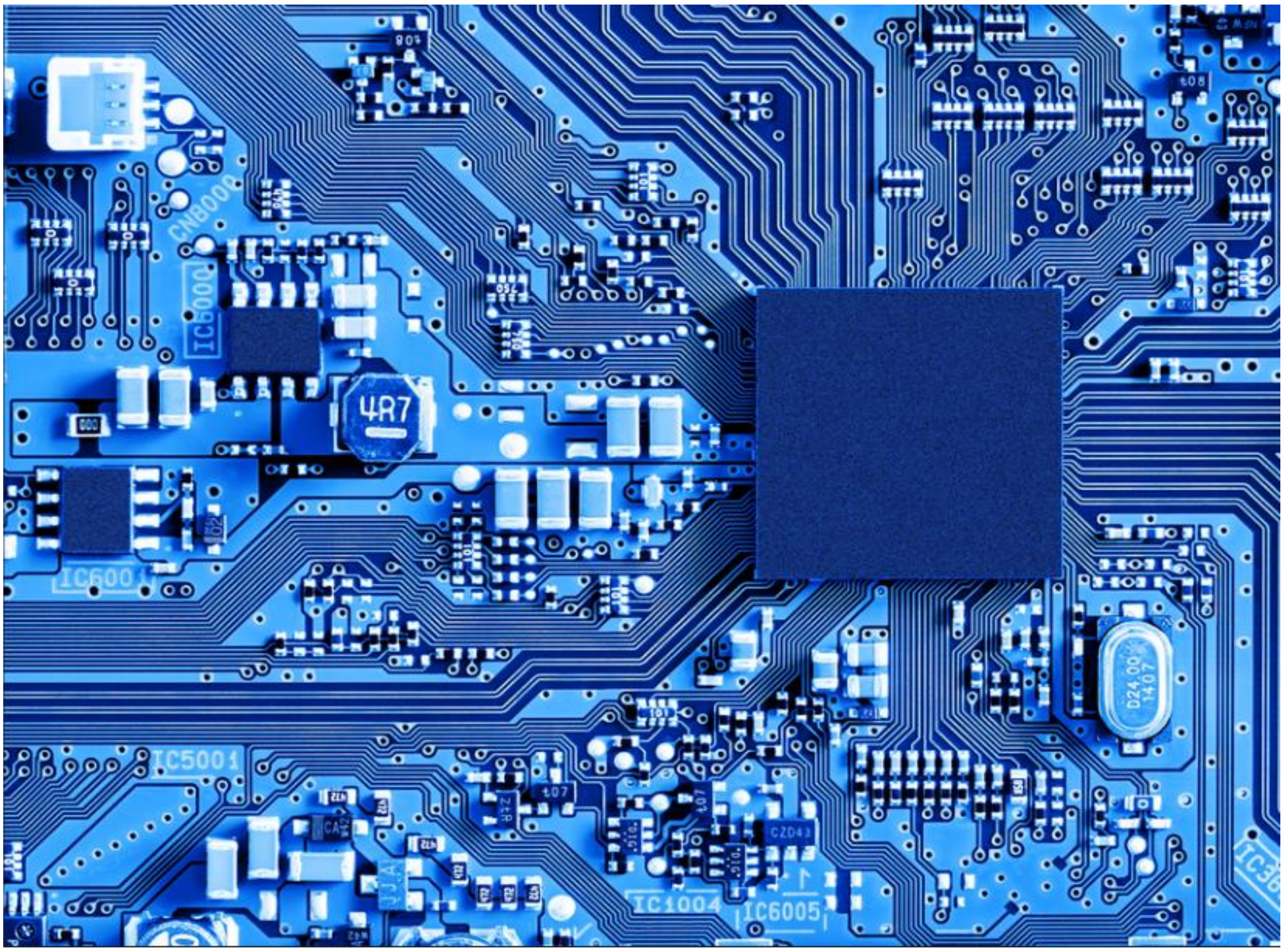
ITI

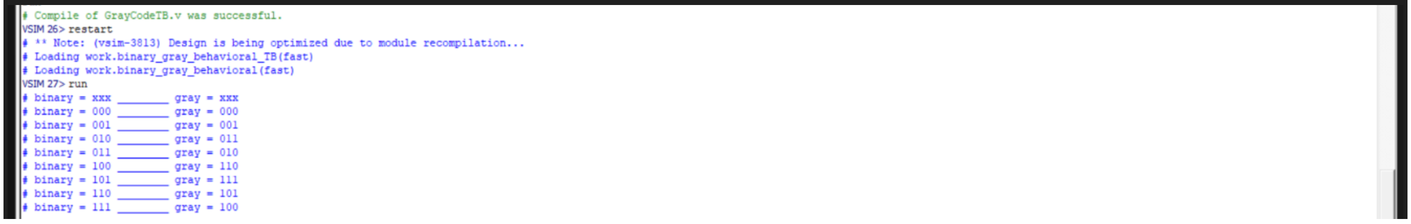
Lab Two
Verilog



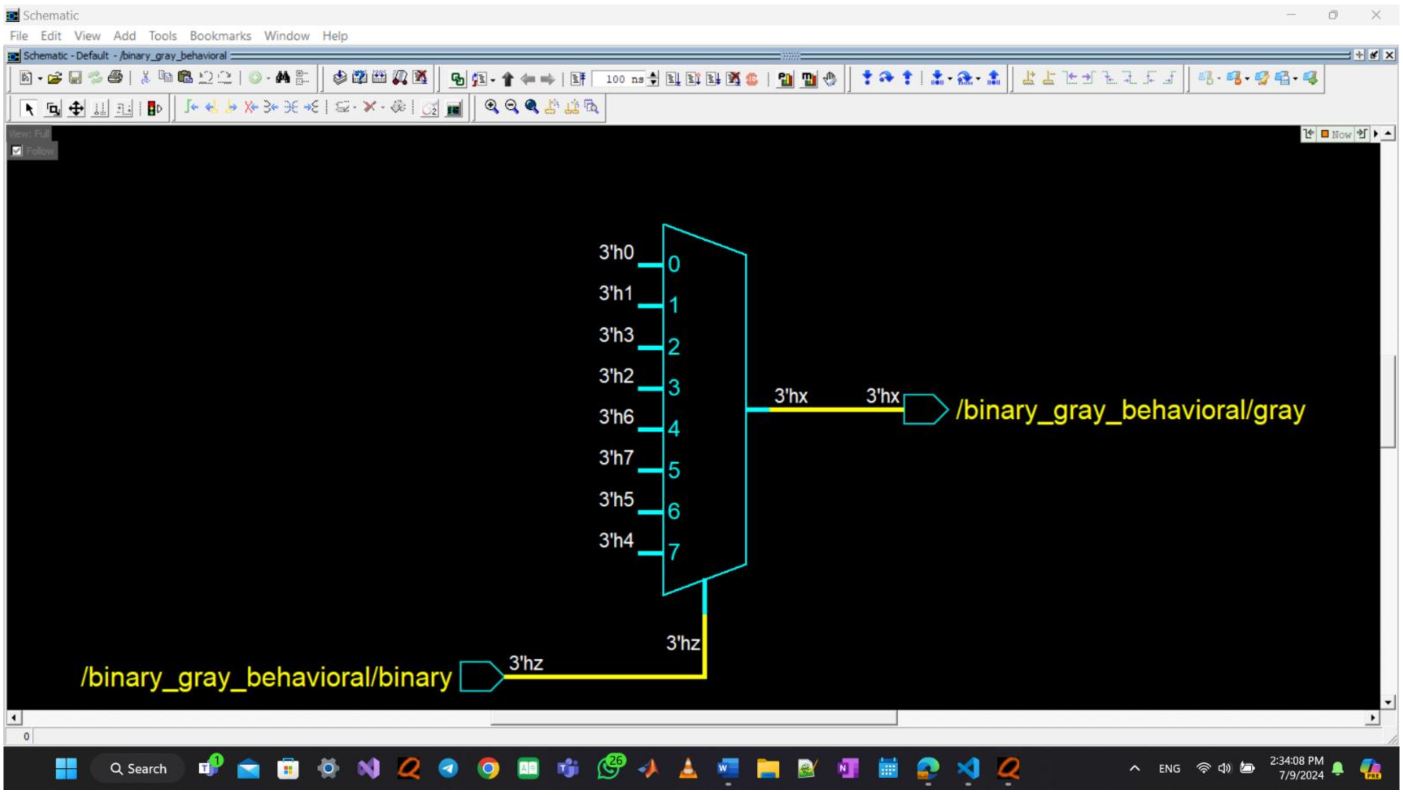
Waleed Emad Yahyaa Zakarya

BINARY TO GRAY

```verilog
module binary_gray_behavioral (binary,gray);

/*input output declaration of port list*/
input wire [2:0] binary;
output reg [2:0] gray;

/*behavioral code for describing the binary to gray code using case*/
always @(*) begin
    case (binary)
    3'b000: gray = 3'b000;
    3'b001: gray = 3'b001;
    3'b010: gray = 3'b011;
    3'b011: gray = 3'b010;
    3'b100: gray = 3'b110;
    3'b101: gray = 3'b111;
    3'b110: gray = 3'b101;
    3'b111: gray = 3'b100;
    endcase
end

endmodule

module binary_gray_behavioral_TB ( );

/*internal variables for using in test bench*/
reg [2:0] binaryTB;
wire [2:0] grayTB;
integer i = 0;

/*instantiaotion of main module*/
binary_gray_behavioral TBinstance(binaryTB,grayTB);

/*appling all possible stimulas*/
initial begin
    $monitor ("binary = %b ________ gray = %b",binaryTB,grayTB);
    for (i=0;i<8;i=i+1)
        #5 binaryTB = i;
end

endmodule
```



