The following are some of useful verilog examples.

Verilog code for an 8-bit shift-left register with a positive-edge clock, serial in and serial out.

Verilog code for an 8-bit shift-left register with a negative-edge clock, a clock enable, a serial in and a serial out.

Verilog code for an 8-bit shift-left register with a positive-edge clock, asynchronous clear, serial in and serial out.

Verilog code for an 8-bit shift-left register with a positive-edge clock, a synchronous set, a serial in and a serial out.

Verilog code for an 8-bit shift-left register with a positive-edge clock, a serial in and a parallel out

8-bit shift-left register with a positive-edge clock,an asynchronous parallel load, a serial in and a serial out

Verilog code for an 8-bit shift-left register with a positive clock,a synchronous parallel load,a serial in and a serial out

Verilog code for an 8-bit shift-left/shift-right register with a positive-edge clock, a serial in and a serial out

Verilog code for a 4-to-1 1-bit MUX using an If statement.

Verilog Code for a 4-to-1 1-bit MUX using a Case statement.

Verilog code for a 3-to-1 1-bit MUX with a 1-bit latch.

Verilog code for a 1-of-8 decoder

Verilog code leads to the inference of a 1-of-8 decoder

Verilog code for a 3-bit 1-of-9 Priority Encoder

Verilog code for a logical shifter

Verilog code for an unsigned 8-bit adder with carry in

Verilog code for an unsigned 8-bit adder with carry out

Verilog code for an unsigned 8-bit adder with carry in and carry out

Verilog code for an unsigned 8-bit adder/subtractor

Verilog code for an unsigned 8-bit greater or equal comparator

Following is the Verilog code for flip-flop with a positive-edge clock.

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

always @(posedge clk)
begin
   q <= d;
end
       endmodule
```

Following is Verilog code for a flip-flop with a negative-edge clock and asynchronous clear.

```
module flop (clk, d, clr, q);
input  clk, d, clr;
output q;
reg    q;
always @(negedge clk or posedge clr)
       begin
   if (clr)
      q <= 1'b0;
   else
      q <= d;
end
       endmodule
```

Following is Verilog code for the flip-flop with a positive-edge clock and synchronous set.

```
       module flop (clk, d, s, q);
       input  clk, d, s;
       output q;
       reg    q;
       always @(posedge clk)
       begin
          if (s)
             q <= 1'b1;
```

```
          else
             q <= d;
       end
       endmodule
```

Following is Verilog code for the flip-flop with a positive-edge clock and clock enable.

```
module flop (clk, d, ce, q);
input  clk, d, ce;
output q;
reg    q;
always @(posedge clk)
       begin
   if (ce)
             q <= d;
end
       endmodule
```

Following is Verilog code for a 4-bit register with a positive-edge clock, asynchronous set and clock enable.

```
module flop (clk, d, ce, pre, q);
input        clk, ce, pre;
input  [3:0] d;
output [3:0] q;
reg    [3:0] q;
always @(posedge clk or posedge pre)
       begin
   if (pre)
      q <= 4'b1111;
   else if (ce)
      q <= d;
       end
       endmodule
```

Following is the Verilog code for a latch with a positive gate.

```
module latch (g, d, q);
       input  g, d;
       output q;
reg    q;
always @(g or d)
       begin
```

```
      if (g)
         q <= d;
   end
   endmodule
```

Following is the Verilog code for a latch with a positive gate and an asynchronous clear.

```
   module latch (g, d, clr, q);
   input  g, d, clr;
   output q;
   reg    q;
always @(g or d or clr)
   begin
      if (clr)
         q <= 1'b0;
      else if (g)
         q <= d;
   end
   endmodule
```

Following is Verilog code for a 4-bit latch with an inverted gate and an asynchronous preset.

```
   module latch (g, d, pre, q);
   input        g, pre;
   input  [3:0] d;
   output [3:0] q;
   reg    [3:0] q;
   always @(g or d or pre)
   begin
      if (pre)
         q <= 4'b1111;
      else if (~g)
         q <= d;
   end
   endmodule
```

Following is Verilog code for a tristate element using a combinatorial process and always block.

