Verilog codes

OR Gate

```
module orgate(a,b,y);
input a,b;
output y;
or g(y,a,b);
endmodule
```

AND Gate

```
module andgate(a,b,y);
input a,b;
output y;
and g(y,a,b);
endmodule
```

Full Adder

```
module fulladder(a, b, c_in, sum, c_out);
input a, b, c_in;
output sum, c_out;
assign sum = (a^b^c);
assign c_out = (a & b)||(b & c)||(a & c);
endmodule


// METHOD 2
module fulladder(a, b, c_in, sum, c_out);
input a, b, c_in;
output sum, c_out;
reg sum, c_out;
always @(a or b or c)
  sum = a^b^c;
  c_out = (a & b)||(b & c)||(a & c);
endmodule
```

Decoder (3 to 8)

```
module decoder3_to_8( in,out, en);
input [2:0]  in;
input en;
```

```
output [7:0] out;
reg [7:0] out;
always @( in or en)
  begin
      if (en)
        begin
          out=8'd0;
          case (in)
              3'b000: out[0]=1'b1;
              3'b001: out[1]=1'b1;
              3'b010: out[2]=1'b1;
              3'b011: out[3]=1'b1;
              3'b100: out[4]=1'b1;
              3'b101: out[5]=1'b1;
              3'b110: out[6]=1'b1;
              3'b111: out[7]=1'b1;
              default: out=8'd0;
          endcase
        end
      else
        out=8'd0;
 end
endmodule
```

Decoder (2 to 4)

```
module dec2_4 (E, A, B, y0, y1, y2, y3)
input A, B, E;
output y0,y1,y2,y3;
assign y0= (~A) & (~B) & E;
assign y1= (~A) & B & E;
assign y2= A & (~ B) & E;
assign y3= A & B & E;
end module
```

Priority Encoder

```
module priority_enoder(en,i,y);
    input en;
    input [7:0]i;
    output [2:0]y;
    assign y[2]=i[4] | i[5] | i[6] | i[7] &en;
    assign y[1]=i[2] | i[3] | i[6] | i[7] &en;
    assign y[0]=i[1] | i[3] | i[5] | i[7] &en;
endmodule
```

2:1 Multiplexer

```
module mux2to1(S, D0, D1, Y);
input S, D0, D1;
output Y;
assign Y = (~S & D0) | (S & D1);
endmodule



// METHOD 2
module mux2to1(S, D0, D1, Y);
input S, D0, D1;
output Y;
reg Y;
always @(S or D0 or D1)
  if(S==1)
    Y = D1;
  else
    Y = D0;
endmodule



// METHOD 3
module mux2to1(S, D0, D1, Y);
input S, D0, D1;
output Y;
reg Y;
always @(S or D0 or D1)
  case (S)
    0: Y=D0;
    1: Y=D1;
  endcase
endmodule
```

4:1 Mulltiplexer

```
module mux4to1(S1,S0,D0,Dl,D2,D3,Y)
input S1,S0,D0,Dl,D2,D3;
output Y;
reg Y;
always @ (S1 or S0 or D0 or D1 or D2 or D3)
  begin
    case ({S1,S0})
        0 : Y=D0;
        1 : Y=D1;
        2 : Y=D2;
        3 : Y=D3;
     endcase
  end
endmodule
```

8:1 Multiplexer

```
module m81(Y, D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2);
input  D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2;
output reg out;
always@(D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2)
  begin
    case(S0 & S1 & S2)
      3'b000: Y=D0;
      3'b001: Y=D1;
      3'b010: Y=D2;
      3'b011: Y=D3;
      3'b100: Y=D4;
      3'b101: Y=D5;
      3'b110: Y=D6;
      3'b111: Y=D7;
      default: Y=1'b0;
    endcase
  end
endmodule
```

1:4 Demultiplexer

```
module demuxlto4(S,D,Y);
input   [1:0] S;
input   D;
output [3 : 0] Y;
reg     [3:0] Y;
always @ (S or D)
 case ( {D, S})
     3'b100 : Y= 4'b0001;
     3'b101 : Y= 4'b0010;
     3'b110 : Y= 4'b0100;
     3'b111 : Y= 4'b1000;
     default : Y= 4'b0000;
  endcase
```

Latch

```
module latch(enable, data, q);
   input  enable, data;
   output q;
   reg    q;
   always @ (enable or data)
      if (enable)
         q <= data;
endmodule
```

D - FLIP FLOP

```
module FlipFlop(D,clk,Q);
input D;
input clk;
output Q;
reg Q;
always @(posedge clk)   // posedge or negedge
  begin
   Q <= D;
  end
endmodul
```

JK - FLIP FLOP

```
module jkflp(J, K, clk, Q);
input J, K;
input clk;
output Q;
reg Q;
always @(posedge clk)
  case({J,K})
    2'b00 : Q <= Q;
    2'b01 : Q <= 0;
    2'b10 : Q <= 1;
    2'b11 : Q <= ~Q;
  endcase
endmodule
```

SR - FLIP FLOP

```
module srff(S,R,clk, Q, Qbar);

input S,R,clk;
output reg Q, Qbar;

always@(posedge clk)
  begin
    if(S == 1)
      begin
        Q <= 1;
        Qbar <= 0;
      end
    else if(R == 1)
      begin
        Q <= 0;
        Qbar <=1;
      end
```

```
    else if(S == 0 & R == 0)
      begin
        Q <= Q;
        Qbar <= Qbar
      end
    end
endmodule
```

T - FLIP FLOP

```
module tff(T, clk, Q, Qbar);
input T, clk;
output Q, Qbar;
reg Q, Qbar;
always @(T or posedge clk)
  begin
    if(T==0)
      begin
        Q = 1'b1;
        Qbar = 1'b0;
      end
    else
      begin
        Q = 1'b0;
        Qbar = 1'b1;
      end
  end
endmodule
```

PIPO Register

```
module pipo(Pi, clk, Po);
input [3:0] Pi, clk;
output [3:0] Po;
reg [3:0] Po;
always @(posedge clk)
  begin
    Po = Pi;   // input is the same as output
  end
endmodule
```

SISO Register

```
module sisomod(Si, clk, So, clear);
input Si, clk, clear;
output So;
output [3:0] temp;
reg So;
```

```
reg [3:0] temp;
always @(posedge clk )
  begin
    if (~clear)
      temp = 4’b0000;
    else
      temp = temp << 1;
      So = tmp[3];
  end
endmodule
```