

SC1005

Tut 7

Tut 7

Q1. A traffic light for a toy car track has a 3-bit one-hot vector input, with MSB corresponding to red and the LSB to green. However, the controller delivered with the track has only a 2-bit output for the lights, encoded as: red = "00", yellow = "01" and green = "10". The controller outputs "11" when there are no lights active.

Design a Verilog interface (using assign statements) to go between the controller and the lights.

Q2. A digital thermostat has two 8-bit unsigned binary inputs representing the target temperature and the actual temperature in degrees centigrade (°C). The thermostat has two outputs: one to turn a heater on when the actual temperature is 4°C below the target, and one to turn a cooler on when the actual temperature is 4°C above the target

(a) Design a Verilog module, *thermo*, with two 8-bit inputs, Tset and Tact and two 1-bit outputs Hon and Con. Use a parameter statement to specify the 8-bit width.

(b) The module designed in part (a) is instantiated in another module. Write the Verilog statement to instantiate the thermo module with identifier U1 using the same signal names in the upper module.

(c) Part way through the design process, the design team finds out that the temperature sensor and the target set-point both have 12-bit outputs. Describe a simple change to the Verilog statement to instantiate the thermo module so that this will not affect the operation.

Q3. The following Verilog module has a number of mistakes/errors that would result in it synthesizing incorrectly.

(a) Identify the mistakes/errors and rewrite the module to correct them.

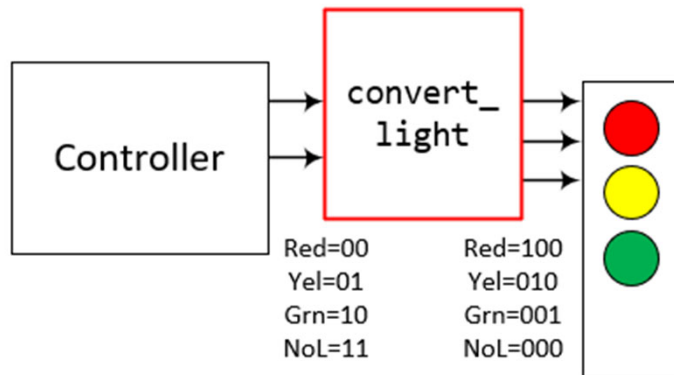
(b) Rewrite the module using a single conditional assignment?

```
module thingamajig (input [3:0] a, b, c, input [1:0] sel,
                   output [5:0] result);
    always @ (a,b)
    begin
        case (sel)
            2'b00 : result = a;
            2'b01 : result = b;
            2'b10 : result = c;
        endcase
    end
endmodule;
```

Q4. (a) Write a Verilog module that implements a 3-to-8 decoder using a case statement in an always block.

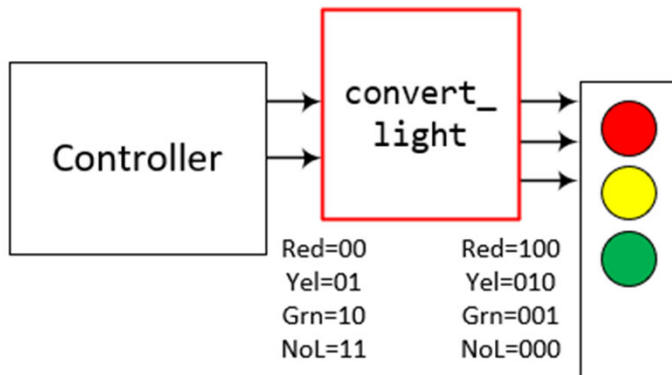
(b) What modification would you make in order to add an enable (en) to the circuit?

Q1: Every Kids worst Christmas nightmare



Q1: Every Kids worst nightmare

- While it would be possible to use a 2 to 4 decoder for this problem (selecting just the required 3 bits), the question asks for assign statements.
- So we just need to work out the logic for each light and encode that in Verilog



```
module convert_light (input [1:0] light_in,  
                     output [2:0] light_out);
```

```
    assign light_out[2] = ~light_in[1] & ~light_in[0]; //red  
    assign light_out[1] = ~light_in[1] & light_in[0]; //yel  
    assign light_out[0] = light_in[1] & ~light_in[0]; //grn
```

```
endmodule
```

Why don't we need to worry about the no light case (eg. 11)?

