Design 2:1 MUX using Dataflow moduling

Design

module mux (I0,I1,S,Y);

input I0,I1,S;

output Y;

assign Y=((~S)&I0)|(S&I1);

endmodule

Testbench

module mux\_test;

// signal declaration

reg I0,I1,S;

wire Y;

mux dut(I0,I1,S,Y);

//DUT

initial begin

I0=1'b0;I1=1'b0;S=1'b0;

#5 I0=1'b0;I1=1'b0;S=1'b1;

#5 I0=1'b0;I1=1'b1;S=1'b0;

#5 I0=1'b0;I1=1'b1;S=1'b1;

#5 I0=1'b1;I1=1'b0;S=1'b0;

#5 I0=1'b1;I1=1'b0;S=1'b1;

#5 I0=1'b1;I1=1'b1;S=1'b0;

#5 I0=1'b1;I1=1'b1;S=1'b1;

end

//capturing result

initial begin

$monitor("simtime =%0t,I0=%b,I1=%b,S=%b,Y=%b",$time,I0,I1,S,Y);

end

//to print waveform

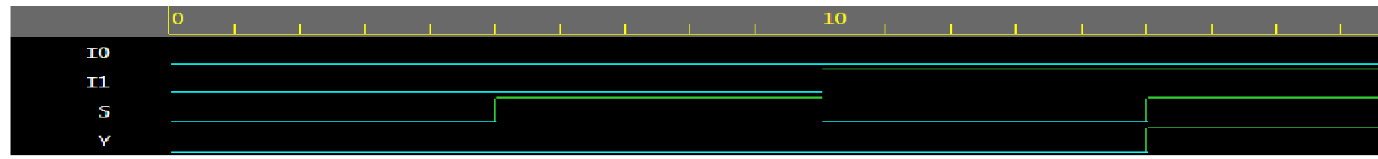
initial begin

$dumpfile("dump.vcd");

$dumpvars(0,I0,I1,S,Y);

end

endmodule



Even odd detector

Design

module detector(A,B,C,Y1,Y2);

input A,B,C;

output Y1,Y2;

assign Y1=~C;

assign Y2=C;

endmodule

Testbench

module detect\_test;

// signal declaration

reg A,B,C;

wire Y1,Y2;

detector dut(A,B,C,Y1,Y2);

// generation of stimulus

initial begin

A=1'b0;B=1'b0;C=1'b0;

#5 A=1'b0;B=1'b0;C=1'b1;

#5 A=1'b0;B=1'b1;C=1'b0;

#5 A=1'b0;B=1'b1;C=1'b1;

#5 A=1'b1;B=1'b0;C=1'b0;

#5 A=1'b1;B=1'b0;C=1'b1;

#5 A=1'b1;B=1'b1;C=1'b0;

#5 A=1'b1;B=1'b1;C=1'b1;

end

// to display

initial begin

$monitor("simtime=%0t,A=%b,B=%b,C=%b,Y1=%b,Y2=%b",$time,A,B,C,Y1,Y2);

end

// waveform

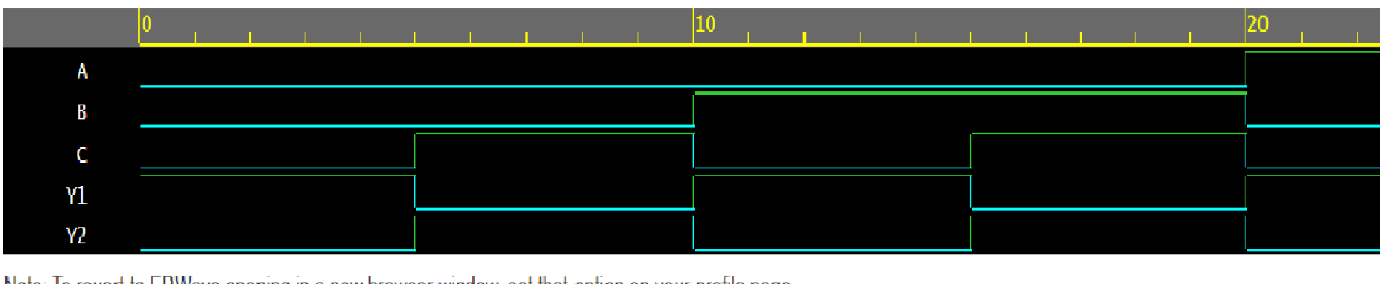
initial begin

$dumpfile("dump.vcd");

