Sequential Circuits in Verilog

Verilog Module Structure

Module STARTING

```
module design
 (
  input  [2:0] data,
  output [2:0] result
  );
```

Module name portlist , port declaration parameter declarations

```
  reg reset_n, clk;
  reg done;
  reg FF_Q, FF_D;
  integer     i;
  wire [2:0]  data_in, data_out;
```

Declaration of wire and reg variables

```
  alg_module alg_instance (
  .clk     (clk ),
  .reset_n (reset_n),
  .d_in     (data_in),
  .d_out    (data_out)
  );
```

Instantiation of other module

```
  assign data_in = 3'b001 + data;
  assign result=data_out ^ data_in;
```

Continuous statements or Data flow statement

```
  initial begin
    forever #(5) clk=~clk;
  end
```

Initial block: blocking assignments

always block: Non-blocking assignments

```
always @(*) begin
  done=0;
  if ( start==1 ) begin
                done=1;
  end else if ( start==0 ) begin
                done=0;
  end else begin
                done=1;
  end

  case(FF_D)
                1'b0: begin
                  FF_D=0;
                end
                1'b1: begin
                  FF_D=1;
                end
  endcase
end
```

always block: Non-blocking assignments

```
always @ ( posedge clk  or negedge reset_n) begin
  if (reset_n ==0) begin
                FF_Q  <=  0 ;
  end  else begin
                FF_Q  <= FF_D;
  end
end
```

tasks and functions

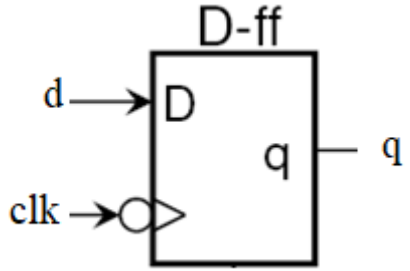
```
task decrypt_task ;
endtask
```

Module ENDING

```
endmodule
```

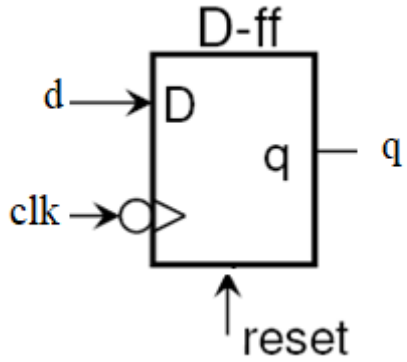
D Flip-Flop

```
module D_ff(input clk, input d, output reg q);
        always @ (negedge clk)
                q <= d; // pronounced “q gets d”
endmodule
```



D Flip-Flop with Asynchronous Reset

```
module D_ff (input clk, input reset, input d, output reg q);
    always @ (negedge clk, posedge reset)
    begin
        if (reset == 1) q <= 0; // when reset
        else q <= d; // when clk
    end
endmodule
```



D Flip-Flop with Synchronous Reset

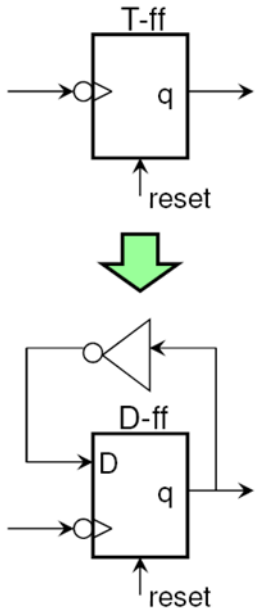
```
module D_ff (input clk, input reset, input d, output reg q);


always @ (posedge clk)
    begin
        if (reset == 1) q <= 0; // when reset
        else q <= d; // when clk
    end
endmodule
```

D Flip-Flop with Enable and Reset

```
module D_ff (input clk, input reset, input d, input en, output reg q);
    always @  (negedge clk, posedge reset)
    begin
        if (reset == 1) q <= 0; // when reset
        else  if (en) q <= d; // when enble and clk
    end
endmodule
```