```
   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
```

Following is the Verilog code for a tristate element using a concurrent assignment.

```
module three_st (t, i, o);
input  t, i;
output o;
   assign o = (~t) ? i: 1'bZ;
      endmodule
```

Following is the Verilog code for a 4-bit unsigned up counter with asynchronous clear.

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

Following is the Verilog code for a 4-bit unsigned down counter with synchronous set.

```
module counter (clk, s, q);
input        clk, s;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk)
begin
```

```
   if (s)
      tmp <= 4'b1111;
   else
      tmp <= tmp - 1'b1;
end
   assign q = tmp;
endmodule
```

Following is the Verilog code for a 4-bit unsigned up counter with an asynchronous load from the primary input.

```
module counter (clk, load, d, q);
input        clk, load;
input  [3:0] d;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge load)
begin
   if (load)
      tmp <= d;
   else
      tmp <= tmp + 1'b1;
end
   assign q = tmp;
endmodule
```

Following is the Verilog code for a 4-bit unsigned up counter with a synchronous load with a constant.

```
module counter (clk, sload, q);
input        clk, sload;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk)
begin
   if (sload)
            tmp <= 4'b1010;
   else
      tmp <= tmp + 1'b1;
end
   assign q = tmp;
      endmodule
```

Following is the Verilog code for a 4-bit unsigned up counter with an asynchronous clear and a clock enable.

```
module counter (clk, clr, ce, q);
input        clk, clr, ce;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge clr)
begin
   if (clr)
      tmp <= 4'b0000;
   else if (ce)
      tmp <= tmp + 1'b1;
end
   assign q = tmp;
      endmodule
```

Following is the Verilog code for a 4-bit unsigned up/down counter with an asynchronous clear.

```
module counter (clk, clr, up_down, q);
input        clk, clr, up_down;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk 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
```

Following is the Verilog code for a 4-bit signed up counter with an asynchronous reset.

```
module counter (clk, clr, q);
input               clk, clr;
output signed [3:0] q;
reg    signed [3:0] tmp;
always @ (posedge clk or posedge clr)
begin
   if (clr)
      tmp <= 4'b0000;
```

```
   else
      tmp <= tmp + 1'b1;
end
   assign q = tmp;
endmodule
```

Following is the Verilog code for a 4-bit signed up counter with an asynchronous reset and a modulo maximum.

```
module counter (clk, clr, q);
parameter MAX_SQRT = 4, MAX = (MAX_SQRT*MAX_SQRT);
input                 clk, clr;
output [MAX_SQRT-1:0] q;
reg    [MAX_SQRT-1:0] cnt;
always @ (posedge clk or posedge clr)
begin
   if (clr)
      cnt <= 0;
   else
      cnt <= (cnt + 1) %MAX;
end
   assign q = cnt;
endmodule
```

Following is the Verilog code for a 4-bit unsigned up accumulator with an asynchronous clear.

```
module accum (clk, clr, d, q);
input        clk, clr;
input  [3:0] d;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge clr)
begin
   if (clr)
      tmp <= 4'b0000;
   else
      tmp <= tmp + d;
end
   assign q = tmp;
endmodule
```

Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, serial in and serial out.

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

Following is the Verilog code for an 8-bit shift-left register with a negative-edge clock, a clock enable, a serial in and a serial out.

```
module shift (clk, ce, si, so);
input        clk, si, ce;
output       so;
reg    [7:0] tmp;
always @(negedge clk)
begin
   if (ce) begin
      tmp    <= tmp << 1;
      tmp[0] <= si;
   end
end
   assign so = tmp[7];
      endmodule
```

Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, asynchronous clear, serial in and serial out.

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

Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, a synchronous set, a serial in and a serial out.