$dumpvars(0,A,B,C,Y1,Y2);

end

endmodule



Design 2x1 mux using ternary operator

Design

module mux1(I,S,Y);

input [3:0]I;

input [1:0]s;

output Y;

assign Y=(s==2`b00)?I[0]:(s==2`b01)?I[1]:((s==2`b10)?I[2]:I[3]);

endmodule

Test bench

module mux1\_test;

// signal declaration

reg [3:0]I,[1:0]S;

wire Y;

mux1 dut(I,S,Y);

//DUT

initial begin

S=2'b00; I=4'b0000;

#5 S=2'b01; I=4'b0001;

#5 S=2'b10; I=4'b0010;

#5 S=2'b11; I=4'b0100;

end

//capturing result

initial begin

$monitor("simtime =%0t,I=%b,S=%b,Y=%b",$time,I,S,Y);

end

//to print waveform

initial begin

$dumpfile("dump.vcd");

$dumpvars(0,I,S,Y);

end

endmodule

Design structural code for full adder using half adder

Design

module HA(a,b,s,c);

input a,b;

output s,c;

assign s=a^b;

assign c=&b;

endmodule

ìnclude"HA.v"

module FA(a,b,cin,sum,cout);

input a,b,cin;

output sum,cout;

wire w1,w2,w3;

HA a1(a,b,w1,w2);

HA a2(w1,cin,sum,w3);

assign cout=w2|w1;

endmodule

testbench

module FA\_test;

reg a,b,cin;

wire sum,cout;

FA dut(a,b,cin,sum,cout);

initial begin

a=1'b0;b=1'b0;Cin=1'b0;

#5 a=1'b0;b=1'b0;Cin=1'b1;

#5 a=1'b0;b=1'b1;Cin=1'b0;

#5 a=1'b0;b=1'b1;Cin=1'b1;

#5 a=1'b1;b=1'b0;Cin=1'b0;

#5 a=1'b1;b=1'b0;Cin=1'b1;

#5 a=1'b1;b=1'b1;Cin=1'b0;

#5 a=1'b1;b=1'b1;Cin=1'b1;

end

// to display

initial begin

$monitor("simtime=%0t,a=%b,b=%b,Cin=%b,sum=%b,cout=%b",$time,a,b,Cin,sum,cout);

end

// waveform

initial begin

$dumpfile("dump.vcd");

$dumpvars(0,a,b,Cin,sum,cout);

end

endmodule

Design a structural code for 4bit adder using full adder

Design

ìnclude"HA.v"

module FA(a,b,cin,sum,cout);

input a,b,cin;

output sum,cout;

wire w1,w2,w3;

HA a1(a,b,w1,w2);

HA a2(w1,cin,sum,w3);

assign cout = w2|w1;

endmodule

`include “FA”

module Add4bit(a,b,cin,s,cout);

input[3:0]a,b;

input cin;

output[3:0]s;

output [2:0]cout;

Add4bit a1(a[0],b[0],cin,s[0],c[0]);

Add4bit a2(a[1],b[1],c[0],s[1],c[1]);

Add4bit a3(a[2],b[2],c[1],s[2],c[2]);

Add4bit a4(a[3],b[3],c[2],s[3],cout);

Endmodule

Testbench

module Add4bit\_test;

reg[3:0]a,b;

reg cin;

wire[3:0]s;

wire [2:0]cout;

Add4bit dut(a,b,cin,sum,cout);

// generation of stimulus

initial begin

a=4'b0000;b=4'b1001;cin=0;

#5 a=4'b0100;b=4'b0001;cin=1;

#5 a=4'b0101;b=4'b1001;cin=0;

end

// to display

initial begin

$monitor("simtime=%0t,a=%b,b=%b,Cin=%b,sum=%b,cout=%b",$time,a,b,Cin,sum,cout);

end

// waveform

initial begin

$dumpfile("dump.vcd");

