Design of FSM to detect the sequence 1011 using Verilog

Subham Ball
Dept. of System-on-chip Design
Indian Institute Of Technology–Palakkad
152202017@smail.iitpkd.ac.in

***Abstract*—Tis project goal is to implement and validate the design of a sequence detector in Verilog. It helps us get a better understanding of the FSM level of abstraction**

I. Introduction

A finite state machine is based on a fictitious sequential circuit having one or more states. There can only ever be one active state of this machine. It suggests that the system must switch between states in order to carry out various activities. The device is in charge of recording a situation's state at a specific moment. The adjustments that are put into place immediately affect the states that generate the desired results. In short, Finite State Machines (FSMs) are a functional abstraction for sequential circuits with centralised ”states” of operation. Combinatorial logic computes outputs and the following state as a function of inputs and the current state at each clock edge. The block diagram of a finite state machine is shown in the following figure (Fig. 1)

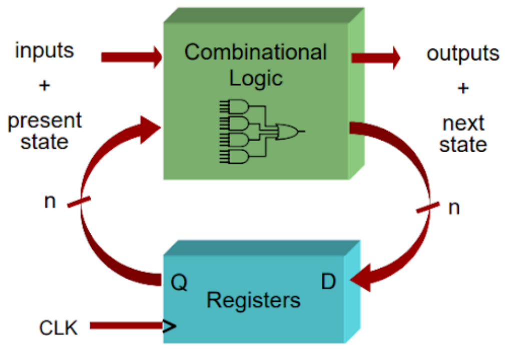


Fig. 1. Block diagram of FSM

A sequence detector, which is a sequential state machine, creates an output 1 whenever the target sequence has been detected after accepting string of bits as input. The current state, as well as any outside input or incoming bit sequences, determine the output. There are two categories of sequence detectors: overlapping and non-overlapping. In contrast to a non-overlapping sequence detector, where the end bit of one sequence does not become the first bit of the following sequence, an overlapping sequence detector assumes the last bit of one sequence as the first bit of the following sequence.

In this experiment, the FSM continually analyses a binary sequence coming from a digital input and only turns on the output when it detects a ”1011” sequence.

We compare the output of the FSM level of abstraction, which is here referred to as the DUT, and the output of the structural description or the reference model. A structural model is defined by a system employing basic parts like digital gates and adders. By integrating different modules and gates, structural modelling explains how a hardware module's components are connected and how the hardware is designed.

II. FSM Level of Abstraction

A. Implementation details

Fig. 2 shows the architectural design of the FSM, which consists of a state register for storing the current state and a combinational network for creating the output and the subsequent state.

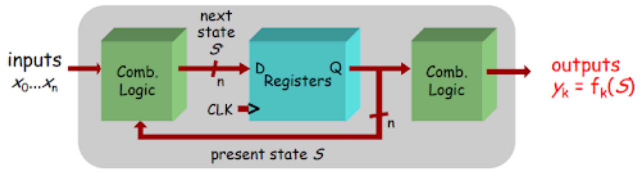


Fig. 2. Architectural layout of FSM

Fig. 3 shows the state transition diagram of the built ”1011” sequence detector. It is considered that the sequence detector in question is non-overlapping in nature. Two three-bit registers, one holding the current state and the other the next state, make up the design. Any one of the five states S0 (sequence: 0), S1 (sequence: 1), S2 (sequence: 10), S3 (sequence: 101), and S4 can be applied to a register (sequence: 1011). The output variable changes to logic 1 when the process reaches the final state S4.

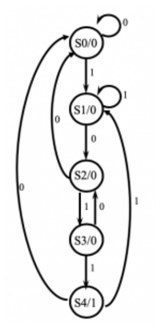
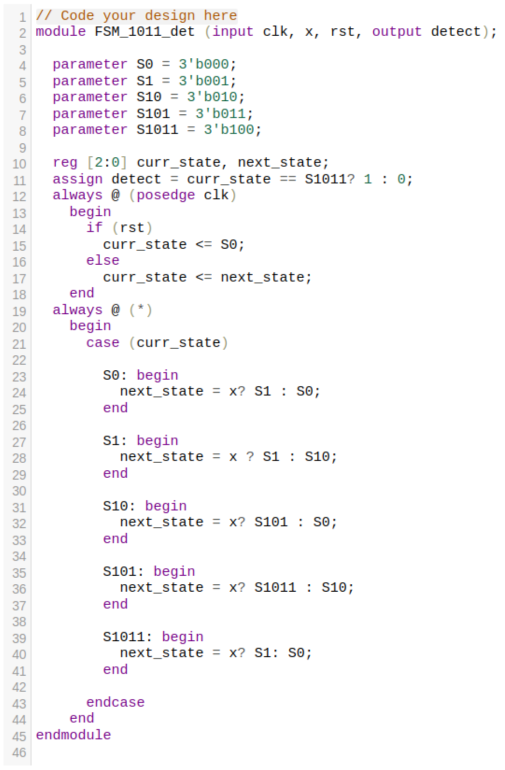


