Solutions to Midterm 1

There were two versions of each question. The values and the answers for both versions are given below.
Version 2 - reversed the first two transition conditions for the binary encoding version of Question 4.
Version 3 - reversed the labelling of the two solutions for Question 1.

Question 1

Fill the table below with the value of each expression as a Verilog numeric literal including the correct width and the correct value in hexadecimal base. Assume the following declarations:

```
logic [7:0] x ;
logic [3:0] y ;
```

and that `x` has the value `8'h3b` (or `8'hb3`) and that `y` has the value `4'b0110`. The first row has been filled in as an example.

expression	value
`x[3:0]`	`4'hb` (or `4'h3`)
`!x[4] | x[3]`	
`{ ~y, x[7:4] }`	
`x << y[3:2]`	
`x[7] ? 3'b1 : 4'd2`	
`x + 1`	

Answers

For `x=8'hb3`:

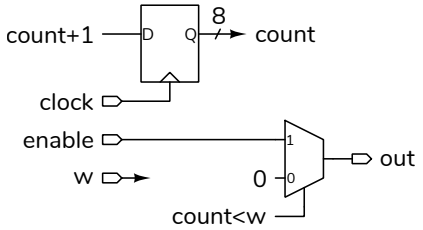
`x[3:0]`	`4'h3`
`!x[4] | x[3]`	`1'h0`
`{ ~y, x[7:4] }`	`8'h9b`
`x << y[3:2]`	`8'h66`
`x[7] ? 3'b1 : 4'd2`	`4'h1`
`x + 1`	`32'hb4`

and for `x=8'h3b`:

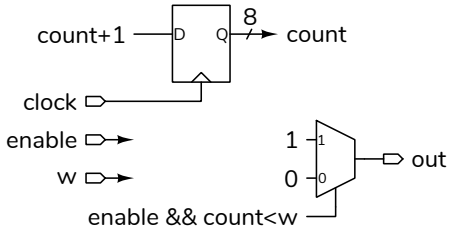
`x[3:0]`	`4'hb`
`!x[4] | x[3]`	`1'h1`
`{ ~y, x[7:4] }`	`8'h93`
`x << y[3:2]`	`8'h76`
`x[7] ? 3'b1 : 4'd2`	`4'h2`
`x + 1`	`32'h3c`

Question 2

Write a System Verilog module named `pwm` with three inputs (`clock`, `enable`, and `w`) and one output (`out`) that implements the schematic below. Follow the course coding conventions.



or



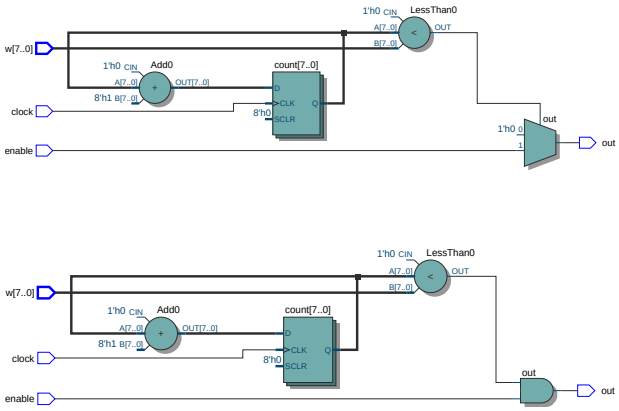
Answers

The question makes more sense if `w` is an 8-bit value, but the schematic shows it as a single bit. Solutions that assume `w` is a 1-bit signal were also marked correct.

The two solutions below correspond to the two schematics.

```
module pwm ( input logic clock, enable,
             input logic [7:0] w,
             output logic out ) ;

   logic [7:0] count ;

   always_ff @(posedge clock)
     count <= count + 1'b1 ;

   assign out = count < w ? enable : '0 ;

endmodule

module pwm_ ( input logic clock, enable,
              input logic [7:0] w,
              output logic out ) ;

   logic [7:0] count ;

   always_ff @(posedge clock)
     count <= count + 1'b1 ;

   assign out = enable && count < w ? '1 : '0 ;

endmodule
```

The schematics generated by Quartus are:

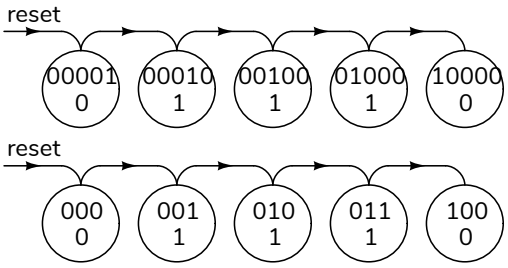


Question 3

A state machine has **reset** and **clock** inputs, and a **pulse** output. Whenever **reset** is asserted, the output is set to **0**. When **reset** is de-asserted, **pulse** is set to **1** for three clock cycles and then it is set to **0** and it stays at **0** until **reset** is asserted again. Write the next-state truth table and the output truth table for this state machine. Use a one-hot (or binary) state encoding.

