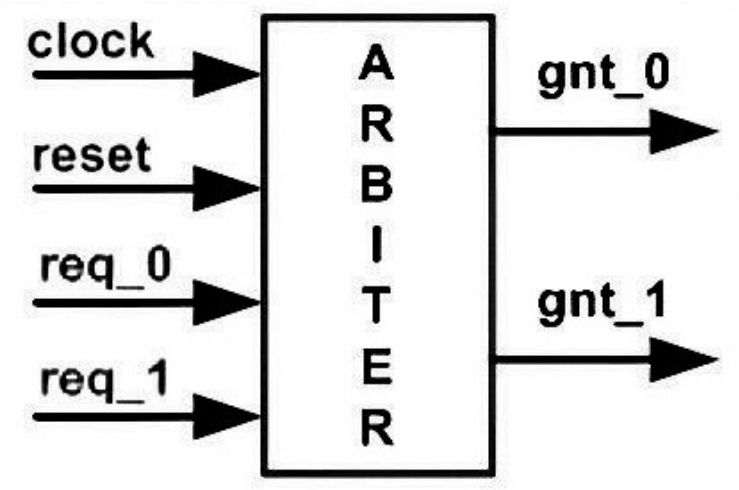
EXPERIMENT- 8
Design of Efficient Synthesizable Finite State Machine

Objective:

RTL coding styles for efficient and synthesizable Finite State Machine (FSM) design using IEEE-compliant Verilog simulators.

Given: An example of Arbiter

A simple arbiter, as the example, has got two request inputs and two grant outputs, as shown in the signal diagram below.



When req_0 is asserted, gnt_0 is asserted

When req_1 is asserted, gnt_1 is asserted

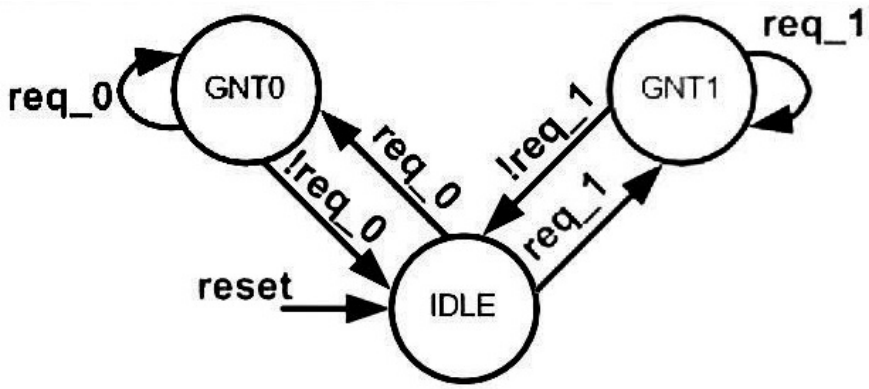
When both req_0 and req_1 are asserted then gnt_0 is asserted i.e. higher priority is given to req_0 over req_1.

We can symbolically translate into a FSM diagram as shown in figure below, here FSM has got following states.

IDLE : In this state FSM waits for the assertion of req_0 or req_1 and drives both gnt_0 and gnt_1 to inactive state (low). This is the default state of the FSM, it is entered after the reset and also during fault recovery condition.

GNT0 : FSM enters this state when req_0 is asserted, and remains here as long as req_0 is asserted. When req_0 is de-asserted, FSM returns to the IDLE state.

GNT1 : FSM enters this state when req_1 is asserted, and remains there as long as req_1 is asserted. When req_1 is de-asserted, FSM returns to the IDLE state.o understand the concepts related to FSM (Moore/Mealy machine) and design the controller for a robot with the given specifications. Write the Verilog Programs for the same using any modeling style.