$dumpvars(0,a,b,Cin,sum,cout);

end

endmodule

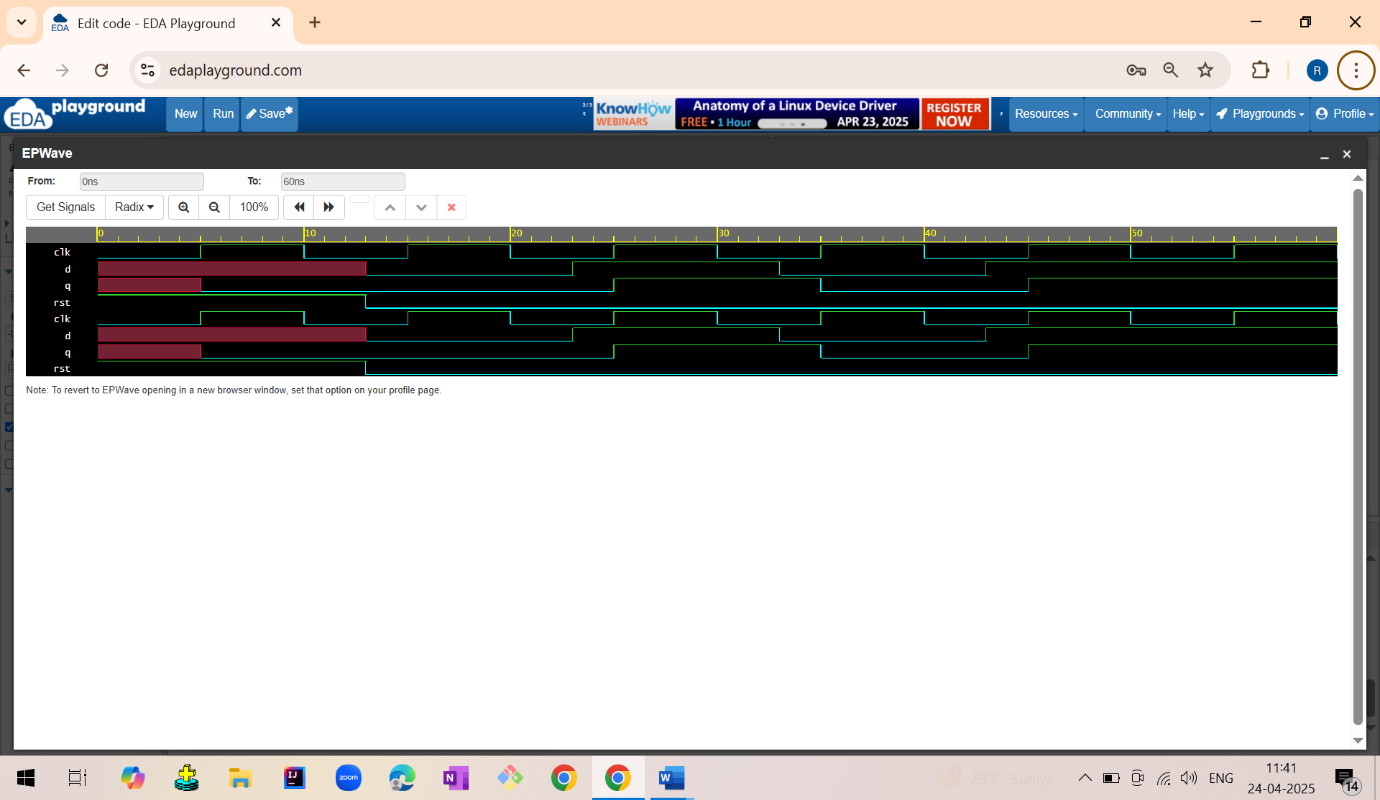
Verilog code of Dff with synchronous rst using behavioral model

|  |  |
| --- | --- |
| TB  module dff\_test;  reg clk,rst,d;  wire q;  dff dut(clk,rst,d,q);  initial begin  clk=0;  rst=1;  #13 rst=0;  d=1'b0;  #10 d=1'b1;  #10 d=1'b0;  #10 d=1'b1;  #20 $finish;  end    always #5 clk=~clk;    initial begin  $dumpfile("dump.vcd");  $dumpvars(0,dff\_test);  end  endmodule | Design  module dff(clk,rst,d,q);  input clk,rst,d;  output reg q;  always @(posedge clk)  begin  if(rst)  q<=0;  else  q<=d;  end  endmodule |