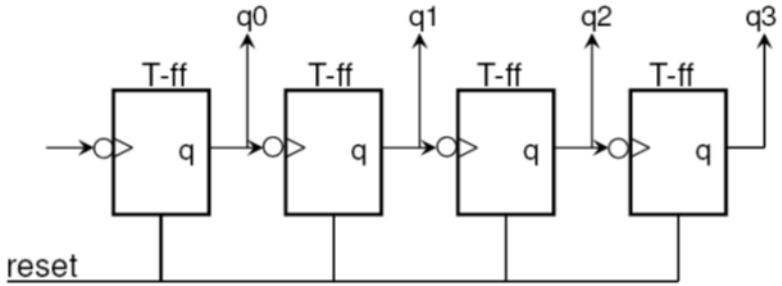
T Flip-Flop

```
module T_ff (input clk, input reset, output q);
wire d;
        D_ff dff1 (clk, reset, d, q);
        not n1(d,q);
endmodule
```



Structural Model : 4-bit Counter

```
module ripple_carry_counter (q, clk, reset);
output [3:0] q;
input clk, reset;
        T_ff tff0(q[0], clk, reset);
        T_ff tff1(q[1], q[0], reset);
        T_ff tff2(q[2], q[1], reset);
        T_ff tff3(q[3], q[2], reset);
endmodule
```

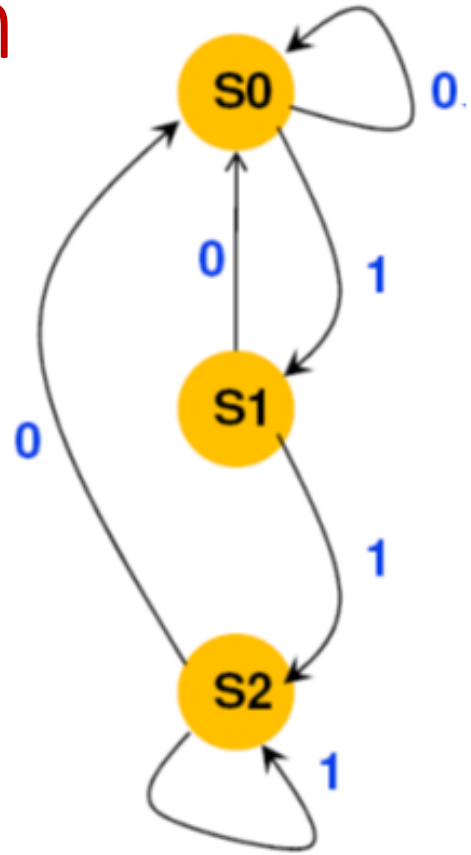


Finite State Machine

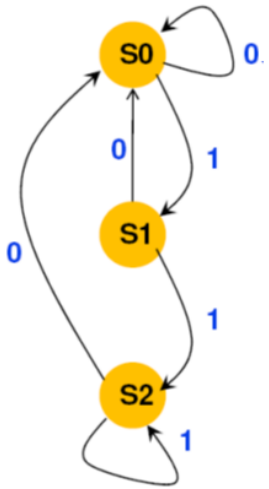
- Each FSM consists of three separate parts:
  - next state logic
  - state register
  - output logic

FSM Verilog Implementation

```
module FSM (q, i, clk, rst);
    input clk,i, rst;
    output q;
    reg q;
    reg [1:0] state, nextstate;

    //The parameter descriptions are optional, it makes reading easier
    parameter S0 = 2'b00, S1 = 2'b01,S2 = 2'b10;
```