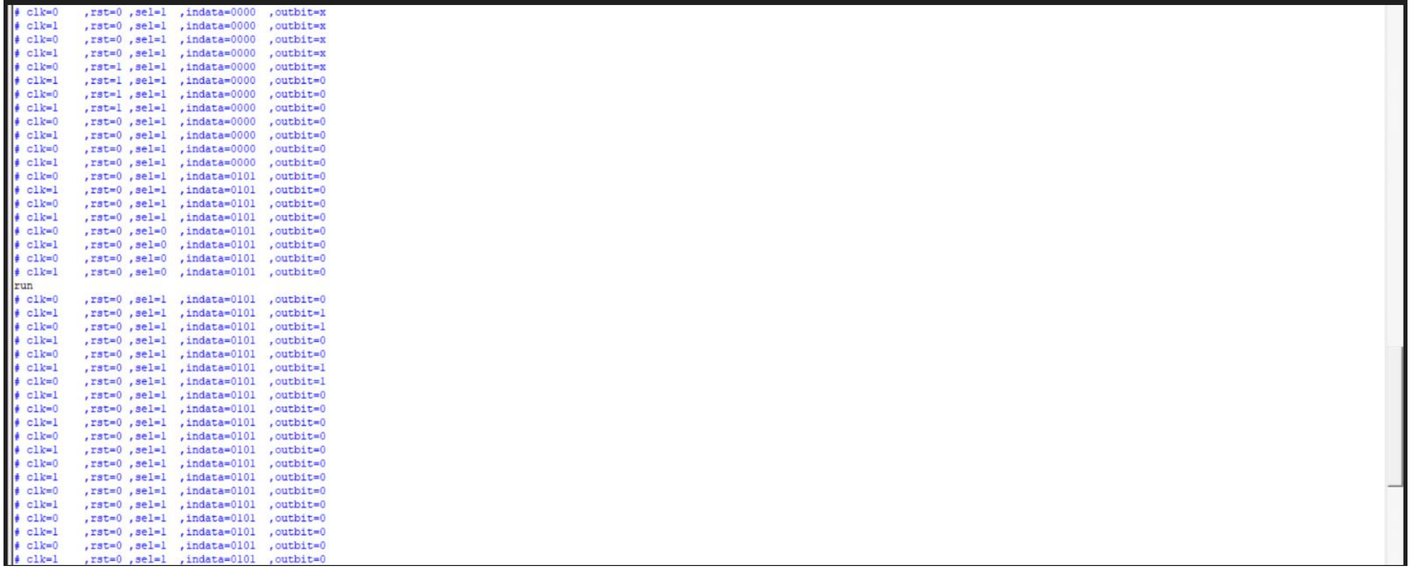
```
# Compile of GrayCodeTB.v was successful.
VSIM 26> restart
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading work.binary_gray_behavioral_TB(fast)
# Loading work.binary_gray_behavioral(fast)
VSIM 27> run
# binary = xxx ________ gray = xxx
# binary = 000 ________ gray = 000
# binary = 001 ________ gray = 001
# binary = 010 ________ gray = 011
# binary = 011 ________ gray = 010
# binary = 100 ________ gray = 110
# binary = 101 ________ gray = 111
# binary = 110 ________ gray = 101
# binary = 111 ________ gray = 100
```


Schematic
File Edit View Add Tools Bookmarks Window Help
Schematic - Default - /binary_gray_behavioral
3'h0
3'h1
3'h3
3'h2
3'h6
3'h7
3'h5
3'h4
0
1
2
3
4
5
6
7
3'hx
3'hx
/binary_gray_behavioral/gray
3'hz
3'hz
/binary_gray_behavioral/binary
2:34:08 PM
7/9/2024

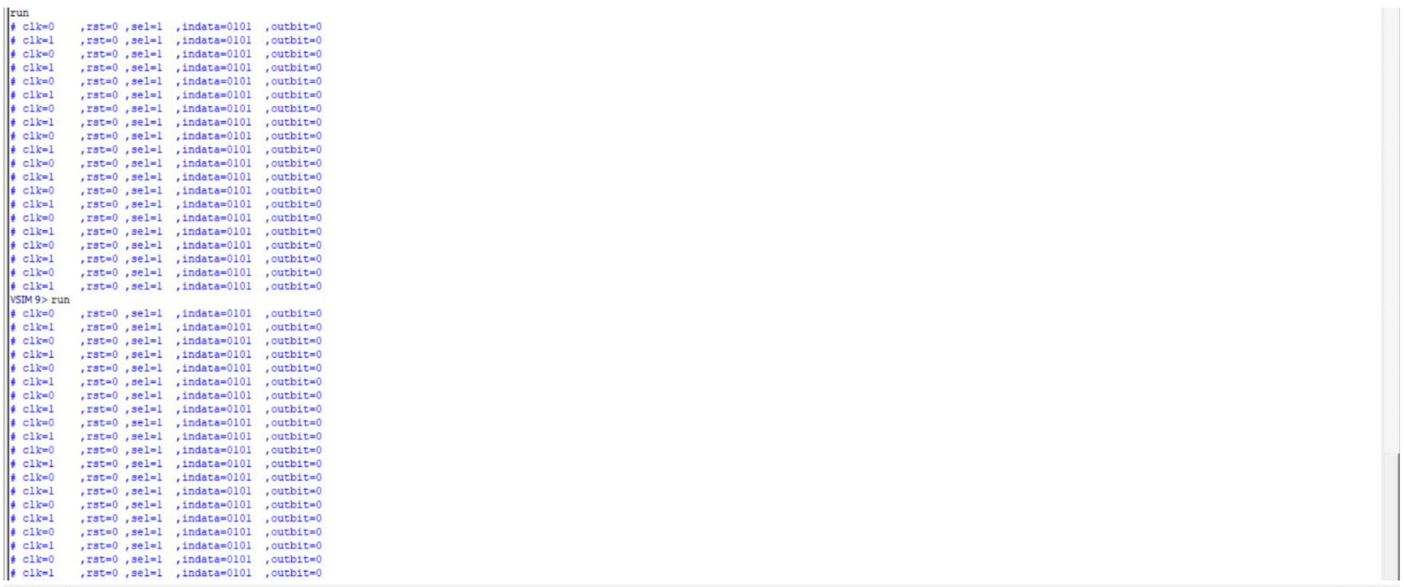
4-BIT PISO (PARALLEL TO SERIAL)