```
module shift (clk, s, si, so);
input        clk, si, s;
output       so;
reg    [7:0] tmp;
always @(posedge clk)
begin
   if (s)
      tmp <= 8'b11111111;
   else
      tmp <= {tmp[6:0], si};
end
   assign so = tmp[7];
      endmodule
```

Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, a serial in and a parallel out.

```
module shift (clk, si, po);
input        clk, si;
output [7:0] po;
reg    [7:0] tmp;
always @(posedge clk)
begin
   tmp <= {tmp[6:0], si};
end
   assign po = tmp;
endmodule
```

Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, an asynchronous parallel load, a serial in and a serial out.

```
module shift (clk, load, si, d, so);
input        clk, si, load;
input  [7:0] d;
output       so;
reg    [7:0] tmp;
always @(posedge clk or posedge load)
begin
   if (load)
            tmp <= d;
   else
      tmp <= {tmp[6:0], si};
```

```
end
   assign so = tmp[7];
      endmodule
```

Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, a synchronous parallel load, a serial in and a serial out.

```
      module shift (clk, sload, si, d, so);
      input        clk, si, sload;
      input  [7:0] d;
      output       so;
      reg    [7:0] tmp;
      always @(posedge clk)
      begin
         if (sload)
            tmp <= d;
         else
            tmp <= {tmp[6:0], si};
      end
         assign so = tmp[7];
      endmodule
```

Following is the Verilog code for an 8-bit shift-left/shift-right register with a positive-edge clock, a serial in and a serial out.

```
module shift (clk, si, left_right, po);
input        clk, si, left_right;
output       po;
reg    [7:0] tmp;
always @(posedge clk)
begin
   if (left_right == 1'b0)
      tmp <= {tmp[6:0], si};
   else
      tmp <= {si, tmp[7:1]};
end
   assign po = tmp;
      endmodule
```

Following is the Verilog code for a 4-to-1 1-bit MUX using an 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
```

Following is the Verilog Code for a 4-to-1 1-bit MUX using a 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
```

Following is the Verilog code for a 3-to-1 1-bit MUX with a 1-bit latch.

```
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;
```

```
        end
        endmodule
```

Following is the Verilog code for a 1-of-8 decoder.

```
        module mux (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
```

Following Verilog code leads to the inference of a 1-of-8 decoder.

```
module mux (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;
      // 110 and 111 selector values are unused
      default : res = 8'bxxxxxxxx;
   endcase
end
       endmodule
```

Following is the Verilog code for a 3-bit 1-of-9 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
```

Following is the Verilog code for a logical shifter.

```
module lshift (di, sel, so);
       input  [7:0] di;
input  [1:0] sel;
output [7:0] so;
reg    [7:0] so;
always @(di or sel)
begin
   case (sel)
      2'b00   : so = di;
      2'b01   : so = di << 1;
      2'b10   : so = di << 2;
      default : so = di << 3;
   endcase
end
       endmodule
```

Following is the Verilog code for an unsigned 8-bit adder with carry in.

```
module adder(a, b, ci, sum);
```

```
input  [7:0] a;
input  [7:0] b;
input        ci;
output [7:0] sum;

   assign sum = a + b + ci;

      endmodule
```

Following is the Verilog code for an 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
```

Following is the Verilog code for an unsigned 8-bit adder with carry in and carry out.

```
      module adder(a, b, ci, sum, co);
      input        ci;
      input  [7:0] a;
      input  [7:0] b;
      output [7:0] sum;
      output       co;
      wire   [8:0] tmp;

         assign tmp = a + b + ci;
         assign sum = tmp [7:0];
         assign co  = tmp [8];

      endmodule
```

Following is the Verilog code for an 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
```

Following is the Verilog code for an unsigned 8-bit greater or equal comparator.

```
module compar(a, b, cmp);
input  [7:0] a;
input  [7:0] b;
output       cmp;

   assign cmp = (a >= b) ?  1'b1 : 1'b0;

      endmodule
```

Following is the Verilog code for an 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
```