Sequence detector

The sequence detector is a single input circuit that will accept a stream of bits (at every clock edge, it will accept one bit) and generate an output '1' whenever the particular sequence is detected. The sequence detector can detect the given sequence either in a non-overlapped manner or in an overlapped manner.

In an overlapping sequence detector, the last bit of one sequence becomes the first bit of the next sequence. However, in a non-overlapping sequence detector, the last bit of one sequence does not become the first bit of the next sequence.

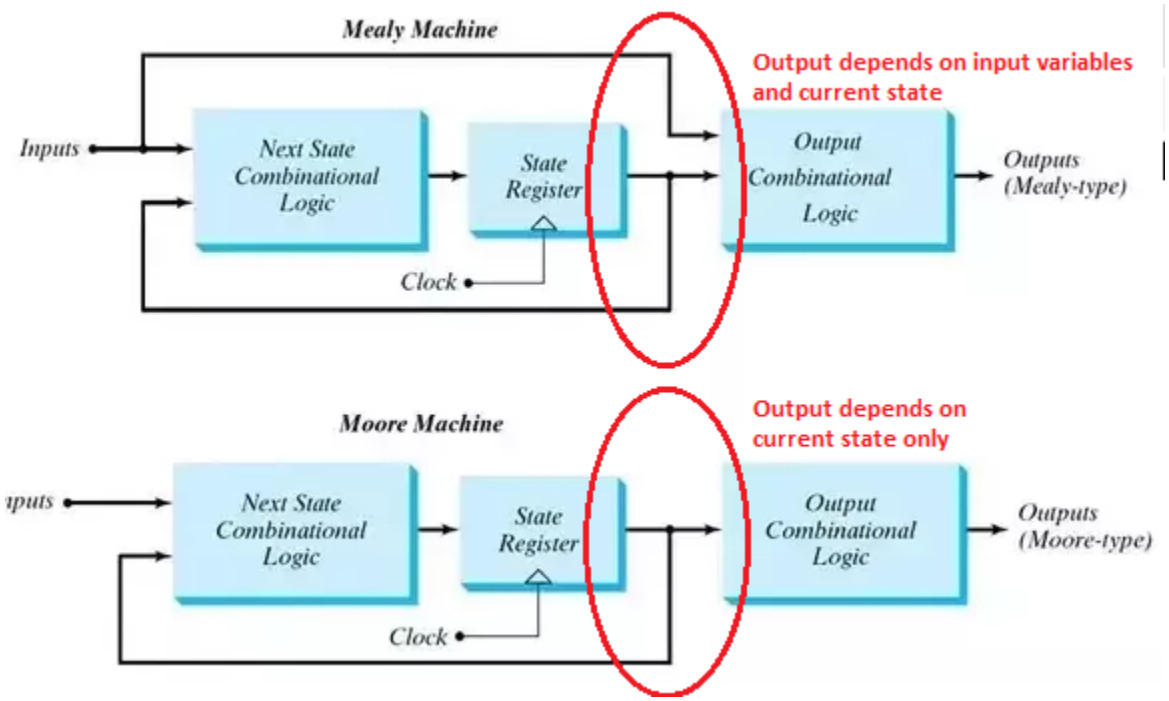
From the word description of the problem, one can understand that the circuit will accept a stream of bits and generate an output 'I' whenever the sequence 1011 is detected. Then, the circuit will go back to the initial state and wait for the next 1011 sequence to generate the output. For example, if the input is 1011011011011, the output generated will be 0001000001000. But in the case of an overlapping sequence detector, the output will be 0001001001001 i.e., additional two 1's are generated due to overlapping sequences.

For non overlapping case
Input : **1011011011011**
Output: **0001000001000**

For overlapping case
Input : **1011011011011**
Output: **0001001001001**

Hints:

- How to design a sequence detector?
  - As you can understand from the given specification, the sequence detector would be a synchronous digital system which essentially be represented as a FSM (Mealy or Moore). So, you must define the state-diagram (consider any type of FSM ~ Mealy or Moore) which will satisfy the given specification.
  - Once you are done with the state diagram, you will have your FSM model which can be seen as a combination of combinational logic and sequential logic (register or memory which will store the previous state).

