Verilog

```
module mytt(A, B, Y);
input [5:0]  A, B;
output[11:0] Y;
reg   [11:0] Y;
parameter C=3'b100;

always@(A or B)
begin
  Y[2:0]=A[2:0];
  Y[3]  = A[3] & B[3];
  Y[5:4]= {A[5]&B[5], A[4]|B[4]};
  Y[8:6]= B[2:0];
  Y[11:9]=C;
end

endmodule
```

--- Concatenation example:

```
module mytt2(A, B, Y);
input [2:0]  A, B;
output[14:0] Y;
parameter C=3'b100;

assign Y = { A, B, {2{C}}, 3'b110};
endmodule
```

Flip-flop with Positive-Edge Clock

```
module flop (CLK, D, Q);
  input CLK, D;
  output Q;
  reg Q;
  always @(posedge CLK)
    begin
      Q <= D;
    end
endmodule
```

Flip-flop with Negative-Edge Clock and Asynchronous Clear

```
module flop (C, D, CLR, Q);
  input C, D, CLR;
  output Q;
  reg Q;
  always@(negedge C or posedge CLR)
    begin
      if (CLR)
          Q <= 1'b0;
      else
          Q <= D;
    end
endmodule
```

Flip-flop with Positive-Edge Clock and Synchronous Set

```
module flop (C, D, S, Q);
  input C, D, S;
  output Q;
  reg Q;
  always @(posedge C)
    begin
      if (S)
          Q <= 1'b1;
      else
          Q <= D;
    end
endmodule
```

4-bit Register with Positive-Edge Clock, Asynchronous Set and Clock Enable

```
module flop (C, D, CE, PRE, Q);
  input C, CE, PRE;
  input [3:0] D;
  output [3:0] Q;
  reg [3:0] Q;
  always @(posedge C or posedge PRE)
    begin
      if (PRE)
          Q <= 4'b1111;
      else if (CE)
          Q <= D;
    end
endmodule
```

Latches

```
module latch (G, D, Q);
  input G, D;
  output Q;
  reg Q;
  always @(G or D)
    begin
      if (G)
          Q <= D;
  end
endmodule
```

Tristates

```
module three_st (T, I, O);
  input T, I;
  output O;
  reg O;
  always @(T or I)
    begin
      if (~T)
          O = I;
      else
          O = 1'bZ;
    end
endmodule
```

```
module three_st (T, I, O);
  input T, I;
  output O;
  assign O = (~T) ? I: 1'bZ;
endmodule
```

Counters .4-bit up/down counter with an asynchronous clear

```
module counter (C, CLR, UP_DOWN, Q);
  input C, CLR, UP_DOWN;
  output [3:0] Q;
  reg [3:0] tmp;
  always @(posedge C or posedge CLR)
    begin
      if (CLR)
          tmp <= 4'b0000;
      else
          if (UP_DOWN)
              tmp <= tmp + 1'b1;
          else
              tmp <= tmp - 1'b1;
    end
  assign Q = tmp;
endmodule
```

8-bit Shift-Left Register with Positive-Edge Clock, Serial In, and Serial Out

```
module shift (C, SI, SO);
  input C,SI;
  output SO;
  reg [7:0] tmp;
  always @(posedge C)
    begin
      tmp <= tmp << 1;
      tmp[0] <= SI;
    end
    assign SO = tmp[7];
endmodule
```

8-bit Shift-Left Register with Positive-Edge Clock, Asynchronous Clear, Serial In, and Serial Out

```
module shift (C, CLR, SI, SO);
  input C, SI, CLR;
  output SO;
  reg [7:0] tmp;
  always @(posedge C or posedge CLR)
  begin
    if (CLR)
        tmp <= 8'b00000000;
    else
        tmp <= {tmp[6:0], SI};
  end
  assign SO = tmp[7];
endmodule
```

8-bit Shift-Left Register with Positive-Edge Clock, Asynchronous Parallel Load, Serial In, and Serial Out

```
module shift (C, ALOAD, SI, D, SO);
  input C, SI, ALOAD;
  input [7:0] D;
  output SO;
  reg [7:0] tmp;
  always @(posedge C or posedge ALOAD)
  begin
    if (ALOAD)
        tmp <= D;
    else
        tmp <= {tmp[6:0], SI};
  end
  assign SO = tmp[7];
endmodule
```

8-bit Shift-Left Register with Negative-Edge Clock, Clock Enable, Serial In, and Serial Out

```
module shift (C, CE, SI, SO);
  input C, SI, CE;
  output SO;
  reg [7:0] tmp;
  always @(negedge C)
    begin
      if (CE)
        begin
          tmp <= tmp << 1;
          tmp[0] <= SI;
        end
    end
    assign SO = tmp[7];
endmodule
```

