پیاده‌سازی مدارهای ترتیبی در Verilog – مقدمه

برای RS-Latch بیان در سطح گیت

```
module SR_latch_gate (input R, input S, output Q, output Qbar);
	nor (Q, R, Qbar);
	nor (Qbar, S, Q);
endmodule
```

برای RS-Latch بیان در سطح جریان داده

```
module SR_latch_dataflow (input R, input S, output Q, output Qbar);
	assign #2 Q_i = Q;
	assign #2 Qbar_i = Qbar;
	assign #2 Q = ~ (R | Qbar);
	assign #2 Qbar = ~ (S | Q);
endmodule
```

برای D-Latch بیان در سطح رفتاری

```
module D_latch_behavior (input D, input Enable, output Q, output Qbar);
	always @ (D or Enable)
	if(Enable)
	begin
		Q <= D;
		Qbar <= ~D;
	end
endmodule
```

برای D-FlipFlop بیان در سطح رفتاری. توجه شود برای حساس کردن رفتار به لبهٔ سیگنال از posedge (لبهٔ بالارونده) یا negedge (لبهٔ پایین‌رونده) استفاده می‌شود.

```
module D_ff_behavior (input D, input Clk, output reg Q);
	always @ (posedge Clk)
	if(Clk)
	begin
		Q <= D;
	end
endmodule
```

برای Master-Slave D-FlipFlop ، بیان در سطح گیت

```
module jk_flip_flop_master_slave(Q, Qn, C, J, K, RESETn);
  output Q;
  output Qn;
  input  C;
  input  J;
  input  K;
  input  RESETn; // Active low reset signal.

  wire   MQ;  // The master's Q output.
  wire   MQn; // The master's Qn output.
  wire   Cn;  // The clock input to the slave shall be the complement of the master's.
  wire   J1;
  wire   K1;
  wire   J2;  // The actual input to the first SR latch (S).
  wire   K2;  // The actual input to the first SR latch (R).
```

```
  assign J2 = !RESETn ? 0 : J1;  // Upon reset force J2 = 0
  assign K2 = !RESETn ? 1 : K1;  // Upon reset force K2 = 1

  and(J1, J, Qn);
  and(K1, K, Q);
  not(Cn, C);
  sr_latch_gated master(MQ, MQn, C, J2, K2);
  sr_latch_gated slave(Q, Qn, Cn, MQ, MQn);
endmodule // jk_flip_flop_master_slave

module sr_latch_gated(Q, Qn, G, S, R);
  output Q;
  output Qn;
  input  G;
  input  S;
  input  R;

  wire   S1;
  wire   R1;

  and(S1, G, S);
  and(R1, G, R);
  nor(Qn, S1, Q);
  nor(Q, R1, Qn);
endmodule // sr_latch_gated
```

برای D-FlipFlop با ری‌ست آسنکرون، بیان در سطح رفتاری

```
module dff_async_reset_b)
        data,   // Data Input
        clk,    // Clock Input
        reset,  // Reset input
        q                   //Q output
        ;(
        input data, clk, reset ;
        output q;
        reg q;

        always @ ( posedge clk or negedge reset)
        if (~reset) begin
                q <= 1'b0;
        end
        else begin
                q <= data;
        end

endmodule
```

برای D-FlipFlop با ری‌ست سنکرون، بیان در سطح رفتاری

```
module D_ff_with_synch_reset_behavior(input D, input Clk, input reset, output reg Q);
        always @(posedge Clk)
                if (reset)
```

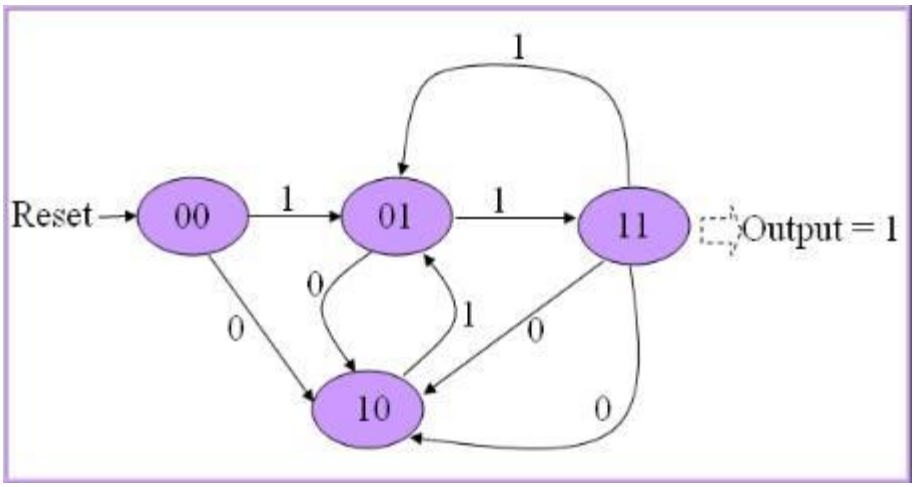
```
        begin
            Q <= 1'b0;
        end
        else
        begin
            Q <= D;
        end
endmodule
```