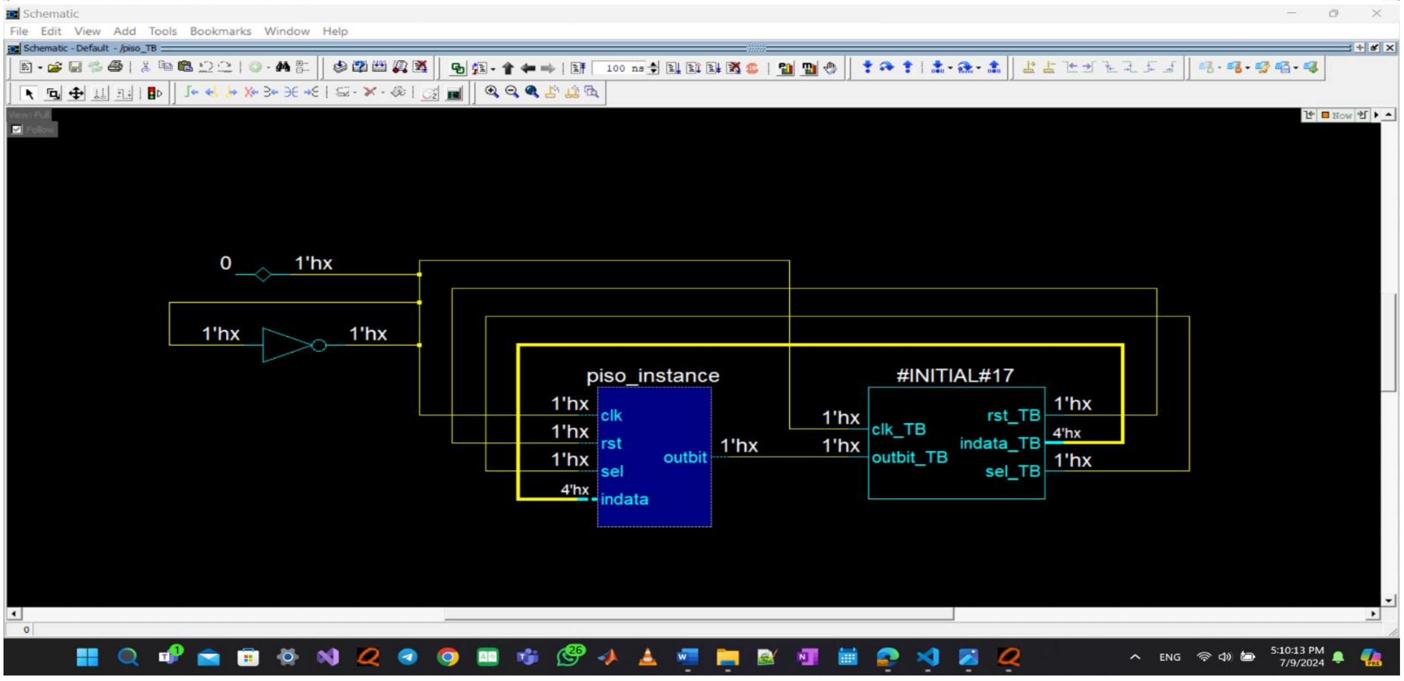
```
module piso (clk,rst,sel,indata,outbit);

input wire clk , rst , sel;
input wire [3:0] indata;
output reg outbit;

reg [3:0] internalreg;

always @(posedge clk )
begin
    if (rst)
    begin
        internalreg <= 4'b000;
        outbit <=0;
    end
    else if (sel == 1'b0)
    begin
        internalreg <= indata;
    end
    else if (sel == 1'b1)
    begin
        outbit <= internalreg[0];
        internalreg[3:0] <= {1'b0,{internalreg[3:1]}};
    end
    else
    begin
        outbit <= 1'b0;
    end
end

endmodule
```

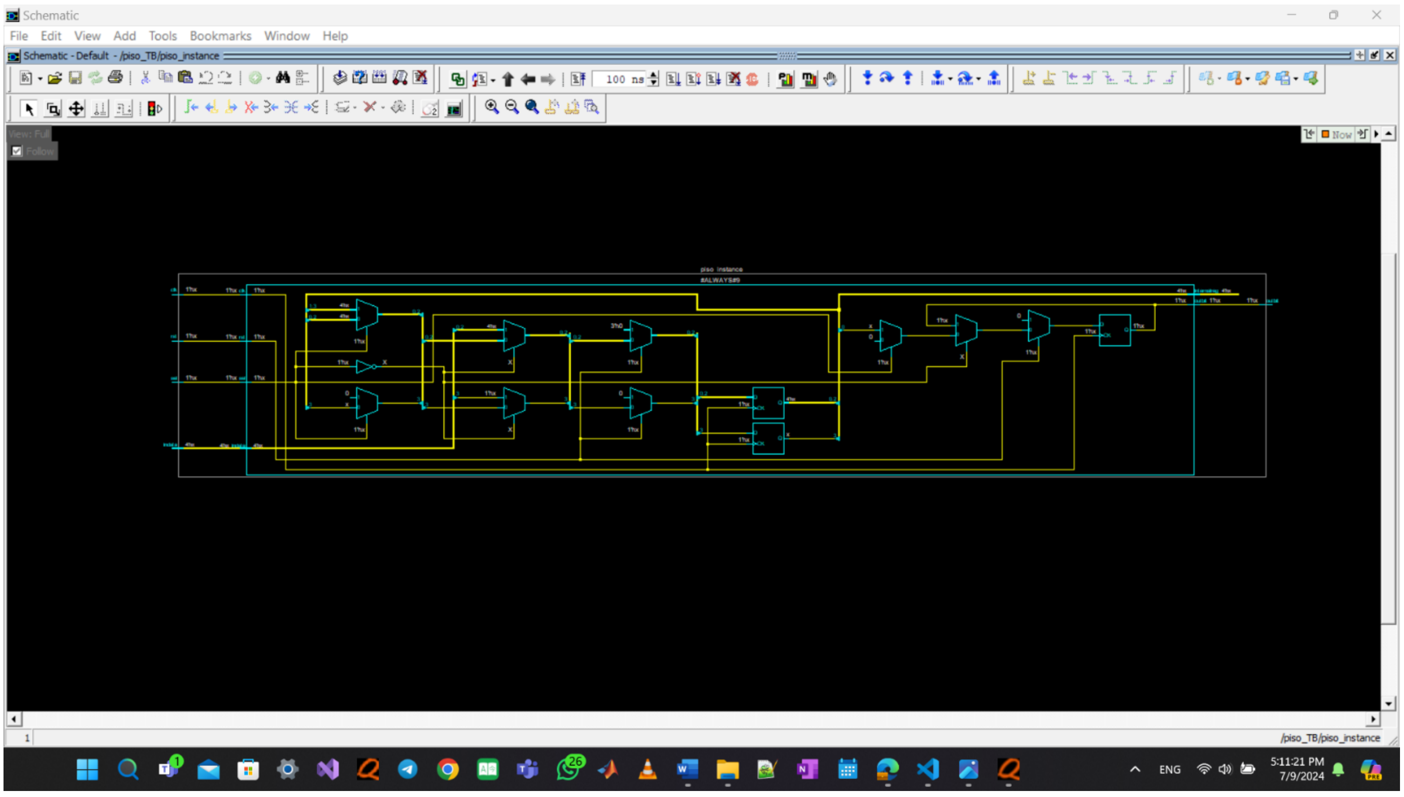
```verilog
module piso_TB ( );

reg clk_TB=0 , rst_TB , sel_TB;
reg [3:0] indata_TB;
wire outbit_TB;

piso piso_instance (clk_TB,rst_TB,sel_TB,indata_TB,outbit_TB);

/*initial
begin
    clk_TB = 0;
    forever #5 clk_TB = ~clk_TB;
end*/
always #5  clk_TB<= ~clk_TB;


initial
begin
    $monitor ("clk=%b    ,rst=%b
,sel=%b  ,indata=%b  ,outbit=%b",clk_TB,rst_TB,sel_TB,indata_TB,outbit_TB);
    rst_TB = 0;
    indata_TB =0;
    sel_TB = 1;
    #20 rst_TB = 1;
    #20 rst_TB = 0;
    #20 indata_TB = 4'b0101;
    #20 sel_TB = 0;
    #20 sel_TB = 1;

end

endmodule
```



```
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=0    ,rst=1 ,sel=1  ,indata=0000  ,outbit=x
# clk=1    ,rst=1 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=1 ,sel=1  ,indata=0000  ,outbit=0
# clk=1    ,rst=1 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
run
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
```

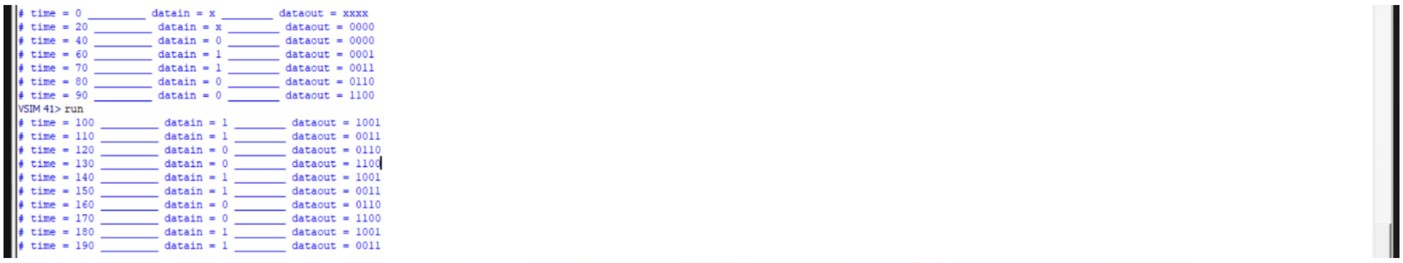

```
run
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
VSIM 9> run
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1     ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
```