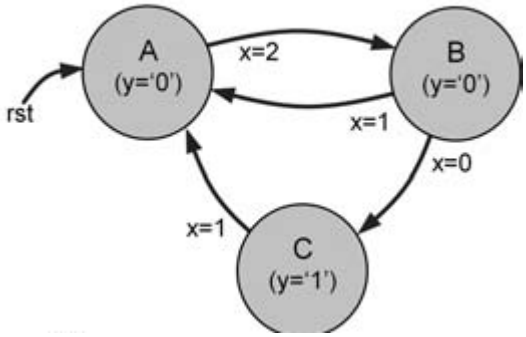


- How to write the codes?

  In your defined FSM (derived from the state diagram), there would be essentially three units that will operate concurrently:

1. A register (driven by the clock)
2. A combinational logic block that will perform state transition
3. Another combinational logic block which will determine output.
   1.

In VHDL, you need to define three concurrent sequential blocks (Processes) in your defined <architecture>. Here is a coding format for a FSM with the following state-transition diagram:



VHDL Template for FSMs

```
---------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------------------
ENTITY <entity_name> IS
   PORT (clk, rst: IN STD_LOGIC;
         input: IN <data_type>;
         output: OUT <data_type>);
END <entity_name>;
---------------------------------------------------------
ARCHITECTURE <architecture_name> OF <entity_name> IS
   TYPE state IS (A, B, C, ...);
   SIGNAL pr_state, nx_state: state;
   ATTRIBUTE ENUM_ENCODING: STRING; --optional attribute
   ATTRIBUTE ENUM_ENCODING OF state: TYPE IS "sequential";
BEGIN
```

-- Process block for the register: (Sequential logic block)

```
18    PROCESS (clk, rst)
19    BEGIN
20       IF (rst='1') THEN
21          pr_state <= A;
22       ELSIF (clk'EVENT AND clk='1') THEN
23          pr_state <= nx_state;
24       END IF;
25    END PROCESS;
```

-- Add another Process block for the Next state logic: (Combinational logic block)

// codes

-- Add another Process block for the FSM output logic: (Combinational logic block)

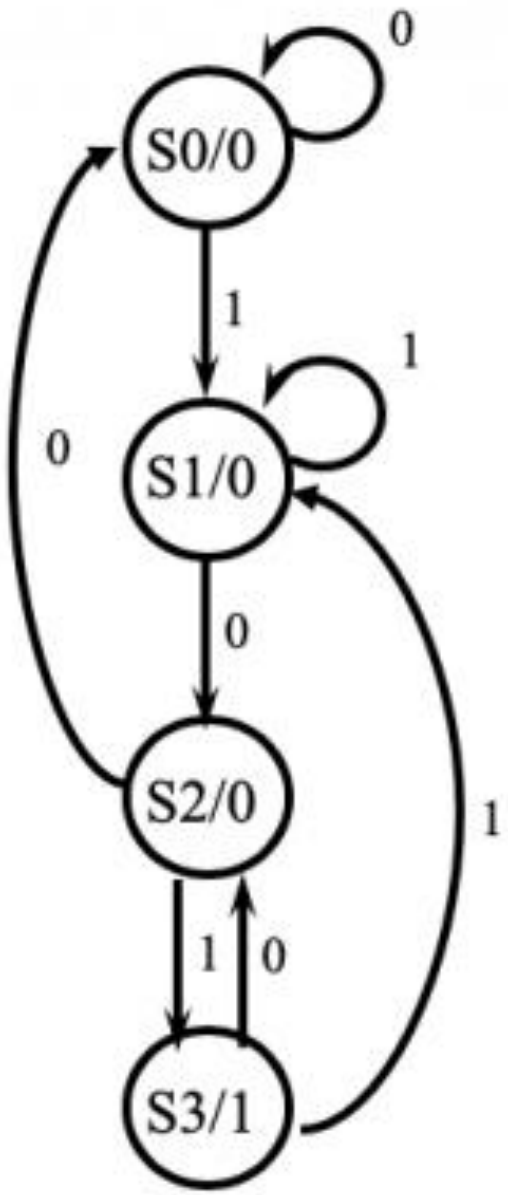
// codes

In line 12, an enumerated data type, called state, is created, then the signals pr_state and nx_state are declared in line 13 as conforming with that data type. Line 14-15 are optional and are used for defining FSM Encoding Styles. For simple behavioural simulation, you may just ignore them or it's okay to use the tool's default FSM encoding style.