Exercise Problems

1. FSM program using a function for combo logic

Source Code

```
module fsm_using_function (clock,reset,req_0,req_1,gnt_0,gnt_1);
  input clock,reset,req_0,req_1;
  output gnt_0,gnt_1;
  wire clock,reset,req_0,req_1;
  reg gnt_0,gnt_1;
  parameter SIZE = 3;
  parameter IDLE = 3'b001,GNT0 = 3'b010,GNT1 = 3'b100;
  reg [SIZE-1:0] state;
  wire [SIZE-1:0] next_state;
  assign next_state = fsm_function(state, req_0, req_1);
  function [SIZE-1:0] fsm_function;
    input [SIZE-1:0] state;
    input req_0;
    input req_1;
    case(state)
      IDLE:
        if (req_0 == 1'b1)
        begin
          fsm_function = GNT0;
```

```
        end
        else if (req_1 == 1'b1)
        begin
          fsm_function = GNT1;
        end
        else
        begin
          fsm_function = IDLE;
        end
      GNT0:
        if (req_0 == 1'b1)
        begin
          fsm_function = GNT0;
        end
        else
        begin
          fsm_function = IDLE;
        end
      GNT1:
        if (req_1 == 1'b1)
        begin
          fsm_function = GNT1;
        end
        else
        begin
          fsm_function = IDLE;
        end
      default:
        fsm_function = IDLE;
    endcase
  endfunction
  always @ (posedge clock)
  begin : FSM_SEQ
    if (reset == 1'b1)
    begin
      state <= #1 IDLE;
    end
    else
    begin
      state <= #1 next_state;
    end
  end
```

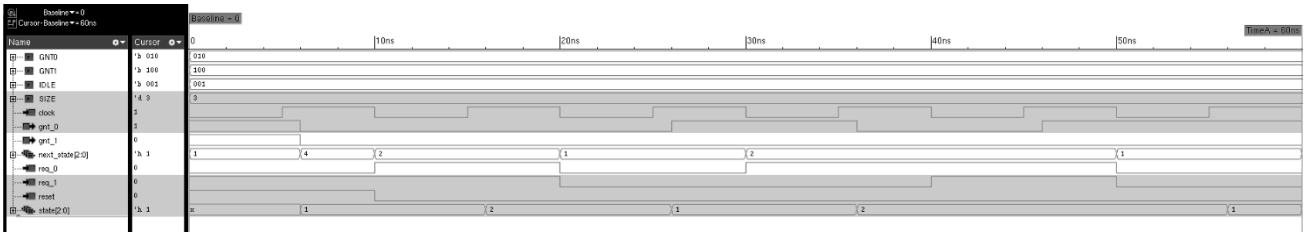
```
  always @ (posedge clock)
  begin : OUTPUT_LOGIC
    if (reset == 1'b1)
    begin
      gnt_0 <= #1 1'b0;
      gnt_1 <= #1 1'b0;
    end
    else
    begin
      case(state)
        IDLE:
        begin
          gnt_0 <= #1 1'b0;
          gnt_1 <= #1 1'b0;
        end
        GNT0:
        begin
          gnt_0 <= #1 1'b1;
          gnt_1 <= #1 1'b0;
        end
        GNT1:
        begin
          gnt_0 <= #1 1'b0;
          gnt_1 <= #1 1'b1;
        end
        default:
        begin
          gnt_0 <= #1 1'b0;
          gnt_1 <= #1 1'b0;
        end
      endcase
    end
  end
endmodule
```

Testbench

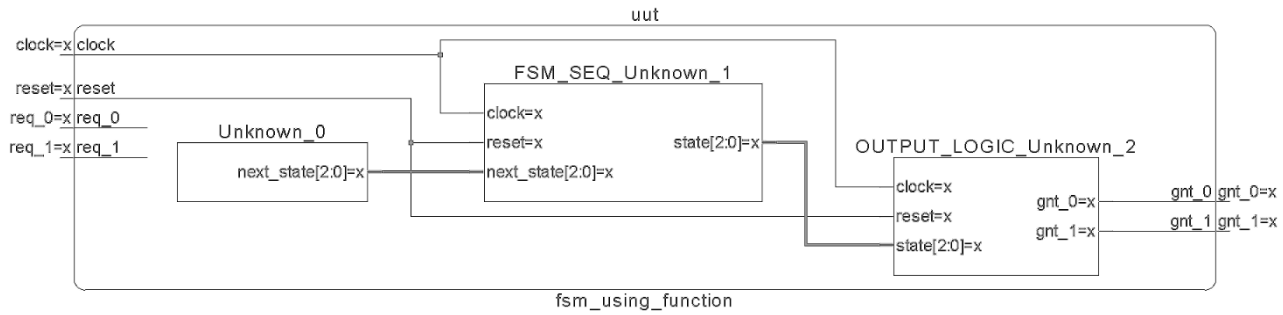
```
module tb();
  reg clock,reset,req_0,req_1;
  wire gnt_0,gnt_1;
  fsm_using_function uut(clock,reset,req_0,req_1,gnt_0,gnt_1);
  initial
```

```
  begin
    clock=0;
    forever
      #5 clock=~clock;
  end
  initial
  begin
    reset=1;
    #10;
    reset=0;
  end
  initial
  begin
    req_1=1'b1;
    req_0=1'b0;
    #10;
    req_1=1'b1;
    req_0=1'b1;
    #10;
    req_1=1'b0;
    req_0=1'b0;
    #10;
    req_1=1'b0;
    req_0=1'b1;
    #10;
    req_1=1'b1;
    req_0=1'b1;
    #10;
    req_1=1'b0;
    req_0=1'b0;
    #10;
    $finish;
  end
endmodule
```

