

Verilog Problems



ORSU VENKATA KRISHNAIAH

Orsu Venkata Krishnaiyah

1. Write a 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
```

2. Write a 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;
```

Orsu Venkata Krishnaiyah

```
end  
endmodule
```

3. Write a 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
```

4. Write a 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)
```

Orsu Venkata Krishnaiyah

```
begin  
if (ce)  
q <= d;  
end  
endmodule
```

5. Write a 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
```

6. Write a Verilog code for a latch with a

Orsu Venkata Krishnaiyah

positive gate.

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

7. Write a Verilog code for a latch with a positive gate and an asynchronous clear.

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

Orsu Venkata Krishnaiyah

end
endmodule

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

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

9. Write a Verilog code for a tristate element using a combinatorial process and always block.

```
module three_st (t, i, o);
```

Orsu Venkata Krishnaiyah

```
input t, i;  
output o;  
reg o;  
always (@(t or i))  
begin  
if ( t )  
o = i;  
else  
o = 1'bZ;  
end  
endmodule
```

10. Write a Verilog code for a tristate element using a concurrent assignment.

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

11. Write a Verilog code for a 4-bit unsigned up counter with asynchronous clear.

```
module counter (clk, clr, q);
```

Orsu Venkata Krishnaiyah

```
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
```

12. Write a 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
```

Orsu Venkata Krishnaiyah

```
tmp <= tmp - 1'bl;  
end  
assign g = tmp;  
endmodule
```

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

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

Orsu Venkata Krishnaiyah

14. Write a Verilog code for a 4-bit unsigned up counter with a synchronous load with a constant.

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

15. Write a Verilog code for a 4-bit unsigned up counter with an asynchronous clear and a clock enable.

```
module counter (clk, clr, ce, g);
    input clk, clr, ce;
    output [3:0] g;
```

Orsu Venkata Krishnaiyah

```
reg [3:0] tmp;
always @ (posedge clk or posedge clr)
begin
if (clr)
tmp <= 4'60000;
else if (ce)
tmp <= tmp + 1'b1;
end
assign q = tmp;
endmodule
```

16. Write a 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 <= 4'60000;
else if (up_down)
tmp <= tmp + 1'b1;
```

Orsu Venkata Krishnaiyah

```
else  
tmp <= tmp - 1'bl;  
end  
assign q = tmp;  
endmodule
```

17. Write a 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 <= 4'60000;  
else  
tmp <= tmp + 1'bl;  
end  
assign q = tmp;  
endmodule
```

18. Write a Verilog code for a 4-bit signed up

Orsu Venkata Krishnaiyah

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
```

19. Write a 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;
```

Orsu Venkata Krishnaiyah

```
reg [3:0] tmp;  
always @ (posedge clk or posedge clr)  
begin  
if (clr)  
tmp <= 4'60000;  
else  
tmp <= tmp + d;  
end  
assign q = tmp;  
endmodule
```

20. Write a 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];
```

Orsu Venkata Krishnaiyah

endmodule

21. Write a 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
```

22. Write a 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);
```

Orsu Venkata Krishnaiyah

```
input clk, si, clr;
output so;
reg [7:0] tmp;
always @ (posedge clk or posedge clr)
begin
if (clr)
tmp <= 8'600000000;
else
tmp <= {tmp[6:0], si};
end
assign so = tmp[7];
endmodule
```

23. Write a 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 <= 8'61111111;
end
assign so = tmp[7];
endmodule
```

Orsu Venkata Krishnaiyah

```
else  
tmp <= {tmp[6:0], si};  
end  
assign so = tmp[7];  
endmodule
```

24. Write a 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
```

25. Write a Verilog code for an 8-bit shift-left register with a positive-edge clock, an asynchronous parallel load, a serial in and a

Orsu Venkata Krishnaiyah

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
```

26. Write a 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;
```

Orsu Venkata Krishnaiyah

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

27. Write a Verilog code for an 8-bit shift-left/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]};
```

Orsu Venkata Krishnaiyah

```
end  
assign po = tmp;  
endmodule
```

28. Write a 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
```

Orsu Venkata Krishnaiyah

29. Write a 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
```

30. Write a 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;
```

Orsu Venkata Krishnaiyah

```
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
```

31. Write a 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;
```