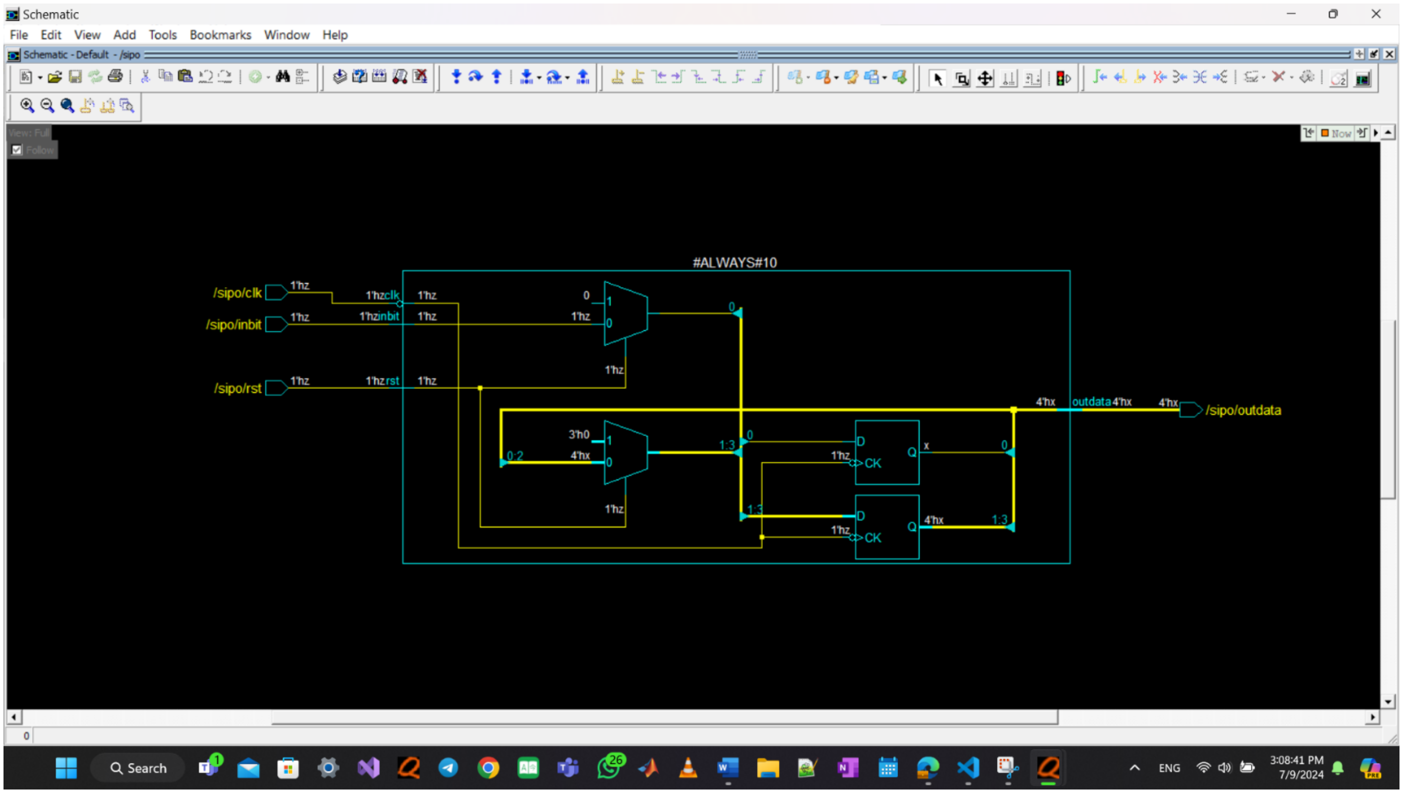

Schematic
File Edit View Add Tools Bookmarks Window Help
Schematic - Default - /piso_TB/piso_instance
piso_instance
#ALWAYS#9
/piso_TB/piso_instance
5:11:21 PM
7/9/2024

4-BIT SIPO (SERIAL TO PARALLEL)

```
module sipo (clk,rst,inbit,outdata);

parameter N = 4;

input wire clk , rst , inbit;
output reg [N-1:0] outdata;

integer i;

always @(negedge clk)
begin
    if (rst)
        begin
            outdata <= 0;
        end
    else
        begin
                outdata <= {inbit,{outdata[N-1:1]}};
        end

end


endmodule
module sipo_TB ( );

reg clk_TB , rst_TB , inbit_TB;
wire [3:0] outdata_TB;

integer i = 0;

sipo sipo_TB_instance (clk_TB,rst_TB,inbit_TB,outdata_TB);

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

initial begin
    $monitor ("time = %0t _________ datain = %b ________ dataout =
%b",$time,inbit_TB,outdata_TB);
    rst_TB = 0;
    #20 rst_TB = 1;
    #20 rst_TB = 0;
    inbit_TB = 0;
    for (i=0;i<8;i=i+1)
        #20 inbit_TB = ~inbit_TB;
end

endmodule
```



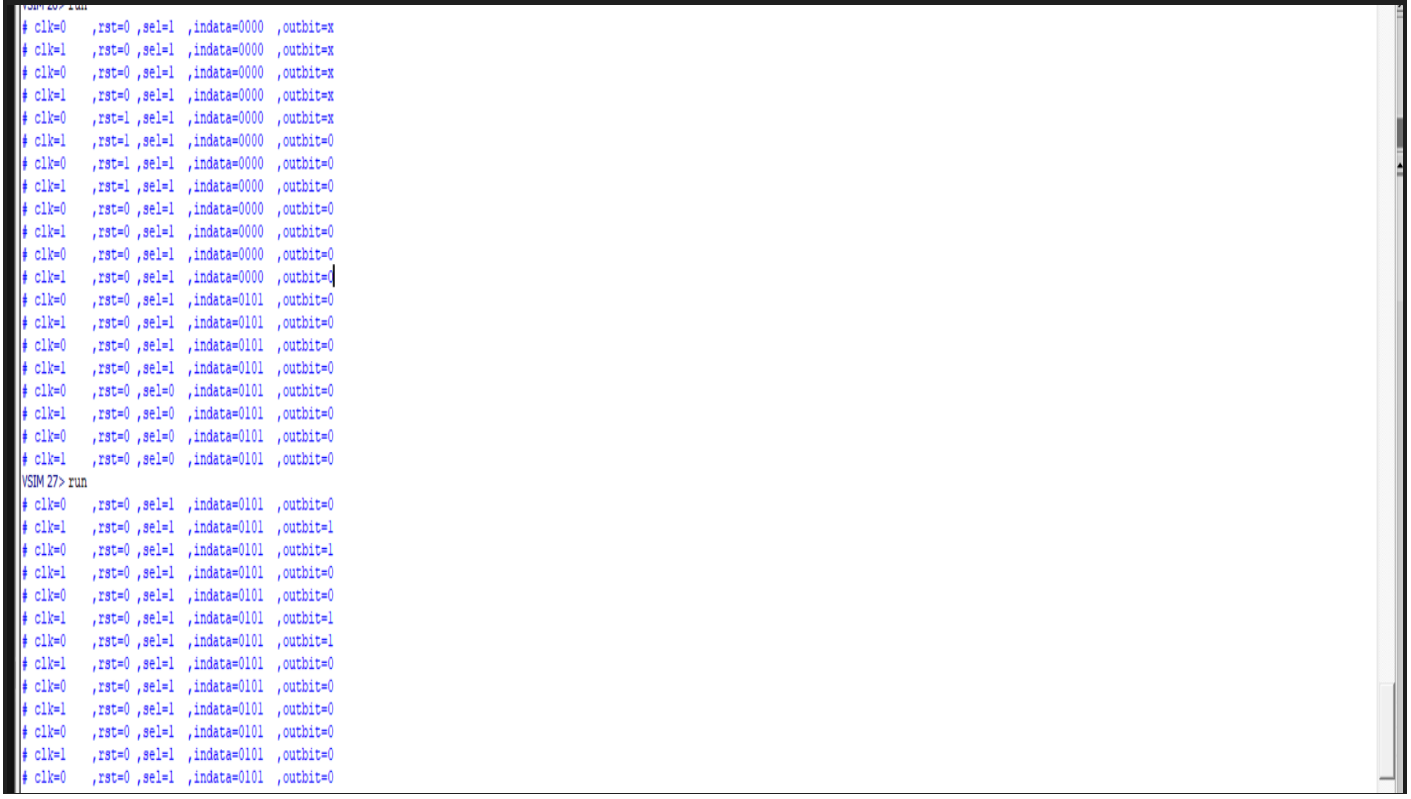
```
# time = 0 __________ datain = x _________ dataout = xxxx
# time = 20 __________ datain = x _________ dataout = 0000
# time = 40 __________ datain = 0 _________ dataout = 0000
# time = 60 __________ datain = 1 _________ dataout = 0001
# time = 70 __________ datain = 1 _________ dataout = 0011
# time = 80 __________ datain = 0 _________ dataout = 0110
# time = 90 __________ datain = 0 _________ dataout = 1100
VSIM 41> run
# time = 100 __________ datain = 1 _________ dataout = 1001
# time = 110 __________ datain = 1 _________ dataout = 0011
# time = 120 __________ datain = 0 _________ dataout = 0110
# time = 130 __________ datain = 0 _________ dataout = 1100
# time = 140 __________ datain = 1 _________ dataout = 1001
# time = 150 __________ datain = 1 _________ dataout = 0011
# time = 160 __________ datain = 0 _________ dataout = 0110
# time = 170 __________ datain = 0 _________ dataout = 1100
# time = 180 __________ datain = 1 _________ dataout = 1001
# time = 190 __________ datain = 1 _________ dataout = 0011
```