4-to-1 MUX using IF Statement

```
module mux (a, b, c, d, s, o);
  input a,b,c,d;
  input [1:0] s;
  output o;
  reg o;
  always @(a or b or c or d or s)
  begin
    if (s == 2'b00)
        o = a;
    else if (s == 2'b01)
        o = b;
    else if (s == 2'b10)
        o = c;
    else
        o = d;
  end
endmodule
```

4-to-1 MUX Using CASE Statement

```
module mux (a, b, c, d, s, o);
  input a, b, c, d;
  input [1:0] s;
  output o;
  reg o;
  always @(a or b or c or d or s)
  begin
    case (s)
      2'b00   : o = a;
      2'b01   : o = b;
      2'b10   : o = c;
      default : o = d;
```

```
    endcase
  end
endmodule
```

Decoders

```
module dec (sel, res);
  input [2:0] sel;
  output [7:0] res;
  reg [7:0] res;
  always @(sel or res)
  begin
    case (sel)
      3'b000 : res = 8'b00000001;
      3'b001 : res = 8'b00000010;
      3'b010 : res = 8'b00000100;
      3'b011 : res = 8'b00001000;
      3'b100 : res = 8'b00010000;
      3'b101 : res = 8'b00100000;
      3'b110 : res = 8'b01000000;
      default : res = 8'b10000000;
    endcase
  end
endmodule
```

Priority Encoder

```
module priority (sel, code);
  input [7:0] sel;
  output [2:0] code;
  reg [2:0] code;
  always @(sel)
  begin
    if (sel[0]) code = 3'b000;
    else if (sel[1]) code = 3'b001;
    else if (sel[2]) code = 3'b010;
    else if (sel[3]) code = 3'b011;
    else if (sel[4]) code = 3'b100;
    else if (sel[5]) code = 3'b101;
    else if (sel[6]) code = 3'b110;
    else if (sel[7]) code = 3'b111;
    else             code = 3'bxxx;
  end
endmodule
```

Unsigned 8-bit Adder with Carry Out

```
module adder(A, B, SUM, CO);
  input [7:0] A;
  input [7:0] B;
  output [7:0] SUM;
  output CO;
  wire [8:0] tmp;
  assign tmp = A + B;
  assign SUM = tmp [7:0];
  assign CO = tmp [8];
endmodule
```

Unsigned 8-bit Adder/Subtractor

```
module addsub(A, B, OPER, RES);
  input  OPER;
  input  [7:0] A;
  input  [7:0] B;
  output [7:0] RES;
  reg    [7:0] RES;
  always @(A or B or OPER)
  begin
    if (OPER==1'b0)
        RES = A + B;
    else
        RES = A - B;
  end
endmodule
```

Comparators

Verilog: (==, !=,<, <=, >, >=)

```
module compar(A, B, CMP);
  input  [7:0] A;
  input  [7:0] B;
  output CMP;
  assign CMP = (A >= B) ? 1'b1 : 1'b0;
endmodule
```

Unsigned 8x4-bit Multiplier

```
module compar(A, B, RES);
  input  [7:0] A;
  input  [3:0] B;
  output [11:0] RES;
  assign RES = A * B;
endmodule
```

Finite State Machine

```
module fsm (clk, reset, x1, outp);
  input clk, reset, x1;
  output outp;
  reg outp;
  reg [1:0] state;
  reg [1:0] next_state;
  parameter s1 = 2'b00,
            s2 = 2'b01,
            s3 = 2'b10,
            s4 = 2'b11;
  always @(posedge clk or posedge reset)
  begin
    if (reset)
        state <= s1;
    else
```

```
        state <= next_state;
  end
  always @(state or x1)
  begin
      case (state)
        s1:
            if (x1 == 1'b1)
                next_state = s2;
            else
                next_state = s3;
        s2: next_state = s4;
        s3: next_state = s4;
        s4: next_state = s1;
      endcase
  end
  always @(state)
  begin
    case (state)
      s1: outp = 1'b1;
      s2: outp = 1'b1;
      s3: outp = 1'b0;
      s4: outp = 1'b0;
    endcase
  end
endmodule
```