Similarly, in Verilog, you may use three <always> blocks:

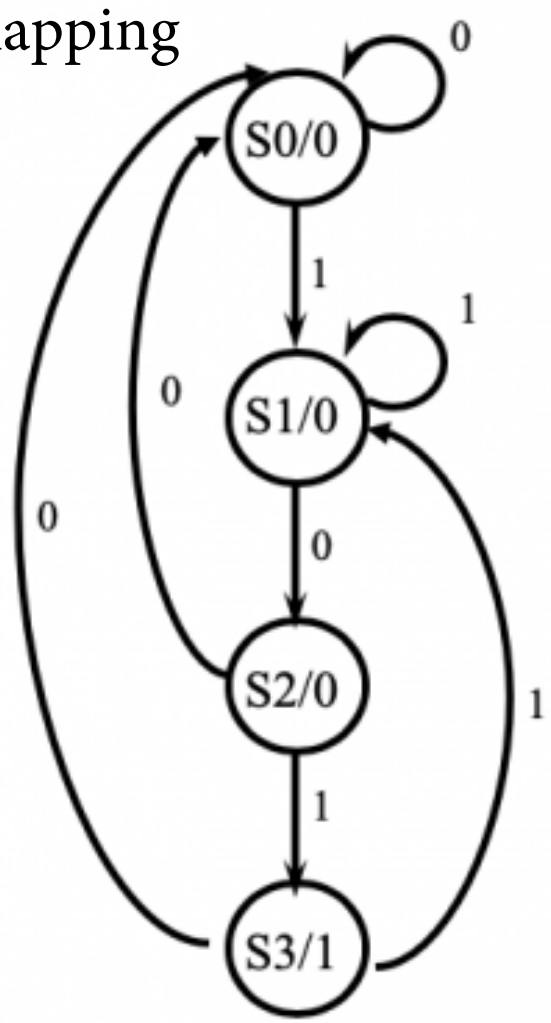
1. One for state transitions
2. One for outputting, and
3. Another one for registering

Moore FSM (101 detector)



Overlapping

Non-Overlapping



Moore FSM (101 detector)

```
1   `timescale 1ns / 1ps
2   module sequence_detector(x,clk,reset,detector_out);
3   input clk; // clock signal
4   input reset; // reset input
5   input x; // binary input
6   output reg detector_out; // output of the sequence detector
7   parameter  s0=2'b00, // "Zero" State
8              s1=2'b01, // "One" State
9              s2=2'b10, // "OneZero" State
10             s3=2'b11; // "OnceZeroOne" State
11  reg [1:0] present_state, next_state; // current state and next state
```

```
// sequential memory of the Moore FSM
always @(posedge clk, posedge reset)
begin
 if(reset==1'b1)
 present_state <= s0;// when reset=1, reset the state of the FSM to "Zero" State
 else
 present_state <= next_state; // otherwise, next state
end
```

```
// combinational logic of the Moore FSM
// to determine next state
always @(present_state,x)
begin
 case(present_state)
 s0:begin
  if(x==1'b1)
   next_state = s1;
  else
   next_state = s0;
 end
 s1:begin
  if(x==1'b0)
   next_state = s2;
  else
   next_state = s1;
 end
 s2:begin
  if(x==1'b0)
   next_state = s0;
  else
   next_state = s3;
 end
 s3:begin
  if(x==1'b0)
   next_state = s0;
  else
   next_state = s1;
 end
 default:next_state = s0;
 endcase
end
```