Orsu Venkata Krishnaiyah

```
3'b011 res = 8'b000001000;  
3'b100 res = 8'b000010000;  
3'b101 res = 8'b001000000;  
3'b110 res = 8'b010000000;  
default res = 8'b100000000;  
endcase  
end  
endmodule
```

32. Write a 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'b000000001;  
3'b001 res = 8'b000000010;  
3'b010 res = 8'b000000100;  
3'b011 res = 8'b000001000;  
3'b100 res = 8'b000010000;  
3'b101 res = 8'b001000000;
```

110 and 111 selector values are unused

Orsu Venkata Krishnaiyah

```
default res = 8'bXXXXXXXXX;  
endcase  
end  
endmodule
```

33. Write a 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])
```

Orsu Venkata Krishnaiyah

```
code = 3'b101;  
else if (sel[6])  
code = 3'b110;  
else if (sel[7])  
code = 3'b111;  
else  
code = 3'bx;xx;  
end  
endmodule
```

34. Write a Verilog code for a logical shifter.

```
module lshift7 (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;
```

Orsu Venkata Krishnaiyah

```
endcase  
end  
endmodule
```

35. Write a 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
```

36. Write a 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;
```

Orsu Venkata Krishnaiyah

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

37. Write a 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
```

Orsu Venkata Krishnaiyah

38. Write a 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
```

39. Write a 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;
```

Orsu Venkata Krishnaiyah

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

40. Write a 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
```

41. Write a Verilog template shows the multiplication operation placed outside the always block and the pipeline stages represented as single registers.

```
module mult(clk, a, b, mult);  
input clk;  
input [17:0] a;
```

Orsu Venkata Krishnaiyah

```
input [17:0] b;  
output [35:0] mult;  
reg [35:0] mult;  
reg [17:0] a_in, b_in;  
wire [35:0] mult_res;  
reg [35:0] pipe_1, pipe_2, pipe_3;
```

```
assign mult_res = a_in * b_in;
```

```
always @ (posedge clk)
```

```
begin
```

```
a_in <= a;
```

```
b_in <= b;
```

```
pipe_1 <= mult_res;
```

```
pipe_2 <= pipe_1;
```

```
pipe_3 <= pipe_2;
```

```
mult <= pipe_3;
```

```
end
```

```
endmodule
```

42. Write a Verilog template shows the multiplication operation placed inside the always block and the pipeline stages are represented as single registers.

Orsu Venkata Krishnaiyah

```
module mult(clk, a, b, mult);
input clk;
input [17:0] a;
input [17:0] b;
output [35:0] mult;
reg [35:0] mult;
reg [17:0] a_in, b_in;
reg [35:0] mult_res;
reg [35:0] pipe_2, pipe_3;
always @(posedge clk)
begin
    a_in <= a;
    b_in <= b;
    mult_res <= a_in * b_in;
    pipe_2 <= mult_res;
    pipe_3 <= pipe_2;
    mult <= pipe_3;
end
endmodule
```

43. Write a Verilog template shows the multiplication operation placed outside the always block and the pipeline stages represented as single registers.

Orsu Venkata Krishnaiyah

```
module mult(clk, a, b, mult);
input clk;
input [17:0] a;
input [17:0] b;
output [35:0] mult;
reg [35:0] mult;
reg [17:0] a_in, b_in;
wire [35:0] mult_res;
reg [35:0] pipe_1, pipe_2, pipe_3;

assign mult_res = a_in * b_in;

always @ (posedge clk)
begin
    a_in <= a;
    b_in <= b;
    pipe_1 <= mult_res;
    pipe_2 <= pipe_1;
    pipe_3 <= pipe_2;
    mult <= pipe_3;
end
endmodule
```

44. Write a Verilog template shows the

Orsu Venkata Krishnaiyah

multiplication operation placed inside the always block and the pipeline stages are represented as single registers.

```
module mult(clk, a, b, mult);
    input clk;
    input [17:0] a;
    input [17:0] b;
    output [35:0] mult;
    reg [35:0] mult;
    reg [17:0] a_in, b_in;
    reg [35:0] mult_res;
    reg [35:0] pipe_2, pipe_3;
    always @ (posedge clk)
    begin
        a_in <= a;
        b_in <= b;
        mult_res <= a_in * b_in;
        pipe_2 <= mult_res;
        pipe_3 <= pipe_2;
        mult <= pipe_3;
    end
endmodule
```