برای D-FlipFlop با ری‌ست آسنکرون، بیان در سطح رفتاری

```
module D_ff_with_asynch_reset_behavior(input D, input Clk, input clear, output reg Q);
    always @(posedge Clk or posedge clear)
        if (clear)
            Q <= 1'b0;
        else
            Q <= D;
endmodule
```

برای T-FlipFlop با ری‌ست آسنکرون، بیان در سطح رفتاری

```
module T_ff_enable_behavior(input Clk, input reset_n, input T, output reg Q);
    always @(negedge Clk)
        if (!reset_n)
            Q <= 1'b0;
        else if (T)
            Q <= ~Q;
endmodule
```



برای ماشین حالت، با توجه به نمودار حالت و تقسیم‌بندی میلی/مور، کد می‌تواند مطابق زیر نوشته شود. همانطور که مشاهده می‌شود تخصیص حالت نیز صورت گرفته است.

مثلاً برای نمودار حالت روبرو کد زیر قابل استفاده است. رجیستر ۲بیتی state مقادیر در نظر گرفته شده برای فلیپ‌فلاپها نشان می‌دهد.

```
module fsm( clk, rst, inp, outp);

  input clk, rst, inp;
  output outp;

  reg [1:0] state;
  reg outp;

  always @( posedge clk, posedge rst )
  begin
  if( rst )
     state <= 2'b00;
  else
```

```
  begin
     case( state )
     2'b00:
     begin
         if( inp ) state <= 2'b01;
         else state <= 2'b10;
     end

     2'b01:
     begin
         if( inp ) state <= 2'b11;
         else state <= 2'b10;
     end

     2'b10:
     begin
         if( inp ) state <= 2'b01;
         else state <= 2'b11;
     end

     2'b11:
     begin
         if( inp ) state <= 2'b01;
         else state <= 2'b10;
     end
     endcase
  end
end


always @(posedge clk, posedge rst)
begin
   if( rst )
      outp <= 0;
   else if( state == 2'b11 )
      outp <= 1;
   else outp <= 0;

end

endmodule
```

برای مقادیر حالت می‌توان از امکان تعریف پارامتر استفاده کرد. مثلاً در متن زیر مقادیر ۴بیتی برای عناوین IDLE، S1، S2 و S3 بترتیب مقادیر ۱، ۲، ۳ و ۴ تعریف شده‌اند.

```
parameter [4:1] // ERROR is 4'b0000
 IDLE = 4'd1,
 S1 = 4'd2,
 S2 = 4'd3,
 S3 = 4'd4;
```

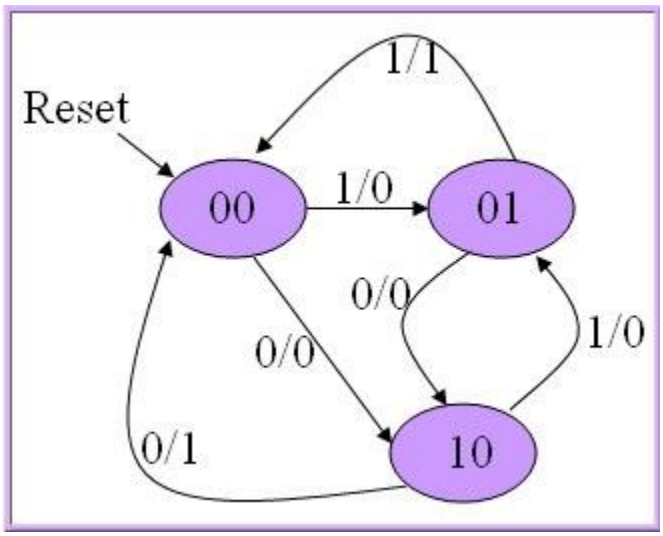
در اینصورت مثلاً اگر مطابق کد قبلی state <= 4'b0010 (یا همان state <= 2'd2) داشته باشیم، می‌توانیم بجای آن state <= S1 بنویسیم.