Output:

sim time=0,clk=0,rst=1,d=x,q=x  
sim time=5,clk=1,rst=1,d=x,q=0  
sim time=10,clk=0,rst=1,d=x,q=0  
sim time=13,clk=0,rst=0,d=0,q=0  
sim time=15,clk=1,rst=0,d=0,q=0  
sim time=20,clk=0,rst=0,d=0,q=0  
sim time=23,clk=0,rst=0,d=1,q=0  
sim time=25,clk=1,rst=0,d=1,q=1  
sim time=30,clk=0,rst=0,d=1,q=1  
sim time=33,clk=0,rst=0,d=0,q=1

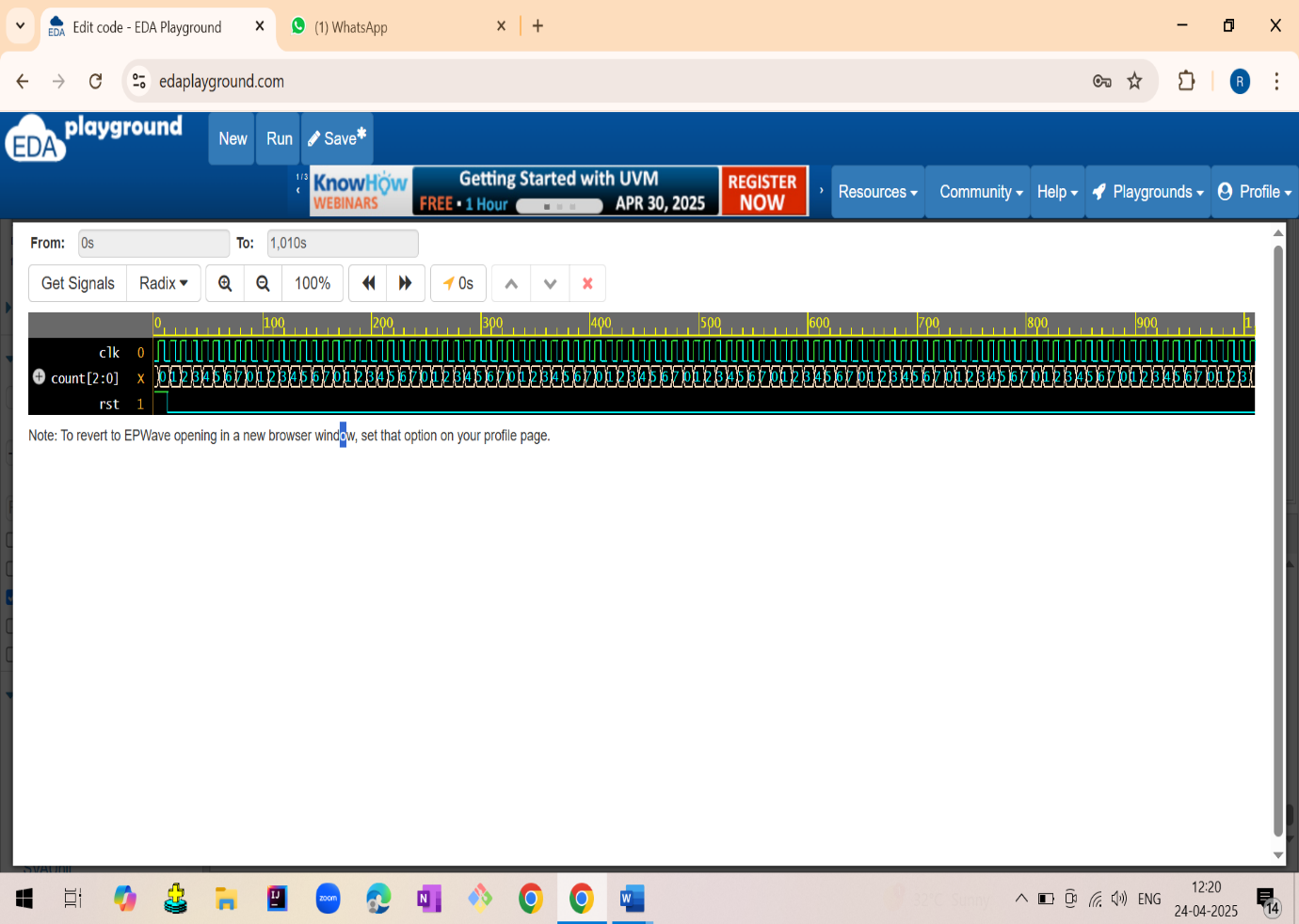
Waveforms



Verilog code for counter with synchronous rst

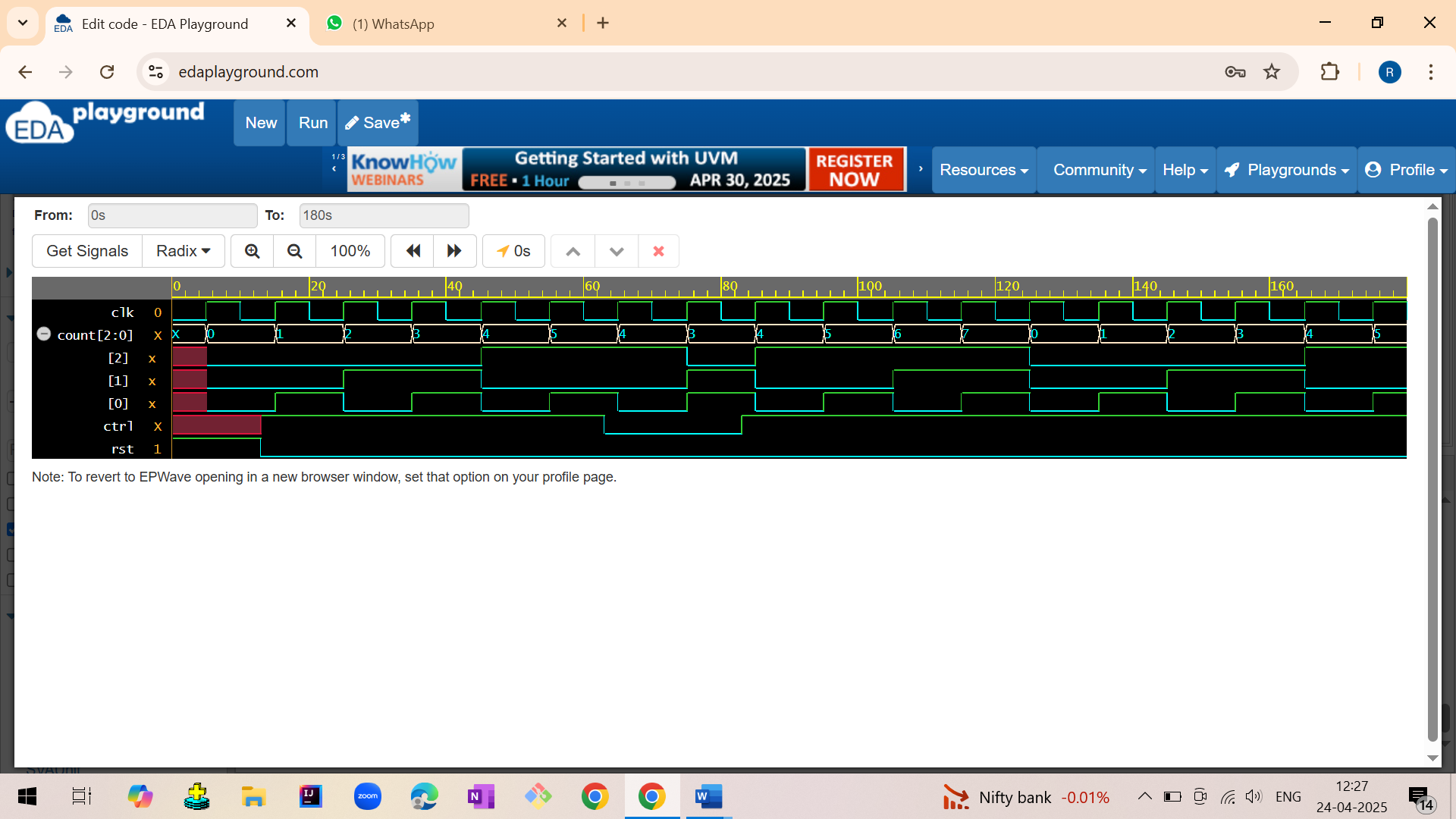
|  |  |
| --- | --- |
| TB  module counter\_test;  reg clk,rst;  wire[2:0]count;    counter dut(clk,rst,count);    initial begin  clk=0;  rst=1;  #13 rst=0;  #1000 $finish;  end    always #5 clk=~clk;    initial begin  $monitor("simtime=%0t,clk=%b,rst=%b",$time,clk,rst);  end      initial begin  $dumpfile("dump.vcd");  $dumpvars(1,counter\_test);  end  endmodule | Code  module counter(clk,rst,count);  input clk,rst;  output[2:0]count;  reg[2:0]temp;  assign count=temp;    always@(posedge clk)  begin  if(rst)  temp<=1'b000;  else  temp<=temp+1;  end  endmodule |

