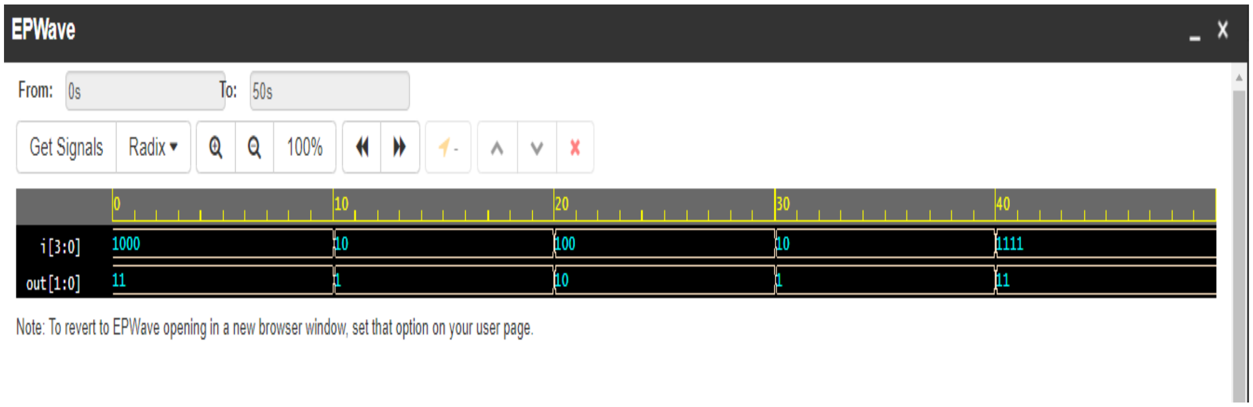
```
////////////////Even Priority encoder///////////////////////

//code

module even_pri_enc(input [3:0] i,output reg [1:0] out);


  always @* begin

    case(i)

      4'b0000: out = 2'b00;

      4'b0010: out = 2'b01;

      4'b0011: out = 2'b01;

      4'b0100: out = 2'b10;

      4'b0101: out = 2'b10;

      4'b0110: out = 2'b10;

      4'b0111: out = 2'b10;

      4'b1000: out = 2'b11;

      default: out = 2'b11;

    endcase

  end


endmodule


//Test Bench

// Code your testbench here

// or browse Examples

module tb;

  reg [3:0] i;
```

```
 wire [1:0] out;


 even_pri_enc uut (.i(i),.out(out));


 initial begin
  $monitor("Time=%0t in_data=%b out=%b", $time, i, out);
  i = 4'b0000; #10;
  i = 4'b0010; #10;
  i = 4'b0100; #10;
  i = 4'b1000; #10;
  i = 4'b1111; #10;
  $finish;
 end


endmodule
```

//
//////////////////



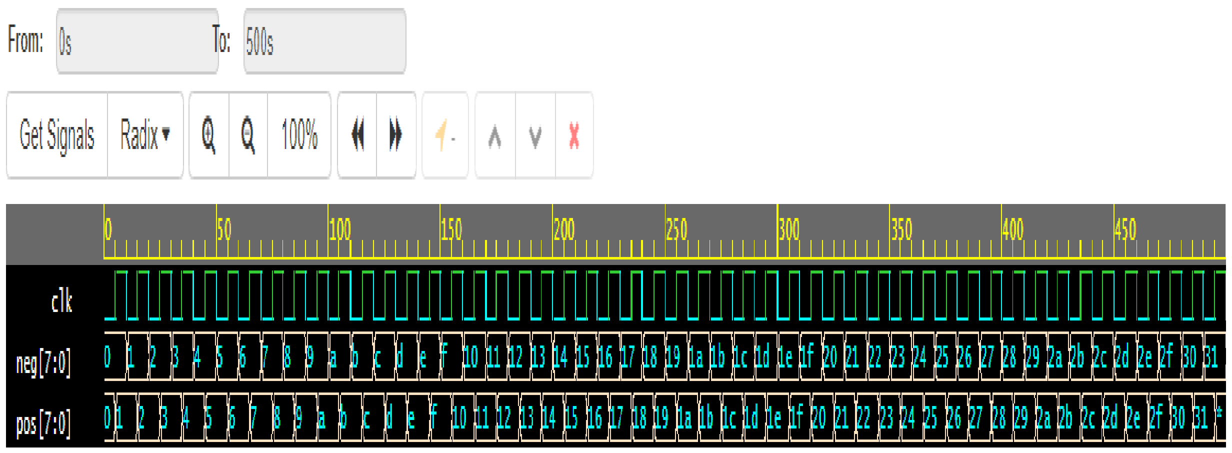
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