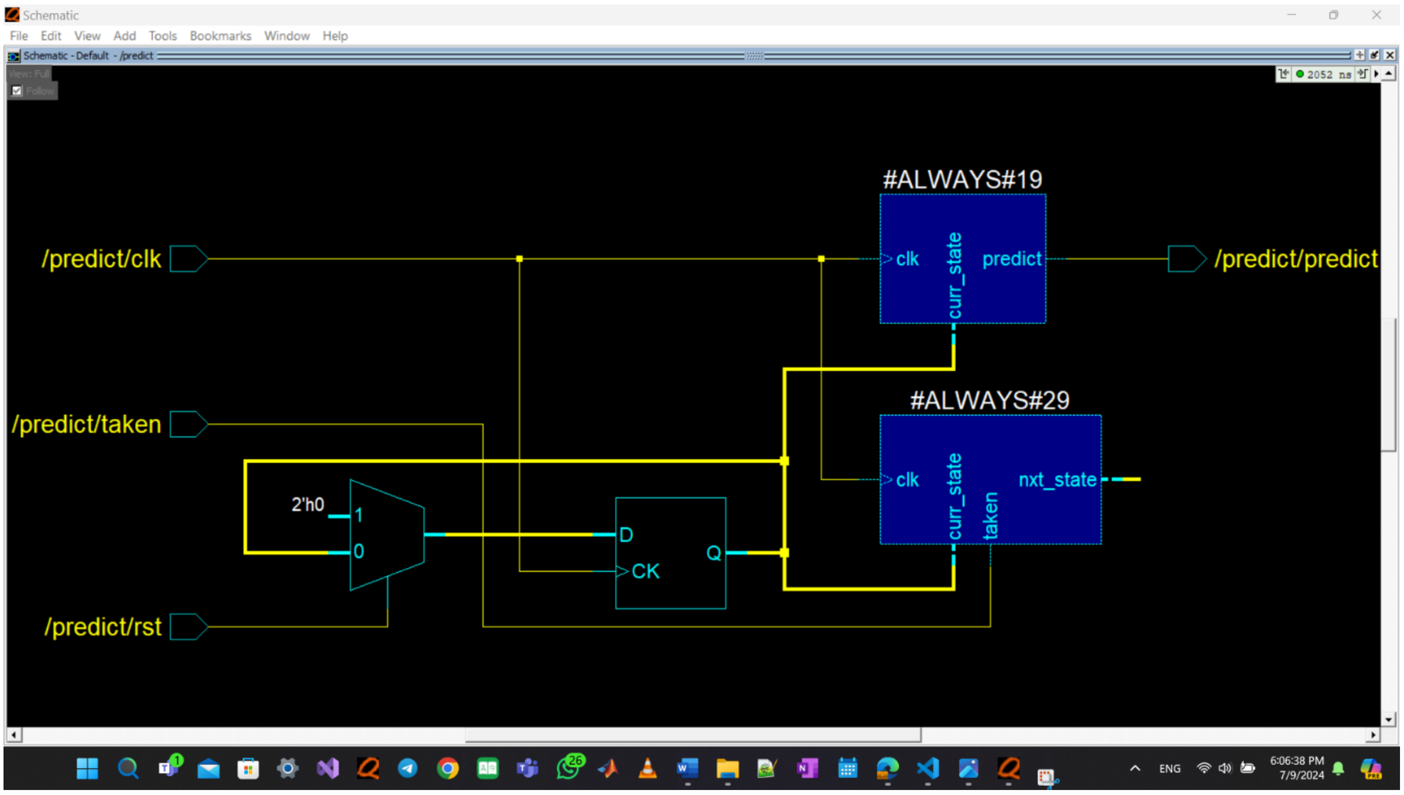
STATE MACHINE

```verilog
module predict (rst , clk , taken , predict);

input rst , clk , taken ;
output reg predict;

localparam A = 00,
           B = 01,
           C = 10,
           D = 11;

reg [1:0] curr_state , nxt_state;

always @(posedge clk)
begin
    if (rst)
    curr_state <= A;
end

always @(posedge clk)
begin
    case (curr_state)
    A: predict <= 1;
    B: predict <= 1;
    C: predict <= 0;
    D: predict <= 0;
    endcase
end

always @(posedge clk)
begin
    case (curr_state)
    A:
    begin
       if (taken == 0)
            nxt_state <= B;
       else if (taken == 1)
            nxt_state <= A;
       else
            nxt_state <= A;
    end
    B:
    begin
       if (taken == 0)
            nxt_state <= C;
       else if (taken == 1)
            nxt_state <= A;
       else
            nxt_state <= A;
    end
    C:
    begin
       if (taken == 0)
            nxt_state <= C;
       else if (taken == 1)
            nxt_state <= D;
       else
            nxt_state <= A;
    end
    D:
    begin
       if (taken == 0)
            nxt_state <= C;
       else if (taken == 1)
            nxt_state <= A;
       else
            nxt_state <= A;
    end
    endcase
end


endmodule
module predict_TB ();

reg rst_TB , clk_TB = 1'b0 , taken_TB ;
wire predict_TB ;

predict predict_instance (rst_TB , clk_TB , taken_TB , predict_TB);
```

```
always #5 clk_TB = ~clk_TB;

initial begin

$monitor("rst_TB = %b , clk_TB = %b , taken_TB = %b , predict_TB = %b",rst_TB , clk_TB ,
taken_TB , predict_TB);

rst_TB=0; taken_TB=0;
#20 rst_TB = 1;
#20 rst_TB = 0;
#20 rst_TB = 1;

#2 taken_TB=1;
#10 taken_TB=1;
#10 taken_TB=0;
#10 taken_TB=1;
#10 taken_TB=0;
#10 taken_TB=0;
#10 taken_TB=1;
#10 taken_TB=1;
#10 taken_TB=1;
#10 taken_TB=0;
#10 taken_TB=0;
#10 taken_TB=1;
#10 taken_TB=1;

end




endmodule
```



```
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=x
# clk=0    ,rst=1 ,sel=1  ,indata=0000  ,outbit=x
# clk=1    ,rst=1 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=1 ,sel=1  ,indata=0000  ,outbit=0
# clk=1    ,rst=1 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0000  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=0  ,indata=0101  ,outbit=0
VSIM 27> run
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=1
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=1    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
# clk=0    ,rst=0 ,sel=1  ,indata=0101  ,outbit=0
```