همچنین دو عبارت state <= 4'b0010 و state = 4'b0010 (که درون بلوک always@ قرار می‌گیرند) در حالت کلی با هم برابرند. اما در یک وضعیت این دو عبارت رفتار متفاوت خواهند داشت. در دو متن زیر که شباهت قابل توجه با هم دارند، در متن سمت راست (استفاده از عملگر <= : non-blocking assignment) یک شیفت‌رجیستر با ۴ فلیپ‌فلاپ (مقادیر a، b، c و d)و در متن سمت چپ (استفاده از عملگر = تنها : blocking assignment) یک شیفت‌رچیستر با تنها ۲ فلیپ‌فلاپ (a و b) سنتز می‌شوند (عملاً مقادیر b و c و d معادل و در واقع یک مقدار هستند).

```
always @(posedge clk)
begin
  b=a;
  c=b;
  d=c;
end
```

```
always @(posedge clk)
 begin
   b<=a;
   c<=b;
   d<=c;
end
```

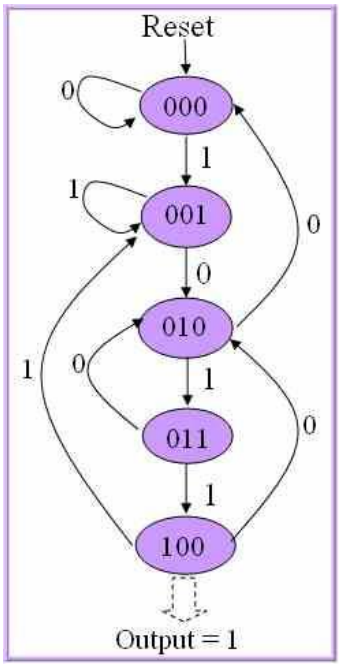
برای ماشین حالت میلی با نمودار حالت روبرو کد می‌تواند مطابق زیر نوشته شود. همانطور که مشاهده می‌شود تخصیص حالت نیز صورت گرفته است. همچنین چانچه در بخش قبل برای ماشین میلی ملاحظه شد، تخصیص مقدار خروجی بر اساس حالت و در بخشی جداگانه از روند جریان حالت صورت گرفت. اما اینجا همزمان با تعیین روند جریان حالت، تعیین خروجی صورت می‌پذیرد. یادآوری می‌شود که در ماشین میلی خروجی در مسیر تعیین جریان حالت رخ می‌دهد.

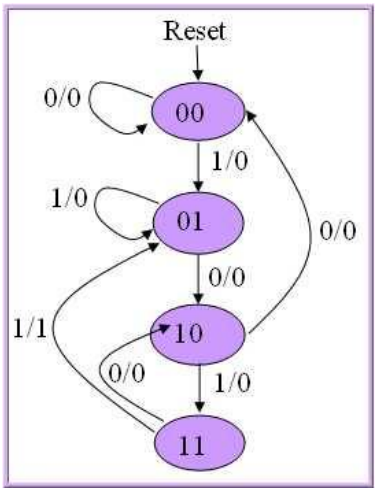



```
module mealy( clk, rst, inp, outp);

  input clk, rst, inp;
  output outp;

  reg [1:0] state;
  reg outp;

  always @( posedge clk, posedge rst ) begin
  if( rst ) begin
     state <= 2'b00;
     outp <= 0;
  end
  else  begin
     case( state )
     2'b00: begin
          if( inp ) begin
            state <= 2'b01;
            outp  <= 0;
          end
          else begin
             state <= 2'b10;
             outp <= 0;
          end
     end
```

```
      2'b01: begin
          if( inp ) begin
              state <= 2'b00;
              outp  <= 1;
          end
          else  begin
             state <= 2'b10;
             outp <= 0;
          end

      end

      2'b10: begin
          if( inp ) begin
              state <= 2'b01;
              outp  <= 0;
          end
          else begin
             state <= 2'b00;
             outp <= 1;
          end

      end

      default: begin
          state <= 2'b00;
          outp <= 0;
      end
    endcase
  end
end

endmodule
```

برای یک ماشین آشکارساز مقادیر ورودی (sequence detector) که نمودار حالت آن مطابق شکل آورده شده کد می‌تواند مطابق زیر باشد:






```
module m1011( clk, rst, inp, outp);

  input clk, rst, inp;
```

```
  output outp;

  reg [1:0] state;
  reg outp;

  always @( posedge clk, rst )
  begin
  if( rst )
     state <= 2'b00;
  else
  begin
     case( {state,inp} )
       3'b000: begin
          state <= 2'b00;
          outp  <= 0;
       end
       3'b001: begin
          state <= 2'b01;
          outp  <= 0;
       end
       3'b010: begin
          state <= 2'b10;
          outp  <= 0;
       end
       3'b011: begin
          state <= 2'b01;
          outp  <= 0;
       end
       3'b100: begin
          state <= 2'b00;
          outp  <= 0;
       end
       3'b101: begin
          state <= 2'b11;
          outp  <= 0;
       end
       3'b110: begin
          state <= 2'b10;
          outp  <= 0;
       end
       3'b111: begin
          state <= 2'b01;
          outp  <= 1;
       end

     endcase
  end
end

endmodule
```