Fig. 3. State transition diagram of 1011 sequence detector

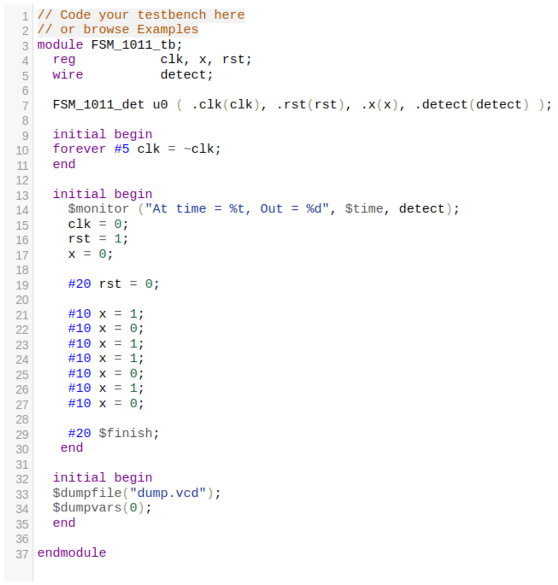
State transition takes place in the always block because at the clock's positive edge, the register holding the current state is changed. In the event of a reset, the register is assigned to the initial state S0; otherwise, it is allocated to the following state, which is determined based on the value of the input sequence bit.

The RTL code for system design at the FSM level of abstraction is shown in Fig. 4. Fig. 5 shows the testbench codes created for verifying the DUT.



```
// Code your design here
module FSM_1011_det (input clk, x, rst, output detect);

  parameter S0 = 3'b000;
  parameter S1 = 3'b001;
  parameter S10 = 3'b010;
  parameter S101 = 3'b011;
  parameter S1011 = 3'b100;

  reg [2:0] curr_state, next_state;
  assign detect = curr_state == S1011? 1 : 0;
  always @ (posedge clk)
    begin
      if (rst)
        curr_state <= S0;
      else
        curr_state <= next_state;
    end
  always @ (*)
    begin
      case (curr_state)

        S0: begin
          next_state = x? S1 : S0;
        end

        S1: begin
          next_state = x ? S1 : S10;
        end

        S10: begin
          next_state = x? S101 : S0;
        end

        S101: begin
          next_state = x? S1011 : S10;
        end

        S1011: begin
          next_state = x? S1: S0;
        end

      endcase
    end
endmodule

```

Fig. 4. RTL code



```
// Code your testbench here
// or browse Examples
module FSM_1011_tb;
  reg           clk, x, rst;
  wire          detect;

  FSM_1011_det u0 ( .clk(clk), .rst(rst), .x(x), .detect(detect) );

  initial begin
  forever #5 clk = ~clk;
  end

  initial begin
    $monitor ("At time = %t, Out = %d", $time, detect);
    clk = 0;
    rst = 1;
    x = 0;

    #20 rst = 0;

    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #10 x = 0;

    #20 $finish;
   end

  initial begin
  $dumpfile("dump.vcd");
  $dumpvars(0);
  end

endmodule
```

Fig. 5. Testbench of DUT

B. *Experimental results*

Using Cadence software, Verilog architecture of the FSM level of abstraction with a clock, reset, and two registers is simulated and synthesised. Figs. 6 and 7 depict the synthesised circuit and observed waveform, respectively.