Waveforms:



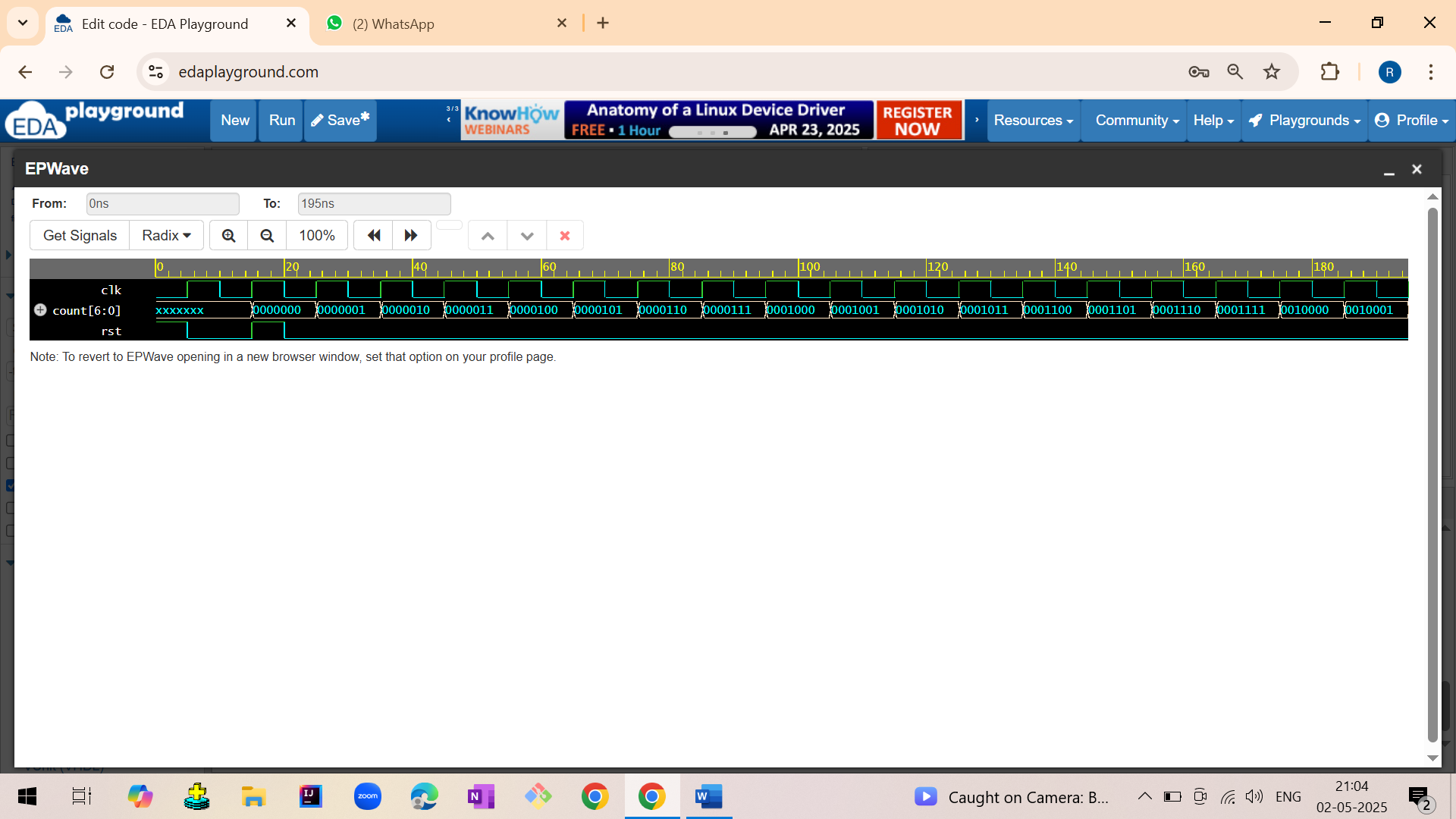
|  |  |
| --- | --- |
| **TB**  module counter\_test;  reg clk,rst,ctrl;  wire[2:0]count;    counter dut(clk,rst,ctrl,count);    initial begin  clk=0;  rst=1;  #13 rst=0;  ctrl=1;  #50 ctrl=0;  #20 ctrl=1;  #100 $finish;  end    always #5 clk=~clk;    initial begin  $monitor("simtime=%0t,clk=%b,rst=%b,ctrl=%b",$time,clk,rst,ctrl);  end      initial begin  $dumpfile("dump.vcd");  $dumpvars(1,counter\_test);  end  endmodule | Design  module counter (clk,rst,ctrl,count);  input clk,rst,ctrl;  output [2:0] count;  reg [2:0] temp;  assign count= temp;      always@(posedge clk)  begin  if(rst)  temp<= 3'b000;  else if(ctrl)  temp<= temp+1;  else  temp<= temp-1;    end  endmodule |

