Solutions to Midterm Exam 1

Question 1

A state machine has a one-bit **`reset`** input and a 3-bit **`value`** output. If **`reset`** is asserted the output is set to **`3'd1`**. When **`reset`** is not asserted the output takes on successive values of **`3'd2`**, **`3'd3`**, and **`3'd7`** (or **`3'd4`**, **`3'd5`**, and **`3'd7`**). Then the output stays at **`3'd7`** until **`reset`** is asserted again. All changes on the output happen on the rising edge of the clock.

Write the state transition *table* for this state machine. Write the state and input values as Verilog literals in any base. You may also use `X` to indicate a "don't care" state or input value.

Solutions

In this problem the output values are unique for each state and so can serve as the state values. In this case the solution to the first version of the question is:

value	reset	next value
x	1	3'd1
3'd1	0	3'd2
3'd2	0	3'd3
3'd3	0	3'd7
3'd7	0	3'd7

and the solution to the second version is:

value	reset	next value
x	1	3'd1
3'd1	0	3'd4
3'd4	0	3'd5
3'd5	0	3'd7
3'd7	0	3'd7

However, since the question does not require that you specify the output for each state, any solution that resets to one state, goes through four successive states and holds that fourth state would be correct.

Question 2

Draw the state transition diagram corresponding to the state transition table below. Label each state using the values shown below. **`in`** is an input declared **`input logic in ;`**. Label each transition with a Verilog expression that would have the value 1. for that transition. You need not include transitions that would not change the state.

current state	input **`in`**	next state
0001	1	0010
0001	0	0001
0010	1	0100
0010	0	0010
0100	x	1000
1000	1	0001
1000	0	1000

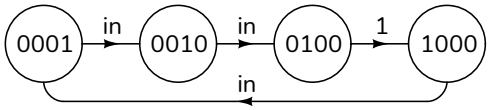
Second version:

current state	input **`in`**	next state
0001	1	0010
0001	0	0001
0010	x	0100
0100	1	1000
0100	0	0100
1000	1	0001
1000	0	1000

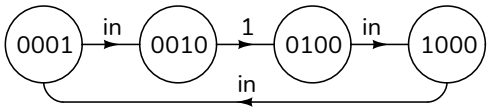
Solutions

A single-bit signal (variable) has only two possible values (0 and 1) so the name itself (**`in`**) is a valid Verilog expression that has the value 1 when the input is 1. Equivalent expressions are **`in == 1`** or **`in != 0`**. An unconditional transition can be labelled with the literal 1 which always has the value 1.

There are four lines that do not cause a change in state so these can be ignored. The diagrams for the remaining transitions is:

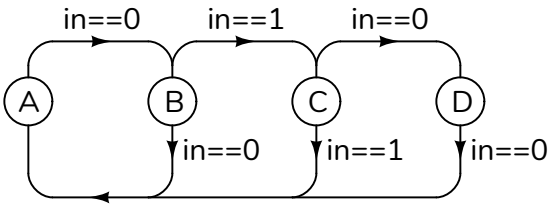


and

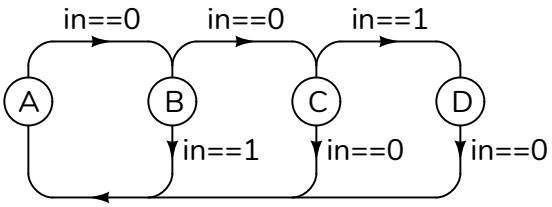


Question 3

Write a Verilog module named **`detector`** that implements the state state transition diagram shown below. It has a one-bit output **`out`**, a one-bit input **`in`**, and a clock input **`clk`**. **`out`** should be set to 1 only in state **`D`**. The output may change only on the rising edge of **`clk`**.



second version:



Follow the mandatory course coding conventions. Declare any signals (variables) required by your solution. You may use any state encoding. You may use any number of **`assign`** and **`always_ff`** statements. You may assume the initial state will be (A). *Hint: come up with a unique encoding for each of the four states.*

Solutions

Any state encoding that has a unique value for each state would work. For example, a binary encoding might be **`2'b00`** for A, **`2'b01`** for B, **`2'b10`** for C, **`2'b11`** for D. A one-hot encoding might be **`4'b0001`** for A, **`4'b0010`** for B, **`4'b0100`** for C, **`4'b1000`** for D. The recommended approach is to use an enumerated type or variable as shown below instead of literals, but any four unique values could be substituted for the four capital letters below.

```
module detector
   ( output logic out,
     input logic in, clk ) ;

   enum int unsigned { A, B, C, D } state ;

   always_ff @(posedge clk)
      state <= state == A && !in ? B :
               state == B &&  in ? C :
               state == C && !in ? D :
               state == B && !in ? A :
               state == C &&  in ? A :
               state == D && !in ? A : state ;

   assign out = state == D ;

endmodule
```

The only change required for the second transition diagram is the **`always_ff`** statement which would change to:

```
   always_ff @(posedge clk)
      state <= state == A && !in ? B :
               state == B && !in ? C :
               state == C &&  in ? D :
               state == B &&  in ? A :
               state == C && !in ? A :
               state == D && !in ? A : state ;
```