////////////////////////////////RISING AND FALLING EDGE COUNTER////////////////////////////////

```
//CODE
module edge_counter(
 input wire clk,
 output reg [7:0] pos,
 output reg [7:0] neg
);

 reg ss;
 initial
  begin
  pos<=0;
  neg<=0;
 end
 always @(posedge clk) begin
  if(ss==0)
   pos <= pos + 1;
  ss=1;
 end

 always @(negedge clk) begin
  if(ss==1)
   neg <= neg + 1;
  ss=0;
 end
```

```
endmodule

/// TEST BENCH



module edge_counter_tb;


 reg clk;

 wire [7:0] pos;

 wire [7:0] neg;


 edge_counter uut (

  .clk(clk),

  .pos(pos),

  .neg(neg)

 );


 initial begin

  clk=0;

  $monitor("clk=%b pos_count=%d neg_count=%d", clk, pos, neg);

  repeat (100) #5 clk = ~clk;

  $finish();

 end


endmodule
```

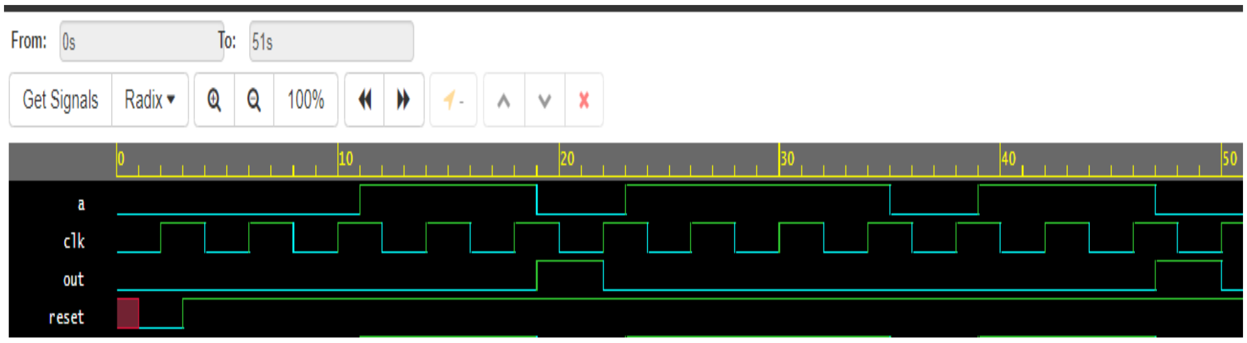
///


From: 0s
To: 500s
Get Signals
Radix
100%
0
50
100
150
200
250
300
350
400
450
clk
neg[7:0]
0 1 2 3 4 5 6 7 8 9 a b c d e f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31
pos[7:0]
0 1 2 3 4 5 6 7 8 9 a b c d e f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 *

```
//////////////////////    FSM SEQ DETECTOR 0110/////////////////////////////////////////////////////////

module seq0110(input bit clk, rst, a, output out);

 bit [3:0] st, n_st;



 always @(posedge clk or negedge rst) begin

  if (!rst) begin

   st <= 4'h1;

  end

  else begin

   st <= n_st;

  end

 end



 always @(st or a) begin
```

```
    case(st)

      4'h1: begin

        if (a == 0) n_st = 4'h2;

        else n_st = 4'h1;

      end

      4'h2: begin

        if (a== 1) n_st = 4'h3;

        else n_st = 4'h2;

      end

      4'h3: begin

        if (a == 1) n_st = 4'h4;

        else n_st = 4'h2;

      end

      4'h4: begin

        if (a == 0) n_st = 4'h1;

        else n_st = 4'h1;

      end

      default: n_st = 4'h1;

    endcase

  end


  assign out = (st == 4'h4) && (a == 0) ? 1 : 0;

endmodule