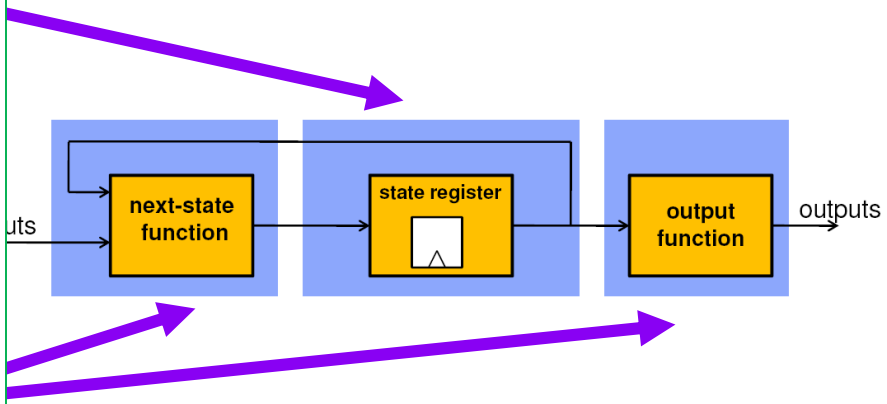


FSM Verilog Implementation: 3-Always Block



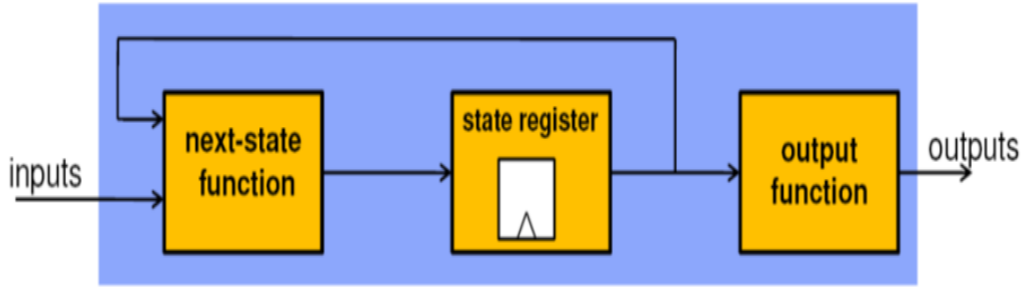

```
module fsm (q, i, clk, rst_n);
    input i, clk, rst_n;
    output q;
    reg    q;
    reg [1:0] state, next_state;
    parameter s0=2'b00, s1=2'b01, s2=2'b10;

    always @(posedge clk or negedge rst_n)
      if(rst_n==0)
        state <= s0;
      else
        state <= next_state;
    always @(*) begin
       next_state=s0
         case(state)
           s0:  begin
                q=0;
                if ( i==1'b1)
                  next_state= s1;
                else
                  next_state=s0;
           end
           s1:  begin
                q=0;
                if ( i==1'b1)
                  next_state= s2;
                else
                  next_state=s0;
           end
           s2:  begin
                q=1;
                if ( i==1'b1)
                  next_state= s2;
                else
                  next_state=s0;
           end
         end
endmodule // fsm
```

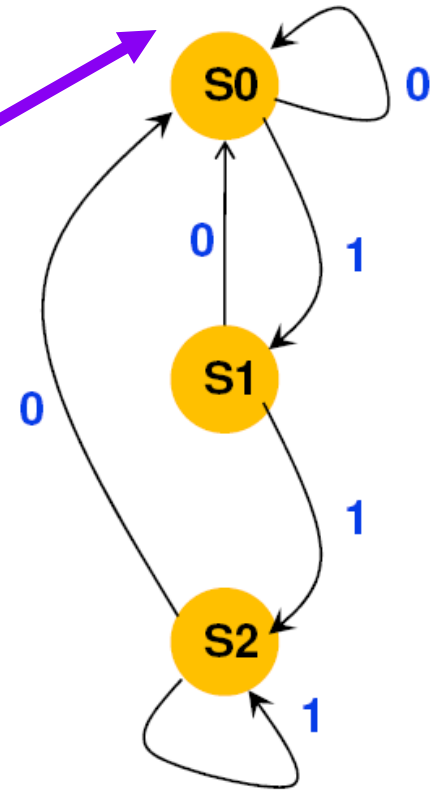
This is the recommended implementation of FSM.
Last always (output) can be coded with assign statement



FSM Verilog Implementation: Single Always Block

```
module fsm(q, i, clk, rst);
  input i, clk, rst;
  output q;
  reg q;
  reg [1:0] state;
parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10;

always @(posedge clk or posedge rst)
  if (rst) begin
    state <= s0; q <= 1'b0;
  end else begin
    case (state)
      s0: if (i == 1'b1) begin
            state <= s1;
            q     <= 1'b0;
          end else begin
            state <= s0;
            q     <= 1'b0;
          end
      s1: ..
      s2: ..
    endcase
  end
```





output

S0 -> 0
S1 -> 0
S2 -> 1