Waveforms:



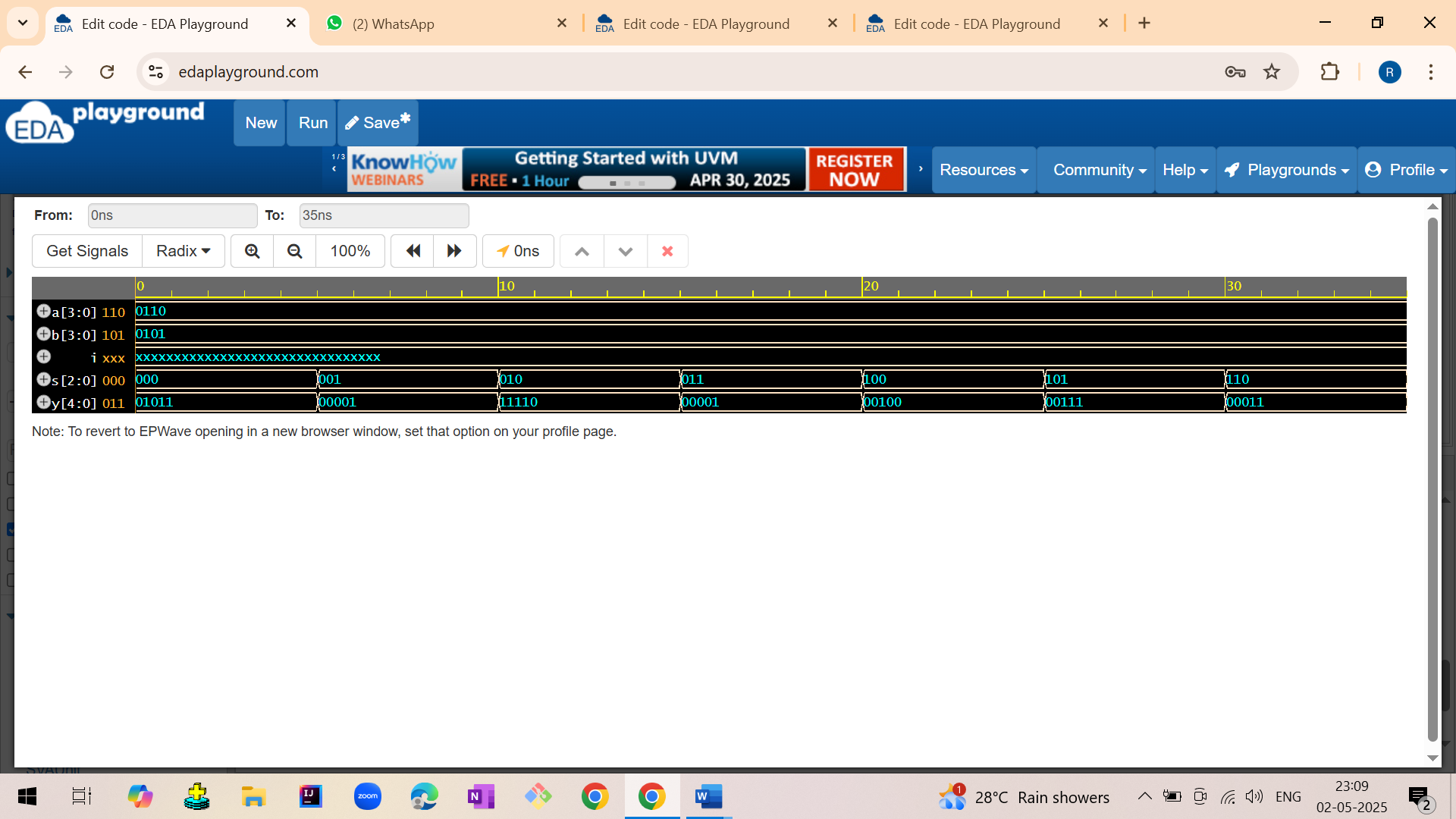
Mod100 counter Verilog code

|  |  |
| --- | --- |
| TB  //testbench  module mod100\_tb;  reg clk,rst;  wire [6:0]count;      mod100 dut(clk,rst,count);    initial begin  clk = 0;  forever #5 clk = ~clk;  end  initial begin  rst = 1;  #5 rst = 0;  #10 rst = 1;  #5 rst = 0;  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(1, mod100\_tb);  end  initial begin  #200 $finish;  end  endmodule | Design  //Mod-100 Counter Verilog Code  module mod100(clk,rst,count);  input clk,rst;  output reg [6:0] count;  reg[6:0] temp;    assign count=temp;  always @(posedge clk)  begin  if (rst)  temp<= 7'd0;  else if (temp == 7'd99)  temp<= 7'd0;  else  temp<= temp + 1;    end  endmodule |



Write a Verilog code to implement ALU .

|  |  |
| --- | --- |
| Tb  module alu\_test;    reg [3:0]a,b;  reg[2:0]s;  wire [4:0]y;  integer i;      alu dut(a,b,s,y);    initial begin  a=4'b0110;b=4'b0101;  for(int i=0;i<8;i++)  begin  {s}=i;  #5;  end  end  initial begin  $monitor("simtime=%0t,a=%b,b=%b,s=%b,y=%b",$time,a,b,s,y);  end    initial begin  $dumpfile("dump.vcd");  $dumpvars(1,alu\_test);  end  endmodule | Design  module alu(a,b,s,y);  input[3:0] a,b;  input[2:0]s;  output[4:0] y;  reg [4:0]y;    always@(s,a,b)  if(!s[2])  