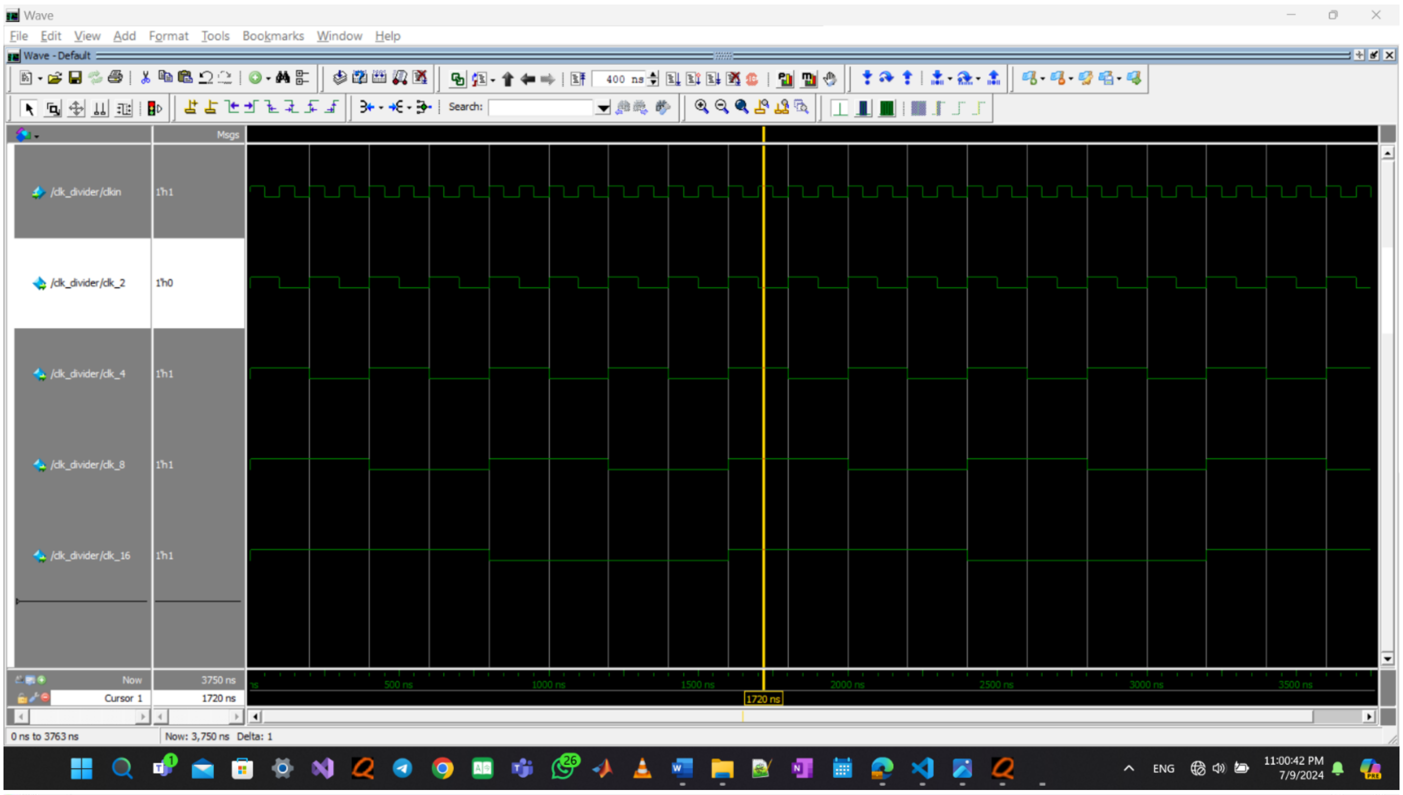


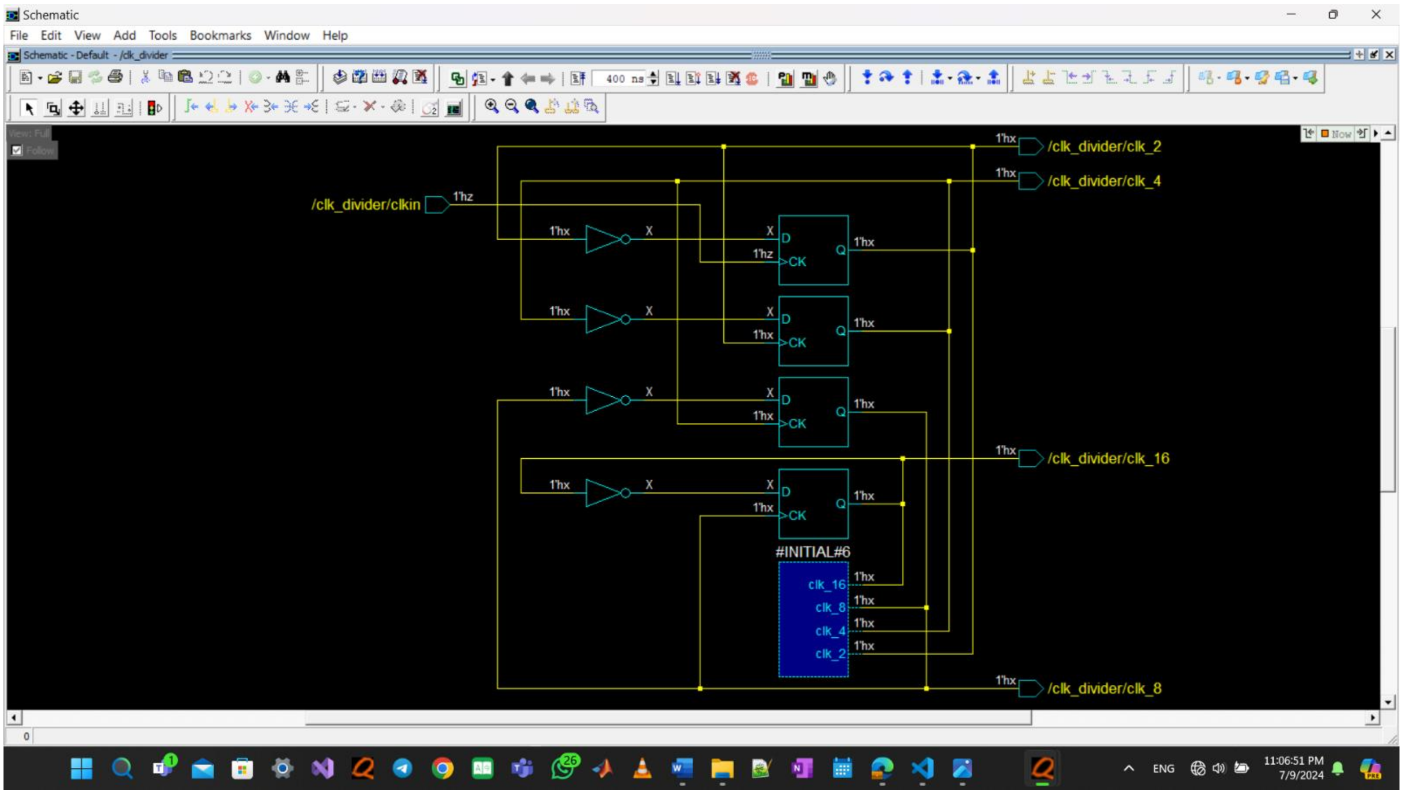
SEQUENCE DETECTOR

```
module seq_dete_method1 (
    input rst,clk,datain,
    output reg dataout );

localparam A = 0,
           B = 1,
           C = 2,
           D = 3,
           E = 4,
           F = 5,
           G = 6,
           H = 7;

reg [2:0] curr_state , nxt_state;

always @ (posedge clk)
begin
    if (rst)
        curr_state <= A;
    else
        curr_state<=nxt_state;
end

always @ (curr_state,datain)
begin
    case (curr_state)

        A:dataout <= 0;
        B:dataout <= 0;
        C:dataout <= 0;
        D:dataout <= 0;
        E:dataout <= 0;
        F:dataout <= 0;
        G:dataout <= 0;
        H: if (datain == 1'b0)
           dataout <= 0;
           else if (datain == 1'b1)
           dataout <= 1;
           else
           dataout <= 0;
    endcase
end

always @ (datain,curr_state)
begin
    case (curr_state)
        A: if (datain == 0)
           nxt_state <= A;
           else if (datain == 1)
           nxt_state <= B;
           else
           nxt_state <= A;
        B:if (datain == 0)
           nxt_state <= C;
           else if (datain == 1)
           nxt_state <= B;
           else
           nxt_state <= A;
        C:if (datain == 0)
           nxt_state <= A;
           else if (datain == 1)
           nxt_state <= D;
           else
           nxt_state <= A;
        D:if (datain == 0)
           nxt_state <= C;
           else if (datain == 1)
           nxt_state <= E;
           else
           nxt_state <= A;
        E:if (datain == 0)
           nxt_state <= F;
           else if (datain == 1)
           nxt_state <= B;
           else
           nxt_state <= A;
        F:if (datain == 0)
           nxt_state <= A;
           else if (datain == 1)
           nxt_state <= G;
           else
           nxt_state <= A;
        G:if (datain == 0)
           nxt_state <= H;
           else if (datain == 1)
           nxt_state <= E;
           else
           nxt_state <= A;
        H: if (datain == 0)
           nxt_state <= A;
           else if (datain == 1)
           nxt_state <= D;
           else
           nxt_state <= A;
    endcase
    curr_state <= nxt_state;

end

endmodule
```

CLOCK DIVIDER BY (2, 4, 8, 16)