45. Write a Verilog template shows the

Orsu Venkata Krishnaiyah

multiplication operation placed outside the always block and the pipeline stages represented as shift registers.

```
module mult3(clk, a, b, mult);
    input clk;
    input [17:0] a;
    input [17:0] b;
    output [35:0] mult;
    reg [35:0] mult;
    reg [17:0] a_in, b_in;
    wire [35:0] mult_res;
    reg [35:0] pipe_regs [3:0];
```

```
assign mult_res = a_in * b_in;
```

```
always @ (posedge clk)
```

```
begin
```

```
    a_in <= a;
```

```
    b_in <= b;
```

```
    {pipe_regs[3], pipe_regs[2], pipe_regs[1], pipe_regs[0]} <=
        {mult,
```

```
        pipe_regs[3], pipe_regs[2], pipe_regs[1]};
```

```
end
```

```
endmodule
```

Orsu Venkata Krishnaiyah

46. Write a templates to implement Multiplier Adder with 2 Register Levels on Multiplier Inputs in Verilog.

```
module mvl_multaddsub1(clk, a, b, c, res);
    input clk;
    input [07:0] a;
    input [07:0] b;
    input [07:0] c;
    output [15:0] res;
    reg [07:0] a_reg1, a_reg2, b_reg1, b_reg2;
    wire [15:0] multaddsub;
    always @(posedge clk)
    begin
        a_reg1 <= a;
        a_reg2 <= a_reg1;
        b_reg1 <= b;
        b_reg2 <= b_reg1;
    end
    assign multaddsub = a_reg2 * b_reg2 + c;
    assign res = multaddsub;
endmodule
```

Orsu Venkata Krishnaiyah

47. Write a Verilog code for resource sharing.

```
module addsub(a, b, c, oper, res);
    input oper;
    input [7:0] a;
    input [7:0] b;
    input [7:0] c;
    output [7:0] res;
    reg [7:0] res;
    always @ (a or b or c or oper)
        begin
            if (oper == 1'b0)
                res = a + b;
            else
                res = a - c;
        end
    endmodule
```

48. Write a templates show a single-port RAM in read-first mode.

```
module ramrfr (clk, en, we, addr, di, do);
    input clk;
    input we;
    input en;
```

Orsu Venkata Krishnaiyah

```
input [4:0] addr;
input [3:0] di;
output [3:0] do;
reg [3:0] RAM[31:0];
reg [3:0] do;
always @(posedge clk)
begin
if (en) begin
if (we)
RAM[addr] <= di;

do <= RAM[addr];
end
end
endmodule
```

49. Write a templates show a single-port RAM in write-first mode.

```
module raminfr (clk, we, en, addr, di, do);
input clk;
input we;
input en;
input [4:0] addr;
input [3:0] di;
```

Orsu Venkata Krishnaiyah

```
output [3:0] do;
reg [3:0] RAM[31:0];
reg [4:0] read_addr;
always @(posedge clk)
begin
if (en) begin
if (we)
RAM[addr] <= di;
read_addr <= addr;
end
end
assign do = RAM[read_addr];
endmodule
```

50. Write a templates show a single-port RAM in no-change mode.

```
module raminfr (clk, we, en, addr, di, do);
input clk;
input we;
input en;
input [4:0] addr;
input [3:0] di;
output [3:0] do;
reg [3:0] RAM[31:0];
```

Orsu Venkata Krishnaiyah

```
reg [3:0] do;
always @(posedge clk)
begin
if (en) begin
if (we)
RAM[addr] <= di;
else
do <= RAM[addr];
end
end
endmodule
```

51. Write a Verilog code for a single-port RAM with asynchronous read.

```
module raminfr (clk, we, a, di, do);
input clk;
input we;
input [4:0] a;
input [3:0] di;
output [3:0] do;
reg [3:0] ram [31:0];
always @(posedge clk)
begin
if (we)
```

Orsu Venkata Krishnaiyah

```
ram[a] <= di;  
end  
assign do = ram[a];  
endmodule
```