Answers

The state transition diagrams for the one-hot and binary encodings are shown below:



A correct solution using only four states (one with an output of zero and three with an output of one) is not possible because there are two states where the output is 0: one state when reset is asserted and one state when reset is not asserted. The question requires that the output stay at 0 indefinitely in both of these states.

Many students gave a solution where the single-bit **pulse** output was used as the state. This is clearly not possible as the state machine needs to output a **1** for three clock cycles. In this respect the state machine is similar to the first state machine described in the lectures and the state machine in the next question.

No marks were awarded for solutions with an insufficient number of states since the state transition table and the output table are necessarily incorrect.

The simplified state transition and output truth tables corresponding to the state transition diagrams above are:

state	reset	next state
xxxxx	1	00001
10000	x	10000
n	x	$n << 1$

state	output
00001	0
10000	0
xxxxx	1

and

state	reset	next state
xxx	1	000
100	x	100
n	x	$n+1$

state	output
x00	0
xxx	1

Question 4

Write a System Verilog module named **esc** that implements the state machine described by the following state transition diagram. The module has four one-bit inputs: **clear**, **even**, **odd**, and **clock** (a clock). It has a one-bit output named **done**. You may use any name(s) for signal(s) internal to the module.

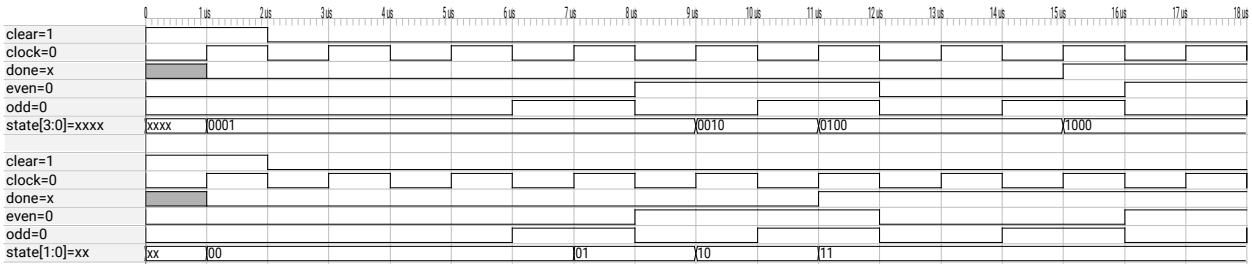
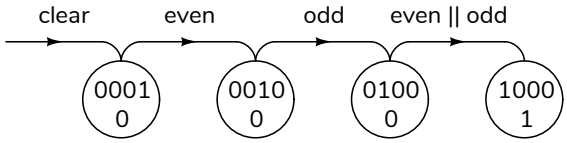
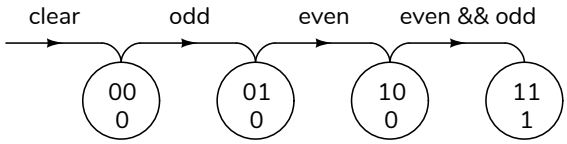


Figure 1: Simulation results for Question 4 solution.

Follow the course coding conventions.



or



Answers

```
module esc ( input logic clear, even, odd, clock,
             output logic done ) ;

   logic [3:0] state ;

   always_ff @(posedge clock)
     state <= clear ? 4'b0001 :
              state == 4'b0001 && even ? 4'b0010 :
              state == 4'b0010 && odd  ? 4'b0100 :
              state == 4'b0100 && ( even || odd ) ?
                   ↪ 4'b1000 :
              state ;

   assign done = state[3] ;


endmodule


module esc_ ( input logic clear, even, odd, clock,
              output logic done ) ;

   logic [1:0] state ;

   always_ff @(posedge clock)
     state <= clear ? 2'b00 :
              state == 2'b00 && odd ? 2'b01 :
              state == 2'b01 && even  ? 2'b10 :
              state == 2'b10 && even && odd ? 2'b11 :
              state ;

   assign done = state == 2'b11 ;


endmodule
```

The simulation results for the two modules are shown in Figure 1.