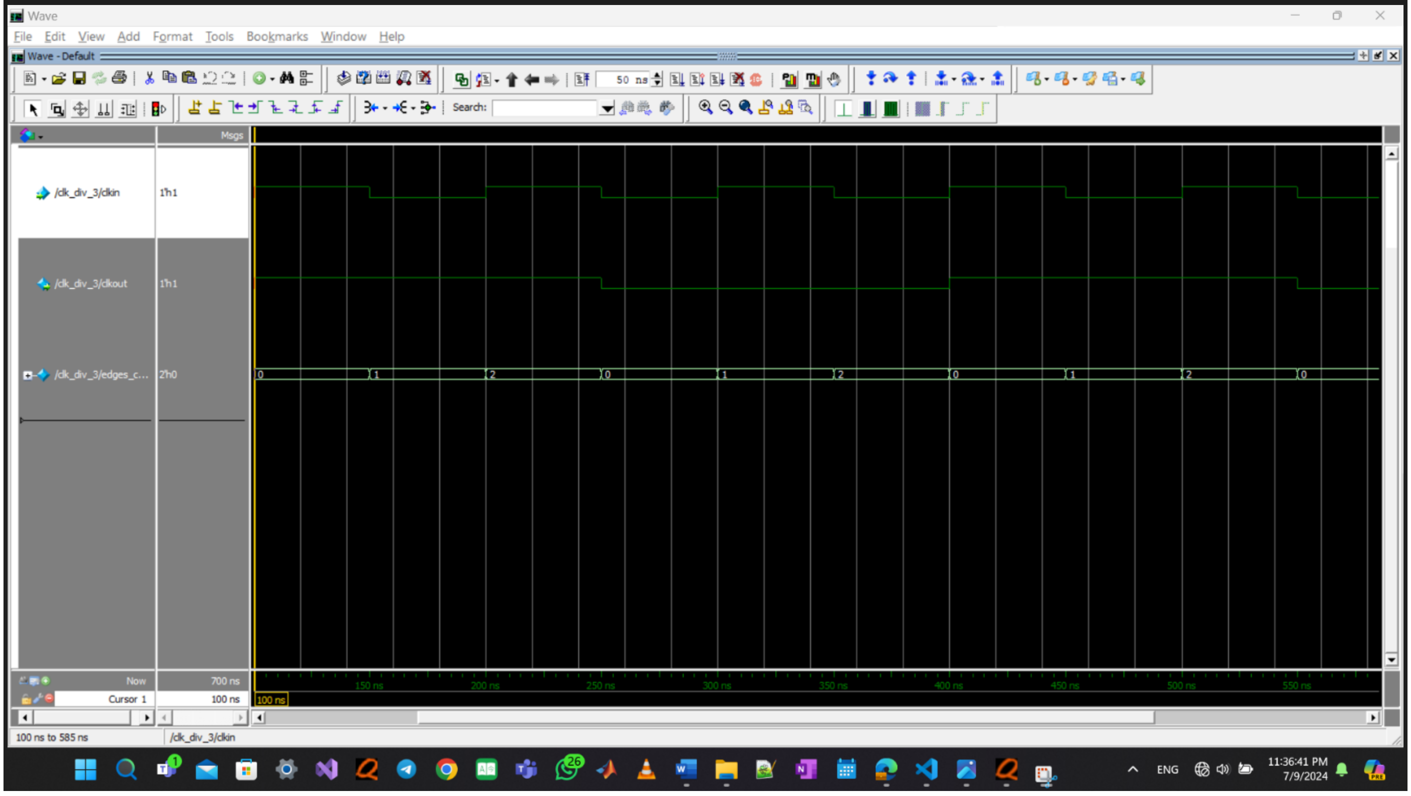
```
module clk_divider (clkin , clk_2 , clk_4 , clk_8 , clk_16);

input clkin;
output reg clk_2 , clk_4 , clk_8 , clk_16;

initial
begin
    clk_2 = 0;
    clk_4 = 0;
    clk_8 = 0;
    clk_16 = 0;
end

always @ (posedge clkin)
clk_2 <= ~clk_2;

always @ (posedge clk_2)
clk_4 <= ~clk_4;

always @ (posedge clk_4)
clk_8 <= ~clk_8;

always @ (posedge clk_8)
clk_16 <= ~clk_16;


endmodule
```





CLOCK DIVIDER BY 3

```
module clk_div_3 (clkin , clkout);

input clkin;
output reg clkout;

reg[1:0] edges_counts ;

initial
begin
    edges_counts = 0;
    clkout =0;
end

always @(clkin)
    edges_counts <= edges_counts+1;

always @(*)
    begin
        if(edges_counts == 2'b11)
        begin
        clkout <= ~clkout;
        edges_counts = 0;
        end
    end




endmodule
```

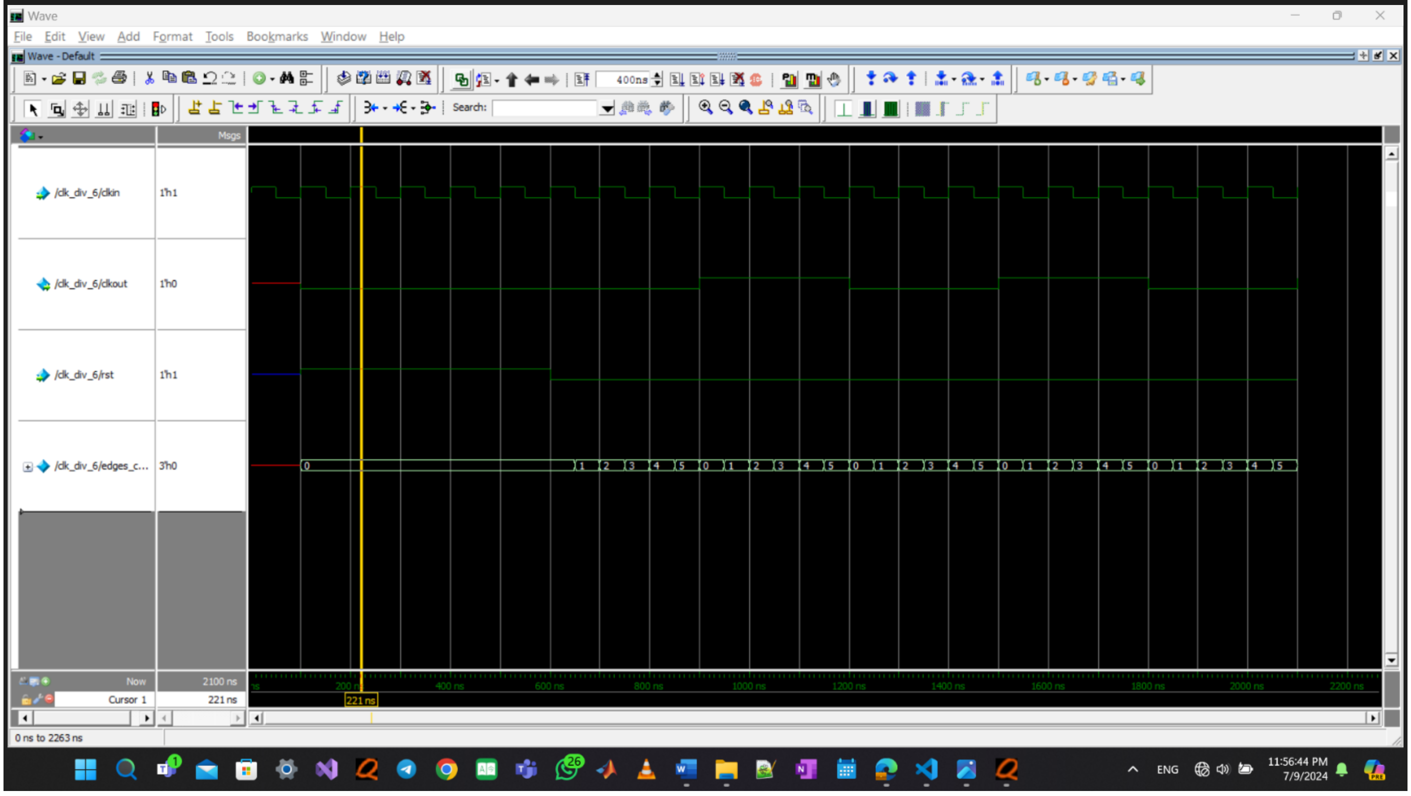


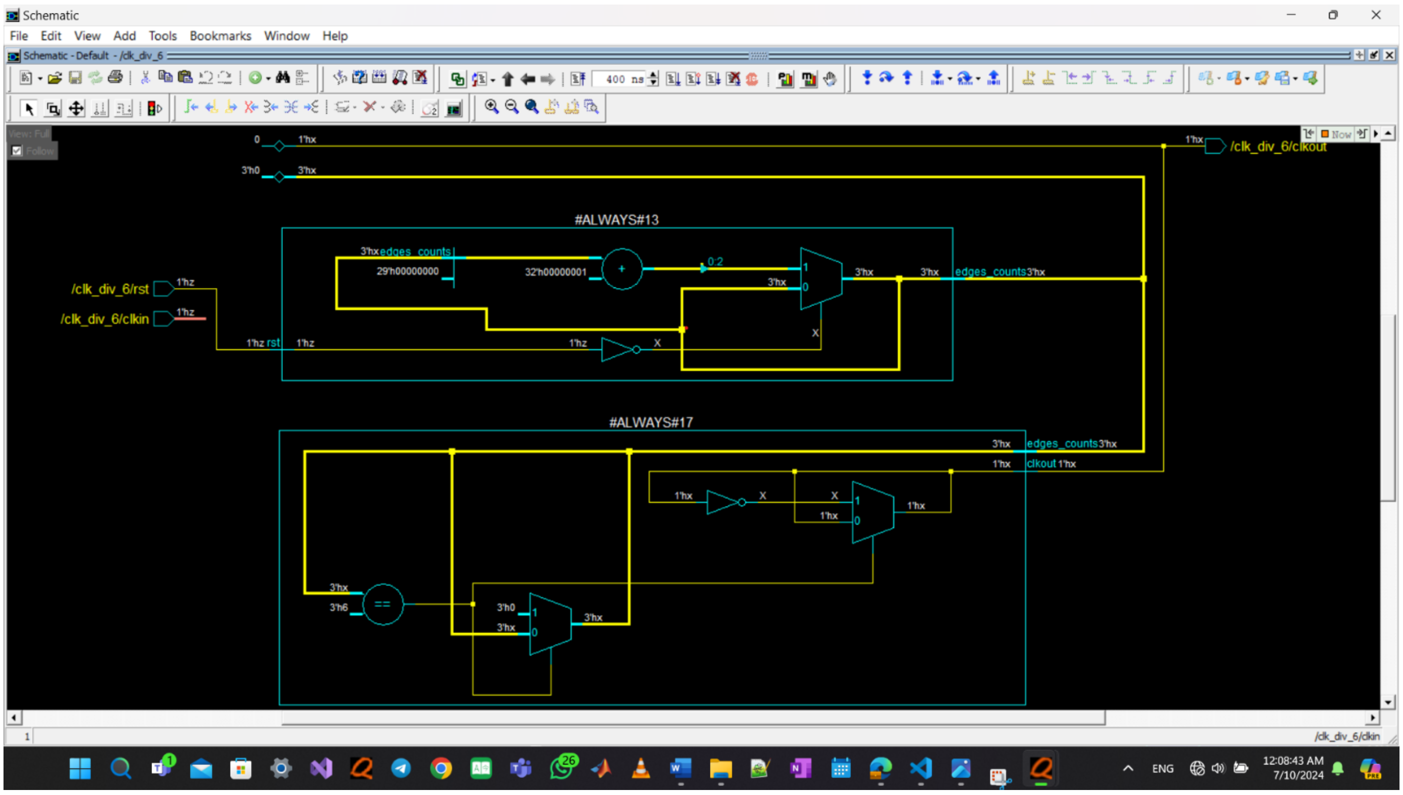
```
/clk_div_3/clkin        1'h1
/clk_div_3/clkout       1'h1
/clk_div_3/edges_c...   2'h0
Now        700 ns
Cursor 1   100 ns    100 ns
100 ns to 585 ns    /clk_div_3/clkin
```

