

SC1005

Tut 9

CE/CZ1005 Digital Logic

Tutorial 9

Q1. (a) Explain what is wrong with the following code for a counter, and correct it:

```
module countwrong (input clk, rst, output [5:0] cnt_out);  
  
    always@*  
    begin  
        cnt_out = cnt_out + 1'b1;  
    end  
endmodule
```

(b) The following combinational module is to be converted into a synchronous module. Add a register after each combinational stage in the original description, by rewriting the module using only a single synchronous always block:

```
module arch1 (input [6:0] a, b, output [13:0] total);  
  
    wire [6:0] int1;  
  
    assign int1 = a + b;  
    assign total = int1 * int1;  
  
endmodule
```

Q2.

A monitoring circuit has an input, evnt, that is high whenever a certain condition is met. It has an internal counter that counts the number of cycles in which evnt is high. When this count exceeds a threshold, determined by the 6-bit thresh input, it sounds an alarm by asserting the alarm output and stops the counter. The human operator can then check for problems and reset the system by asserting rst. Design a Verilog module that implements this circuit. (Hint: alarm should be a combinational circuit determined from the count value and threshold. It can be used to determine whether or not the counter counts in any given cycle).

Q3.

- (a) Write a Verilog description of a 5-bit binary counter that counts up to 20 and then wraps round to zero.
- (b) Modify the counter in (a), adding a new 5-bit input, countmax. The counter should now wrap around at the countmax value.
- (c) Write a Verilog module that implements a 6-bit counter that counts down from an initial value. The initial value should be loaded on reset from a 6-bit input, start_val.

- Q1** (a) For a synchronous block, we must use `always@(posedge clk)`, and for synchronous blocks, we always add a reset condition. We also need to change the output to a reg. Note a circuit as described would not synthesise due to a combinational loop (i.e. the output of the adder is connected to its input.)

```
module countwrong (input clk, rst, output [5:0] cnt_out);  
  
always@*  
begin  
    cnt_out = cnt_out + 1'b1;  
end  
endmodule
```

The corrected version

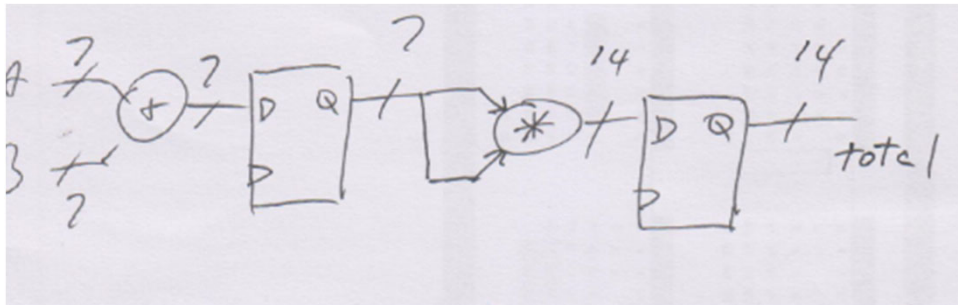
```
module countwrong (input clk, rst,  
                    output reg [5:0] cnt_out);  
  
always@(posedge clk)  
begin  
    if(rst)  
        cnt_out <= 6'b0;  
    else  
        cnt_out <= cnt_out + 1'b1;  
end  
endmodule
```

Q1(b) The circuit computes $(a+b)^2$.

We need to change the output and internal signal to a reg. Then put the two assigns as assignments inside a synchronous always block.:

```
module arch1 (input [6:0] a, b, output [13:0] total);  
  
wire [6:0] int1;  
  
assign int1 = a + b;  
assign total = int1 * int1;  
  
endmodule
```

What is actually generated?



What about the size of int1?

```
module arch1 (input [6:0] a, b, input clk, rst,  
               output reg [13:0] total);  
  
    reg [6:0] int1;  
  
    always @ (posedge clk)  
    begin  
        if (rst) begin  
            int1 <= 7'b0000_0000;  
            total <= 14'd0;  
        end  
        else  
        begin  
            int1 <= a + b;  
            total <= int1 * int1;  
        end  
    end  
endmodule
```

Q2 For this question, we need a counter that only counts when event is 1 and alarm output is not 1. Alarm should be set to the result of a comparison between the count value and thresh. We should make the counter large enough for any thresh. As thresh could be 111111, we need a 7 bit counter as in 6 bits, $111111 + 1 = 000000$, which is not greater than thresh.

```
module sysmon (input clk, rst, evnt, input [5:0] thresh, output alarm);  
    reg [6:0] int_count;  
    always @ (posedge clk)  
    begin  
        if (rst)  
            int_count <= 7'b0000000;  
        else  
            if (evnt && !alarm)  
                int_count <= int_count + 1;  
    end  
    assign alarm = (int_count > thresh);  
endmodule
```

Q3 (a) Here, we simply need to ensure that when we reach the maximum value, the next output is zero. In other cases, the counter counts up.

```
module count20 (input clk, rst,  
                output reg [4:0] count_out);  
  
    always @ (posedge clk)  
    begin  
        if (rst)  
            count_out <= 5'd0;  
        else  
        begin  
            if (count_out == 5'd20)  
                count_out <= 5'd0;  
            else  
                count_out <= count_out - 1;  
        end  
    end  
endmodule
```

What is the counting sequence?

0, 1, 2, ..., 18, 19, 20, 0, 1, 2, ...

Q3 (b) We add an extra input, and use that as the condition in the wrap around in (a)

```
module count20 (input clk, rst, input [4:0] maxcnt,  
                output reg [4:0] count_out);  
  
    always @ (posedge clk)  
    begin  
        if (rst)  
            count_out <= 5'd0;  
        else  
            begin  
                if (count_out == maxcnt)  
                    count_out <= 5'd0;  
                else  
                    count_out <= count_out + 1;  
            end  
        end  
    end  
endmodule
```

What is the counting sequence?

If maxcnt = 5:

0, 1, 2, 3, 4, 5, 0, 1, 2, ...

Q3(c) We can replace a standard counter reset assignment to use the new input:

```
module count20 (input clk, rst, input [5:0] startcnt,  
                output reg [5:0] count_out);  
  
    always @ (posedge clk)  
    begin  
        if (rst)  
            count_out <= startcnt;  
        else  
            count_out <= count_out - 1;  
        end  
    end  
endmodule
```

What is the counting sequence?

If startcnt = 5:

5, 4, 3, 2, 1, 0, 63, 62, 61, ...

Q3 (c) If we wanted to restrict to between [startcnt and 0] what would we need to do?

```
module count20 (input clk, rst, input [5:0] startcnt,  
                output reg [5:0] count_out);  
  
    always @ (posedge clk)  
    begin  
        if (rst)  
            count_out <= startcnt;  
        else  
            begin  
                if (count_out == 5'd0)  
                    count_out <= startcnt;  
                else  
                    count_out <= count_out - 1;  
            end  
        end  
    endmodule
```

What is the counting sequence?

If startcnt = 5:

5, 4, 3, 2, 1, 0, 5, 4, 3, ...