52. Write a Verilog code for a single-port RAM with "False" synchronous read.

```
module raminfr (clk, we, a, di, do);  
input clk;  
input we;  
input [4:0] a;  
input [3:0] di;  
output [3:0] do;  
reg [3:0] ram [31:0];  
reg [3:0] do;  
always @(posedge clk)  
begin  
if (we)  
ram[a] <= di;  
do <= ram[a];  
end  
endmodule
```

Orsu Venkata Krishnaiyah

53. Write a Verilog code for a single-port RAM with synchronous read (read through).

```
module ramintfr (clk, we, a, di, do);
    input clk;
    input we;
    input [4:0] a;
    input [3:0] di;
    output [3:0] do;
    reg [3:0] ram [31:0];
    reg [4:0] read_a;
    always @ (posedge clk)
    begin
        if (we)
            ram[a] <= di;
        read_a <= a;
    end
    assign do = ram[read_a];
endmodule
```

54. Write a Verilog code for a single-port block RAM with enable.

```
module ramintfr (clk, en, we, a, di, do);
    input clk;
    input en;
```

Orsu Venkata Krishnaiyah

```
input en;
input we;
input [4:0] a;
input [3:0] di;
output [3:0] do;
reg [3:0] ram[31:0];
reg [4:0] read_a;
always @(posedge clk)
begin
if(en) begin
if(we)
ram[a] <= di;
read_a <= a;
end
end
assign do = ram[read_a];
endmodule
```

55. Write a Verilog code for a dual-port RAM with asynchronous read.

```
module raminfr (clk, we, a, dpra, di, spo,
dpo);
input clk;
input we;
```

Orsu Venkata Krishnaiyah

```
input [4:0] a;
input [4:0] dpra;
input [3:0] di;
output [3:0] spo;
output [3:0] dpo;
reg [3:0] ram[31:0];
always @ (posedge clk)
begin
if (we)
ram[a] <= di;
end
assign spo = ram[a];
assign dpo = ram[dpra];
endmodule
```

56. Write a Verilog code for a dual-port RAM with false synchronous read.

```
module ramiffr (clk, we, a, dpra, di, spo,
dpo);
input clk;
input we;
input [4:0] a;
input [4:0] dpra;
input [3:0] di;
```

Orsu Venkata Krishnaiyah

```
output [3:0] spo;
output [3:0] dpo;
reg [3:0] ram[31:0];
reg [3:0] spo;
reg [3:0] dpo;
always @(posedge clk)
begin
if (we)
ram[a] <= di;
```

```
spo = ram[a];
dpo = ram[dpra];
end
endmodule
```

57. Write a Verilog code for a dual-port RAM with synchronous read (read through).

```
module raminfr (clk, we, a, dpra, di, spo,
dpo);
input clk;
input we;
input [4:0] a;
input [4:0] dpra;
```

Orsu Venkata Krishnaiyah

```
input [3:0] di;
output [3:0] spo;
output [3:0] dpo;
reg [3:0] ram [31:0];
reg [4:0] read_a;
reg [4:0] read_dpra;
always @(posedge clk)
begin
if (we)
ram[a] <= di;
read_a <= a;
read_dpra <= dpra;
end
assign spo = ram[read_a];
assign dpo = ram[read_dpra];
endmodule
```

58. Write a Verilog code for a dual-port RAM with enable on each port.

```
module raminfr (clk, ena, enb, wea, addra,
addrb, dia, doa, dob);
input clk, ena, enb, wea;
input [4:0] addra, addrb;
input [3:0] dia;
```

Orsu Venkata Krishnaiyah

```
output [3:0] doa, dob;
reg [3:0] ram[31:0];
reg [4:0] read_addr_a, read_addr_b;
always @(posedge clk)
begin
if (ena) begin
if (wea) begin
ram[addr_a] <= dia;
end
end
end
```

```
always @(posedge clk)
begin
if (enb) begin
read_addr_b <= addr_b;
end
end
assign doa = ram[read_addr_a];
assign dob = ram[read_addr_b];
endmodule
```