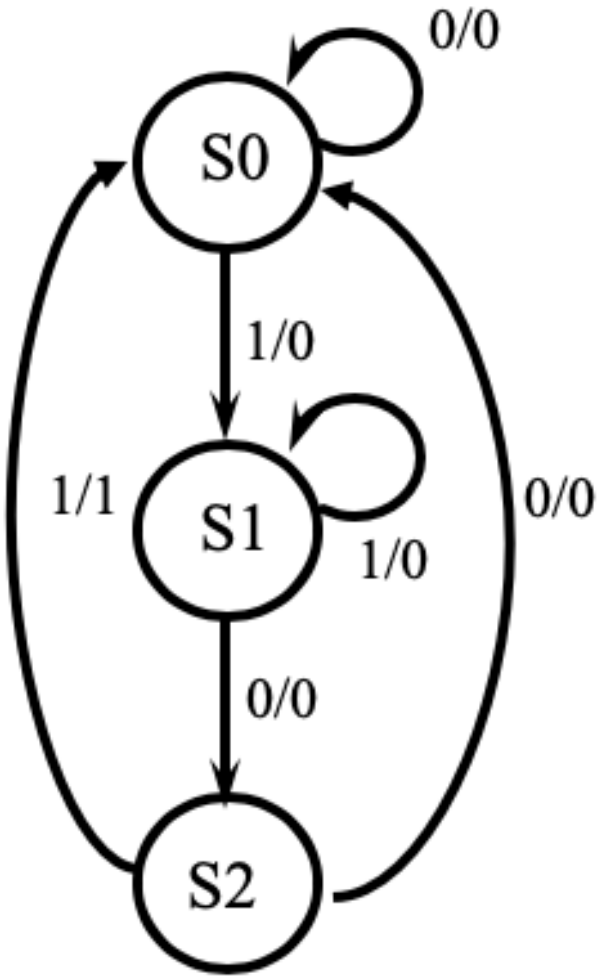
```
// combinational logic to determine the output
// of the Moore FSM, output only depends on current state
always @(present_state)
begin
case(present_state)
 s0:   detector_out = 1'b0;
 s1:   detector_out = 1'b0;
 s2:  detector_out = 1'b0;
 s3:  detector_out = 1'b1;
default:  detector_out = 1'b0;
endcase
end
endmodule
```

Test Bench

```
1   `timescale 1ns / 1ps
2   module sequence_detector_tb;
3
4       // Inputs
5       reg x;
6       reg clk;
7       reg reset;
8
9       // Outputs
10      wire detector_out;
11
12      // Instantiate the Unit Under Test (UUT)
13      sequence_detector uut (
14          .x(x),
15          .clk(clk),
16          .reset(reset),
17          .detector_out(detector_out)
18      );
19
20  initial begin
21   clk = 0;
22   forever #5 clk = ~clk;
23   end
24   initial begin
25    // Initialize Inputs
26    x = 0;
27    reset = 1;
28    // Wait 100 ns for global reset to finish
29    #30;
30    reset = 0;
31    #40;
32    x = 1;
33    #10;
34    x = 0;
35    #10;
36    x = 1;
37    #20;
38    x = 0;
39    #20;
40    x = 1;
41    #20;
42    x = 0;
43    // Add stimulus here
44   end
45  endmodule
```