begin  if(!s[1])  begin  if(!s[0])  y=a+b;//addition  else  y=a-b;//subtraction  end  else  begin  if(!s[0])  y=a\*b;//multipictaion  else  y=a/b;//division  end  end  else  begin  if(!s[1])  begin  if(!s[0])  y=a&b;//logical and  else  y=a|b;//logical or  end  else  begin  if(!s[0])  y=a^b;//logical xor  else  y=~(a^b);//logical xnor  end  end  endmodule |



Clock generation

|  |  |
| --- | --- |
| // 1st using always block  module clk\_gen1;  reg clk;  initial begin  clk=0;  #200 $finish;  end  always #5clk=~clk;  initial begin  $dumpfile("dump.vcd");  $dumpvars(1,clk\_gen1);  end    endmodule  //2nd using initial  module clk\_gen2;  reg clk;  initial begin  clk=0;  #100 $finish;  end  initial begin  forever #5clk=~clk;  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(1,clk\_gen2);  end  endmodule | //3rd using while  module clk\_gen3;  reg clk;  initial begin  clk=0;  #100 $finish;  end  initial begin  while(1)begin  #5clk=~clk;  end  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(1,clk\_gen3);  end  endmodule  //4th using forever loop  module clk\_gen4;  reg clk;  initial begin  clk=0;  #100 $finish;  end    initial begin  clk = 0;  forever #5 clk = ~clk;  end  initial begin  $dumpfile("dump.vcd");  $dumpvars(1,clk\_gen4);  end  endmodule |