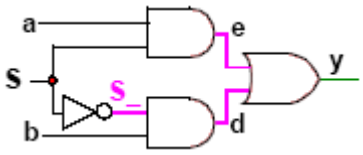
Hierarchical Design

```
module my_block (in1, in2, dout);
  input in1, in2;
  output dout;
  . . .
  . . .

endmodule

module top (DI_1, DI_2, DI_3,DI_4,
   DOUT1, DOUT2);
  input DI_1, DI_2, DI_3, DI_4;
  output DOUT1, DOUT2;
  my_block inst1 (
        .in1(DI_1),
        .in2(DI_2),
        .dout(DOUT)
        );

  my_block inst2(DI_3,DI_4, DOUT);
  . . .
  . . .

endmodule
```

More Verilog Examples:

1.

```
module mux_2to1(a,b,d,e, y, S_, S);
input    a,b,S,d,e,S_;
output  y;
      not       (S_,S);
      and       (e,a,S);
      and       (d,b,S_);
      or        (y,e,d);
endmodule
```




```
module mux_2to1(Y, A, B, sel);
output Y;
input  A, B, sel;

assigny=(sel)?A:B;

endmodule
```

2.

```
module mux_2to1(Y, a,b,S);
input a,b,S;
output y;
reg y;
always@(a or b)
begin:
  case(S)
    0:y=b;
    1:y=a;
   endcase
end
endmodule
```

3.

```
module mux_2to1(Y, A, B, sel);
output [15:0] Y;
input [15:0] A, B;
input sel;
reg [15:0] Y;
always @(A or B or sel)
begin
```

```
    if (sel == 1'b0)
            Y = A;
    else
            Y = B;
end
endmodule
```

4.

```
module mux_4to1(Y, A, B, C, D, sel);
output [15:0] Y;
input [15:0] A, B, C, D;
input [1:0] sel;
reg [15:0] Y;
always @(A or B or C or D or sel)
begin
    case ( sel )
        2'b00: Y = A;
        2'b01: Y = B;
        2'b10: Y = C;
        2'b11: Y = D;
        default: Y = 16'hxxxx;
    endcase
end
endmodule
```

5.

```
module mux_4to1(y,a,b,c,d,S1,S0);
input a,b,c,d;
input S1,S0;
output y;
assign y=(~S1&~S0&a)|(~S1&S0&b)|(S1&~S0&c)|(S1&S0&d);
endmodule
```

6.

```
module mux4to1 (w0, w1, w2, w3, S, f);
input w0, w1, w2, w3;
input [1:0] S;
output f;
assign f = S[1] ? (S[0] ? w3 : w2) : (S[0] ? w1 : w0);
endmodule
```

7. 2-to-4 decoder:

```
module dec2to4(W, Y, En);
input       [1:0] W;
input       En;
```

```
output      [0:3] Y;
reg         [0:3] Y;

always @(en or W)
begin
case ({en,W})
     3'b100: Y = 4'b1000;
     3'b101: Y = 4'b0100;
     3'b110: Y = 4'b0010;
     3'b111: Y = 4'b0001;
default: Y = 4'b0000;
endcase
end
endmodule
```

8. Module of a 3 to 8 Decoder with an active high enable input and active low outputs.

```
module decoder_3to8(Y, A, B, C, en);
output [7:0] Y;
input A, B, C;
input en;
assign Y = ( {en,A,B,C} == 4'b1000) ? 8'b0000_0001 :
           ( {en,A,B,C} == 4'b1001) ? 8'b0000_0010 :
           ( {en,A,B,C} == 4'b1010) ? 8'b0000_0100 :
           ( {en,A,B,C} == 4'b1011) ? 8'b0000_1000 :
           ( {en,A,B,C} == 4'b1100) ? 8'b0001_0000 :
           ( {en,A,B,C} == 4'b1101) ? 8'b0010_0000 :
           ( {en,A,B,C} == 4'b1110) ? 8'b0100_0000 :
           ( {en,A,B,C} == 4'b1111) ? 8'b1000_0000 :
                                      8'b0000_0000;
endmodule
```

9.

```
module demux_3to8(a,b,c,,d,e,f,g,h,S2,S1,S0,Din);
input  S2,S1,S0,Din;
output a,b,c,d,e,f,g,h;
reg a,b,c,d,e,f,g,h;
always@(S2 or S1 or S0 or Din)
begin
     {a,b,c,d,e,f,g,h}=8'd0;
     case({S2,S1,S0})
          3'd0:       a= Din;
          3'd1:       b= Din;
          3'd2:       c= Din;
          3'd3:       d= Din;
          3'd4:       e= Din;
          3'd5:       f= Din;
          3'd6:       g= Din;
          3'd7:       h= Din;
```

```
            default:{a,b,c,d,e,f,g,h}=8'd0;
        endcase
    end
    endmodule
```