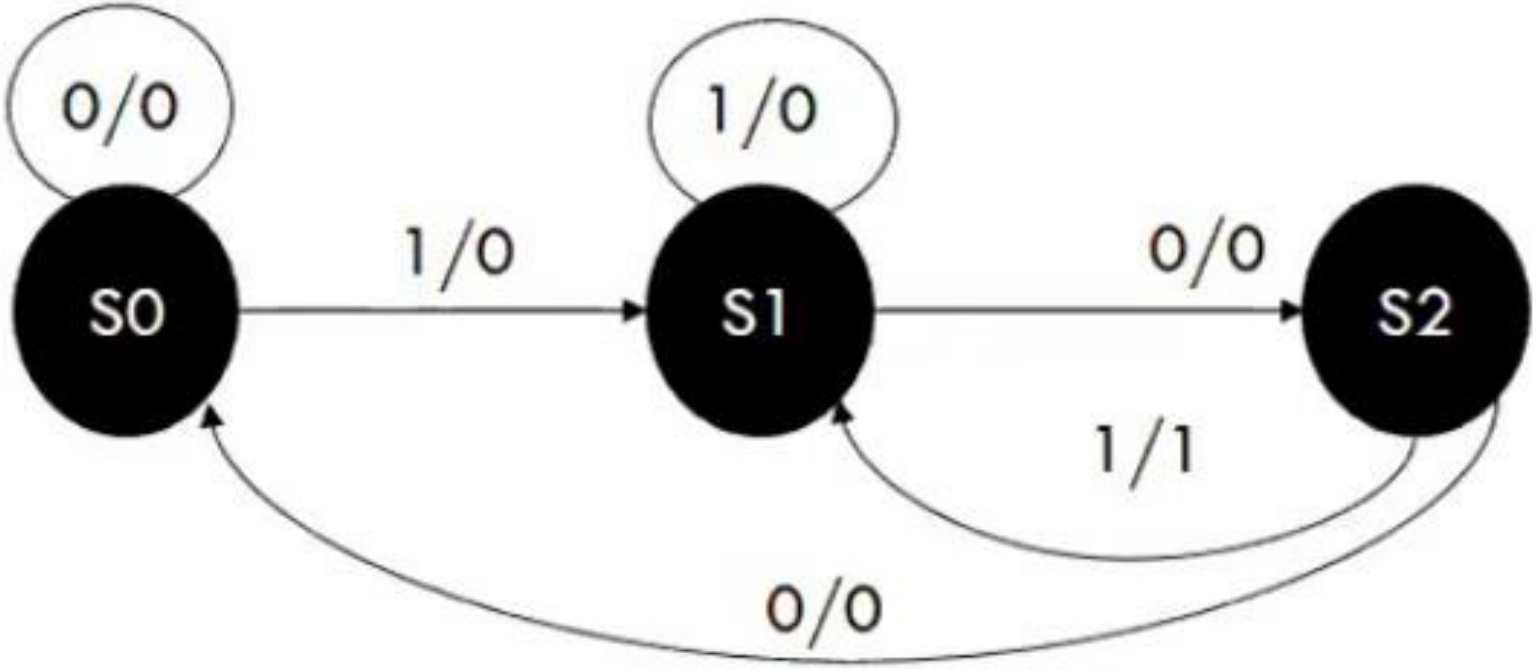
Mealy FSM (101 detector)

Non-overlapping



Mealy FSM (101 detector)