//TEST BENCH
```

```
module tb();
  reg clk, reset, a;
  wire out;

  seq0110 d(clk, reset, a, out);
  initial clk = 0;
  always #2 clk = ~clk;

  initial begin
    a = 0;
    #1 reset = 0;
    #2 reset = 1;



    #4 a = 0;
    #4 a = 1;
    #4 a = 1;
    #4 a = 0;

     #4 a = 1;
    #4 a = 1;
    #4 a = 1;
    #4 a = 0;

     #4 a = 1;
    #4 a = 1;

    #4 a = 0;

    #4 a = 0;




    $finish;

  end


  initial begin


    $dumpfile("dump.vcd");

    $dumpvars(0);

  end

endmodule
```

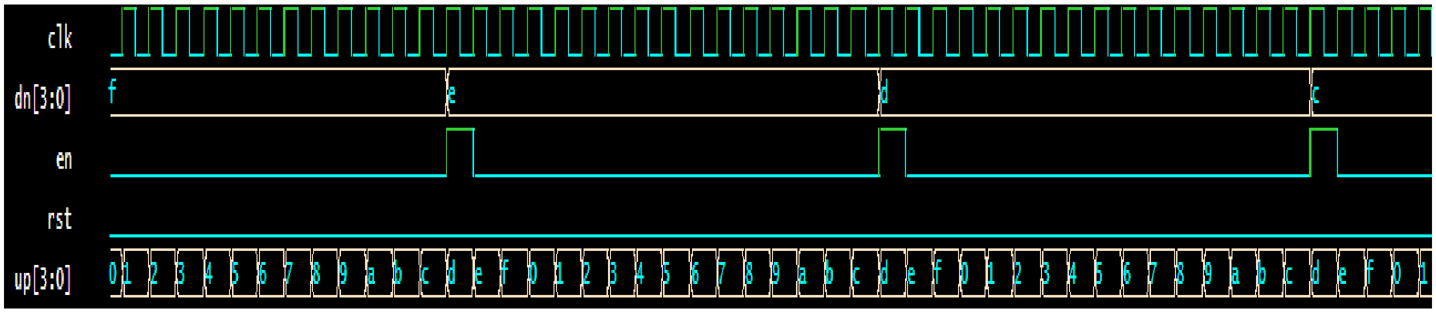


```
////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////

//////////////Counter

// Code your design here

module counter (

  input rst,

  input clk,

  output reg en,

  output reg [3:0] up,

  output reg [3:0] dn

);


  initial begin

    up <= 4'b0000;

    dn <= 4'b1111;

    en <= 1'b0;

  end


  always @(posedge rst or posedge clk) begin

    if (rst) begin

      up <= 4'b0000;

      dn <= 4'b1111;

      en <= 1'b0;

    end

    else begin
```

```
      up <= up + 1;

      if (up == 4'b1100) begin

        en <= 1;

      end

      else begin

        en <= 0;

      end

    end

  end


  always @(posedge en) begin

    dn <= dn - 1;

  end


endmodule


////////////////TEST BENCH


module top;

  reg clk, rst;

  reg en;

  reg [3:0] up, dn;

  integer count = 0;

  integer start_time, end_time;

  real frequency;
```

```
counter uut (.rst(rst), .clk(clk), .en(en), .up(up), .dn(dn));

initial begin
  clk = 0;
  forever #5 clk = ~clk;
end

initial begin
  rst = 1;
  #10 rst = 0;
  $dumpfile("dump.vcd"); $dumpvars;
  $monitor(" up=%b, dn=%b, en=%b frequency=%f", up, dn, en, frequency);
end

always @(posedge en) begin
  if (!count) begin
    start_time = $time;
    count = count + 1;
  end
  else begin
    end_time = $time;
    frequency = 1 / (($time - start_time) * 1e-9);
    count = 0;
  end
end
```