Waveform



Schematic



2. FSM program Using two always blocks

Source Code

```
module fsm_using_always (clock,reset,req_0,req_1,gnt_0,gnt_1);
  input clock,reset,req_0,req_1;
  output gnt_0,gnt_1;
  wire clock,reset,req_0,req_1;
  reg gnt_0,gnt_1;
  parameter SIZE = 3;
  parameter IDLE = 3'b001,GNT0 = 3'b010,GNT1 = 3'b100;
  reg [SIZE-1:0] state;
  reg [SIZE-1:0] next_state;
  always @ (state or req_0 or req_1)
  begin : FSM_COMBO
    next_state = 3'b000;
    case(state)
      IDLE:
        if (req_0 == 1'b1)
        begin
          next_state = GNT0;
        end
        else if (req_1 == 1'b1)
        begin
          next_state= GNT1;
        end
        else
        begin
          next_state = IDLE;
        end
      GNT0:
        if (req_0 == 1'b1)
        begin
```

```
          next_state = GNT0;
        end
        else
        begin
          next_state = IDLE;
        end
      GNT1:
        if (req_1 == 1'b1)
        begin
          next_state = GNT1;
        end
        else
        begin
          next_state = IDLE;
        end
      default:
        next_state = IDLE;
    endcase
  end
  always @ (posedge clock)
  begin : FSM_SEQ
    if (reset == 1'b1)
    begin
      state <= #1 IDLE;
    end
    else
    begin
      state <= #1 next_state;
    end
  end
  always @ (posedge clock)
  begin : OUTPUT_LOGIC
    if (reset == 1'b1)
    begin
      gnt_0 <= #1 1'b0;
      gnt_1 <= #1 1'b0;
    end
    else
    begin
      case(state)
        IDLE:
        begin
```

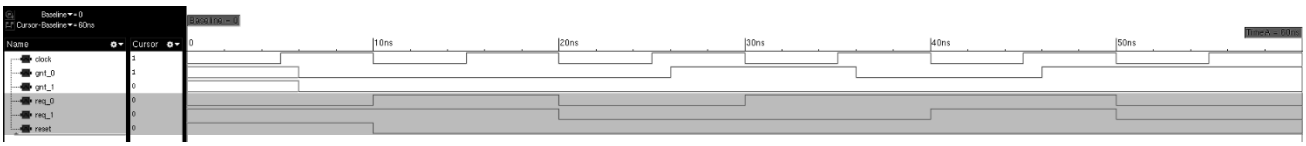
```
          gnt_0 <= #1 1'b0;
          gnt_1 <= #1 1'b0;
        end
        GNT0:
        begin
          gnt_0 <= #1 1'b1;
          gnt_1 <= #1 1'b0;
        end
        GNT1:
        begin
          gnt_0 <= #1 1'b0;
          gnt_1 <= #1 1'b1;
        end
        default:
        begin
          gnt_0 <= #1 1'b0;
          gnt_1 <= #1 1'b0;
        end
      endcase
    end
  end
endmodule
```

