Digital Logic Design Experiment *5*

Name: SPARSH ARYA

Registration Number: 17BEC0656

FSM:

```
module fsm_melay_101 (clk,rst,in,out);
input clk,rst,in;
output reg out;
reg [1:0] pre_state, next_state;
always@(posedge clk)
if (rst)
pre_state<=2'b00;
else
pre_state<=next_state;
 always @(posedge clk or in)
begin
case({pre_state,in})
3'b000: begin
                    next_state<=2'b00;
                    out<=1'b0;
                    end
3'b001: begin

                    next_state<=2'b01;
                    out<=1'b0;
                    end
3'b010: begin

                    next_state<=2'b10;
                    out<=1'b0;
                    end
3'b011: begin

                    next_state<=2'b01;
                    out<=1'b0;
                    end
3'b100: begin

                    next_state<=2'b00;
                    out<=1'b0;
                    end
3'b101: begin

                    next_state<=2'b01;
                    out<=1'b1;
                    end
```

```
endcase
end
endmodule
```

FSM testbench:

```
module fsm_101_tst;
wire out;
reg clk,rst,in;
fsm_melay_101 uut (clk,rst,in,out);
initial
begin
clk=0;
rst=1;
in=0;
#10 rst=0;

#10 in=1;
#10 in=0;
#10 in=1;
#10 in=1;
#10 in=0;
#10 in=1;
#10 in=1;
#10 in=0;
#10 in=1;

#10 $finish;
end
always #5 clk=~clk;
endmodule
```

FSM 101:

```
module pan_fsm_101 (clk,rst,in,out);
input clk,rst,in;
output out;
reg out;
reg [1:0]pre_state,nxt_state;
parameter S0 = 2'b00;
parameter S1 = 2'b01;
```

```
parameter S2 = 2'b10;
parameter S3 = 2'b11;

always@(posedge clk)
if(rst)
pre_state <= S0;
else
pre_state <= nxt_state;

always@(pre_state or in)
begin
case(pre_state)
S0 :    if(in)
                begin
                nxt_state <= S1;
                out <=0;
                end
        else
                begin
                nxt_state <= S0;
                out <= 0;
                end

S1 :    if(in)
                begin
                nxt_state <= S1;
                out <=0;
                end
        else
                begin
                nxt_state <= S2;
                out <= 0;
                end

S2 :    if(in)
                begin
                nxt_state <= S3;
                out <=0;
                end
        else
                begin
                nxt_state <= S0;
                out <= 0;
                end
```

```
S3 :    if(in)
                        begin
                        nxt_state <= S1;
                        out <=1;
                        end
                else
                        begin
                        nxt_state <= S2;
                        out <= 0;
                        end
endcase
end
endmodule
```

FSM testbench:

```
module pan_fsm_101_tst;
 wire out;
 reg rst,clk,in;
 pan_fsm_101 m1 (clk,rst,in,out);
 initial
 begin
 clk=0;rst=1;in=0;
 #10 rst =0;
 #10 in =1;
 #10 in =0;#10 in =1;#10 in =1;#10 in =0;
 #10 in =1;#10 in =1;#10 in =0;#10 in =1;
#10 $finish;
end
always
#5 clk=~clk;

Endmodule
```

Counters:

```
module up_counter(input clk, reset, output[3:0] counter
  );
```

```
reg [3:0] counter_up;

// up counter
always @(posedge clk or posedge reset)
begin
if(reset)
 counter_up <= 4'd0;
else
 counter_up <= counter_up + 4'd1;
end
assign counter = counter_up;
endmodule
```

Counters testbench:

```
module upcounter_testbench();
reg clk, reset;
wire [3:0] counter;

up_counter dut(clk, reset, counter);
initial begin
clk=0;
forever #5 clk=~clk;
end
initial begin
reset=1;
#20;
reset=0;
end
endmodule
```

Down counter:

```
module down_counter(input clk, reset, output [3:0] counter
  );
reg [3:0] counter_down;
```

```
// down counter
always @(posedge clk or posedge reset)
begin
if(reset)
 counter_down <= 4'hf;
else
 counter_down <= counter_down - 4'd1;
end
assign counter = counter_down;
endmodule
```

Down counter testbench:

```
module downcounter_testbench();
reg clk, reset;
wire [3:0] counter;

down_counter dut(clk, reset, counter);
initial begin
clk=0;
forever #5 clk=~clk;
end
initial begin
reset=1;
#20;
reset=0;
end
endmodule
```

Up-down counter:

```
module up_down_counter(input clk, reset,up_down, output[3:0]  counter
   );
reg [3:0] counter_up_down;

// down counter
always @(posedge clk or posedge reset)
begin
if(reset)
 counter_up_down <= 4'h0;
else if(~up_down)
 counter_up_down <= counter_up_down + 4'd1;
```

```
else
 counter_up_down <= counter_up_down - 4'd1;
end
assign counter = counter_up_down;
endmodule
```

Up-down counter testbench:

```
module updowncounter_testbench();
reg clk, reset,up_down;
wire [3:0] counter;

up_down_counter dut(clk, reset,up_down, counter);
initial begin
clk=0;
forever #5 clk=~clk;
end
initial begin
reset=1;
up_down=0;
#20;
reset=0;
#200;
up_down=1;
end
endmodule
```

END.