Q2 (a):

```
module thermo #(parameter SIZE = 8) (  
    input [SIZE-1:0] Tact, Tset,  
    output Hon, Con);  
  
    assign Hon = (Tact + 4) < Tset;    //Heater On  
    assign Con = Tact > (Tset + 4);    //Cooler On  
  
endmodule
```

What problem could potentially occur if we used:

```
assign Hon = Tact < (Tset - 4);
```

Q2 (b) & (c):

(b) If instantiated inside another module

```
thermo U1 (.Tact(Tact), .Tset(Tset), .Hon(Hon), .Con(Con));
```

(c) If there was a design change from 8-bit to 12-bit

- We override the parameter

```
thermo #(.SIZE(12)) U1 (.Tact(Tact), .Tset(Tset),  
    .Hon(Hon), .Con(Con));
```

Q3 (a):

- Identify the mistakes/errors in the code

```
module something (input [3:0] a, b, c,  
                 input [1:0] sel,  
                 output [5:0] result);  
always @ (a,b)  
begin  
    case (sel)  
        2'b00 : result = a;  
        2'b01 : result = b;  
        2'b10 : result = c;  
    endcase  
end  
endmodule;
```

- There are 2 syntax errors, 2 coding mistakes which would result in incorrect operation and 1 which may be a coding mistake (or it may also be correct)
 - Error: *result* is assigned from within an **always** block so must be declared as **reg**.
 - Error: there is no semicolon after **endmodule**
 - Mistake: the sensitivity list in the always statement does not include all inputs (a, b, c, sel). Better to use "**always @ ***"
 - Mistake: all possible combinations in the case statement are not specified. An unwanted latch would be synthesized.
 - Coding: *result* may be incorrectly declared as you are assigning a 4-bit value (*a*, *b* or *c*) to a 6-bit *result*. This is not an error and may be intended. It would generate a synthesis warning.

Q3 (a):

- Corrected code

```
module something (input [3:0] a, b, c,  
                 input [1:0] sel,  
                 output reg [5:0] result);  
    always @ (a,b,c,sel)  
        //or better use "always @ *")  
    begin  
        result = a;  
        case (sel)  
            2'b00 : result = a;  
            2'b01 : result = b;  
            2'b10 : result = c;  
            // or 2'b11 : result = a;  
            // or default statement  
        endcase  
    end  
endmodule
```

- There are 2 syntax errors, 2 coding mistakes which would result in incorrect operation and 1 which may be a coding mistake (or it may also be correct)
1. Error: *result* is assigned from within an **always** block so must be declared as **reg**.
 2. Error: there is no semicolon after **endmodule**
 3. Mistake: the sensitivity list in the always statement does not include all inputs. Better to use "**always @ ***"
 4. Mistake: all possible combinations in the case statement are not specified. An unwanted latch would be synthesized.
 5. Coding: *result* may be incorrectly declared as you are assigning a 4-bit value (*a*, *b* or *c*) to a 6-bit *result*. This is not an error and may be intended. It would generate a synthesis warning.

The corrected code is shown on the left

Q3 (a):

- Corrected code

```
module something (input [3:0] a, b, c,  
                 input [1:0] sel,  
                 output reg [5:0] result);  
  
    always @ *  
  
    begin  
        result = 6'd0;  
        case (sel)  
            2'b00 : result = {2'd0,a};  
            2'b01 : result = {2'd0,b};  
            2'b10 : result = {2'd0,c};  
        endcase  
    end  
endmodule
```

- There are 2 syntax errors, 2 coding mistakes which would result in incorrect operation and 1 which may be a coding mistake (or it may also be correct)
1. Error: *result* is assigned from within an **always** block so must be declared as **reg**.
 2. Error: there is no semicolon after **endmodule**
 3. Mistake: the sensitivity list in the always statement does not include all inputs. Better to use "**always @ ***"
 4. Mistake: all possible combinations in the case statement are not specified. An unwanted latch would be synthesized.
 5. Coding: *result* may be incorrectly declared as you are assigning a 4-bit value (*a*, *b* or *c*) to a 6-bit *result*. This is not an error and may be intended. It would generate a synthesis warning.
- If you wanted *result* to be 6-bit and to get rid of the warning , use:

Q3 (a):

- Corrected code

```
module something (input [3:0] a, b, c,  
                 input [1:0] sel,  
                 output reg [5:0] result);  
  
    always @ *  
  
    begin  
        result = 6'd0;  
        case (sel)  
            2'b00: result= {{2{a[3]}},a};  
            2'b01: result= {{2{b[3]}},b};  
            2'b10: result= {{2{c[3]}},c};  
        endcase  
    end  
endmodule
```

- There are 2 syntax errors, 2 coding mistakes which would result in incorrect operation and 1 which may be a coding mistake (or it may also be correct)
 1. Error: *result* is assigned from within an **always** block so must be declared as **reg**.
 2. Error: there is no semicolon after **endmodule**
 3. Mistake: the sensitivity list in the always statement does not include all inputs. Better to use "**always @ ***"
 4. Mistake: all possible combinations in the case statement are not specified. An unwanted latch would be synthesized.
 5. Coding: *result* may be incorrectly declared as you are assigning a 4-bit value (*a*, *b* or *c*) to a 6-bit *result*. This is not an error and may be intended. It would generate a synthesis warning.