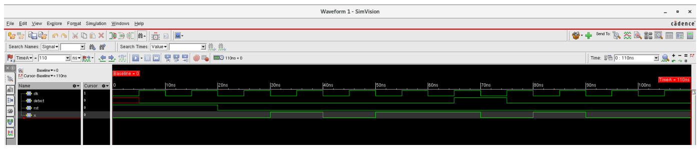


Fig. 6. Output Waveform

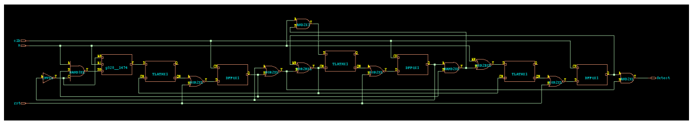
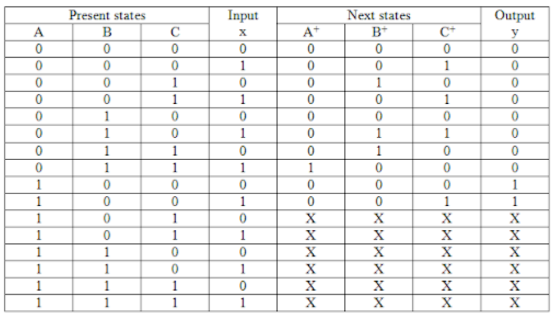


Fig. 7. Synthesized Circuit Diagram

III. Structural description

A. *Implementation details:*

The state transition diagram in Fig. 3 is used to create the state table of the sequence detector, depicted in Fig. 8.

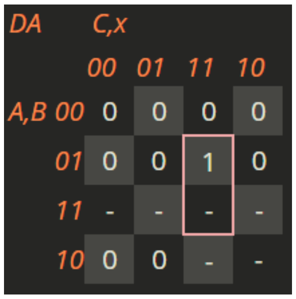


Present states			Input	Next states			Output
A	B	C	x	A^+	B^+	C^+	y
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0
0	0	1	1	0	0	1	0
0	1	0	0	0	0	0	0
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Fig. 8. State Table

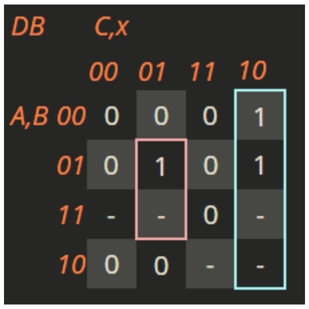
The three-bit state parameter of the FSM level of abstraction has distinct registers assigned to each of its three bits. Three one-bit registers are needed to handle this, and an additional one-bit register is needed for the output. The state table, which is applicable to the development of the combinational logic, serves as the source of the Karnaugh Maps for each of these registers.

Figures 9, 10, 11, and 12 show the Karnaugh Maps for the updated values of the state parameter's first bit (A), second bit (B), last bit (C), and output, which would be assigned to the registers, accordingly.



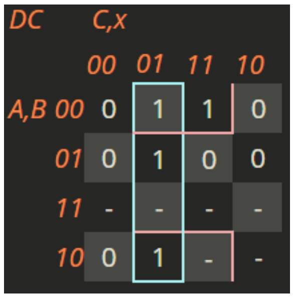
DA	C,x			
	00	01	11	10
A,B 00	0	0	0	0
01	0	0	1	0
11	-	-	-	-
10	0	0	-	-

Fig. 9. Karnaugh Map of first bit A



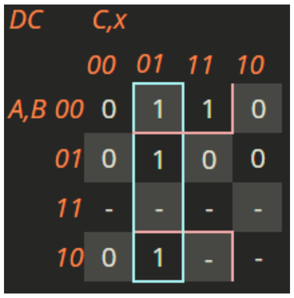
DB	C,x			
	00	01	11	10
A,B 00	0	0	0	1
01	0	1	0	1
11	-	-	0	-
10	0	0	-	-

Fig. 10. Karnaugh Map of second bit B



DC	C,x			
	00	01	11	10
A,B 00	0	1	1	0
01	0	1	0	0
11	-	-	-	-
10	0	1	-	-

Fig. 11. Karnaugh Map of last bit C



DC	C,x			
	00	01	11	10
A,B 00	0	1	1	0
01	0	1	0	0
11	-	-	-	-
10	0	1	-	-

Fig. 12. Karnaugh Map of output Y

The following formulations represent the logical expressions generated from the Karnaugh Maps:

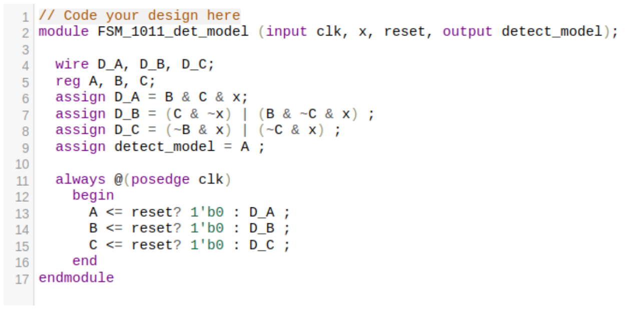
$$DA(A,B,C,x) = BCx$$

$$DB(A,B,C,x) = Cx' + BC'x$$

$$DC(A,B,C,x) = B'x + C'x$$

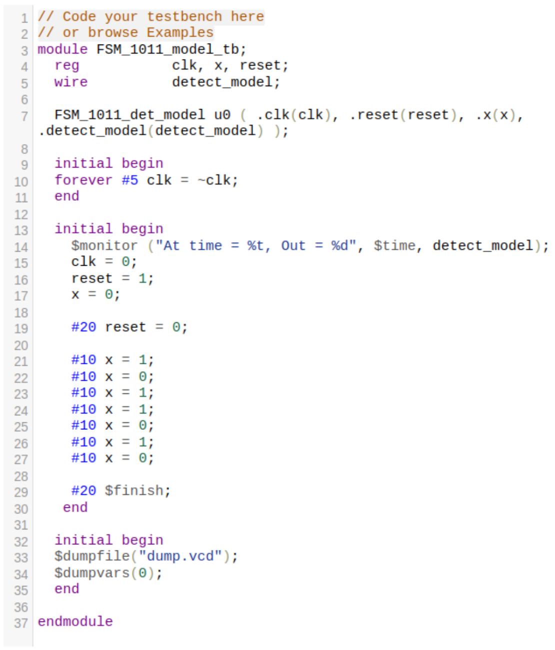
$$Y(A,B,C,x) = A$$

The RTL code for the structural design of the "1011" sequence detector is shown in Fig. 13. Fig. 14 shows the testbench codes created for verifying the reference model.



```
// Code your design here
module FSM_1011_det_model (input clk, x, reset, output detect_model);

  wire D_A, D_B, D_C;
  reg A, B, C;
  assign D_A = B & C & x;
  assign D_B = (C & ~x) | (B & ~C & x) ;
  assign D_C = (~B & x) | (~C & x) ;
  assign detect_model = A ;

  always @(posedge clk)
    begin
      A <= reset? 1'b0 : D_A ;
      B <= reset? 1'b0 : D_B ;
      C <= reset? 1'b0 : D_C ;
    end
endmodule
```

Fig. 13. RTL code



```verilog
// Code your testbench here
// or browse Examples
module FSM_1011_model_tb;
  reg           clk, x, reset;
  wire          detect_model;

  FSM_1011_det_model u0 ( .clk(clk), .reset(reset), .x(x),
.detect_model(detect_model) );

  initial begin
  forever #5 clk = ~clk;
  end

  initial begin
    $monitor ("At time = %t, Out = %d", $time, detect_model);
    clk = 0;
    reset = 1;
    x = 0;

    #20 reset = 0;

    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #10 x = 0;

    #20 $finish;
   end

  initial begin
  $dumpfile("dump.vcd");
  $dumpvars(0);
  end

endmodule
```

Fig. 14. Testbench code

B. *Experimental results*

Cadence software is used to simulate and synthesis the Verilog design of the structural description, which is the golden reference model. The circuit has a clock and four registers. In Figures 15 and 16, respectively, the observed waveform and the synthesised circuit are displayed.