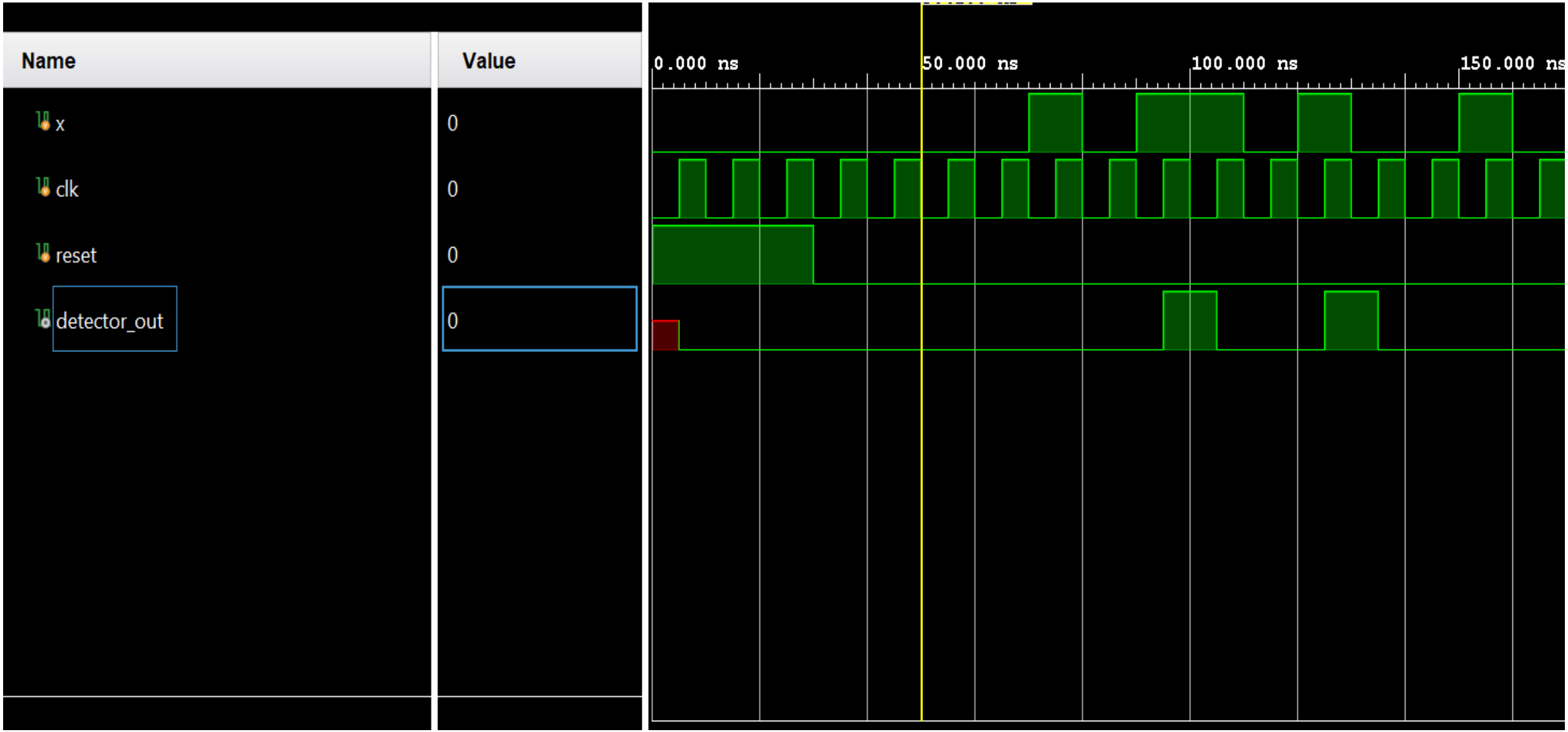
Overlapping



```
`timescale 1ns / 1ps
module sequence_detector_mealy(x,clk,reset,detector_out);
input clk; // clock signal
input reset; // reset input
input x; // binary input
output reg detector_out; // output of the sequence detector
parameter  s0=2'b00, // "Zero" State
           s1=2'b01, // "One" State
           s2=2'b10; // "OneZero" State
reg [1:0] present_state, next_state; // current state and next state
// sequential memory of the Moore FSM
always @(posedge clk, posedge reset)
begin
 if(reset==1'b1)
 present_state <= s0;// when reset=1, reset the state of the FSM to "Zero" State
 else
 present_state <= next_state; // otherwise, next state
end
```

```
// combinational logic of the Mealy FSM
// to determine next state
always @(present_state,x)
begin
case(present_state)
s0:begin
  if(x==1'b1)
   next_state = s1;
  else
   next_state = s0;
end
s1:begin
  if(x==1'b1)
   next_state = s1;
  else
   next_state = s2;
end
s2:begin
  if(x==1'b1)
   next_state = s1;
  else
   next_state = s0;
end
default:next_state = s0;
endcase
end
```

```
// combinational logic to determine the output
// of the Mealy FSM, output depends on current state and external input
always @(present_state,x)
begin
case(present_state)
 s0:   detector_out = 1'b0;
 s1:   detector_out = 1'b0;
 s2:  begin
      if (x==1'b1)
         detector_out = 1'b1;
      else
         detector_out = 1'b0;
      end
default:  detector_out = 1'b0;
endcase
end
endmodule
```


Name
Value
x
0
clk
0
reset
0
detector_out
0
0.000 ns
50.000 ns
100.000 ns
150.000 ns