CLOCK DIVIDER BY 6

```
module clk_div_6 (rst , clkin , clkout);

input clkin,rst;
output reg clkout;

reg[2:0] edges_counts ;

always @(rst)
begin
    clkout = 0;
    edges_counts = 0;
end
```

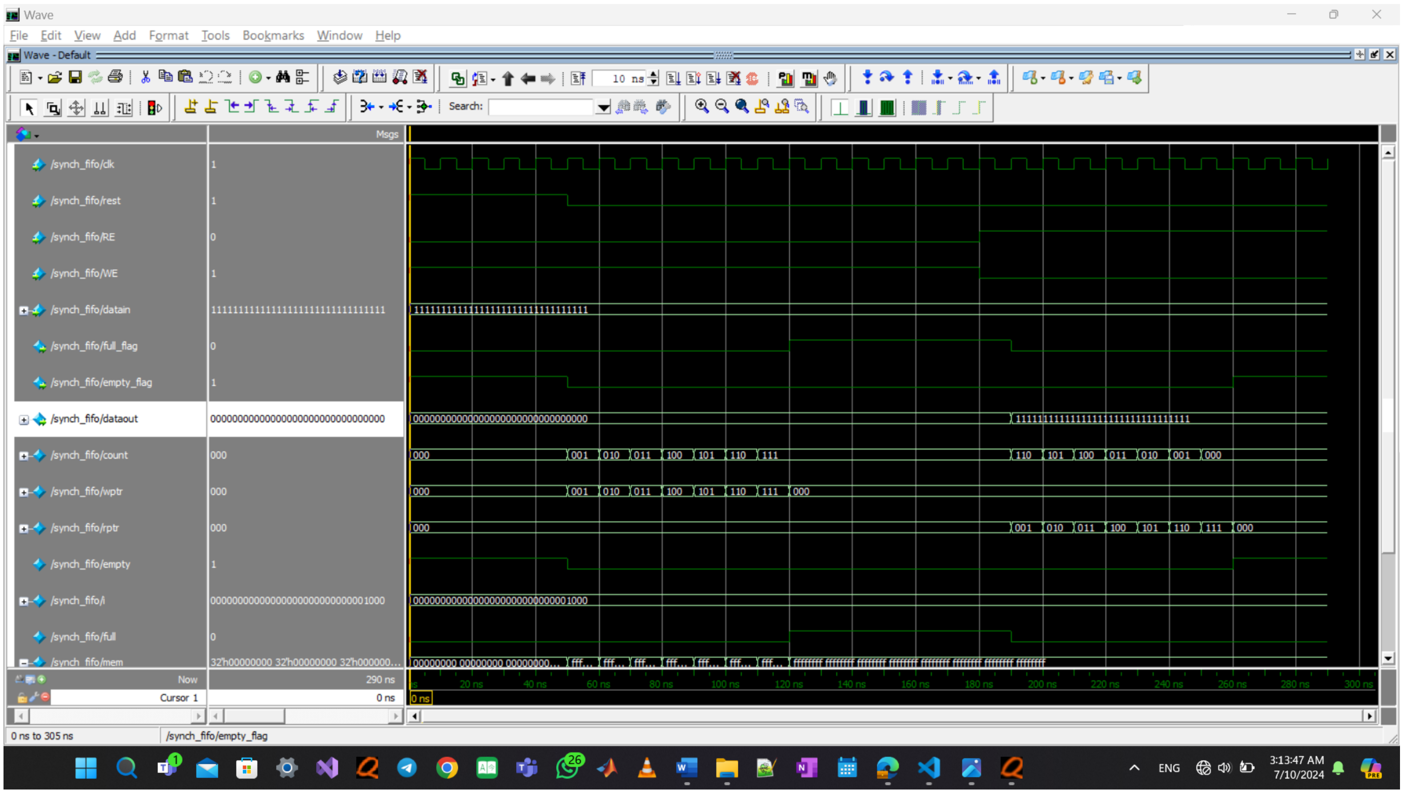
```verilog
always @(clkin)
    if(~rst)
    edges_counts <= edges_counts+1;

always @(*)
    begin
        if(edges_counts == 3'b110)
        begin
        clkout = ~clkout;
        edges_counts = 0;
        end
    end



endmodule
```





FIFO SYNCH

```
module synch_fifo (
    rest , clk , RE , WE , datain ,
    full_flag , empty_flag , dataout
);

input rest , clk , RE , WE ;
input [31:0] datain;
output reg full_flag , empty_flag;
output reg [31:0] dataout;

reg [31:0] mem [0:7];
reg [2:0] count;
reg [2:0] wptr;
reg [2:0] rptr;
reg empty;
reg full;

integer i;

always @(posedge clk , rest) begin
    if (rest)
    begin
        full_flag <= 1'b0;
        empty_flag <= 1'b1;
        dataout <= 'b0;
        count <= 'b0;
        empty <= 'b1;
        full <= 'b0;
        wptr <= 'b0;
        rptr <= 'b0;
        for (i=0;i<8;i=i+1)     mem[i] <= 'b0;
    end
    else
    begin
        if (WE && (count<8) && !(full))
        begin
            mem[wptr] <= datain;
            wptr <= wptr + 1;
            count <= count + 1;
            if(count == 3'b111)
            begin
                full_flag <= 1'b1;
                empty_flag <= 1'b0;
                empty <= 'b0;
                full <= 'b1;
            end
            else
            begin
                full_flag <= 1'b0;
                empty_flag <= 1'b0;
                empty <= 'b0;
                full <= 'b0;
            end
        end
        else if (RE && (count>0) && !(empty))
        begin
            dataout <= mem[rptr];
            rptr <= rptr + 1;
            count <= count - 1;
            if(count == 'b0)
            begin
                full_flag <= 1'b0;
                empty_flag <= 1'b1;
                empty <= 'b1;
                full <= 'b0;
            end
            else
            begin
                full_flag <= 1'b0;
                empty_flag <= 1'b0;
                empty <= 'b0;
                full <= 'b0;
            end
        end
        else
        begin
            full_flag <= 1'b0;
            empty_flag <= 1'b1;
            dataout <= 'b0;
            count <= 'b0;
            empty <= 'b1;
            full <= 'b0;
            wptr <= 'b0;
            rptr <= 'b0;
            for (i=0;i<8;i=i+1)     mem[i] <= 'b0;
        end
    end


end


endmodule
```



```
/synch_fifo/clk          1
/synch_fifo/rest         1
/synch_fifo/RE           0
/synch_fifo/WE           1
/synch_fifo/datain       11111111111111111111111111111111
/synch_fifo/full_flag    0
/synch_fifo/empty_flag   1
/synch_fifo/dataout      00000000000000000000000000000000
/synch_fifo/count        000
/synch_fifo/wptr         000
/synch_fifo/rptr         000
/synch_fifo/empty        1
/synch_fifo/i            00000000000000000000000000001000
/synch_fifo/full         0
/synch_fifo/mem          32'h00000000 32'h00000000 32'h000000...
Now                      290 ns
Cursor 1                 0 ns
```

```
count: 000 001 010 011 100 101 110 111 ... 110 101 100 011 010 001 000
wptr:  000 001 010 011 100 101 110 111 000
rptr:  000 ... 001 010 011 100 101 110 111 000
dataout: 00000000000000000000000000000000 ... 11111111111111111111111111111111
```

0 ns to 305 ns   /synch_fifo/empty_flag