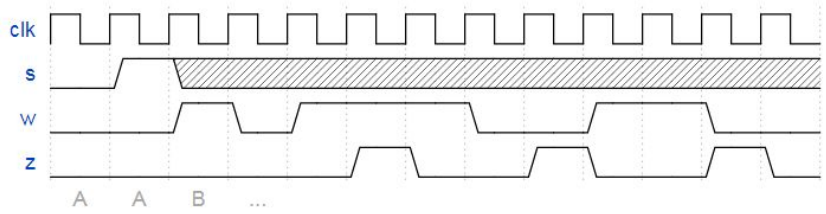
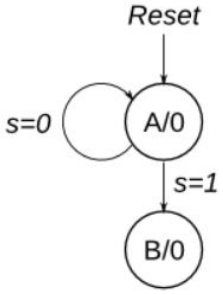
Exams/2014 q3fsm

Consider a finite state machine with inputs *s* and *w*. Assume that the FSM begins in a reset state called *A*, as depicted below. The FSM remains in state *A* as long as *s* = 0, and it moves to state *B* when *s* = 1. Once in state *B* the FSM examines the value of the input *w* in the next three clock cycles. If *w* = 1 in exactly two of these clock cycles, then the FSM has to set an output *z* to 1 in the following clock cycle. Otherwise *z* has to be 0. The FSM continues checking *w* for the next three clock cycles, and so on. The timing diagram below illustrates the required values of *z* for different values of *w*.

Use as few states as possible. Note that the *s* input is used only in state *A*, so you need to consider just the *w* input.





FSM還算單純 主要是在狀態B要另外設一個**counter** 123 123一直跑一直跑 (每次數到3歸零 每次數到3歸零)

處理z的話 很顯然要要紀錄數3個cycle總共收到了幾個w
我的做法是數前2個cycle搭配第3個cycle的w來做一個z_trigger訊號 條件如下:
1.前2個cycle總共收到1個w 此時第3cycle, w必需=1 才有接下來的z
2.前2個cycle總共收到2個w 此時第3cycle, w要等於0 才有接下來的z
我們的z_trigger就設成上面兩個條件的任一個成立

剩下的code就很簡單就能code出來了

```
module top_module (
    input clk,
    input reset,   // Synchronous reset
    input s,
    input w,
    output z
);
    parameter A = 1'b0;
    parameter B = 1'b1;

    reg curr_state, next_state;
```

```
always @ (posedge clk)begin
    if(reset)begin
        curr_state <= A;
    end
    else begin
        curr_state <= next_state;
    end
end

always @ (*) begin
    case(curr_state)
        A: next_state = s? B : A;
        B: next_state = B;
    endcase
end

reg [1:0] loop_cnt;
wire      set_loop_cnt = ((curr_state==A) & (next_state==B))|
                         ((curr_state==B) & (loop_cnt==2'd3)); //1 2 3 1 2 3 1 2 3

always @ (posedge clk)begin
    if(reset)begin
        loop_cnt <= 2'd0;
    end
    else if(set_loop_cnt)begin
        loop_cnt <= 2'd1;
    end
    else if(curr_state==B)begin
        loop_cnt <= loop_cnt + 2'd1;
    end
end

reg [1:0] w_cnt;
always @ (posedge clk)begin
    if(reset)begin
        w_cnt <= 2'd0;
    end
    else if(set_loop_cnt) begin
        w_cnt <= 2'd0;
    end
    else if(curr_state==B & w==1'b1)begin
        w_cnt <= w_cnt + 2'd1;
    end
end

wire z_trigger;
assign z_trigger = ((w_cnt == 2'd1)& ((loop_cnt==2'd3) & w)) |
                   ((w_cnt == 2'd2)& ((loop_cnt==2'd3) & ~w));
reg z_valid;
always @ (posedge clk)begin
    if(reset)begin
        z_valid <= 1'b0;
    end
    else if (z_trigger)begin
        z_valid <= 1'b1;
    end
    else if(set_loop_cnt)begin
```

```
            z_valid <= 1'b0;
        end
    end
    assign z = (loop_cnt==2'd1) & z_valid;


endmodule
```