Examples of State Machines

Exercise 1:

```
module ex83 ( input reset, clk,
              output logic [7:0] y ) ;

   logic [2:0] n ;
   logic [7:0] seq [0:7] = '{ 3, 5, 1, 7, 6, 1, 8, 9 } ;

   always_ff @(posedge clk) n
      <= reset ? '0 : n+1 ;

   assign y = seq[n] ;

endmodule
```

(a) change these

How would you: (a) change the values in the sequence? (b) change the length of the sequence to 6? (c) stop at the last value?

(b) → count 0 to 5 & repeat:

```
always_ff @(posedge clk) n <=
   reset || n == 5 ? '0 : n+1 ;
```

(c) → stop at n==7

```
always_ff @(posedge clk) n <=
   reset ? '0 :
   n == 7 ? 7 :
   n+1 ;
```

Exercise 2:

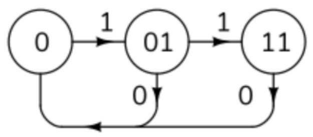
```
module ex82 ( output logic unlock,
              input logic [3:0] in,
              input logic clk ) ;

  logic [2:0] n ;

  always_ff @(posedge clk) n
     <= n == 0 && in == 4 ? 1 :
        n == 1 && in == 7 ? 2 :
        n == 2 && in == 2 ? 3 :
        n == 3 && in == 4 ? 4 :
        0 ;

  assign unlock = n == 4 ;

endmodule
```

logic [3:0] lut[0:3] = '{4, 7, 2, 4};

<= in == lut[n] ? n+1 : 0 ;

Rewrite the module to store the sequence in an unpacked array as in the previous example.

Exercise 3: Write the body of a Verilog module named **edge** with input **`in`** and output **`out`** that implements the rising-edge detector.

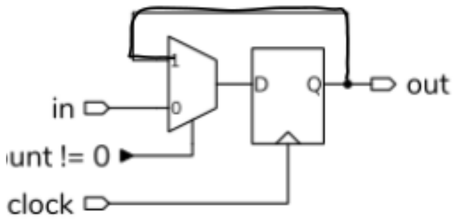


```
logic [1:0] d;

alway-ff @(posedge clk) d <=
    d==2'b00 && in==1 ? 2'b01 :
    d ==2'b01 && in==1 ? 2'b11 :
    d ==2'b11 && in ==0 ? 2'b0 :
    d == 2b01 && in ==0 ? 2'b0 :
    d ;
assign out = d==2'b01 ? 1:0;
```

Exercise 4: Write the state transition table for this state machine.

state	reset	count	next state
X X	1	X X	0 0
0 0	0	0	0 1
0 1	0	0	1 0
1 0	0	0	1 1
1 1		0	0 0

Exercise 5: Write **`always_ff`** statements that implement these state machines.



```
alway-ff @(posedge clock)
    out <= count != 0 ? out : in;
```

count	in == out	next count
→ x	1	$N-1$
→ 0	x	$N-1$
n	0	$n-1$

```
localparam N = 50_000;

always_ff@(posedge clock)
    count <= in == out ? N-1 :
        count==0 ? N-1 :
        in != out ? count-1 :
        count;
```