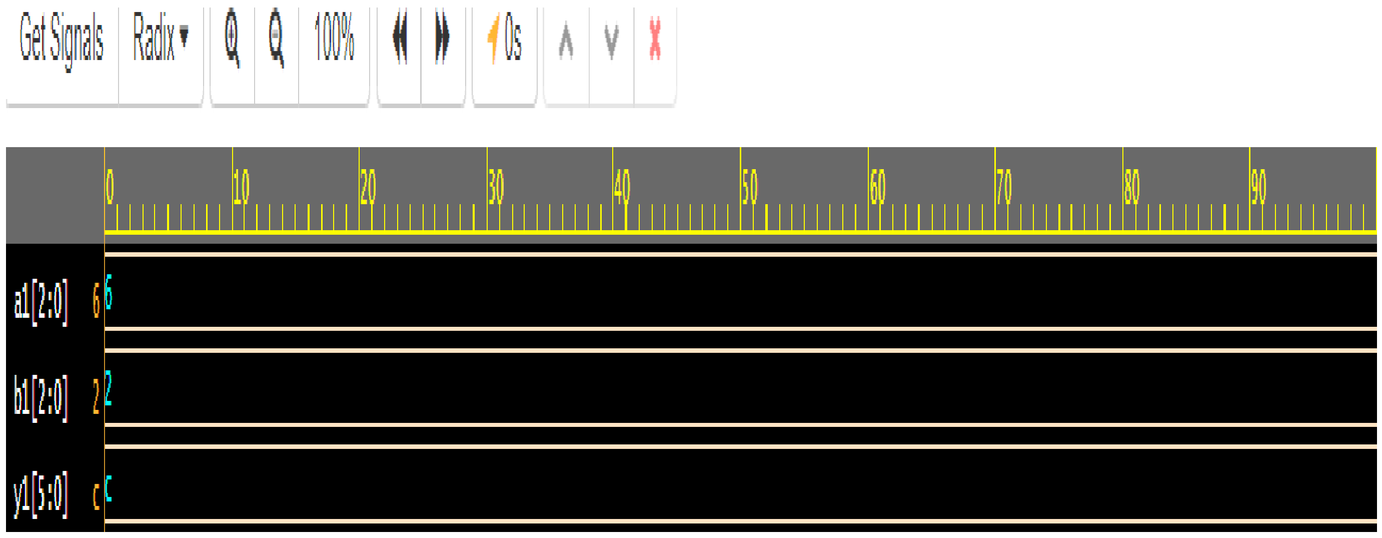
```
  initial #500 $finish;


endmodule/////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
```



```
////////////////////////////////////////////////////////////////////////


module Multiplier3bit(

  input wire [2:0] a1,

  input wire [2:0] b1,

  output reg [5:0] y1

);

  reg [5:0] y1_pro;

  initial begin

    y1_pro=0;

    y1=0;

  end

  always @(a1 or b1) begin

    for (int i = 0; i < 3; i = i + 1) begin

      y1_pro = y1_pro + (a1[i]*(b1<<i));
    end
    y1 = y1_pro;
  end
endmodule
```

TESET BENCH

```
module testbench;

  reg [2:0] a1, b1;
  wire [5:0] y1;

  Multiplier3bit  uut(
    .a1(a1),
    .b1(b1),
    .y1(y1)
  );

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1);
    a1 = 3'b101;
    b1 = 3'b011;
    $display("a1=%b b1=%b y1=%b", a1, b1, y1);
    a1 = 3'b110;

    b1 = 3'b010;

    $display("a1=%b b1=%b y1=%b", a1, b1, y1);


    #100 $finish();

  end


endmodule
```


Get Signals
Radix
100%
0s
0
10
20
30
40
50
60
70
80
90
a1[2:0]
6
6
b1[2:0]
2
2
y1[5:0]
c
c