Testbench

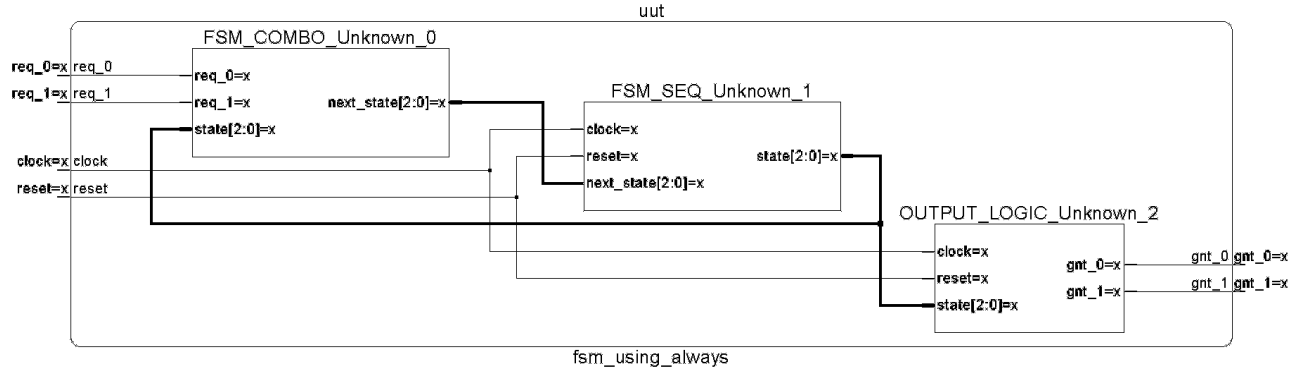
```
module tb();
  reg clock,reset,req_0,req_1;
  wire gnt_0,gnt_1;
  fsm_using_always uut(clock,reset,req_0,req_1,gnt_0,gnt_1);
  initial
  begin
    clock=0;
    forever
      #5 clock=~clock;
  end
  initial
  begin
    reset=1;
    #10;
    reset=0;
  end
  initial
```

```
  begin
    req_1=1'b1;
    req_0=1'b0;
    #10;
    req_1=1'b1;
    req_0=1'b1;
    #10;
    req_1=1'b0;
    req_0=1'b0;
    #10;
    req_1=1'b0;
    req_0=1'b1;
    #10;
    req_1=1'b1;
    req_0=1'b1;
    #10;
    req_1=1'b0;
    req_0=1'b0;
    #10;
    $finish;
  end
endmodule
```

Waveform



Schematic



3. FSM program using single always for both sequential and combinational logic and output logice

Source Code

```
module fsm_using_single_always (clock,reset,req_0,req_1,gnt_0,gnt_1);
  input clock,reset,req_0,req_1;
  output gnt_0,gnt_1;
  wire clock,reset,req_0,req_1;
  reg gnt_0,gnt_1;
  parameter SIZE = 3;
  parameter IDLE = 3'b001,GNT0 = 3'b010,GNT1 = 3'b100;
  reg [SIZE-1:0] state;
  reg [SIZE-1:0] next_state;
  always @ (posedge clock)
  begin : FSM
    if (reset == 1'b1)
    begin
      state <= #1 IDLE;
      gnt_0 <= 0;
      gnt_1 <= 0;
    end
    else
    case(state)
      IDLE:
        if (req_0 == 1'b1)
        begin
          state <= #1 GNT0;
          gnt_0 <= 1;
        end
        else if (req_1 == 1'b1)
        begin
          gnt_1 <= 1;
          state <= #1 GNT1;
        end
        else
        begin
          state <= #1 IDLE;
        end
      GNT0:
        if (req_0 == 1'b1)
        begin
```

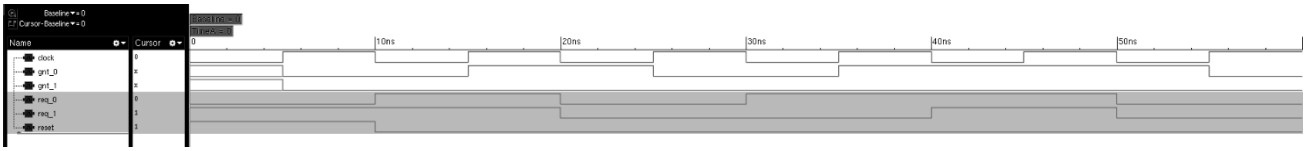
```
          state <= #1 GNT0;
        end
        else
        begin
          gnt_0 <= 0;
          state <= #1 IDLE;
        end
      GNT1:
        if (req_1 == 1'b1)
        begin
          state <= #1 GNT1;
        end
        else
        begin
          gnt_1 <= 0;
          state <= #1 IDLE;
        end
      default:
        state <= #1 IDLE;
    endcase
  end
endmodule
```

Testbench

```
module tb();
  reg clock,reset,req_0,req_1;
  wire gnt_0,gnt_1;
  fsm_using_single_always uut(clock,reset,req_0,req_1,gnt_0,gnt_1);
  initial
  begin
    clock=0;
    forever
      #5 clock=~clock;
  end
  initial
  begin
    reset=1;
    #10;
    reset=0;
  end
  initial
```

```
  begin
    req_1=1'b1;
    req_0=1'b0;
    #10;
    req_1=1'b1;
    req_0=1'b1;
    #10;
    req_1=1'b0;
    req_0=1'b0;
    #10;
    req_1=1'b0;
    req_0=1'b1;
    #10;
    req_1=1'b1;
    req_0=1'b1;
    #10;
    req_1=1'b0;
    req_0=1'b0;
    #10;
    $finish;
  end
endmodule
```

Waveform



Schematic