59. Write a Verilog code for a ROM with registered output.

Orsu Venkata Krishnaiyah

```
module rominfr (clk, en, addr, data);
input clk;
input en;
input [4:0] addr;
output reg [3:0] data;
always (@(posedge clk))
begin
if (en)
case(addr)
4'b0000: data <= 4'b00010;
4'b0001: data <= 4'b00010;
4'b0010: data <= 4'b1110;
4'b0011: data <= 4'b00010;
4'b0100: data <= 4'b01000;
4'b0101: data <= 4'b1010;
4'b0110: data <= 4'b1100;
4'b0111: data <= 4'b00000;
4'b1000: data <= 4'b1010;
4'b1001: data <= 4'b00010;
4'b1010: data <= 4'b1110;
4'b1011: data <= 4'b00010;
4'b1100: data <= 4'b01000;
4'b1101: data <= 4'b1010;
4'b1110: data <= 4'b1100;
4'b1111: data <= 4'b00000;
```

Orsu Venkata Krishnaiyah

```
default: data <= 4'bXXXX;  
endcase  
end  
endmodule
```

60. Write a Verilog code for a ROM with registered address.

```
module rominfr (clk, en, addr, data);  
input clk;  
input en;  
input [4:0] addr;  
output reg [3:0] data;  
reg [4:0] raddr;  
always @ (posedge clk)  
begin  
if (en)  
raddr <= addr;  
end
```

```
always @ (raddr)  
begin  
if (en)  
casel(raddr)  
4'b0000: data = 4'b0010;
```

Orsu Venkata Krishnaiyah

```
4'b0001: data = 4'b0010;  
4'b0010: data = 4'b1110;  
4'b0011: data = 4'b0010;  
4'b0100: data = 4'b0100;  
4'b0101: data = 4'b1010;  
4'b0110: data = 4'b1100;  
4'b0111: data = 4'b0000;  
4'b1000: data = 4'b1010;  
4'b1001: data = 4'b0010;  
4'b1010: data = 4'b1110;  
4'b1011: data = 4'b0010;  
4'b1100: data = 4'b0100;  
4'b1101: data = 4'b1010;  
4'b1110: data = 4'b1100;  
4'b1111: data = 4'b0000;  
default: data = 4'bXXXX;  
endcase  
end  
endmodule
```

61. Write a Verilog code for an FSM with a single process.

```
module fsm(clk, reset, x1, outp);  
input clk, reset, x1;
```

Orsu Venkata Krishnaiyah

```
output outp;
reg outp;
reg [1:0] state;
parameter s1 = 2'b00; parameter s2 =
2'b01;
parameter s3 = 2'b10; parameter s4 = 2'b11;
always @ (posedge clk or posedge reset)
begin
if (reset) begin
state <= s1; outp <= 1'b1;
end
else begin
case (state)
s1: begin
if (x == 1'b1) begin
state <= s2;
outp <= 1'b1;
end
else begin
state <= s3;
outp <= 1'b1;
end
end
s2: begin
state <= s4;
```

Orsu Venkata Krishnaiyah

```
outp <= 1'b0;  
end  
s3: begin  
state <= s4;  
outp <= 1'b0;  
end  
s4: begin  
state <= s1;  
outp <= 1'b1;  
end  
endcase  
end  
end  
endmodule
```

62. Write a Verilog code for an FSM with two processes.

```
module fsm (clk, reset, xl, outp);  
input clk, reset, xl;  
output outp;  
reg outp;  
reg [1:0] state;  
parameter s1 = 2'b00; parameter s2 =  
2'b01;
```

Orsu Venkata Krishnaiyah

```
parameter s3 = 2'b10; parameter s4 = 2'b11;
always @ (posedge clk or posedge reset)
begin
if (reset)
state <= s1;
else begin
case (state)
s1: if (x1 == 1'b1)
state <= s2;
else
state <= s3;
s2: state <= s4;
s3: state <= s4;
s4: state <= s1;
endcase
end
end
always @ (state) begin
case (state)
s1: outp = 1'b1;
s2: outp = 1'b1;
s3: outp = 1'b0;
s4: outp = 1'b0;
endcase
end
```

Orsu Venkata Krishnaiyah

endmodule

63. Write a Verilog code for an FSM with three processes.

```
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; parameter s2 =
    2'b01;
    parameter s3 = 2'b10; parameter 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)
```