر سیگنال write فعال باشد، داده ورودی در block قرار گرفته، valid bit یک می شود و block در خانه مورد نظر از cache قرار می گیرد.

اگر سیگنال read فعال باشد، داده موجود آدرس مورد نظر، در block قرار میگیرد، اگر valid bit یک باشد، و hit رخ دهد، سیگنال hit یک شده و با توجه به block_offset بلاک مورد نظر خوانده میشود و در data_out قرار می گیرد.

```
always @(*)
begin
    index = address[15:4];
    tag = address[31:16];
    block_offset = address[3:2];
    block = 145'd0;

    if(write_sig)
    begin
        block[143:128] = tag;
        case(block_offset)
            2'b00 : block[31:0] = data;
            2'b01 : block[63:32] = data;
            2'b10 : block[95:64] = data;
            2'b11 : block[127:96] = data;
        endcase

        block[144] = 1'd1;
        cache[index] = block;
    end
```

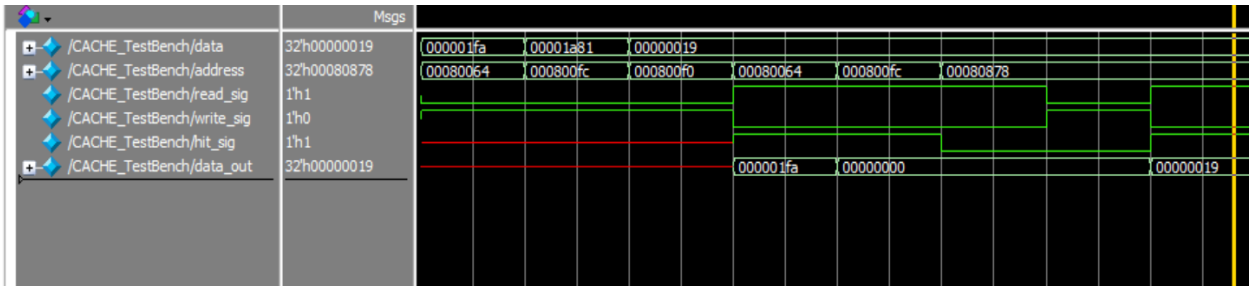
```
    else if(read_sig)
    begin
        block = cache[index];
        if(block[143:128] == tag)
        begin
            hit_sig = block[144];
            case(block_offset)
                2'b00 : data_out = block[31:0];
                2'b01 : data_out = block[63:32];
                2'b10 : data_out = block[95:64];
                2'b11 : data_out = block[127:96];
            endcase
        end
        else
            hit_sig = 0;
    end
end

endmodule
```

برای تست cache controller در فایل Q2_Test یک testbench می نویسیم:

```
module CACHE_TestBench;
    reg [31:0] data, address;
    reg read_sig, write_sig;
    wire hit_sig; wire [31:0] data_out;
    Cache_Controller CC(
        data_out, hit_sig, address,
        data, read_sig, write_sig);

    initial
    begin
        write_sig = 1'd1; read_sig = 1'd0;
        address = 32'b0000000000001000_000000000110_01_00;
        data = 32'd506;

        #10
        write_sig = 1'd1; read_sig = 1'd0;
        address = 32'b0000000000001000_000000001111_11_00;
        data = 32'd6785;

        #10
        write_sig = 1'd1; read_sig = 1'd0;
        address = 32'b0000000000001000_000000001111_00_00;
        data = 32'd25;

        #10
        write_sig = 1'd0; read_sig = 1'd1;
        address = 32'b0000000000001000_000000000110_01_00;

        #10
        write_sig = 1'd0; read_sig = 1'd1;
        address = 32'b0000000000001000_000000001111_11_00;

        #10
        write_sig = 1'd0; read_sig = 1'd1;
        address = 32'b0000000000001000_000010000111_10_00;

        #10
        write_sig = 1'd1; read_sig = 1'd0;
        address = 32'b0000000000001000_000010000111_10_00;
        data = 32'd25;

        #10
        write_sig = 1'd0; read_sig = 1'd1;
        address = 32'b0000000000001000_000010000111_10_00;
    end
endmodule
```

ابتدا سیگنال write را فعال کرده و 3 داده در آدرس های مشخص درون cache قرار می دهیم. سپس سیگنال read را فعال کرده و سه داده از cache می خوانیم که در دو تا از آن ها hit رخ داده و در آخری miss رخ می دهد و سیگنال hit صفر می شود، چون این داده قبلا در cache قرار نگرفته است. سپس همین داده را روی cache مینویسیم و دوباره آن را می خوانیم این دفعه hit رخ می دهد.