If you wanted a 6-bit 2's complement *result* and to get rid of the warning, use:

Q3 (b):

```
module something (input [3:0] a, b, c,  
                 input [1:0] sel,  
                 output [5:0] result);  
  
    assign result = (sel==2'b00) ? a :  
                   (sel==2'b01) ? b : c;  
  
endmodule
```

Note: You would get the size mismatch warning with this code

In the above, *c* is assigned to *result* when *sel* is `10` and `11`.

If you specifically wanted `0` to be assigned for the 11 case, then you would need to modify the assign statement to:

```
assign result = (sel==2'b00) ? a :  
               (sel==2'b01) ? b :  
               (sel==2'b10) ? c : 4'b0000;
```

Q4 (a): 3-to-8 decoder using a case statement.

```
module dec3to8 (input [2:0] i, output reg [7:0] o);  
    always @ * begin  
        case (i)  
            3'b000 : o = 8'b0000_0001;  
            3'b001 : o = 8'b0000_0010;  
            3'b010 : o = 8'b0000_0100;  
            3'b011 : o = 8'b0000_1000;  
            3'b100 : o = 8'b0001_0000;  
            3'b101 : o = 8'b0010_0000;  
            3'b110 : o = 8'b0100_0000;  
            3'b111 : o = 8'b1000_0000;  
            default: o = 8'b0000_0000;  
        endcase  
    end  
endmodule
```

Does the **default** statement
ever get executed?

Q4 (b): Add the enable within the always block.

```
module dec3to8 (input en, input [2:0] i, output reg [7:0] o);  
    always @ * begin  
        o = 8'b0000_0000;    // default assignment to o  
        if (en) begin  
            case (i)  
                3'b000 : o = 8'b0000_0001;  
                3'b001 : o = 8'b0000_0010;  
                3'b010 : o = 8'b0000_0100;  
                3'b011 : o = 8'b0000_1000;  
                3'b100 : o = 8'b0001_0000;  
                3'b101 : o = 8'b0010_0000;  
                3'b110 : o = 8'b0100_0000;  
                3'b111 : o = 8'b1000_0000;  
                //default: o = 8'b0000_0000;    //not needed due to default assignment.  
            endcase  
        end  
    end  
endmodule
```

Is this correct?

Note: The **begin** and **end** keywords with the **if** statement are not needed.

Q4 (b): Alternative solution (does not use **if** statement).

This alternative solution does not use if statement. It uses an assign statement outside the always block. Note that the **if** version is better.

Add enable to port declaration, and remove **reg** from the output declaration for o:

```
module dec3to8 (input en, input [2:0] i, output [7:0] o);
```

Declare an 8-bit variable in the module body, allocated from inside an **always** so must be **reg**:

```
reg [7:0] o_tmp;
```

Assign to o_tmp from inside the case statement (instead of assigning to o):


```
3'b011 : o_tmp = 8'b0000_1000;
```

Add a conditional assignment to assign to o (outside of the **always** block), so must remove reg from the output declaration:

```
assign o = en ? o_tmp : 8'b0000_0000;
```

Q4 (a): Alternative decoder with enable.

```
module dec3to8 (input en, input [2:0] i, output [7:0] o);  
    reg [7:0] o_tmp;  
    assign o = en ? o_tmp : 8'b0000_0000;  
    always @ * begin  
        case (i)  
            3'b000 : o_tmp = 8'b0000_0001;  
            3'b001 : o_tmp = 8'b0000_0010;  
            3'b010 : o_tmp = 8'b0000_0100;  
            3'b011 : o_tmp = 8'b0000_1000;  
            3'b100 : o_tmp = 8'b0001_0000;  
            3'b101 : o_tmp = 8'b0010_0000;  
            3'b110 : o_tmp = 8'b0100_0000;  
            3'b111 : o_tmp = 8'b1000_0000;  
            default: o_tmp = 8'b0000_0000;  
        endcase  
    end  
endmodule
```



Does the assign statement, “assign o = en ? ...”
need to be below the always block?
That is, are we using o_tmp before it is allocated in the case?