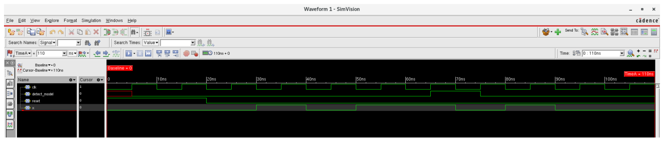


Fig. 15. Output Waveform

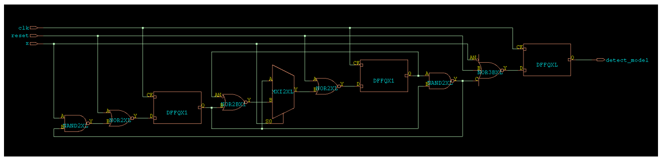
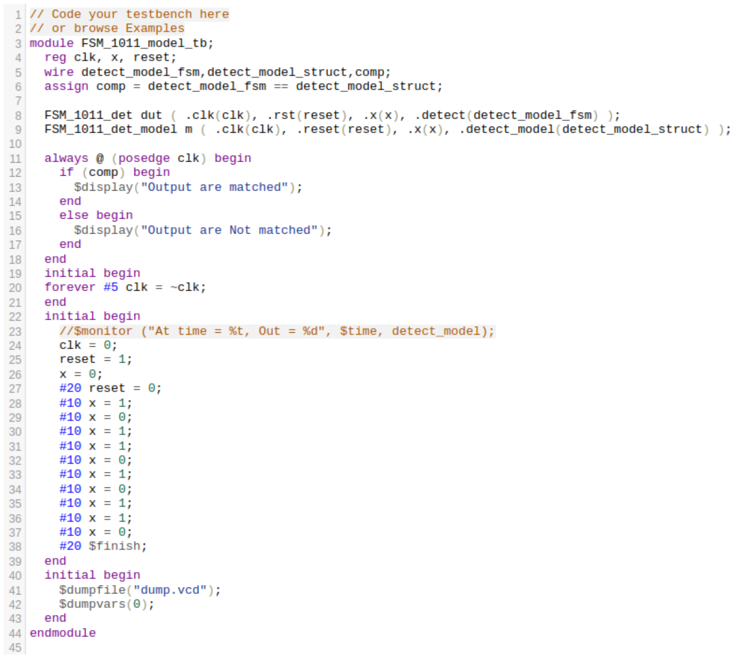


Fig. 16. Synthesized Circuit Diagram

IV. Comparison of the two models

A. *Implementation details:*

The FSM level of abstraction and structural description modules are both instantiated and their outputs are compared. The testbench which executes this task is shown below in Fig. 17.



```verilog
// Code your testbench here
// or browse Examples
module FSM_1011_model_tb;
  reg clk, x, reset;
  wire detect_model_fsm,detect_model_struct,comp;
  assign comp = detect_model_fsm == detect_model_struct;

  FSM_1011_det dut ( .clk(clk), .rst(reset), .x(x), .detect(detect_model_fsm) );
  FSM_1011_det_model m ( .clk(clk), .reset(reset), .x(x), .detect_model(detect_model_struct) );

  always @ (posedge clk) begin
    if (comp) begin
      $display("Output are matched");
    end
    else begin
      $display("Output are Not matched");
    end
  end
  initial begin
  forever #5 clk = ~clk;
  end
  initial begin
    //$monitor ("At time = %t, Out = %d", $time, detect_model);
    clk = 0;
    reset = 1;
    x = 0;
    #20 reset = 0;
    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #10 x = 1;
    #10 x = 0;
    #20 $finish;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end
endmodule
```

Fig. 17. Testbench performing the comparison

B. *Experimental results*

Utilizing Cadence software, the Verilog design that compares the structural description modules and FSM degree of abstraction is simulated. In Fig.18, the observed waveform is displayed.

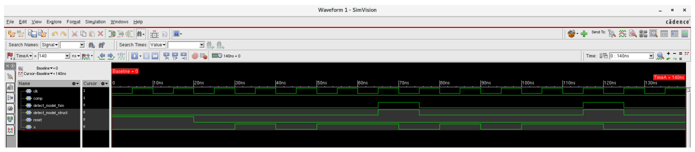
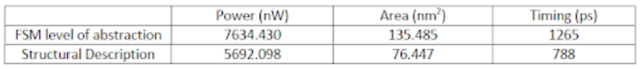


Fig. 18. Output Waveform

V. Analysis

The following table (Fig. 19) shows the comparison of the DUT and reference model with regards to power, area and time.



	Power (nW)	Area (nm^2)	Timing (ps)
FSM level of abstraction	7634.430	135.485	1265
Structural Description	5692.098	76.447	788

Fig. 19. Comparative analysis

VI. Conclusion

The ”1011” sequence detector’s FSM level of abstraction is effectively applied in Verilog design and is synthesised and simulated using Cadence software. The DUT is also validated using the structural description, which functioned as the golden reference model. The reference model exhibits minimal time, area and power parameters when compared to the FSM level of abstraction. We consider the FSM level of abstraction to be the best way to build the sequence detector since constructing the schematic diagram from scratch in structural description prevents the compiler from optimising the design to meet requirements that are not merely concerned with exactness.