شبیه سازی waveform آن به صورت زیر است:



سوال سوم:

برای این سوال ماژول PriorityQ خود را طبق الگوریتم لینک زیر، اما در verilog پیاده سازی می کنیم:

/https://www.geeksforgeeks.org/priority-queue-in-python

ابتدا یک priority queue با 16 خانه 8 بیتی می سازیم. رجیستر 16 بیتی empty را تعریف می کنیم که نشان می دهد هر خانه متناظر از صف اولویت پر است یا خالی، و در ابتدا همه bit های آن یک است.

اگر سیگنال enqueue فعال باشد، و تعداد داده های درون cache (size) کمتر از 16 تا باشد، داده را در بلاک خالی اضافه می کنیم، و سایز pqueue را یکی زیاد می کنیم.

```
module PriorityQ(data_out, data_in, enqueue_sig, dequeue_sig);

    reg [7:0] queue [15:0]; reg done; reg [7:0] max;
    reg[15:0] empty = 15'b111111111111111;
    output reg [7:0] data_out;
    input [7:0] data_in; input enqueue_sig, dequeue_sig;
    integer size = 0, i;

    always @(*)
    begin
        if(enqueue_sig && size < 16)
        begin
            done = 1'b0;
            for(i = 0; !done && i < 16; i = i+1)
            begin
                if(empty[i])
                begin
                    empty[i] = 1'b0;
                    queue[i] = data_in;
                    done = 1'b1;
                    size = size + 1;
                end
            end
        end
```

اگر سیگنال dequeue فعال باشد، و تعداد داده های درون cache (size) بزرگتر از 0 باشد، بزرگترین داده موجود در صف (داده دارای اولویت بیشتر) از صف خارج می شود و در data_out قرار می گیرد، سایز pqueue را یکی کم می کنیم.

```
        else if(dequeue_sig && size > 0)
        begin
            max = 8'd0;
            for(i = 0; i < 16; i = i+1)
            begin
                if(queue[max] < queue[i])
                begin
                    max = i;
                end
            end
            data_out = queue[max];
            queue[max] = 8'd0;
            empty[max] = 1'b1;
            size = size - 1;
        end
    end
```

برای تست کد خود در فایل Q3_Test یک testbench می نویسیم:

```
module PriorityQ_TestBench;

    reg enqueue_sig, dequeue_sig;
    reg [7:0] data_in;
    wire [7:0] data_out;
    PriorityQ PQ(
        data_out, data_in,
        enqueue_sig, dequeue_sig
        );

    initial
    begin
        enqueue_sig = 1;
        dequeue_sig = 0;
        data_in = 8'd50;

        #10;
        enqueue_sig = 1;
        dequeue_sig = 0;
        data_in = 8'd20;

        #10;
        enqueue_sig = 0;
        dequeue_sig = 1;

        #10;
        enqueue_sig = 1;
        dequeue_sig = 0;
        data_in = 8'd14;

        #10;
        enqueue_sig = 0;
        dequeue_sig = 1;

        #10;
        enqueue_sig = 1;
        dequeue_sig = 0;
        data_in = 8'd24;

        #10;
        enqueue_sig = 0;
        dequeue_sig = 1;

        #10;
        enqueue_sig = 0;
        dequeue_sig = 0;
    end

endmodule
```

با چند بار enqueue و dequeue داده های مختلف می بینیم هر دفعه برای dequeue داده بزرگتر را خروجی می دهد.

شبیه سازی waveform آن به صورت زیر می شود: