TASK-1 Design and Implementation of Digital Circuits in Verilog

VLSI Internship

Submitted By:
Aareen Kadam

To the
Hardware Technology Group

Centre for Development of Advanced Computing
Centre In North East
(CDAC-CINE)

Research Park, IIT Guwahati,
Guwahati, Assam

June, 2025

Contents

1 Task 1

1.1 Question1 : Logic Gates and Modelling

Question 1 is about desgining and simulating basic gates using gate-level ,dataflow and behavioral modelling .I have defined different outputs for all the gates for all the designing styles in my code and created a testbench which verifies all the inputs for the gates.

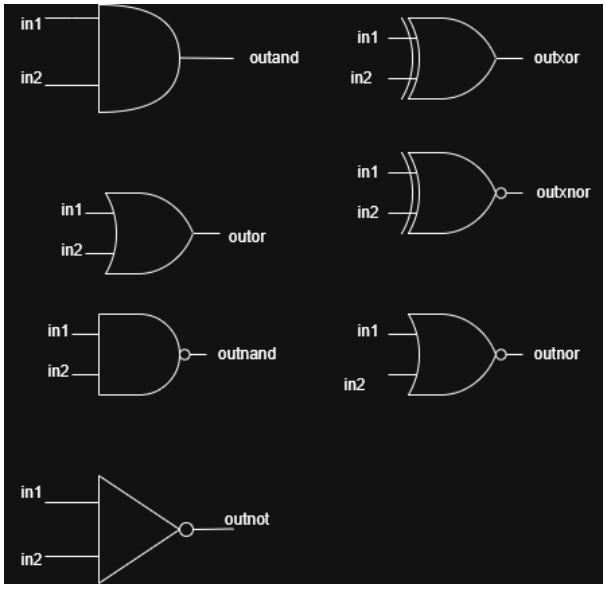
```
1  `timescale 1ns / 1ps
2
3  //////////////////////////////////////////////////////////////////////////////////
4  // Company:
5  // Engineer:
6  //
7  // Create Date:   12:12:07 06/11/2025
8  // Design Name:   ques1
9  // Module Name:   /home/aareen/CDAC/ques1test.v
10 // Project Name:  CDAC
11 // Target Device:
12 // Tool versions:
13 // Description:
14 //
15 // Verilog Test Fixture created by ISE for module: ques1
16 //
17 // Dependencies:
18 //
19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 //////////////////////////////////////////////////////////////////////////////////
24
25 module ques1test;
26
27         // Outputs
28         wire outand,outnot1,outnot2,outor,outnand,outnor,outxor,outxnor,
29         outand1,outor1,not1in1,not1in2,outnand1,outnor1,outxor1,outxnor1,
30         outand2,outor2,not2in1,not2in2,outnand2,outnor2,outxor2,outxnor2;
31         reg in1_tb,in2_tb;
32         // Instantiate the Unit Under Test (UUT)
33         ques1 uut (
34                 .in1(in1_tb),.in2(in2_tb),.outand(outand),.outor(outor),.notin1(outnot1),.no
35                 .outand1(outand1),.outand2(outand2),.outor1(outor1),.outor2(outor2),.outnand
36         );
37         initial
38         begin
39                 // Initialize Inputs
40
41                 // Wait 100 ns for global reset to finish
42                 #100;
43
44                 // Add stimulus here
45                 #5in1_tb<=0;in2_tb<=0;
46                 #5in1_tb<=1;in2_tb<=0;
47                 #5in1_tb<=0;in2_tb<=1;
48                 #5in1_tb<=1;in2_tb<=1;
49                 #5 $finish;
50         end
```

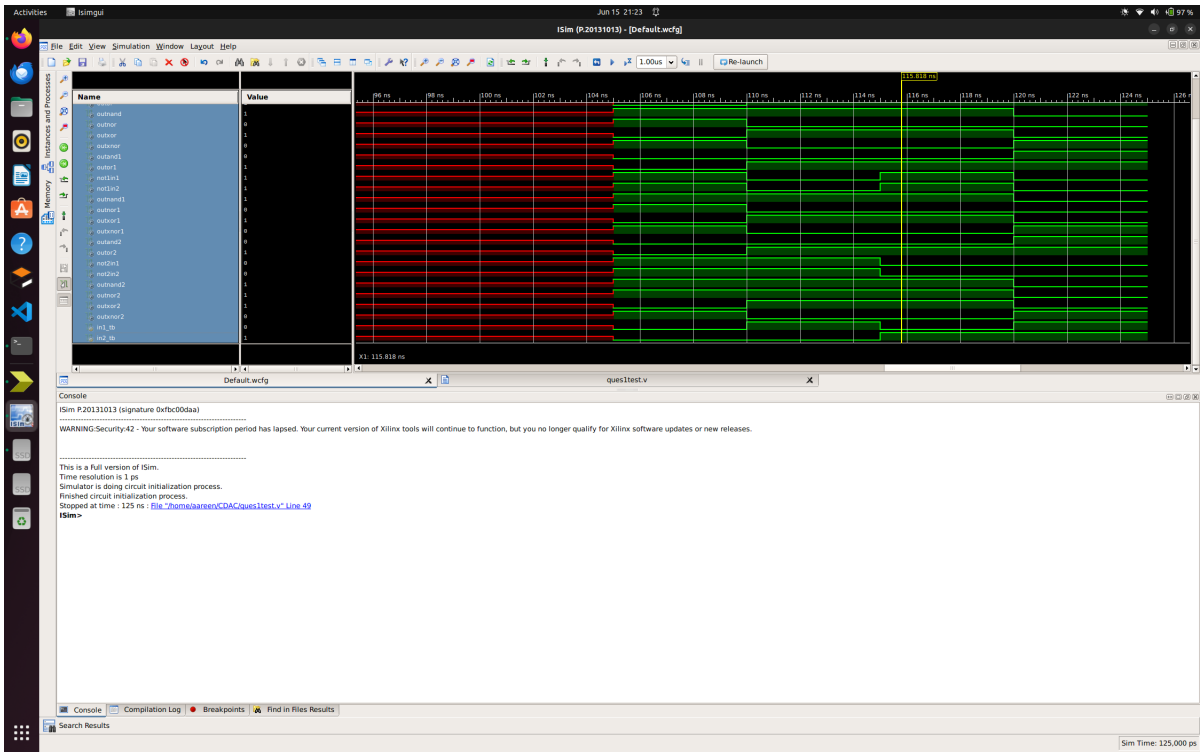
```
51
52 endmodule
```

The following is the TestBench for my code

```
1  `timescale 1ns / 1ps
2
3  ////////////////////////////////////////////////////////////////////////////////
4  // Company:
5  // Engineer:
6  //
7  // Create Date:   12:12:07 06/11/2025
8  // Design Name:   ques1
9  // Module Name:   /home/aareen/CDAC/ques1test.v
10 // Project Name:  CDAC
11 // Target Device:
12 // Tool versions:
13 // Description:
14 //
15 // Verilog Test Fixture created by ISE for module: ques1
16 //
17 // Dependencies:
18 //
19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 ////////////////////////////////////////////////////////////////////////////////
24
25 module ques1test;
26
27         // Outputs
28         wire outand,outnot1,outnot2,outor,outnand,outnor,outxor,outxnor,
29         outand1,outor1,not1in1,not1in2,outnand1,outnor1,outxor1,outxnor1,
30         outand2,outor2,not2in1,not2in2,outnand2,outnor2,outxor2,outxnor2;
31         reg in1_tb,in2_tb;
32         // Instantiate the Unit Under Test (UUT)
33         ques1 uut (
34                 .in1(in1_tb),.in2(in2_tb),.outand(outand),.outor(outor),.notin1(outnot1),.no
35                 .outand1(outand1),.outand2(outand2),.outor1(outor1),.outor2(outor2),.outnand
36         );
37         initial
38         begin
39                 // Initialize Inputs
40
41                 // Wait 100 ns for global reset to finish
42                 #100;
43
44                 // Add stimulus here
45                 #5in1_tb<=0;in2_tb<=0;
46                 #5in1_tb<=1;in2_tb<=0;
47                 #5in1_tb<=0;in2_tb<=1;
48                 #5in1_tb<=1;in2_tb<=1;
49                 #5 $finish;
50         end
51
52 endmodule
```

. These are the block diagrams for my circuit followed by the timing diagrams for it.


in1
in2
outand
in1
in2
outxor
in1
in2
outor
in1
in2
outxnor
in1
in2
outnand
in1
in2
outnor
in1
in2
outnot


ISim (P.20131013) - [Default.wcfg]
Default.wcfg
ques1test.v
Console
ISim P.20131013 (signature 0xfbc00daa)
WARNING:Security:42 - Your software subscription period has lapsed. Your current version of Xilinx tools will continue to function, but you no longer qualify for Xilinx software updates or new releases.
This is a full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 125 ns : File "/home/aareen/CDAC/ques1test.v" Line 49
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results

1.2 Question2 : MUX

Question 2. In this question we are asked to design a 2:1 MUX and then using that module we are supposed to design 4:1 ,7:1 and 13:1 MUX. The ques2.v file has the 2:1 mux module called 'mux' and the 4:1 mux called 'ques2' . 2:1 mux has been designed using dataflow modeling ,and the 4:1 mux has been designed using 3 modules of a 2:1 mux ,i.e. using structural modeling. 7:1 mux and 13:1 mux have also been designed using structural modeling. 7:1 has 7 2:1 muxs and 13:1 has 14 2:1 muxs.One of the selecct inputs for the select lines has been shorted to 1 to prevent inputting 000 ,and for the other 13:1 mux I have given a don't care input which will ignore the incorrect inputs, shown in Fig4.

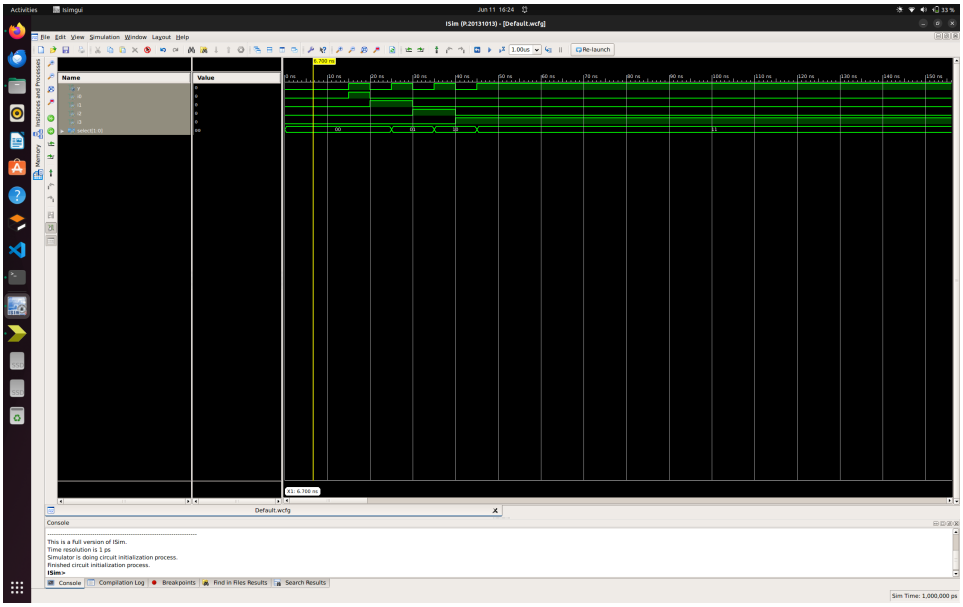


Fig. 1: 4:1 MUX Simulation

```
1
2
3  ////// 2:1 MUX
4   module mux(input i0,input i1,input s,output y);
5   begin
6           assign y=(i0&~s)|(i1&s);
7           end
8           endmodule
9
```

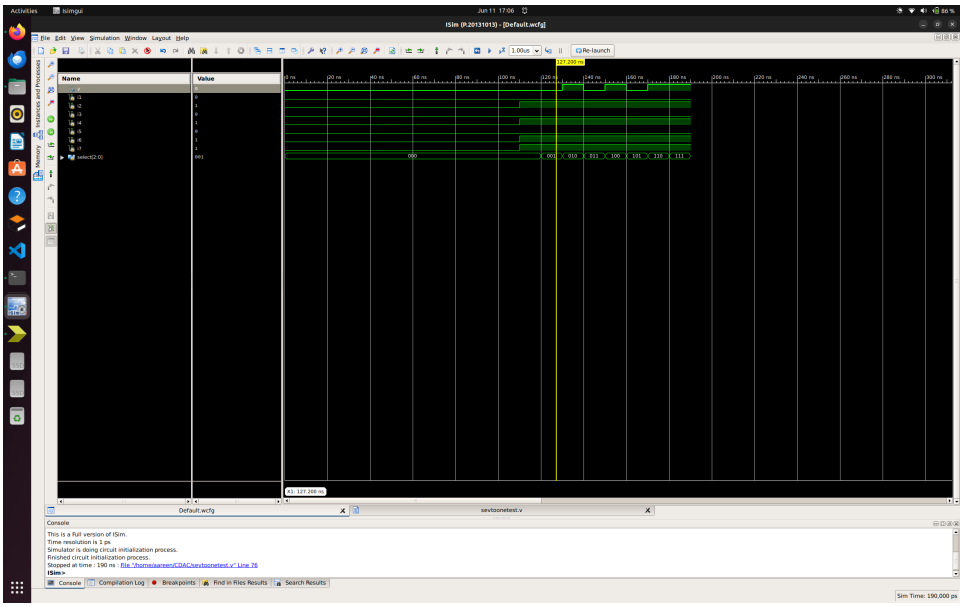


Fig. 2: 7:1 MUX Simulation

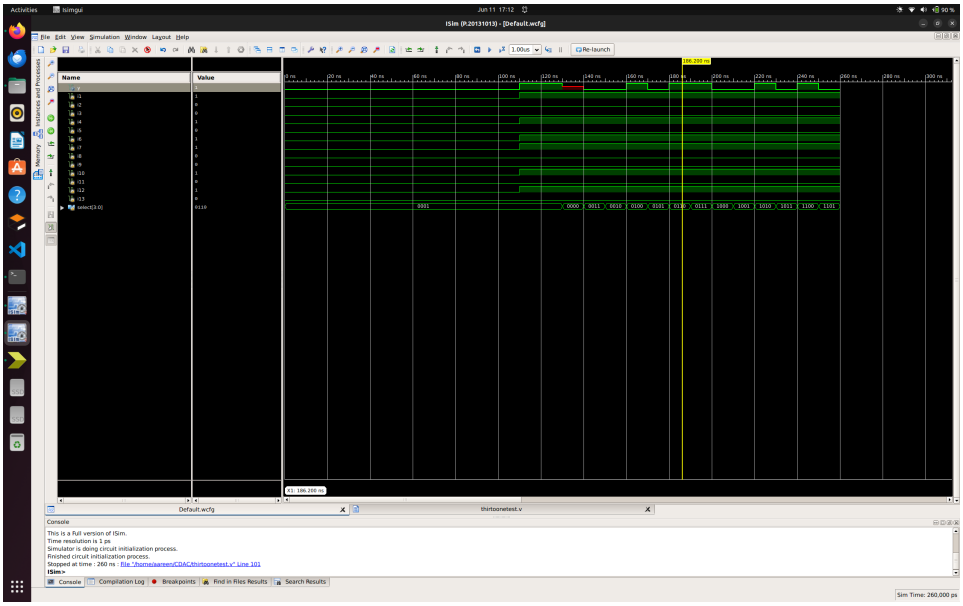


Fig. 3: 13:1 MUX Simulation

```
10
11
12
13  ///////4:1 mux
14
15  module ques2(input i0,input i1,input i2,input i3,input [1:0]select,output y
16      );
17            begin
18            wire w1,w2;
19                    mux mux0(i0,i1,select[0],w1);
20                    mux mux1(i2,i3,select[0],w2);
21                    mux mux2(w1,w2,select[1],y);
22            end
23
24
25  endmodule
26
27  module seventoone(input i1,input i2,input i3,input i4,input i5,input i6,input i7, input [2:0
28  begin
29           wire w1,w2,w3,w4,w5,w6;
30           mux mux0(i0,i1,1'b1,w1);
31           mux mux1(i2,i3,select[0],w2);
32           mux mux2(i4,i5,select[0],w3);
33           mux mux3(i6,i7,select[0],w4);
34           mux mux4(w1,w2,select[1],w5);
35           mux mux5(w3,w4,select[1],w6);
36           mux mux6(w5,w6,select[2],y);
37  end
38  endmodule
39
40
41  module thirtoone(input i1,input i2,input i3,input i4,input i5, input i6 ,input i7, input i8
42  input i11 ,input i12 ,input i13,input [3:0]select,output y);
43  begin
44
45  wire w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14;
46  mux mux0(1'bx,i1,select[0],w1);
47  mux mux1(i2,i3,select[0],w2);
48  mux mux2(i4,i5,select[0],w3);
49  mux mux3(i6,i7,select[0],w4);
50  mux mux4(i8,i9,select[0],w5);
51  mux mux5(i10,i11,select[0],w6);
52  mux mux6(i12,i13,select[0],w7);
53  mux mux7(1'bx,1'bx,select[0],w8);
54  mux mux8(w1,w2,select[1],w9);
55  mux mux9(w3,w4,select[1],w10);
56  mux mux10(w5,w6,select[1],w11);
57  mux mux11(w7,w8,select[1],w12);
58  mux mux12(w9,w10,select[2],w13);
59  mux mux13(w11,w12,select[2],w14);
60  mux mux14(w13,w14,select[3],y);
61
62
63
64  end
65  endmodule
```

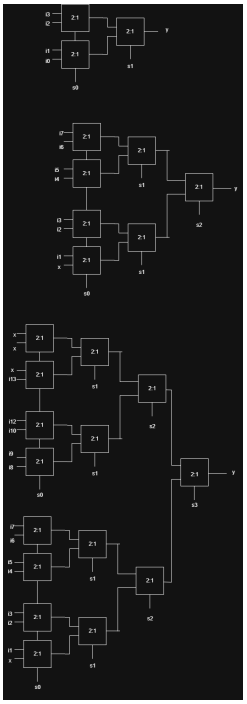


Fig. 4: Question 2 :Circuit of 4:1 ,7:1 and 13:1 MUX

1.3 Question3 : Binary to Gray and Gray to Binary Converters

Gray to Binary and Binary Converters have been implemented using Dataflow Modeling. The equations used to generate gray bits from binary code bits are

$$g[3] = b[3]$$

$$g[2] = b[3] \oplus b[2]$$

$$g[1] = b[2] \oplus b[1]$$

$$g[0] = b[1] \oplus b[0]$$

and the equations use to generate binary bits from gray code are

$$b[3] = g[3]$$

$$b[2] = g[3] \oplus g[2]$$

$$b[1] = g[3] \oplus g[2] \oplus g[1]$$

$$b[0] = g[3] \oplus g[2] \oplus g[1] \oplus g[0]$$

Code for binary to gray converter :

```
1  module bintgrayconvques3(input [3:0]b,output [3:0]g
2      );
3            assign g[3]=b[3];
```

```
          assign g[2]=b[2]^b[3];
          assign g[1]=b[1]^b[2];
          assign g[0]=b[1]^b[0];


endmodule
```

Testbench for the binary to gray converter :

```

module testbintog;

        // Inputs
        reg [3:0] b;

        // Outputs
        wire [3:0] g;

        // Instantiate the Unit Under Test (UUT)
        bintgrayconvques3 uut (
                .b(b),
                .g(g)
        );

        initial begin
                // Initialize Inputs
                b = 0;

                // Wait 100 ns for global reset to finish
                #100;
                end

                // Add stimulus here
                always #10 b=b+1;

                always @(b)
                begin
                #1;
                $display("␣Input␣(bin):␣%4b␣|␣Output␣(gray):␣%4b␣|␣Expected␣:␣%4b",b,g,{b[3]


endmodule
```

Code for gray to binary converter :

```
module graytobinques3(
    output [3:0] b,
    input [3:0] g
    );
          assign b[0]=^g;
          assign b[1]=g[1]^g[2]^g[3];
          assign b[2]=g[2]^g[3];
```

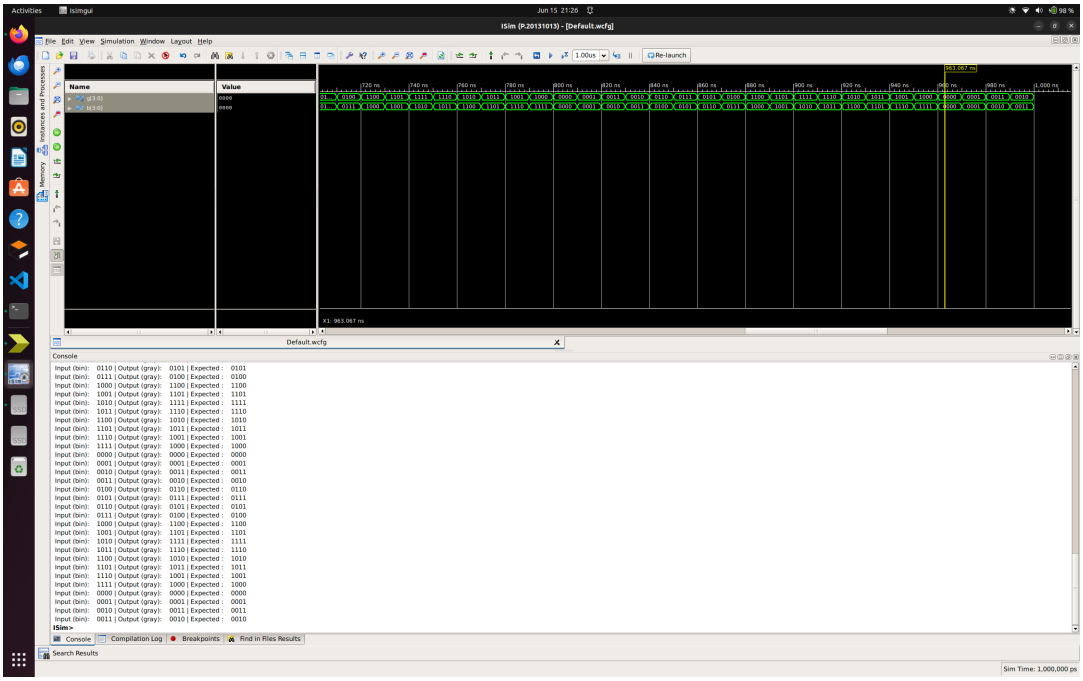


Fig. 5: Simulation of Binary to Gray Converter

```
9           assign b[3]=g[3];
10
11
12  endmodule
```

Testbench for gray to binary converter :

```
1
2
3   module gtobques3test;
4
5           // Inputs
6           //reg [3:0] g;
7           reg [3:0] bin;
8
9           // Outputs
10          wire [3:0] b;
11          wire [3:0]gray;
12
13          // Instantiate the Unit Under Test (UUT)
14          graytobinques3 uut (
15                  .b(b),
16                  .g(gray)
17          );
18
19          bintgrayconvques3 uut2(.b(bin),.g(gray));
20
21          initial begin
22                  // Initialize Inputs
23                  bin = 0;
24                  #10;
25
26                  // Wait 100 ns for global reset to finish
```

```
27
28                  for(bin=0;bin<16;bin=bin+1)
29                  begin
30                  #10;
31                  if(bin!=b)
32                  $display("Mismatch␣␣at␣gray␣=%4b,␣b␣=%4b␣|␣Expected␣:␣%4b",gray,b,bin);
33                  else
34                  $display("Gray␣Code␣:␣%4b␣␣|␣Binary␣Code␣:␣%4b",gray,b);
35                  end
36
37
38                  // Add stimulus here
39                  end
40
41
42
43
44  endmodule
```

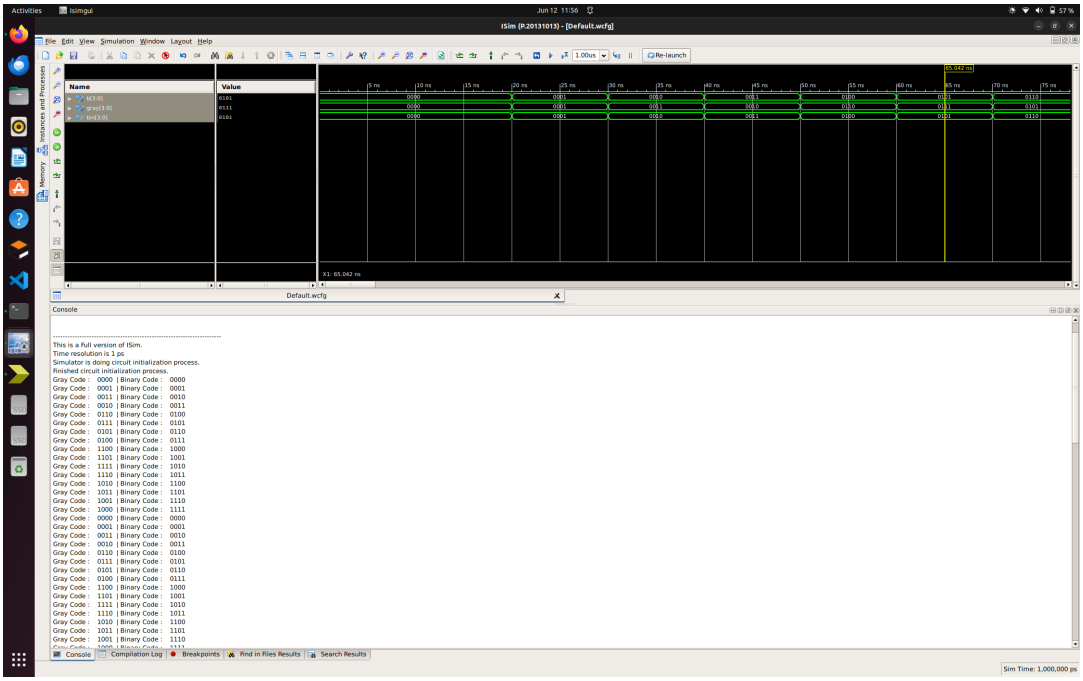


Fig. 6: Simulaition of Gray to Binary Converter

Block Diagram for both the Circuits :

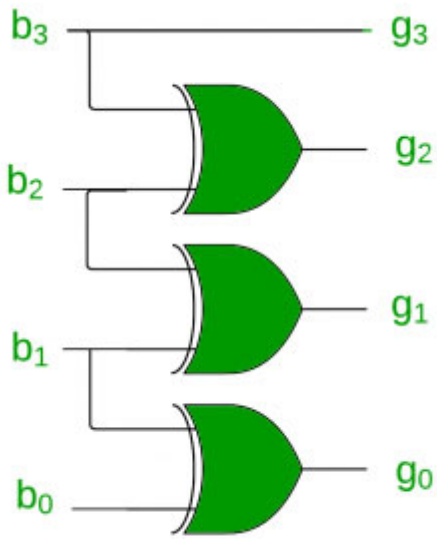


Fig. 7: Block Diagram of Binary to Gray Converter

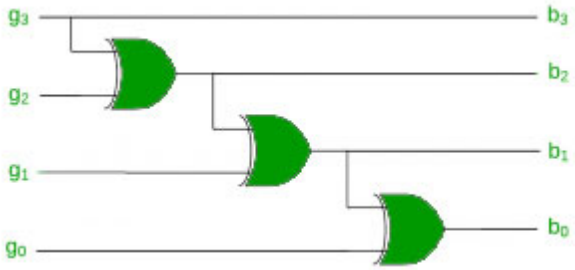


Fig. 8: Block Diagram of Gray to Binary Converter

1.4 Question 4 : Adders and Subtractors

1.4.1 Half Adder

The half adder was implemented using Dataflow modeling in Verilog . Equations used used for sum and carry bits are

$$Sum = a \oplus b$$

$$Carry = a \wedge b$$

Given below is it's code , testbench and simulation image in verilog along with block diagram

```
1 `timescale 1ns / 1ps
2 module halfadder(
3     input a,
4     input b,
5          output sum,
6          output carry
```

```
    );

assign sum = a^b;
assign carry=a&b;

endmodule
```

Testbench for Half Adder

```
`timescale 1ns / 1ps


module haddertest;

        // Inputs
        reg a;
        reg b;

        // Instantiate the Unit Under Test (UUT)
        halfadder uut (
                .a(a),
                .b(b),
                .sum(sum),
                .carry(carry)
        );

        initial begin
                // Initialize Inputs
                a = 0;
                b = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here
                a=0;b=0;
                #1;
                $display("a : %b| b : %b | sum : %b | carry : %b",a,b,sum,carry);
                #10;
                a=1;b=0;
                #1;
                $display("a : %b| b : %b | sum : %b | carry : %b",a,b,sum,carry);
                #10;
                a=0;b=1;
                #1;
                $display("a : %b| b : %b | sum : %b | carry : %b",a,b,sum,carry);
                #10
                a=1;b=1;
                #1;
                $display("a : %b| b : %b | sum : %b | carry : %b",a,b,sum,carry);
                #100 $finish;
        end

endmodule
```

Simulation for Half Adder : Block Diagram for Half Adder

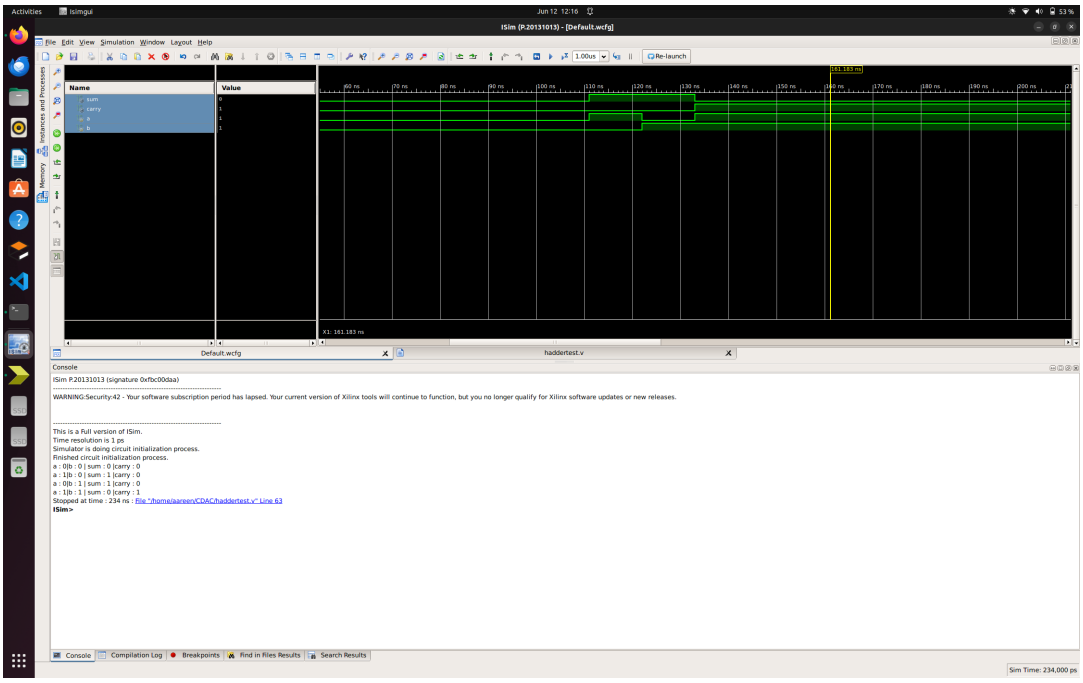


Fig. 9: Simulation of Half Adder

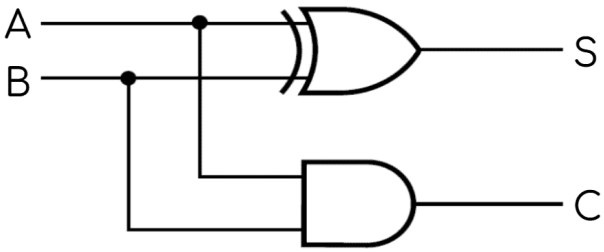


Fig. 10: Block Diagram of Half Adder

1.4.2 Full Adder

The full Adder was implemented using Dataflow Modeling. The equations used for Sum and Carry were

$$Sum = (a \oplus b) \vee (a \oplus Cin) \vee (b \oplus Cin)$$

$$Carry = (a \oplus b) \vee (a \oplus Cin) \vee (b \oplus Cin)$$

The code,testbench ,Simulation Diagram and Block Diagram are shared below

```
`timescale 1ns / 1ps

module fulladder(
    input cin,
    input a,
    input b,
    output sum,
    output cout
    );
         assign sum= (a^b) | (b^cin) | (a^cin);
         assign cout = (a&b) | (a&cin)|( b&cin);


endmodule
```

Testbench

```
`timescale 1ns / 1ps


module fulladdertest;

        // Inputs
        reg cin;
        reg a;
        reg b;

        // Outputs
        wire sum;
        wire cout;

        // Instantiate the Unit Under Test (UUT)
        fulladder uut (
                .cin(cin),
                .a(a),
                .b(b),
                .sum(sum),
                .cout(cout)
        );

        initial begin
                // Initialize Inputs
                cin = 0;
                a = 0;
                b = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here
                #1;
                $display ("a:␣%b␣|␣b:␣%b␣|␣cin:␣%b␣|␣sum:␣%b␣|␣cout:␣%b",a,b,cin,sum,cout);
                #10 a=0;b=1;cin=0;
                #1;
        $display ("a:␣%b␣|␣b:␣%b␣|␣cin:␣%b␣|␣sum:␣%b␣|␣cout:␣%b",a,b,cin,sum,cout);
                 #10 a=1;b=0;cin=0;  // Removed stray `
    #1;
    $display ("a:␣%b␣|␣b:␣%b␣|␣cin:␣%b␣|␣sum:␣%b␣|␣cout:␣%b",a,b,cin,sum,cout);
```

```
42                  #10 a=1;b=1;cin=0;
43                  #1;
44                  $display ("a:␣%b␣|␣b:␣%b␣|␣cin:␣%b␣|␣sum:␣%b␣|␣cout:␣%b",a,b,cin,sum,cout);
45                  #10 a=0;b=0;cin=1;
46                  #1;
47                  $display ("a:␣%b␣|␣b:␣%b␣|␣cin:␣%b␣|␣sum:␣%b␣|␣cout:␣%b",a,b,cin,sum,cout);
48                  #10 a=1;b=0;cin=1;
49                  #1;
50                  $display ("a:␣%b␣|␣b:␣%b␣|␣cin:␣%b␣|␣sum:␣%b␣|␣cout:␣%b",a,b,cin,sum,cout);
51
52                  #10 a=1;b=1;cin=1;
53                  #1;
54                  $display ("a:␣%b␣|␣b:␣%b␣|␣cin:␣%b␣|␣sum:␣%b␣|␣cout:␣%b",a,b,cin,sum,cout);
55                  #10 $finish;
56
57
58
59          end
60
61  endmodule
```

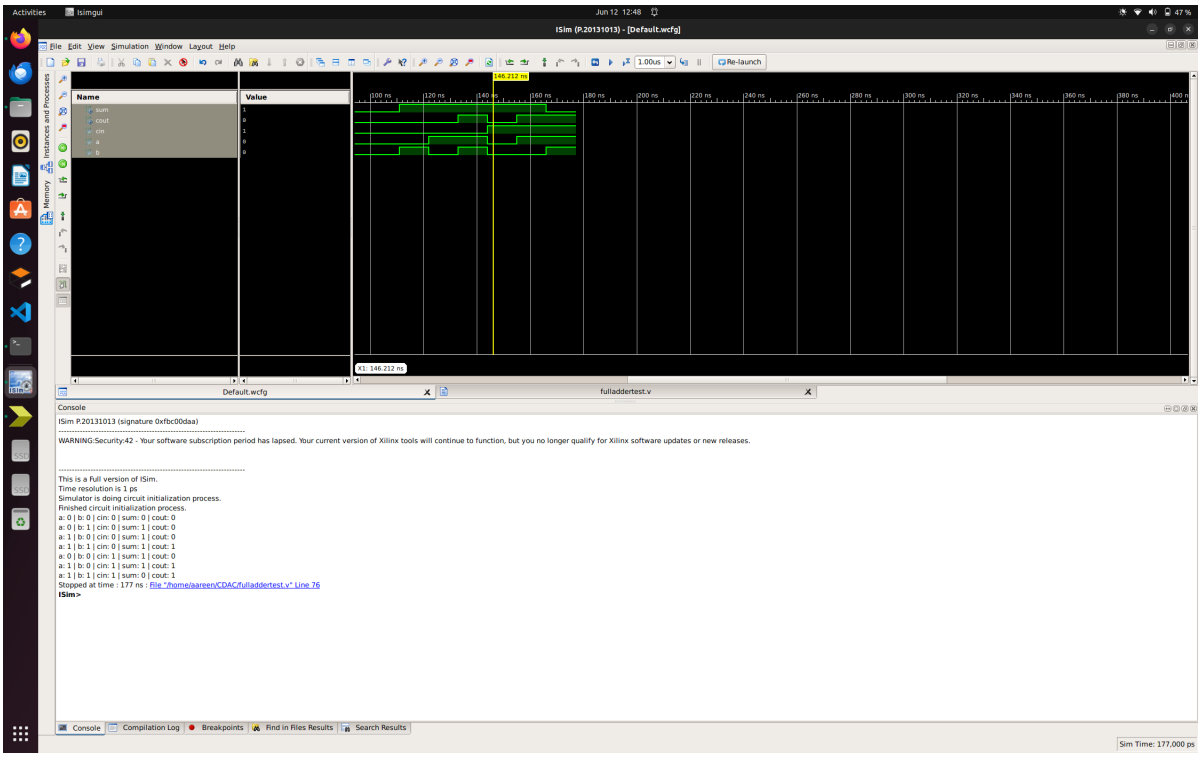


Fig. 11: Simulation of Full Adder

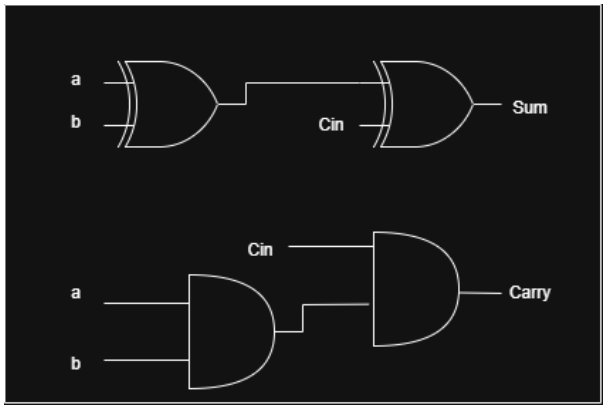


Fig. 12: Block Diagram of Full Adder

1.4.3 Half Subtractor

Thr half Subtractor was implemented using Dataflow Modeling in verilog . The equations used for Difference and Borrow were :

$$Diff = a \oplus b$$

$$Borrow = \sim a \wedge b$$

Code for Half Subtractor

```
1  `timescale 1ns / 1ps
2  module halSubtractor(
3      input a,
4      input b,
5      output diff,
6      output borrow
7      );
8           assign diff=a^b;
9           assign borrow =(~a)&b;
10
11
12 endmodule
```

Testbench

```
1  `timescale 1ns / 1ps
2
3
4  module halfSubTest;
5
6          // Inputs
7          reg a;
8          reg b;
9
10         // Outputs
11         wire diff;
12         wire borrow;
13
14         // Instantiate the Unit Under Test (UUT)
15         halSubtractor uut (
16                 .a(a),
```

```
                .b(b),
                .diff(diff),
                .borrow(borrow)
        );

        initial begin
                // Initialize Inputs
                a = 0;
                b = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here
                $display("Testing␣Half␣Subtractor:");
        $display("a␣|␣b␣|␣diff␣|␣borrow␣|␣Expected␣diff␣|␣Expected␣borrow");

        a = 0; b = 0; #10;
        $display("%b␣|␣%b␣|␣␣%b␣␣␣|␣␣␣%b␣␣␣␣|␣␣␣␣␣␣␣0␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣0", a, b, diff, borrow)

        a = 0; b = 1; #10;
        $display("%b␣|␣%b␣|␣␣%b␣␣␣|␣␣␣%b␣␣␣␣|␣␣␣␣␣␣␣1␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣1", a, b, diff, borrow)

        a = 1; b = 0; #10;
        $display("%b␣|␣%b␣|␣␣%b␣␣␣|␣␣␣%b␣␣␣␣|␣␣␣␣␣␣␣1␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣0", a, b, diff, borrow)

        a = 1; b = 1; #10;
        $display("%b␣|␣%b␣|␣␣%b␣␣␣|␣␣␣%b␣␣␣␣|␣␣␣␣␣␣␣0␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣0", a, b, diff, borrow)

        $finish;

        end

endmodule
```

1.4.4 Full Subtractor

Full Subtractor was implemented using Dataflow modeling. The equations used to generate the carry and difference are

$$Diff = Bin \oplus (a \oplus b)$$

$$Borrow = Bin \wedge (\sim (a \oplus b) \vee (\sim a \wedge b)$$

Code for Full Subtractor

```
‘timescale 1ns / 1ps

module fullSub(
    input a,
    input b,
    input Bin,
    output Bout,
```

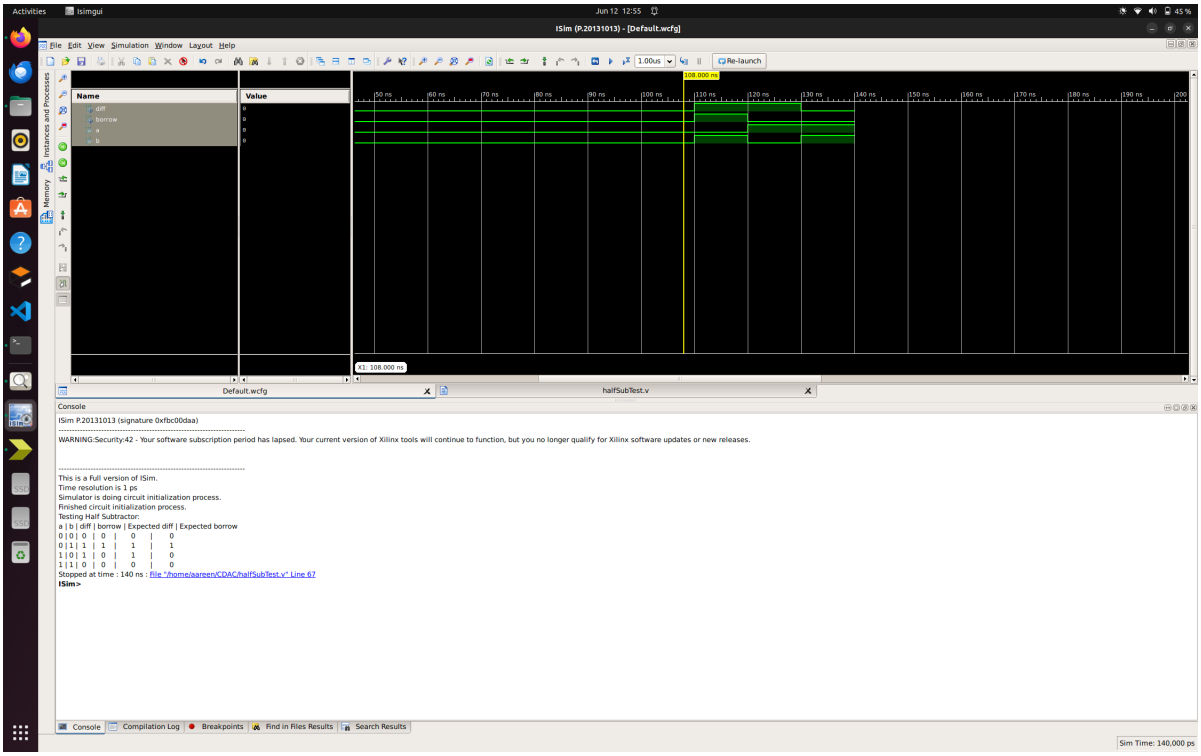


Fig. 13: Simulation of Half Subtractor

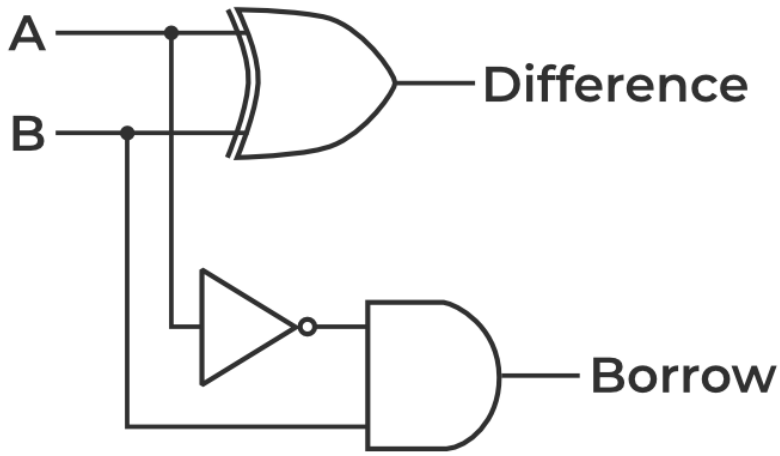


Fig. 14: Block Diagram of Half Subtractor

```
8       output diff
9       );
10  assign diff= Bin^(a^b);
11  assign Bout =Bin&(~(a^b))| (~a&b);
12
13  endmodule
```

Testbench

```
`timescale 1ns / 1ps


module fullsubtest;

        // Inputs
        reg a;
        reg b;
        reg Bin;

        // Outputs
        wire Bout;
        wire diff;

        // Instantiate the Unit Under Test (UUT)
        fullSub uut (
                .a(a),
                .b(b),
                .Bin(Bin),
                .Bout(Bout),
                .diff(diff)
        );
        integer i;
        initial begin
                // Initialize Inputs
                a = 0;
                b = 0;
                Bin = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here
                $display("a | b | Bin | diff | Bout | Expected diff | Expected Bout");

        for (i = 0; i < 8; i = i + 1) begin
            {a, b, Bin} = i; // Assign 3-bit input from loop variable
            #10; // Wait for outputs to stabilize

            // Calculate expected values using the same logic as your code
            $display(
                "%b | %b | %b | %b | %b | %b | %b | %b",
                a, b, Bin, diff, Bout,
                Bin ^ (a ^ b),                // Expected diff
                (Bin & ~(a ^ b)) | (~a & b)  // Expected Bout
            );
        end
        $finish;

        end

endmodule
```

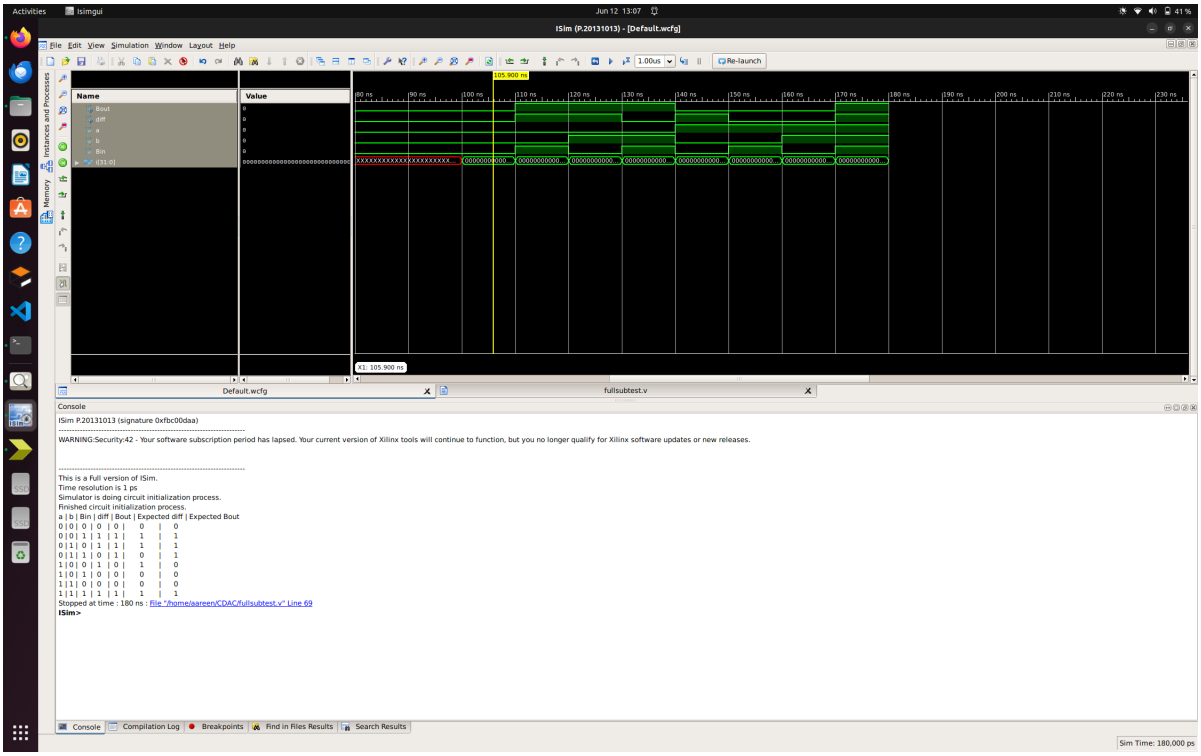


Fig. 15: Simulation of Full Subtractor

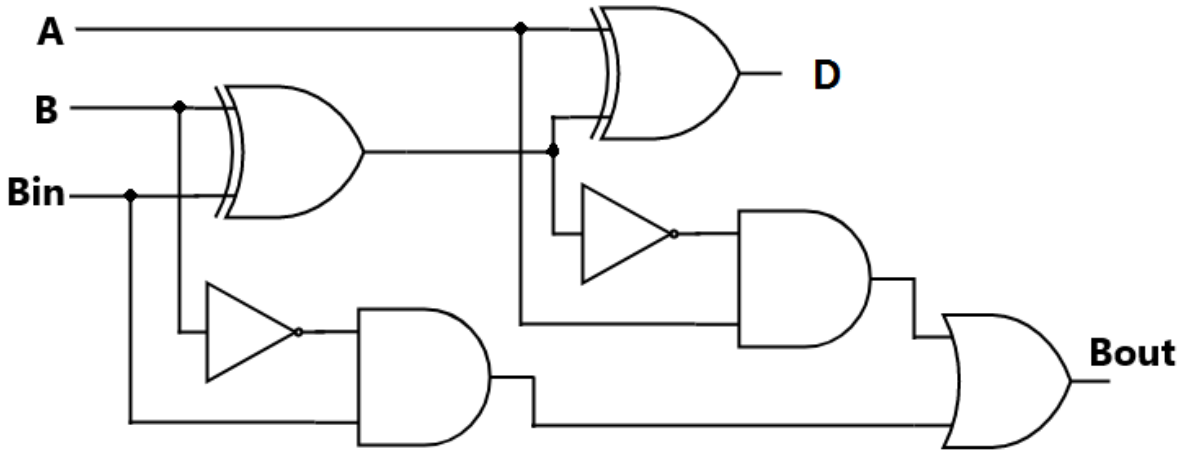


Fig. 16: Block Diagram of Full Subtractor

1.5 Question 5 : 4 Bit Full Adder and Ripple Carry Adder

1.5.1 4-Bit Full Adder

The 4 bit full adder was implemented using Dataflow Modeling. Given below is the code for the same along with it's testbench, block diagram and simulation.

```
1 `timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////////////////
module fourbitfulladderNonRipple(

    input [3:0] a,
    input [3:0] b,
    input cin,
    output reg [3:0] sum,
    output reg cout
);
    always @(a or b or cin) begin
        {cout, sum} = a + b + cin;
    end
endmodule
```

Testbench:

```
`timescale 1ns / 1ps

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

module foubitadderNonRippleTest;

        // Outputs
        reg [3:0] a, b;
    reg cin;
wire [3:0] sum;
    wire cout;

        // Instantiate the Unit Under Test (UUT)
        fourbitfulladderNonRipple uut (
                .a(a),
        .b(b),
        .cin(cin),
        .sum(sum),
        .cout(cout)
        );

        initial begin
                // Initialize Inputs

                // Wait 100 ns for global reset to finish
                //#100;
                 $monitor("Time=%0t: a=%d, b=%d, cin=%b    sum=%d, cout=%b",
                 $time, a, b, cin, sum, cout);

        a = 4'b0011; b = 4'b0010; cin = 0; #10;
        a = 4'b1111; b = 4'b0001; cin = 0; #10;
        a = 4'b1000; b = 4'b1000; cin = 1; #10;

        $finish;

                // Add stimulus here

        end

endmodule
```

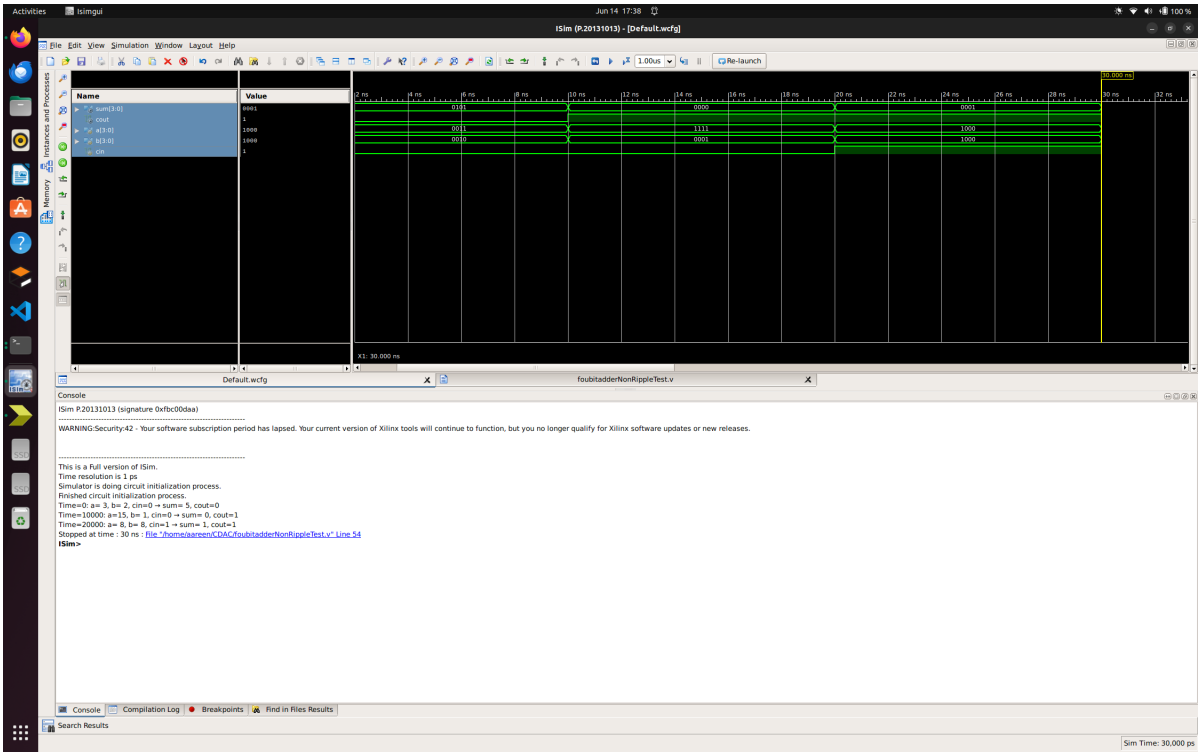


Fig. 17: Simulation of 4 -Bit fulll Adder

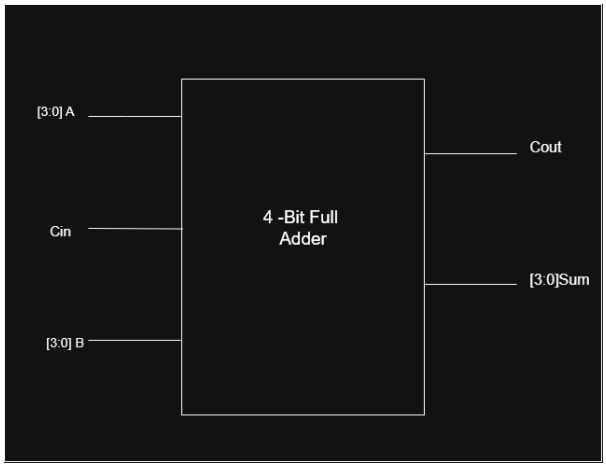


Fig. 18: Block Diagram of 4 bit Full Adder

1.5.2 4-Bit Ripple Carry Adder

The 4 bit ripple carry adder was designed using 4 full adder modules . Code for the ripple carry adder

```
1  `timescale 1ns / 1ps
2  module fourbitfulladder(
3      input cin,
4      input [3:0] a,
5      input [3:0] b,
6      output cout,
```

```
    output [3:0] sum
    );
         wire w1,w2,w3;
         fulladder fa0(cin,a[0],b[0],sum[0],w1);
         fulladder fa1(w1,a[1],b[1],sum[1],w2);
         fulladder fa2(w2,a[2],b[2],sum[2],w3);
         fulladder fa3(w3,a[3],b[3],sum[3],cout);


endmodule
```

Testbench

```
`timescale 1ns / 1ps

module fourbitaddertest;

        // Inputs
        reg cin;
        reg [3:0] a;
        reg [3:0] b;

        // Outputs
        wire cout;
        wire [3:0] sum;

        // Instantiate the Unit Under Test (UUT)
        fourbitfulladder uut (
                .cin(cin),
                .a(a),
                .b(b),
                .cout(cout),
                .sum(sum)
        );
        integer i,j;
        initial begin
                // Initialize Inputs
                cin = 0;
                a = 0;
                b = 0;
                        i=0;
                // Wait 100 ns for global reset to finish

                 #1
                // Add stimulus here
                for(i=0;i<16;i=i+1)
                begin
                a=i;
                        for(j=0;j<16;j=j+1)
                        begin
                           b=j;
                                #10;
                                $display("a: %4b , b: %4b , sum : %4b , cout : %b ",a,b,sum,c

                        end

                end

                #10;
                cin=1;
```

```
48                  for(i=0;i<16;i=i+1)
49                  begin
50                  a=i;
51                          for(j=0;j<16;j=j+1)
52                          begin
53                              b=j;
54                                  #10;
55                                  $display("a: %4b , b: %4b , sum : %4b , cout : %b ",a,b,sum,c
56
57                          end
58
59                  end
60
61          end
62
63  endmodule
```

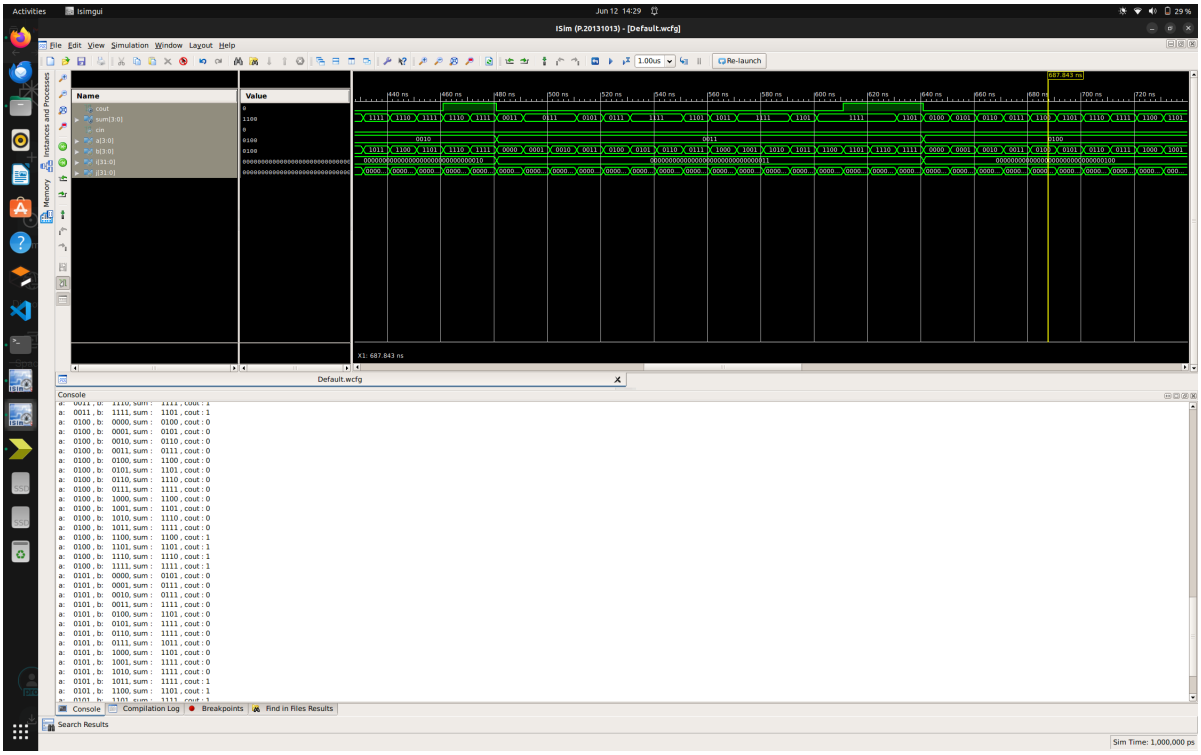


Fig. 19: Simulation of 4 -Bit ripple carry Adder

1.6 Question 6 : Priority Encoder

The Priority Encoder was designed using Behavioral modeling . The module has 4 inputs and 2 -Bit output .

```
1  `timescale 1ns / 1ps
2
3  module priorityEncoder(
4      input i0,
5      input i1,
6      input i2,
7      input i3,
```

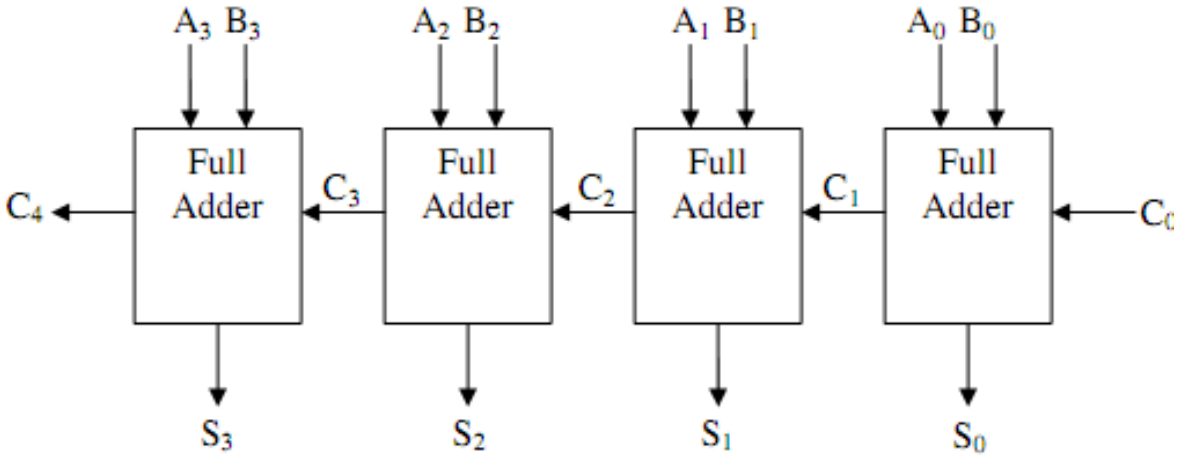


Fig. 20: Block Diagram of 4-Bit Ripple Carry Adder

```
    output reg [1:0] y
    );
          always @(*)
          begin
                  if(i3)
                  y=3;
                  else if(i2)
                  y=2;
                  else if(i1)
                  y=1;
                  else
                  y=0;
          end


endmodule
```

Testbench

```
`timescale 1ns / 1ps


module prioEncTest;

        // Inputs
        reg i0;
        reg i1;
        reg i2;
        reg i3;

        // Outputs
        wire [1:0] y;

        // Instantiate the Unit Under Test (UUT)
        priorityEncoder uut (
                .i0(i0),
                .i1(i1),
                .i2(i2),
```

```
21                  .i3(i3),
22                  .y(y)
23          );
24          integer i;
25          initial begin
26                  // Initialize Inputs
27                  i0 = 0;
28                  i1 = 0;
29                  i2 = 0;
30                  i3 = 0;
31
32                  // Wait 100 ns for global reset to finish
33                  #100;
34
35                  // Add stimulus here
36                  #10
37                  for(i=0;i<16;i=i+1)
38                  begin
39                  #10
40                  {i3,i2,i1,i0}=i;
41                  end
42
43
44          end
45
46  endmodule
```

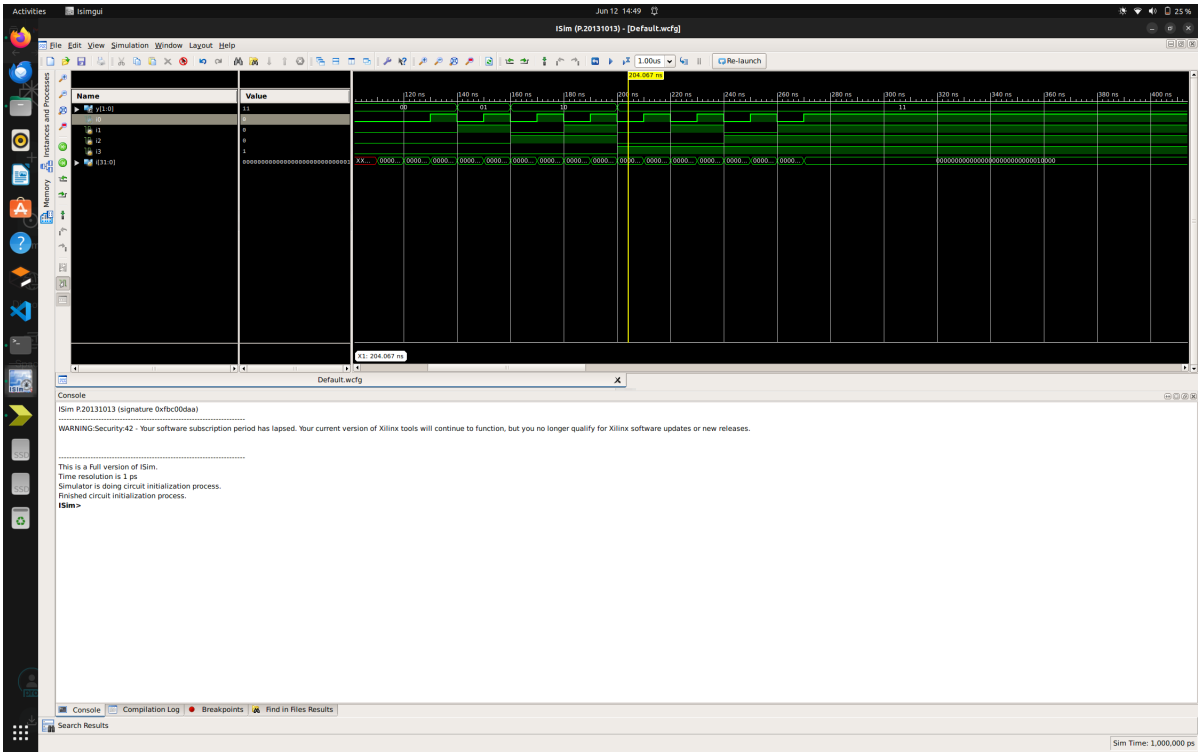


Fig. 21: Simulation of Priority Encoder

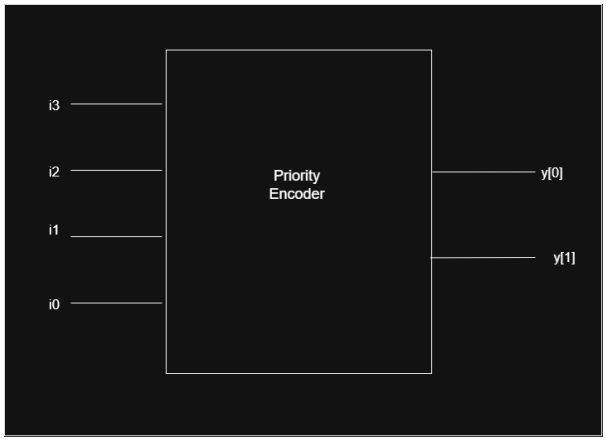


Fig. 22: Block Diagram of Priority Encoder

1.7 Question 7 : Latches

1.7.1 S-R Latch

The modeling of SR Latch has been done using behavioral modeling in verilog . The function that an SR Latch performs is described below

Table 1: SR Latch Truth Table

S (Set)	**R (Reset)**	**Q (Output)**	**$\overline{Q}$ (Complement)**
0	0	Previous Q	Previous $\overline{Q}$
0	1	0	1
1	0	1	0
1	1	invalid	

Code used :

```
1  `timescale 1ns / 1ps
2
3  module SRLatch(
4      input s,
5      input r,
6           input en,
7      output reg q,
8      output reg qbar
9      );
10         always @(*)
11         begin
12                  if(en)
13                  begin
14                  case ({s,r})
15                  00:begin
16                          q<=q;
17                          qbar<=qbar;
18                  end
19                  01:begin
20                  q<=0;
21                  qbar<=1;
22                  end
```

```
                10: begin
                        q<=1;
                        qbar<=0;
                        end
                11: begin
                q<=0;
                qbar<=0;
                                end
                endcase
                end
                else
                begin
                        q<=q;
                        qbar<=qbar;
                end



        end

endmodule
```

Testbench :

```
`timescale 1ns / 1ps



module srtest;

        // Inputs
        reg s;
        reg r;
        reg en;

        // Outputs
        wire q;
        wire qbar;

        // Instantiate the Unit Under Test (UUT)
        SRLatch uut (
                .s(s),
                .r(r),
                .en(en),
                .q(q),
                .qbar(qbar)
        );

        initial begin
                // Initialize Inputs
                s = 0;
                r = 1;
                en = 1;

                // Wait 100 ns for global reset to finish
                #50;


                // Add stimulus here
```

```
36                  s<=0;r<=0;
37                  #30;
38                  s<=0;r<=1;
39                  #30;
40                  s<=1;r<=1;
41                  #30
42                  en=0;
43                  s<=1;r<=0;
44                  #30 $finish;
45
46
47          end
48
49
50  endmodule
```

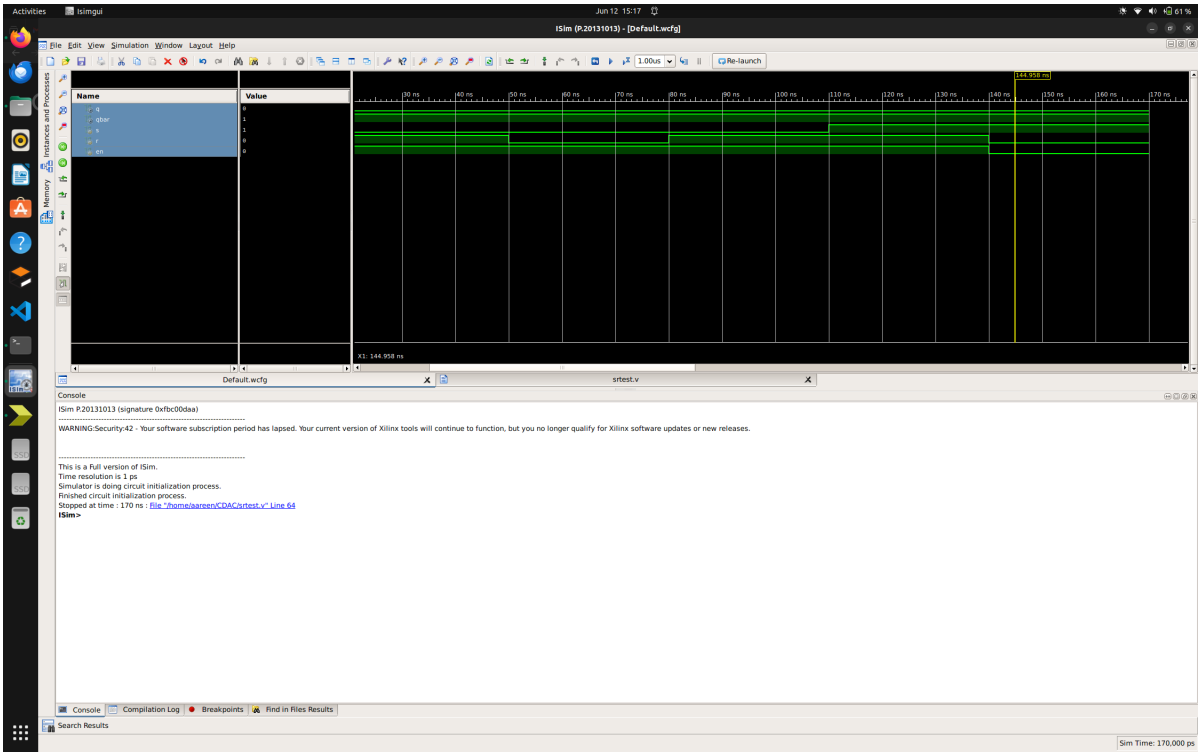


Fig. 23: Simulation SR Latch

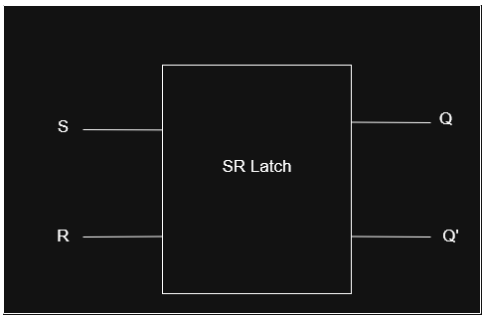


Fig. 24: Block Diagram SR Latch

1.7.2 JK Latch

The JK Latch is implemented usig Behavioral Modeling.The truth table for the JK Latch is given below

Table 2: JK Latch Truth Table (Level-Sensitive, clk = 1)

clk	J	K	Q (Current)	Q+ (Next State)
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Code for JK Latch

```
`timescale 1ns / 1ps
module jkLatcch(
    input j,
    input k,
         input clk,
    output reg q,
    output reg qbar
    );
        always @(clk, j, k) begin
        if (clk) begin
            // Standard JK latch logic
            q    <= (j & ~q) | (~k & q);
            qbar <= ~q;
        end
        // Else: latch holds previous value (implicit in reg behavior)
    end


endmodule
```

Testbench for the JK Latch

```
`timescale 1ns / 1ps


module jkLatchTest;

        // Inputs
        reg j;
        reg k;
        reg clk;

        // Outputs
        wire q;
        wire qbar;

        // Instantiate the Unit Under Test (UUT)
        jkLatcch uut (
```

```
17                  .j(j),
18                  .k(k),
19                  .clk(clk),
20                  .q(q),
21                  .qbar(qbar)
22          );
23          initial begin
24          clk = 0;
25          forever #10 clk = ~clk;
26      end
27
28      // Stimulus generation
29      initial begin
30          // Initialize inputs
31          j = 0;
32          k = 0;
33
34          // Test sequence
35          // 1. Hold mode (J=0, K=0)
36          $display("Testing␣Hold␣mode␣(J=0,␣K=0)");
37          #20; // Wait for 2 clock cycles
38
39          // 2. Set mode (J=1, K=0)
40          j = 1; k = 0;
41          $display("Testing␣Set␣mode␣(J=1,␣K=0)");
42          #20;
43
44          // 3. Reset mode (J=0, K=1)
45          j = 0; k = 1;
46          $display("Testing␣Reset␣mode␣(J=0,␣K=1)");
47          #20;
48
49          // 4. Toggle mode (J=1, K=1)
50          j = 1; k = 1;
51          $display("Testing␣Toggle␣mode␣(J=1,␣K=1)");
52          #20;
53
54          // 5. Hold mode again (J=0, K=0)
55          j = 0; k = 0;
56          $display("Testing␣Hold␣mode␣again␣(J=0,␣K=0)");
57          #20;
58
59          // End simulation
60          $display("Simulation␣completed.");
61          $finish;
62      end
63
64      // Optional: Monitor outputs for all time
65      initial begin
66          $monitor("Time=%0t:␣clk=%b,␣j=%b,␣k=%b,␣q=%b,␣qbar=%b",
67                   $time, clk, j, k, q, qbar);
68      end
69
70
71  endmodule
```

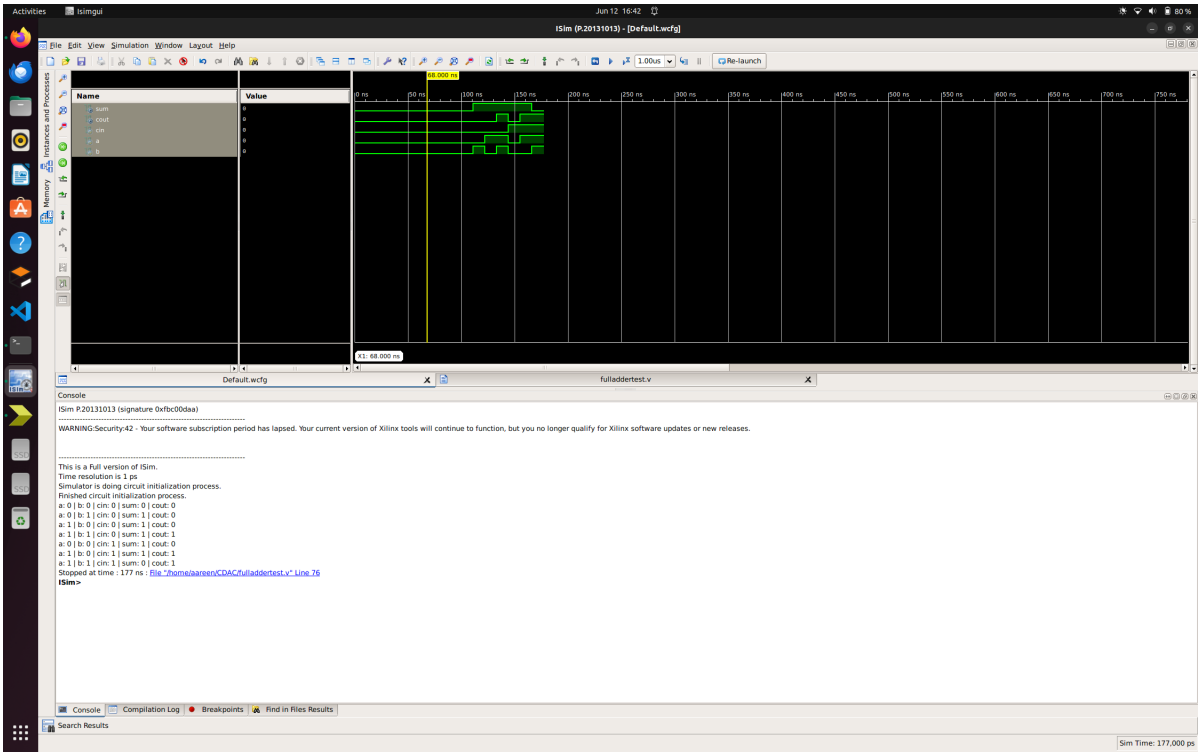


Fig. 25: Simulation JK Latch

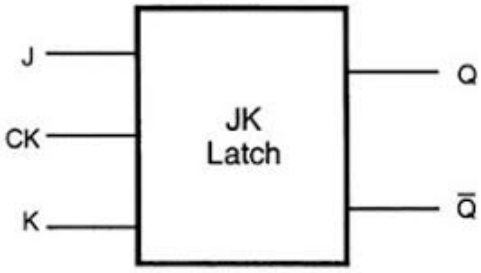


Fig. 26: Block Diagram JK Latch

1.7.3 T Latch

The T-latch is designed using Behavioral Modeling. the truth table for the same is provided below

Code for T Latch

```
1 `timescale 1ns / 1ps
2 module TLatch(
3     input t,
4     input en,
5          input reset,
6     output reg q
7     );
8          always @(en,t,reset)
9          begin
```

Table 3: T Latch Truth Table (with Reset and Enable Inputs)

R (Reset)	clk (Enable)	T	Q (Current)	Q+ (Next State)
1	X	X	X	0
0	0	X	0	0
0	0	X	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

```
10          if(reset)
11          begin
12          q<=0;
13          end
14
15          else if(en)
16           begin
17                  if(t)
18                          q<=~q;
19                  else
20                          q<=q;
21                  end
22                  else
23                  q<=q;
24
25
26          end
27
28
29  endmodule
```

TestBench

```
1   `timescale 1ns / 1ps
2
3
4   module TLatchTest;
5
6           // Inputs
7           reg t;
8           reg en;
9           reg reset;
10
11          // Outputs
12          wire q;
13
14          // Instantiate the Unit Under Test (UUT)
15          TLatch uut (
16                  .t(t),
17                  .en(en),
18                  .reset(reset),
19                  .q(q)
20          );
21
22          initial begin
23                  // Initialize Inputs
24                  t = 0;
```

```
25                  en = 0;
26                  reset = 0;
27                  #5;
28                  reset=1;//generating a pulse edge to reset the latch to  a valid initial sta
29                  #10
30                  reset=0;
31                  en=1;
32
33
34                  // Wait 100 ns for global reset to finish
35                  #100;
36          t=1;
37                    #20;
38                    t=1;
39                    en=0;
40                    #20
41                    t=0;
42                    #20
43                    $finish;
44                  // Add stimulus here
45
46          end
47
48  endmodule
```

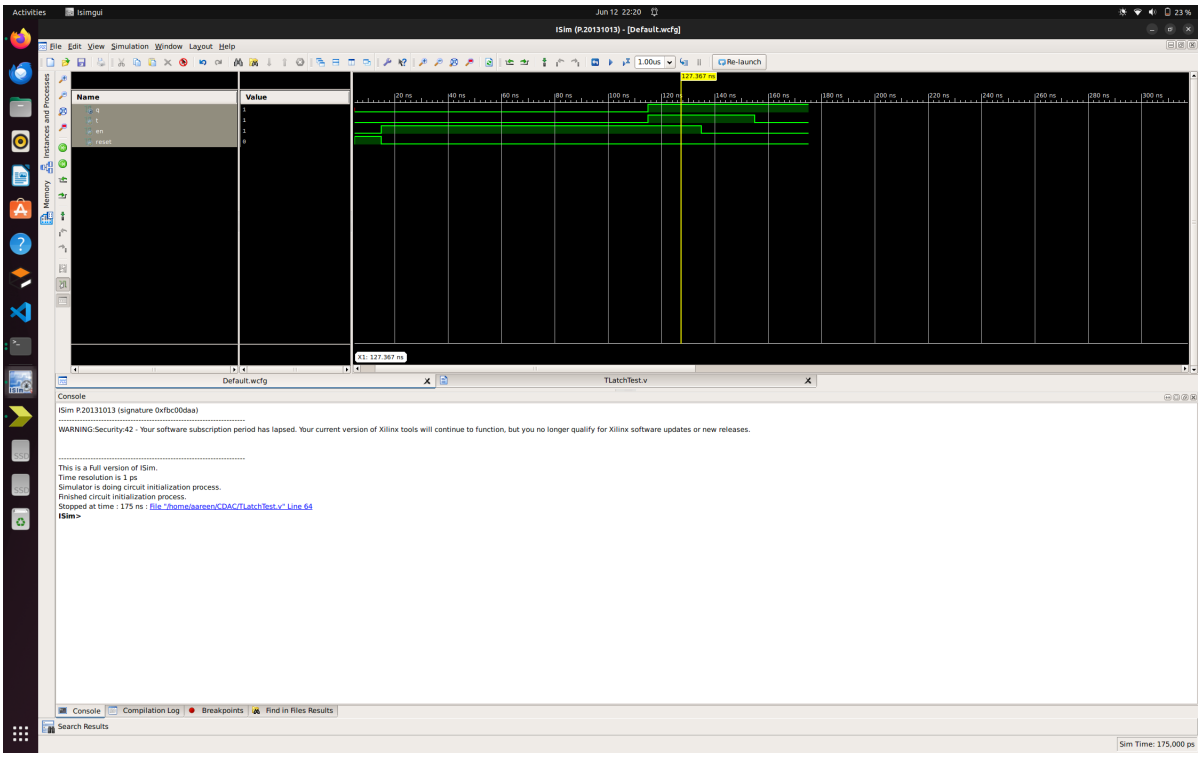


Fig. 27: Simulation of T Latch

1.7.4 D Latch

The D Latch is implemented using Behavioral Modeling . The truth table for the same is provided below

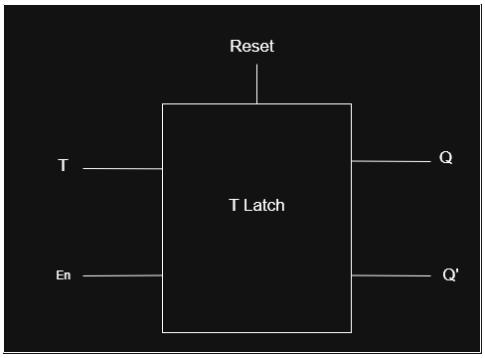


Fig. 28: Block Diagram T Latch

Table 4: D Latch Truth Table (with Enable Input)

clk (Enable)	D	Q (Current)	Q+ (Next State)
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

Code for D Latch

```
1  `timescale 1ns / 1ps
2  module DLatch(
3      input d,
4      input en,
5      output reg q
6      );
7            always @(en,d)
8            begin
9                   if(en)
10                          q=d;
11           end
12
13
14 endmodule
```

Testbench

```
1  `timescale 1ns / 1ps
2
3  module DLatchTest;
4
5          // Inputs
6          reg d;
7          reg en;
8
9          // Outputs
10         wire q;
11
12         // Instantiate the Unit Under Test (UUT)
13         DLatch uut (
14                 .d(d),
15                 .en(en),
16                 .q(q)
17         );
```

```
18
19          initial begin
20                  // Initialize Inputs
21                  d = 0;
22                  en = 0;
23
24
25                  // Wait 100 ns for global reset to finish
26                  #100;
27                  en=1;
28                  #10;
29                  d=1;
30                  #10;
31                  en=0;
32                  #10
33                  d=0;
34                  #10;
35                  $finish;
36
37
38                  // Add stimulus here
39
40          end
41
42  endmodule
```

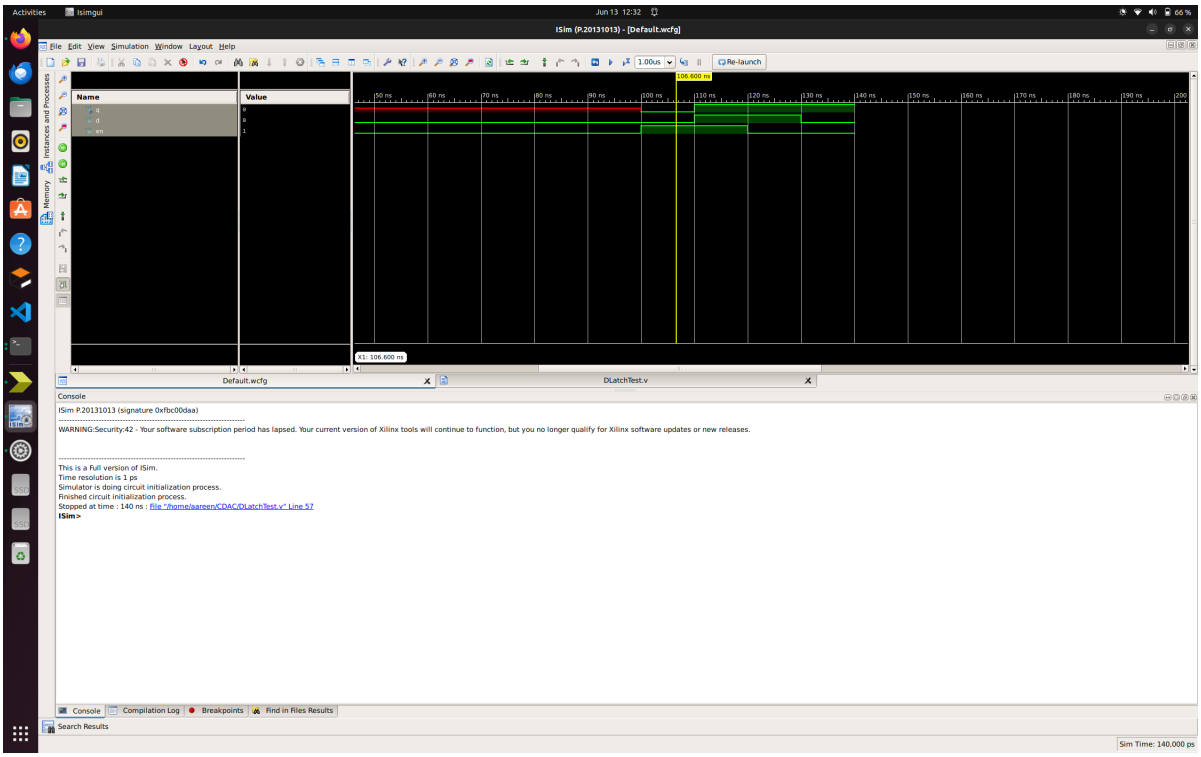


Fig. 29: Simulation of D Latch

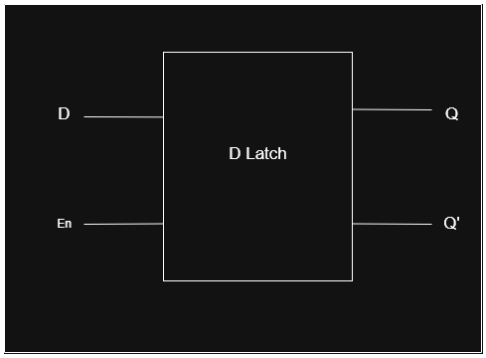


Fig. 30: Block Diagram D Latch

1.8 Flip Flops

1.8.1 SR Flip Flop,JK Flip Flop , T Flip Flop, D Flip FLop

All 4 four flip flops have been in the same module behaviorally.

Their truth tables are as follows

Table 5: T Flip-Flop Truth Table (Positive Edge-Triggered, Synchronous Clear)

clr	clk	T	Q (Current)	Q+ (Next State)
X	-	X	0	0
X	-	X	1	1
1	↑	X	X	0
0	↑	0	0	0
0	↑	0	1	1
0	↑	1	0	1
0	↑	1	1	0

Table 6: D Flip-Flop Truth Table (Positive Edge-Triggered)

clk	D	Q (Current)	Q+ (Next State)
-	X	0	0
-	X	1	1
↑	0	X	0
↑	1	X	1

Code for the Flip Flops

```
1  `timescale 1ns / 1ps
2  module Ques8ffs(input s,input r,output reg qsr,output reg qsrbar,input clk,
3  input d,output reg qd, output reg qdbar,
4  input j,input k,output reg qjk,output reg qjkbar,
5  input t,output reg qt,output reg qtbar,input tclr
6      );
7            //////////////SR Flip Flop//////////////////////
8            always @(posedge clk)
9            begin
10           qsr=s|((~r)&qsr);
11           qsrbar=~qsr;
```

Table 7: T Flip-Flop Truth Table (Positive Edge-Triggered, Synchronous Clear)

clr	clk	T	Q (Current)	Q+ (Next State)
X	-	X	0	0
X	-	X	1	1
1	↑	X	X	0
0	↑	0	0	0
0	↑	0	1	1
0	↑	1	0	1
0	↑	1	1	0

Table 8: SR Flip-Flop Truth Table (Positive Edge-Triggered)

clk	S	R	Q (Current)	Q+ (Next State)
-	X	X	0	0
-	X	X	1	1
↑	0	0	0	0
↑	0	0	1	1
↑	0	1	0	0
↑	0	1	1	0
↑	1	0	0	1
↑	1	0	1	1
↑	1	1	X	invalid

```
12          end
13
14          ////////////D Flip Flop//////////////////////////
15          always @(posedge clk)
16          begin
17          qd=d;
18          qdbar=~qd;
19          end
20
21          ////////////JK FlipFlop////////////////////////
22          always @(posedge clk)
23          begin
24          qjk=(j&(qjkbar))|((~k)&qjk);
25          qjkbar=~qjk;
26
27
28          end
29
30          ////////T Flip Flop//////////////////
31          always @(posedge clk)
32          begin
33          if(tclr)
34                begin
35                qt=0;
36                qtbar=1;
37                end
38                else begin
39          qt=t^qt;
40          qtbar=~qt;
41          end
42          end
43
44
```

```
endmodule
```

Testbench

```
`timescale 1ns / 1ps


module ques8test;

        // Inputs
        reg s;
        reg r;
        reg clk;

        reg d;

        reg j;
        reg k;

        reg t;
        reg tclr;
        // Outputs
        wire qsr;
        wire qsrbar;

        wire qd;
        wire qdbar;

        wire qjk;
        wire qjkbar;

        wire qt;
        wire qtbar;

        // Instantiate the Unit Under Test (UUT)
        Ques8ffs uut (
                .s(s),
                .r(r),
                .qsr(qsr),
                .qsrbar(qsrbar),
                .clk(clk),

                .d(d),
                .qd(qd),
                .qdbar(qdbar),

                .j(j),
                .k(k),
                .qjk(qjk),
                .qjkbar(qjkbar),


                .t(t),
                .tclr(tclr),
                .qt(qt),
                .qtbar(qtbar)

        );
        initial
        begin
```

```
57          #10;
58          forever #10 clk=~clk;
59          end
60
61          initial begin
62                  // Initialize Inputs
63                  s = 0;
64                  r = 0;
65                  clk = 0;
66                  tclr=1;
67                  d=0;
68
69                  j=1;
70                  k=0;
71
72                  t=0;
73                  // Wait 100 ns for global reset to finish
74                  #30;
75                  s<=1;
76                  r<=0;
77                  tclr<=0;
78
79                  d<=1;
80
81                  j<=0;
82                  k<=1;
83                  t<=1;
84
85                  #20;
86                  s<=0;
87                  r<=1;
88                  d<=1;
89
90                  j<=1;
91                  k<=1;
92                  t<=0;
93                  #20;
94                  s<=0;
95                  r<=0;
96                  t<=1;
97                  d<=0;
98                  j<=0;
99                  k<=0;
100                 #20;
101
102
103       $finish;
104                 // Add stimulus here
105
106         end
107
108 endmodule
```

1.8.2 Question 8 : Implementation with T Flip Flops

The following code gives implementation of all Flip Flops using the T Flip Flop .The Coding has been done in the Dataflow and Structural Format.

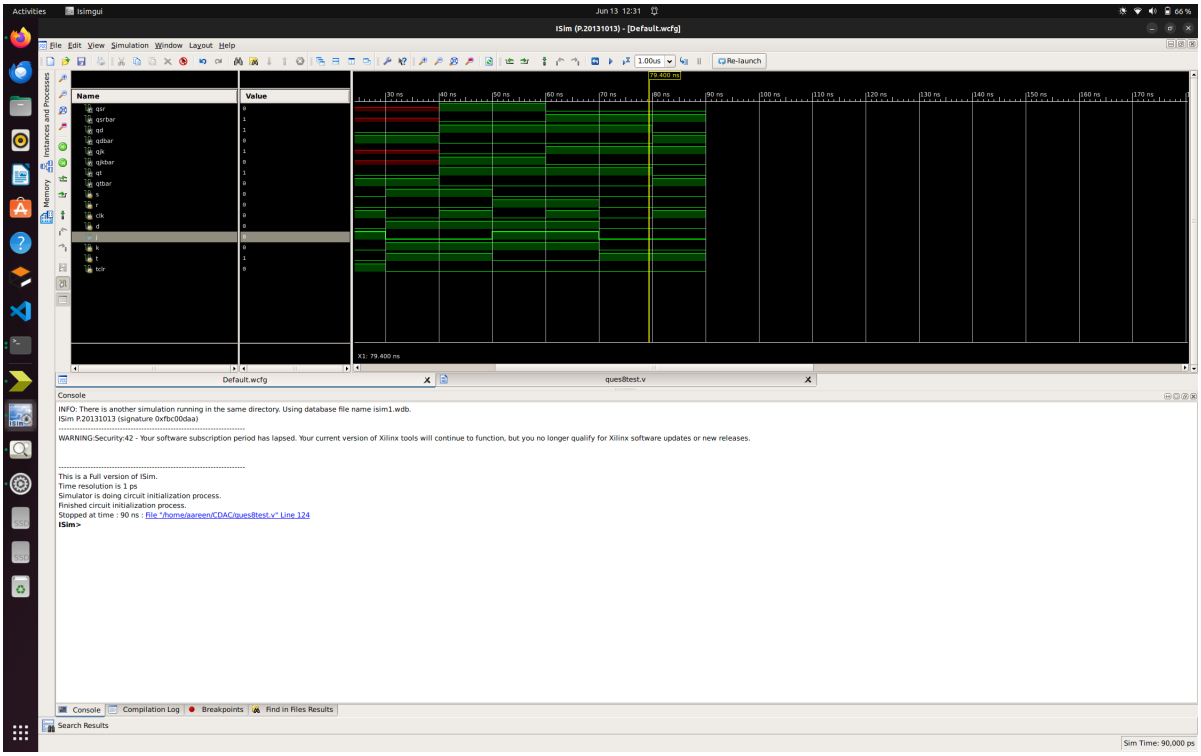


Fig. 31: Simulation for all Flip Flops

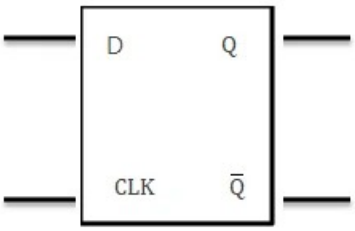


Fig. 32: Block Diagram of D Flip flop

```
1  `timescale 1ns / 1ps
2  module SRusingtff(
3      input s,
4      input r,
5      output qsr,
6      output qsrbar,
7      input clr,
8      input clk
9  );
10     wire w1;
11     assign w1 = (s & ~r & ~qsr) | (r & ~s & qsr);
12     Tff tff1(w1, clr, clk, qsr, qsrbar);
13 endmodule
```

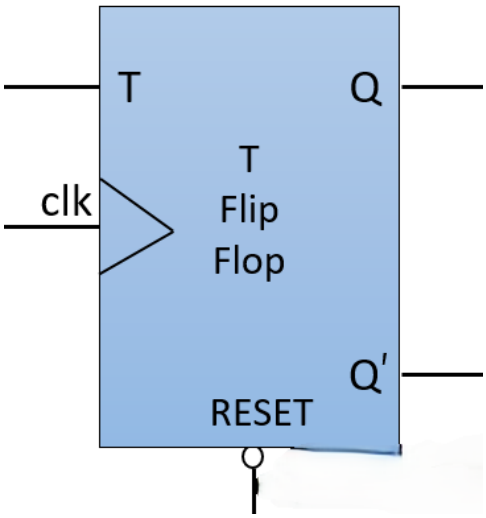


Fig. 33: Block Diagram of T Flip flop

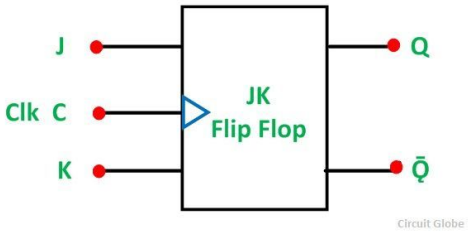


Fig. 34: Block Diagram of JK Flip flop

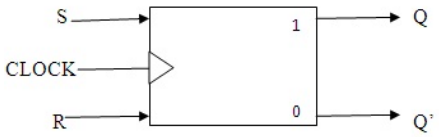


Fig. 35: Block Diagram of SR Flip flop

Testbench for SR using T Flip Flop

```
1  `timescale 1ns / 1ps
2
3
4
5  module SRUsingTffTestFile;
6
7          // Inputs
8          reg s;
9          reg r;
10         reg clr;
11         reg clk;
12
13         // Outputs
14         wire qsr;
15         wire qsrbar;
16
```

```
17          // Instantiate the Unit Under Test (UUT)
18          SRusingtff uut (
19                  .s(s),
20                  .r(r),
21                  .qsr(qsr),
22                  .qsrbar(qsrbar),
23                  .clr(clr),
24                  .clk(clk)
25          );
26          initial
27          begin
28          clk=0;
29          forever #10 clk=~clk;
30          end
31
32          initial begin
33                  // Initialize Inputs
34                  s = 0;
35                  r = 0;
36                  //clr = 0;
37                  clr=1;
38
39
40                  // Wait 100 ns for global reset to finish
41                  #50;
42                  clr=0;
43                  #50;
44                  s=1;
45                  r=0;
46                  #30
47                  s=1;
48                  r=0;
49                  #40;
50                  $finish;
51
52                  // Add stimulus here
53
54          end
55
56  endmodule
```

Code for JK Flip flop Modelled using T flip flop

```
1  `timescale 1ns / 1ps
2  module jkffTff(
3      input j,
4      input k,
5      input reset,
6      input clk,
7      output qjk,
8      output qjkbar
9      );
10
11           wire w3=(j&(~qjk))|(qjk&k);
12          Tff t3(w3,reset,clk,qjk,qjkbar);
13
14  endmodule
```

Testbench for JK Flip flop modelled using Tff

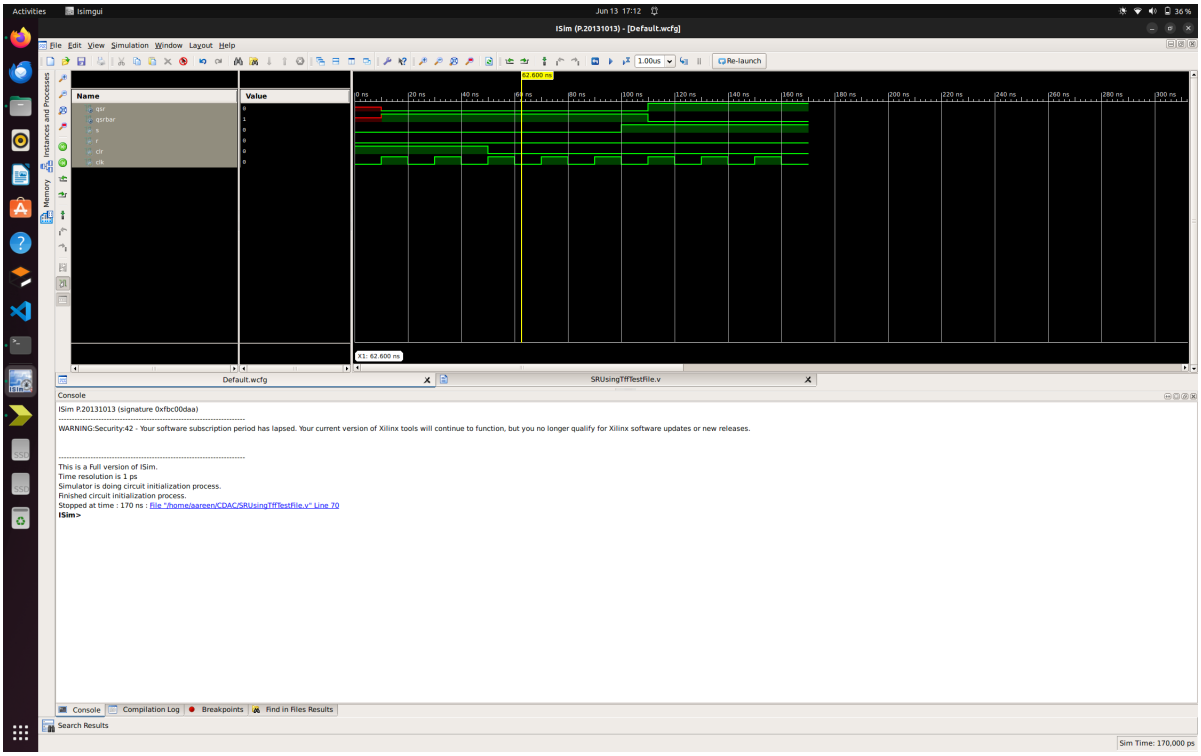


Fig. 36: Simulation of SR using T F.F.

```
1  `timescale 1ns / 1ps
2
3
4  module jkfftfftest;
5
6          // Inputs
7          reg j;
8          reg k;
9          reg reset;
10         reg clk;
11
12         // Outputs
13         wire qjk;
14         wire qjkbar;
15
16         // Instantiate the Unit Under Test (UUT)
17         jkffTff uut (
18                 .j(j),
19                 .k(k),
20                 .reset(reset),
21                 .clk(clk),
22                 .qjk(qjk),
23                 .qjkbar(qjkbar)
24         );
25         initial
26         begin
27         clk=0;
28         forever #10 clk=~clk;
29         end
30
```

```
31
32          initial begin
33                  // Initialize Inputs
34                  j = 0;
35                  k = 0;
36                  reset = 1;
37                  #20
38
39                  // Wait 100 ns for global reset to finish
40                  reset=0;
41                  j=0;
42                  k=1;
43                  #30;
44                  j=1;
45                  k=0;
46                  #30;
47                  j=1;
48                  k=1;
49                  #20;
50                  $finish;
51
52                  // Add stimulus here
53
54          end
55
56  endmodule
```

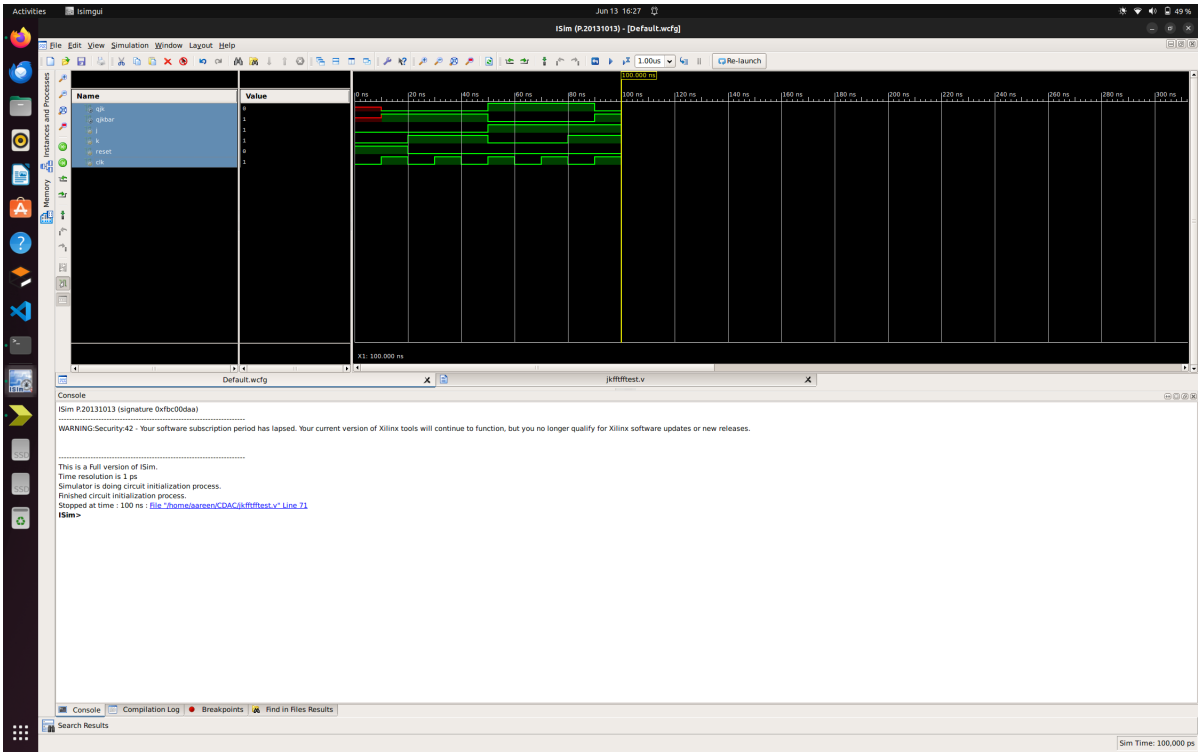


Fig. 37: Simulation of JK using T F.F.

Code for D F.F. using T F.F.

```
1  `timescale 1ns / 1ps
2  module DusingTff(
```

```
    input d,
    input clk,
    input reset,
    output qd,
    output qdbar
);

    wire t_in = d ^ qd;


    Tff tff_inst(t_in,reset,clk,qd,qdbar);
endmodule
```

Testbench

```
`timescale 1ns / 1ps


module DusingTffTest;

        // Inputs
        reg d;
        reg clk;
        reg reset;

        // Outputs
        wire qd;
        wire qdbar;

        // Instantiate the Unit Under Test (UUT)
        DusingTff uut (
                .d(d),
                .clk(clk),
                .reset(reset),
                .qd(qd),
                .qdbar(qdbar)
        );
        initial
        begin
        clk=0;
        #1;
        forever #10 clk=~clk;
        end

        initial begin
        // Initialize inputs
        d = 0;
        reset = 1;

        // Apply reset for 1 clock cycle
        #20 reset = 0; // Deassert reset at 20

        // Test case 1: d=1 (after reset)
        @(posedge clk); // Wait for next clock edge (time 30)
        d = 1;

        // Test case 2: d=0
        @(posedge clk); // Wait for next clock edge (time 50)
        d = 0;

```

```
46          // Test case 3: Re-apply reset
47          @(posedge clk); // Time 70
48          //reset = 1;
49          #20 reset = 0; // Release reset at 90
50
51          // Test case 4: d=1 after reset
52          @(posedge clk); // Time 90
53          d = 1;
54
55          // End simulation
56          #200 $finish; // Time 110
57
58          end
59
60  endmodule
```

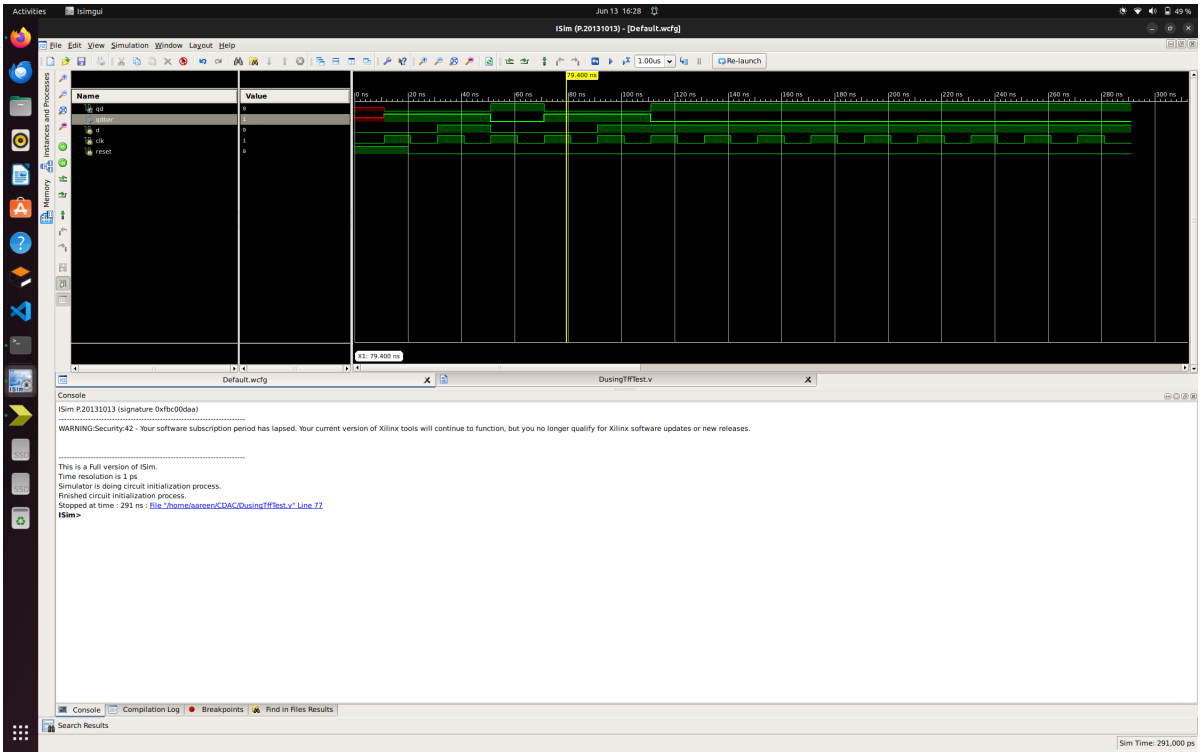


Fig. 38: Simulation of D using T F.F.

1.9 Mod-10 Counter

1.9.1 Mod-10 Asynchronous Counter

The Asynchronous Counter is modeled using Behavioral Modeling . It has one reset pin which resets it's count to 0.

Code :

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
```

```
// Engineer:
//
// Create Date:    19:11:45 06/13/2025
// Design Name:
// Module Name:    mod10asych
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module mod10asych(
    input reset,
    input clk,
    output reg [3:0] q
    );
         wire internal_reset;
    assign internal_reset = reset | (q == 4'b1010);

    /
    always @(negedge clk or posedge internal_reset) begin
        if (internal_reset) q[0] <= 0;
        else q[0] <= ~q[0];
    end

    always @(negedge q[0] or posedge internal_reset) begin
        if (internal_reset) q[1] <= 0;
        else q[1] <= ~q[1];
    end

    always @(negedge q[1] or posedge internal_reset) begin
        if (internal_reset) q[2] <= 0;
        else q[2] <= ~q[2];
    end

    always @(negedge q[2] or posedge internal_reset) begin
        if (internal_reset) q[3] <= 0;
        else q[3] <= ~q[3];
    end

endmodule
```

Testbench :

```
`timescale 1ns / 1ps


module mod10asynchTest;

        // Inputs
        reg reset;
        reg clk;

        // Outputs
```

```
        wire [3:0] q;

        // Instantiate the Unit Under Test (UUT)
        mod10asych uut (
                .reset(reset),
                .clk(clk),
                .q(q)
        );
        initial begin
        clk=0;
        forever #10 clk=~clk;
        end

        initial begin
                // Initialize Inputs
                reset = 1;
                //clk = 0;

                // Wait 100 ns for global reset to finish
                #100;
        reset=0;
                  #300;
                  reset=1;
                  #50;
                  reset=0;
                  #500;
                  $finish;

                // Add stimulus here

        end

endmodule
```

1.9.2 Mod-10 Synchronous Up Counter

The synchronous Mod-10 up counter has been designed using Behavioral Modeling.

Code :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:00:02 06/13/2025
// Design Name:
// Module Name:    syncMod10
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

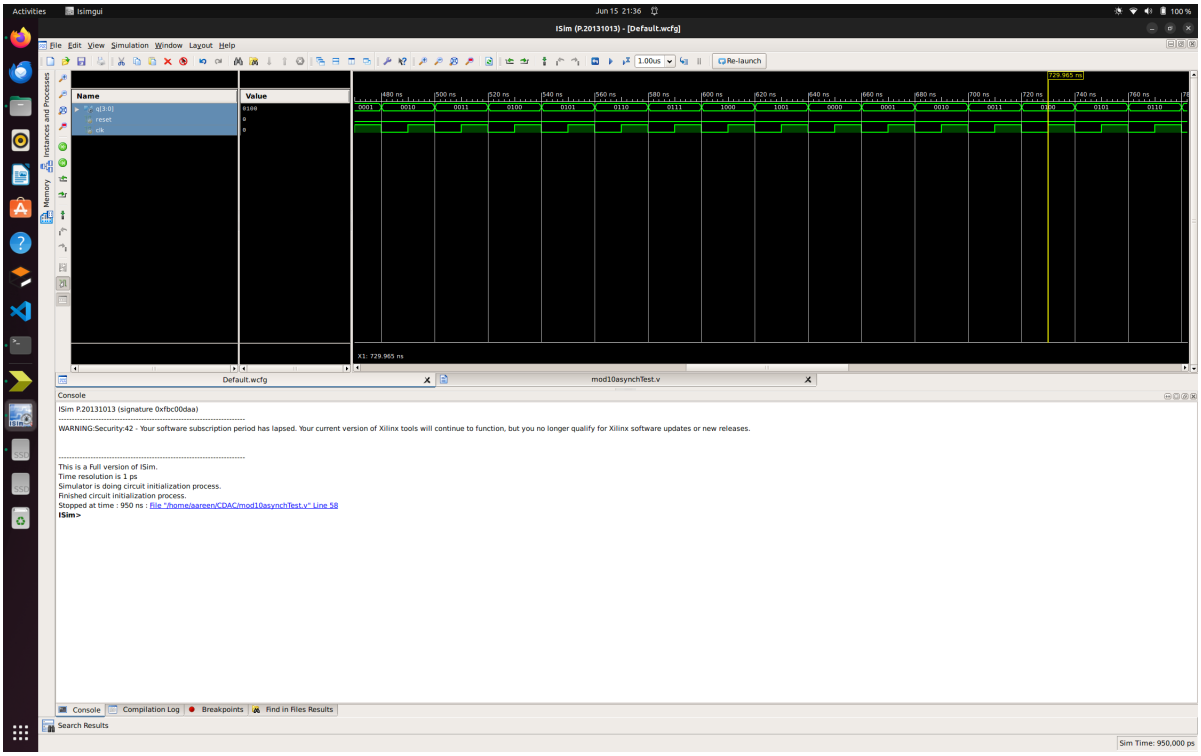


Fig. 39: Simulation of mod 10 asynch Counter

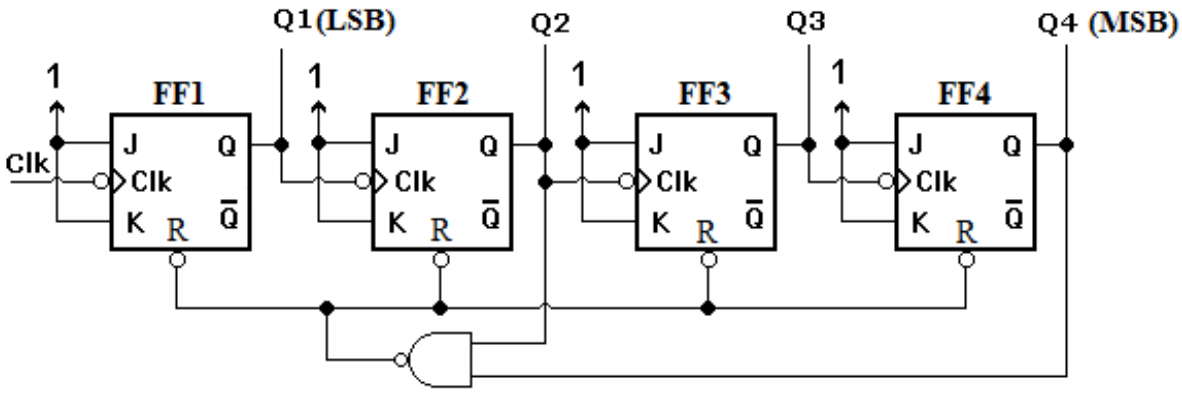


Fig. 40: Block Diagram of mod 10 asynch Counter

```
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module syncMod10(
22      input clk,input reset ,
23      output reg [3:0] q
24      );
25          always @(posedge clk)
26          begin
27          if(reset)
28          q=0;
29          else
30          q=(q+1)%10;
31          end
```

```
32
33 endmodule
```

Testbench

```
1  `timescale 1ns / 1ps
2
3  //////////////////////////////////////////////////////////////////////////////////
4  // Company:
5  // Engineer:
6  //
7  // Create Date:   19:02:42 06/13/2025
8  // Design Name:   syncMod10
9  // Module Name:   /home/aareen/CDAC/mod10syncTest.v
10 // Project Name:  CDAC
11 // Target Device:
12 // Tool versions:
13 // Description:
14 //
15 // Verilog Test Fixture created by ISE for module: syncMod10
16 //
17 // Dependencies:
18 //
19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 //////////////////////////////////////////////////////////////////////////////////
24
25 module mod10syncTest;
26
27         // Inputs
28         reg clk;
29         reg reset;
30
31         // Outputs
32         wire [3:0] q;
33
34         // Instantiate the Unit Under Test (UUT)
35         syncMod10 uut (
36                 .clk(clk),
37                 .reset(reset),
38                 .q(q)
39         );
40         initial begin
41         clk=0;
42         forever#5 clk=~clk;
43         end
44
45         initial begin
46                 // Initialize Inputs
47         //      clk = 0;
48                 reset = 1;
49
50                 // Wait 100 ns for global reset to finish
51                 #100;
52                 reset=0;
53
54                 #150;
55                 reset=1;
```

```
56                  #30;
57                  reset=0;
58                  #50;
59                  $finish;
60
61
62                  // Add stimulus here
63
64          end
65
66  endmodule
```

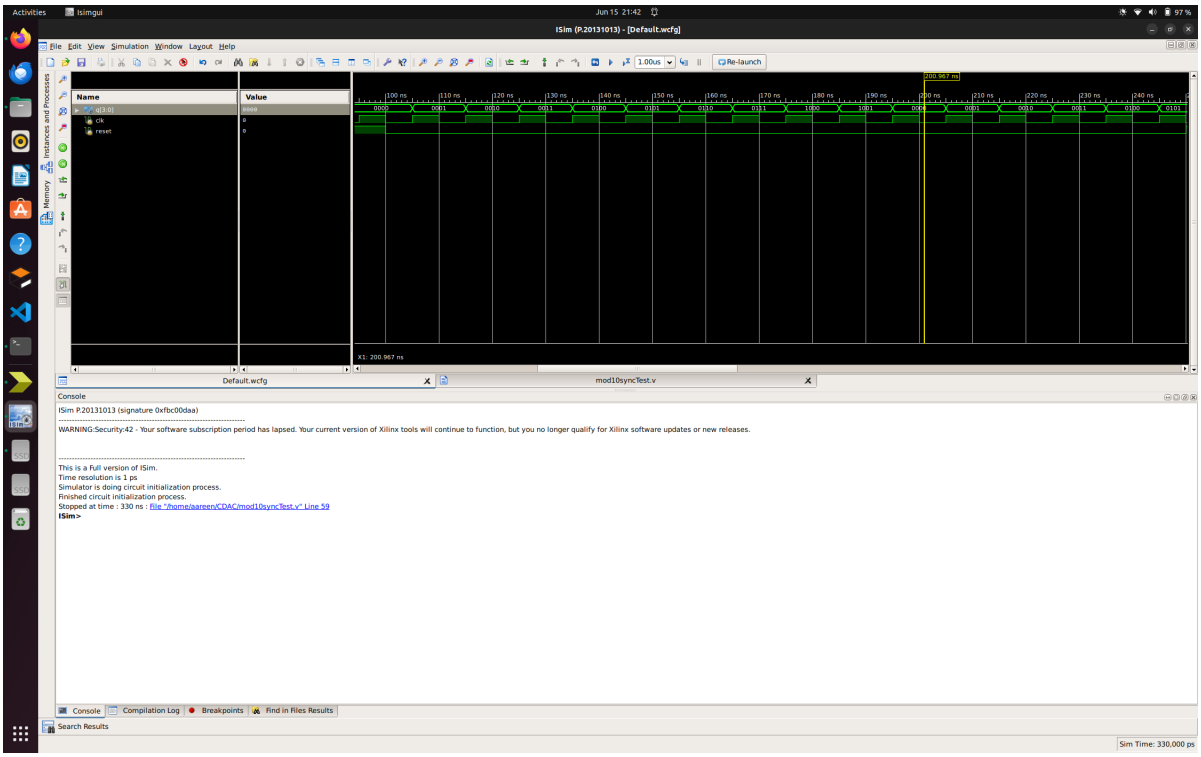


Fig. 41: Simulation of mod 10 Synch Counter

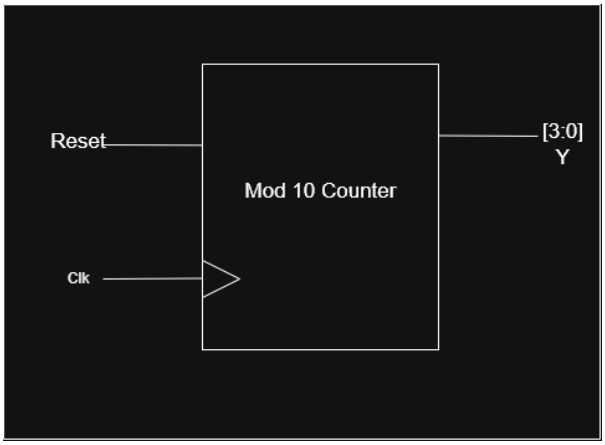


Fig. 42: Block Diagram of mod 10 Synch Counter

1.10 Question 10 : Ring and Johnson Counter

The Ring and Johnson counters were designed using Behavioral Modeling .

Code :

```
`timescale 1ns / 1ps

module johnsonCounter(
    output reg [3:0] q,
    input reset,
    input clk
    );
        always @(posedge clk)
        begin
                if(reset)
                q<=0;
                else
                q<={q[2:0],~q[3]};
        end

endmodule
```

```
`timescale 1ns / 1ps
module rincounter(
    input clk,
    input reset,
    output reg [3:0] q
    );
         always @(posedge clk)
         begin
         if(reset)
         q=1;
         else
         begin
         q[0]<=q[3];
         q[1]<=q[0];
         q[2]<=q[1];
         q[3]<=q[2];
         end

         end


endmodule
```

Testbenches:

```
`timescale 1ns / 1ps


```

```
module johnsonCounterTest;

        // Inputs
        reg reset;
        reg clk;

        // Outputs
        wire [3:0] q;

        // Instantiate the Unit Under Test (UUT)
        johnsonCounter uut (
                .q(q),
                .reset(reset),
                .clk(clk)
        );
        initial begin
        clk=0;
        forever #10 clk=~clk;
        end

        initial begin
                // Initialize Inputs
                reset = 1;
                //clk = 0;

                // Wait 100 ns for global reset to finish
                #100;
        reset=0;
                  #150;
                  reset=1;
                  #30;
                        reset=0;
                        #200;
                        $finish;
                // Add stimulus here

        end

endmodule
```

```
`timescale 1ns / 1ps



module ringCounterTest;

        // Inputs
        reg clk;
        reg reset;

        // Outputs
        wire [3:0] q;

        // Instantiate the Unit Under Test (UUT)
        rincounter uut (
                .clk(clk),
                .reset(reset),
```

```
18                      .q(q)
19              );
20              initial begin
21              clk=0;
22              forever #10 clk=~clk;
23              end
24
25              initial begin
26                      // Initialize Inputs
27                      //clk = 0;
28                      reset = 1;
29
30                      // Wait 100 ns for global reset to finish
31                      #100;
32                      reset=0;
33                      #50
34                      reset=1;
35                      #50
36                      reset=0;
37                      #50
38                      $finish;
39
40
41                      // Add stimulus here
42
43              end
44
45      endmodule
```

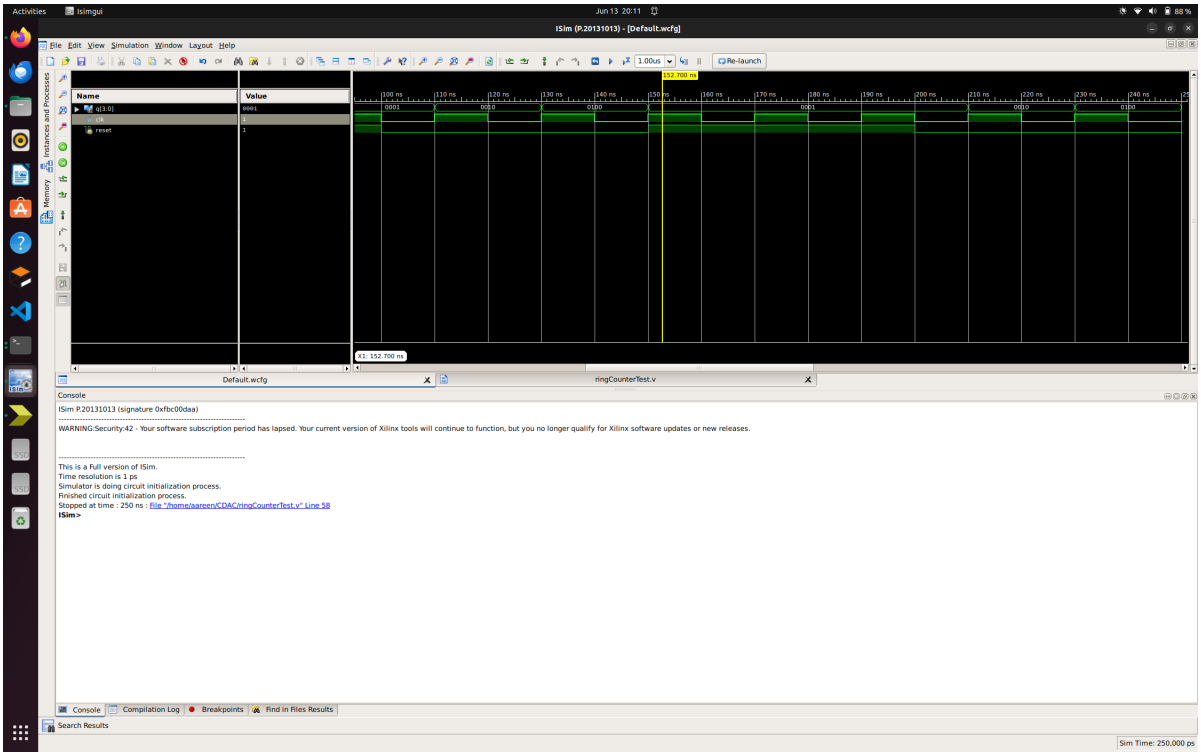


Fig. 43: Simulation of Ring Counter

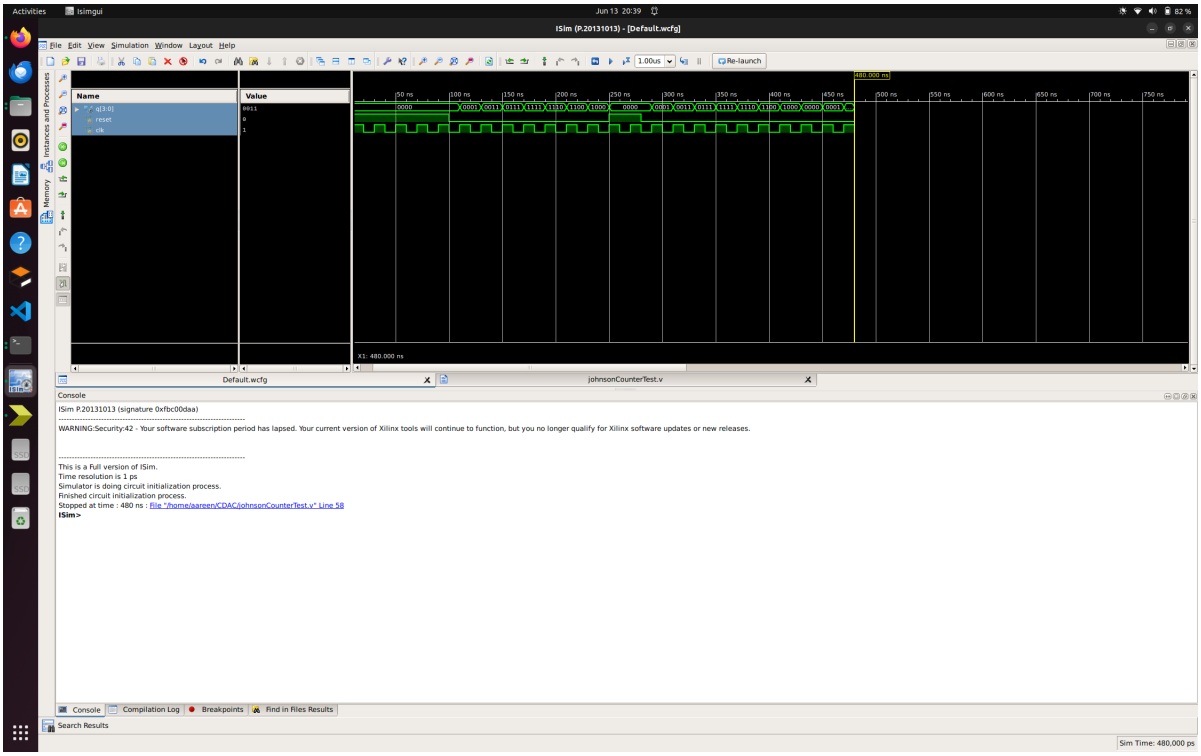


Fig. 44: Simulation of Johnson Counter

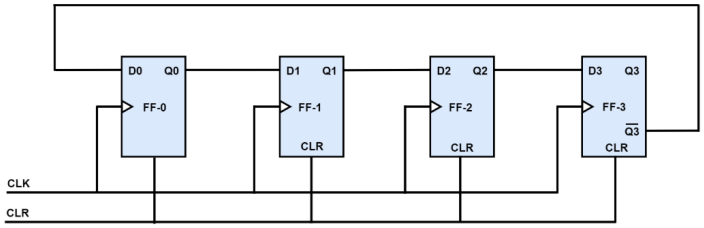


Fig. 45: Block Diagram of Johnson Counter

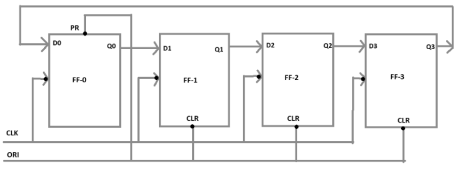


Fig. 46: Block Diagram Ring Counter

1.11 Question 11 : Universal Shift Register

The universal Shift Register has 4 modes , serial right shift , serial left shift, parallel load and memory mode in which it keeps the written data stored in itself. The designing has been done using Behavioral

style of Modeling . Code :

```
`timescale 1ns / 1ps
module UniShiftReg(
    output reg[3:0] q,
    input serialright,
    input serialleft,
    input [3:0] in,
         input clk,
         input clr,
         input [1:0]sel
    );

         always @(posedge clk)
         begin
         if(clr)
         q=0;
         else
                case(sel)
                        2'b00:begin
                                        q<=q;
                                        end
                        2'b01:begin///shift right
                                                q<={q[2:0],serialright};

                                        end

                        2'b10: begin
                                        q<={serialleft,q[3:1]};
                                                end
                        2'b11: begin//parallel load
                                                q<=in;
                                 end
                                 endcase


         end


endmodule
```

```
`timescale 1ns / 1ps


module UniShiftRegTest;

        // Inputs
        reg serialright;
        reg serialleft;
        reg [3:0] in;
        reg clk;
        reg clr;
        reg [1:0] sel;
```

```
13
14          // Outputs
15          wire [3:0] q;
16
17          // Instantiate the Unit Under Test (UUT)
18          UniShiftReg uut (
19                  .q(q),
20                  .serialright(serialright),
21                  .serialleft(serialleft),
22                  .in(in),
23                  .clk(clk),
24                  .clr(clr),
25                  .sel(sel)
26          );
27          initial
28          begin
29          clk=0;
30          forever #10 clk=~clk;
31
32          end
33
34          initial begin
35                  // Initialize Inputs
36                  serialright = 0;
37                  serialleft = 0;
38                  in = 0;
39                  clk = 0;
40                  clr = 1;
41                  sel = 0;
42
43                  // Wait 100 ns for global reset to finish
44                  #60;
45                  clr=0;
46                  sel<=3;
47                  in<=4'b0110;
48                  #20;
49                  serialright=1;
50                  sel=1;
51                  #60;
52                  sel=0;
53                  #40;
54                  clr=1;
55                  #20
56                  clr=0;
57                  #30
58                  sel=2;
59                  serialleft=1;
60                  #90;
61                  sel=0;
62                  #50;
63
64
65
66                  $finish;
67
68                  // Add stimulus here
69
70          end
71
72  endmodule
```

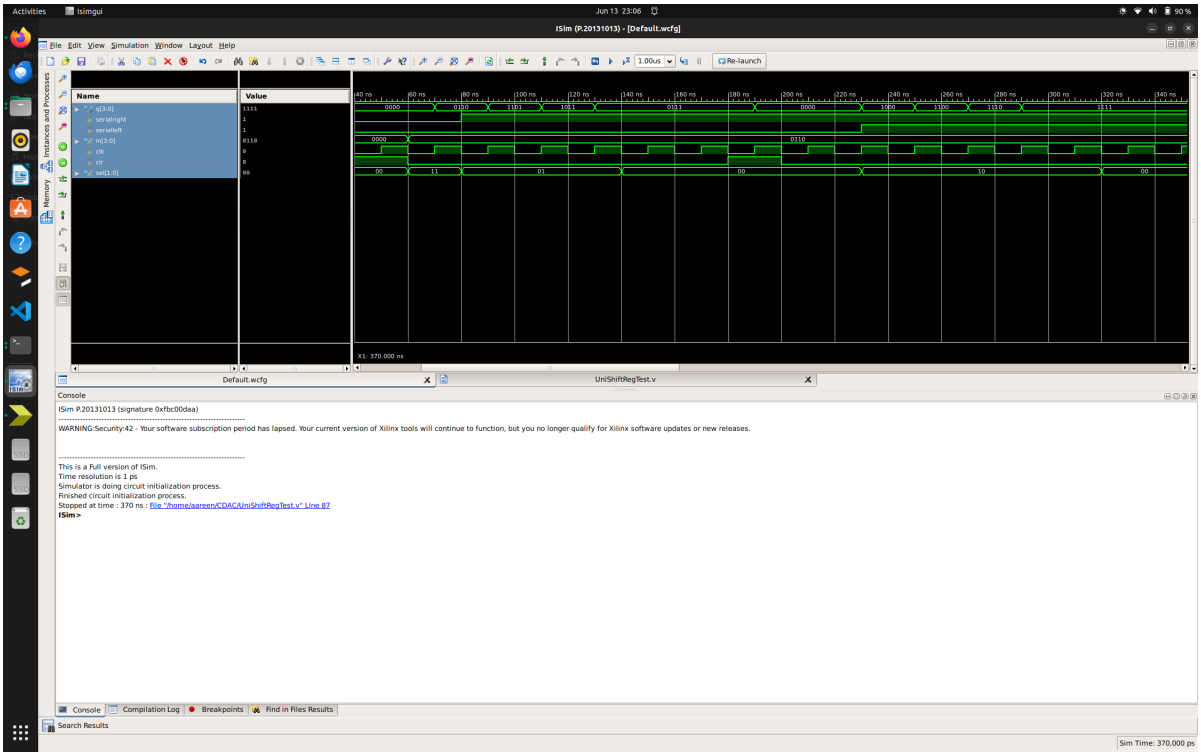


Fig. 47: Simulation of Univarsal Shift Register

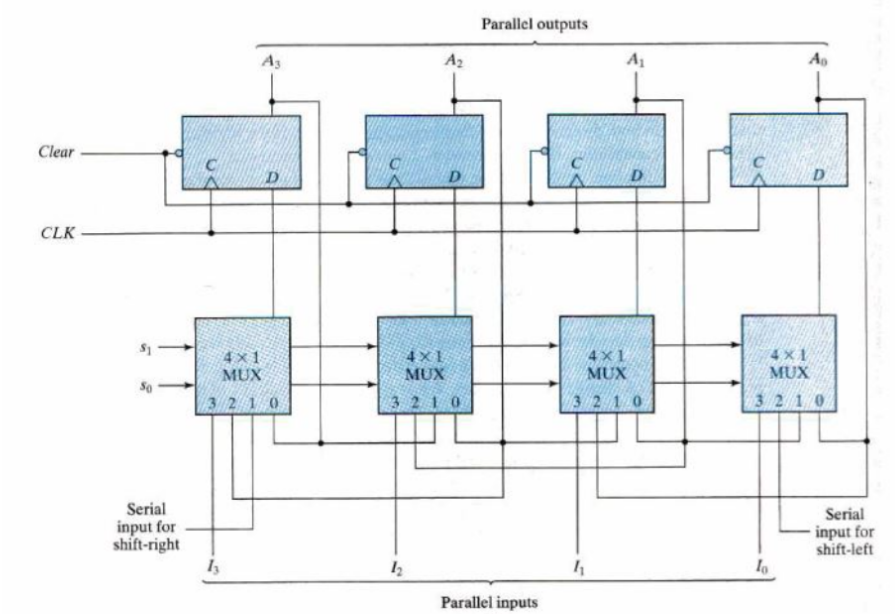


Fig. 48: Block Diagram of Univarsal Shift Register

1.12 Sequence Detector

1.12.1 Mealy Machine 10011 : Non - Overlapping Case

Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////

module Mealy10011NonOverlapping(
    input clk,
    input reset,
    input din,
    output reg seq_detected
    );
localparam S0 = 3'b000,
             S1 = 3'b001,
             S2 = 3'b010,
             S3 = 3'b011,
             S4 = 3'b100;

  reg [2:0] state;

  always @(posedge clk or posedge reset) begin
    if (reset) begin
      state <= S0;
      seq_detected <= 1'b0;
    end else begin
      case (state)
        S0: begin
          if (din)
            state <= S1;
          else
            state <= S0;
          seq_detected <= 1'b0;
        end

        S1: begin
          if (~din)
            state <= S2;
          else
            state <= S1;
          seq_detected <= 1'b0;
        end

        S2: begin
          if (~din)
            state <= S3;
          else
            state <= S1;
          seq_detected <= 1'b0;
        end

        S3: begin
          if (din)
            state <= S4;
          else
            state <= S0;
          seq_detected <= 1'b0;
```

```
54          end
55
56          S4: begin
57            if (din) begin
58              seq_detected <= 1'b1;
59              state <= S0;
60            end else begin
61              seq_detected <= 1'b0;
62              state <= S0;
63            end
64          end
65
66          default: begin
67            state <= S0;
68            seq_detected <= 1'b0;
69          end
70        endcase
71      end
72    end
73
74  endmodule
```

Testbench:

```
1   `timescale 1ns / 1ps
2
3   ////////////////////////////////////////////////////////////////////////////////
4
5
6   module nonOverMoore10011;
7
8           // Inputs
9           reg clk;
10          reg reset;
11          reg din;
12
13          // Outputs
14          wire seq_detected;
15
16          // Instantiate the Unit Under Test (UUT)
17          nonOverlappingMoore10011 uut (
18                  .clk(clk),
19                  .reset(reset),
20                  .din(din),
21                  .seq_detected(seq_detected)
22          );
23
24          initial begin
25      clk = 0;
26      forever #5 clk = ~clk;
27    end
28
29    initial begin
30      reset = 1;
31      din = 0;
32      #12;
33      reset = 0;
34
35      din = 1; #10;
36      din = 0; #10;
```

```
37        din = 0; #10;
38        din = 1; #10;
39        din = 1; #10;
40        din = 0; #10;
41        din = 0; #10;
42        din = 1; #10;
43        din = 1; #10;
44        din = 1; #10;
45
46        #20;
47        $stop;
48      end
49
50  endmodule
```

Simulation:

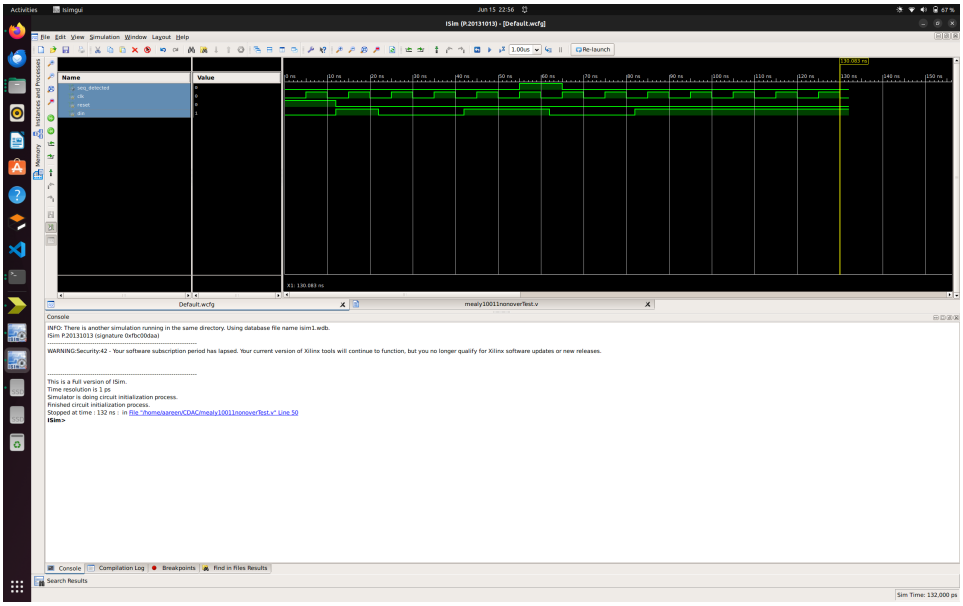


Fig. 49: Simulation for the Mealy Non - Overlapping case

State Diagram :

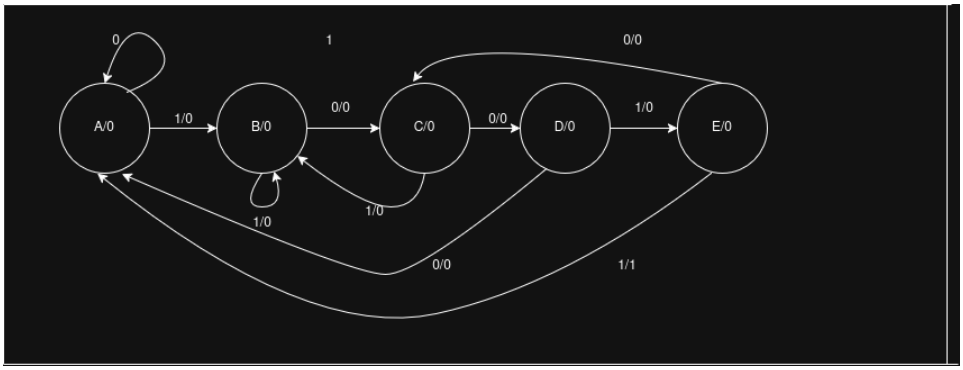


Fig. 50: State Diagram for the Mealy Non Overlapping case

1.12.2 Mealy Machine 10011 : Overlapping Case

Code :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////

module Mealy10011Overlapping(
input clk,
    input reset,
    input din,
    output reg seq_detected
    );
         localparam S0 = 3'b000,
             S1 = 3'b001,
             S2 = 3'b010,
             S3 = 3'b011,
             S4 = 3'b100;

  reg [2:0] state;

  always @(posedge clk or posedge reset) begin
    if (reset) begin
      state <= S0;
      seq_detected <= 1'b0;
    end else begin
      case (state)
        S0: begin
          if (din)
            state <= S1;
          else
            state <= S0;
          seq_detected <= 1'b0;
        end

        S1: begin
          if (~din)
            state <= S2;
          else
            state <= S1;
          seq_detected <= 1'b0;
        end

        S2: begin
          if (~din)
            state <= S3;
          else
            state <= S1;
          seq_detected <= 1'b0;
        end

        S3: begin
          if (din)
            state <= S4;
          else
            state <= S0;
          seq_detected <= 1'b0;
        end

```

```
56          S4: begin
57            if (din) begin
58              seq_detected <= 1'b1;
59              state <= S1;
60            end else begin
61              seq_detected <= 1'b0;
62              state <= S2;
63            end
64          end
65
66          default: begin
67            state <= S0;
68            seq_detected <= 1'b0;
69          end
70        endcase
71      end
72    end
73
74
75  endmodule
```

Testbench :

```
1   `timescale 1ns / 1ps
2
3   ////////////////////////////////////////////////////////////////////////////////
4
5
6   module nonOverMoore10011;
7
8           // Inputs
9           reg clk;
10          reg reset;
11          reg din;
12
13          // Outputs
14          wire seq_detected;
15
16          // Instantiate the Unit Under Test (UUT)
17          nonOverlappingMoore10011 uut (
18                  .clk(clk),
19                  .reset(reset),
20                  .din(din),
21                  .seq_detected(seq_detected)
22          );
23
24          initial begin
25      clk = 0;
26      forever #5 clk = ~clk;
27    end
28
29    initial begin
30      reset = 1;
31      din = 0;
32      #12;
33      reset = 0;
34
35      din = 1; #10;
36      din = 0; #10;
37      din = 0; #10;
```

```
38      din = 1; #10;
39      din = 1; #10;
40      din = 0; #10;
41      din = 0; #10;
42      din = 1; #10;
43      din = 1; #10;
44      din = 1; #10;
45
46      #20;
47      $stop;
48    end
49
50  endmodule
```

Simulation:

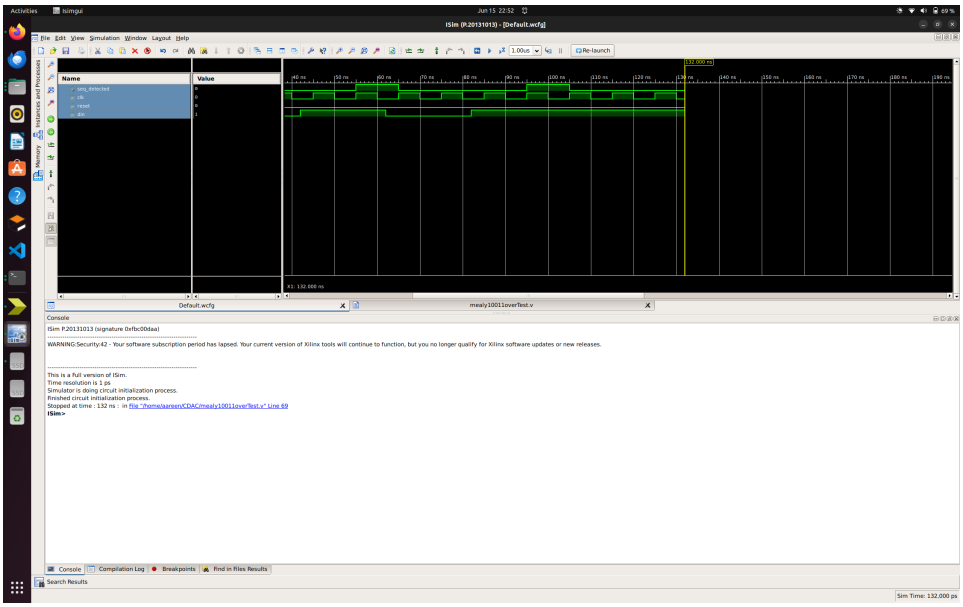


Fig. 51: Simulation for the Mealy Overlapping case

State Diagram :

1.12.3 Moore Machine 10011 : Non Overlapping Case

Code :

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    22:27:41 06/15/2025
7  // Design Name:
8  // Module Name:    nonOverlappingMoore10011
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
```

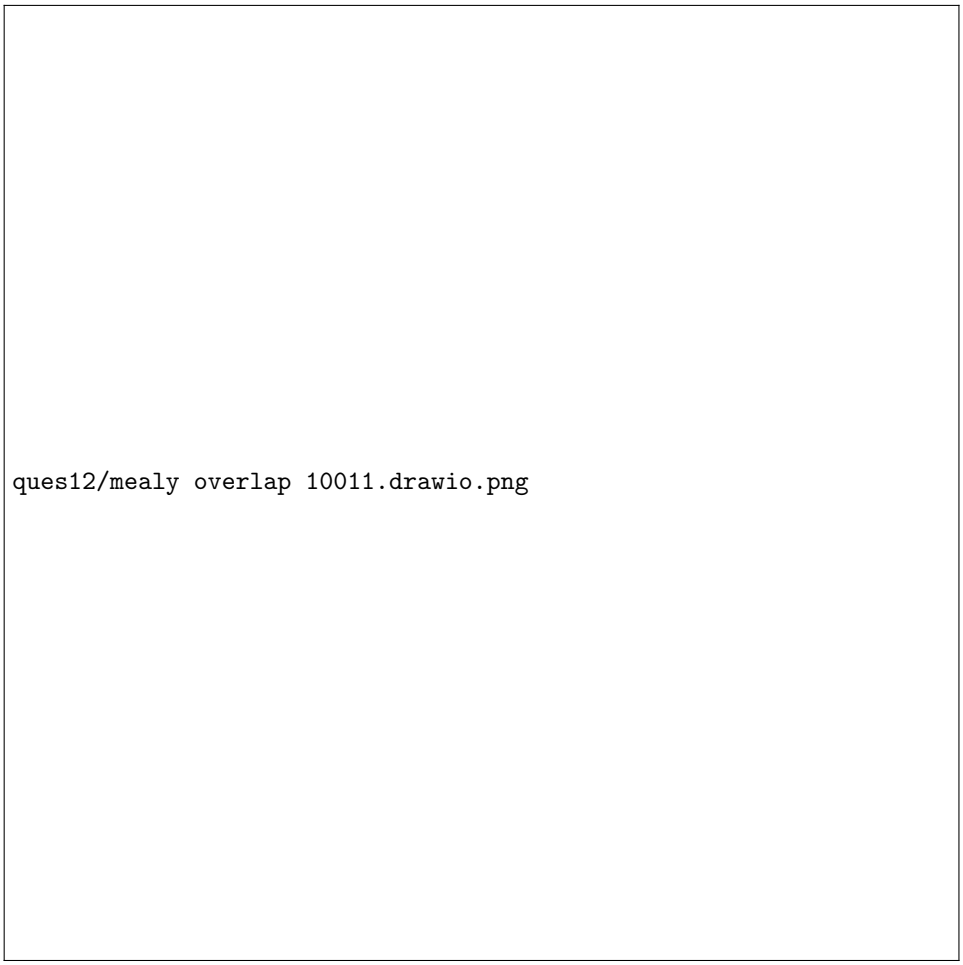


Fig. 52: State Diagram for the Mealy Overlapping case

```
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module nonOverlappingMoore10011(
22     input clk,
23     input reset,
24     input din,
25     output reg seq_detected
26     );
27
28   localparam S0 = 3'b000,
29              S1 = 3'b001,
30              S2 = 3'b010,
31              S3 = 3'b011,
32              S4 = 3'b100,
33              S5 = 3'b101;
34
35   reg [2:0] state;
```

```
36
37    always @(posedge clk or posedge reset) begin
38      if (reset) begin
39        state <= S0;
40        seq_detected <= 1'b0;
41      end
42      else begin
43        case (state)
44          S0: begin
45            seq_detected <= 1'b0;
46            if (din)
47              state <= S1;
48            else
49              state <= S0;
50          end
51
52          S1: begin
53            seq_detected <= 1'b0;
54            if (~din)
55              state <= S2;
56            else
57              state <= S1;
58          end
59
60          S2: begin
61            seq_detected <= 1'b0;
62            if (~din)
63              state <= S3;
64            else
65              state <= S1;
66          end
67
68          S3: begin
69            seq_detected <= 1'b0;
70            if (din)
71              state <= S4;
72            else
73              state <= S0;
74          end
75
76          S4: begin
77            seq_detected <= 1'b0;
78            if (din)
79              state <= S5;
80            else
81              state <= S2;
82          end
83
84          S5: begin
85            seq_detected <= 1'b1;
86            if (din)
87              state <= S1;
88            else
89              state <= S0;
90          end
91
92          default: begin
93            state <= S0;
94            seq_detected <= 1'b0;
95          end
```

```
96        endcase
97      end
98    end
99
100 endmodule
```

Testbench :

```
1  `timescale 1ns / 1ps
2
3  ////////////////////////////////////////////////////////////////////////////////
4
5
6  module non0verMoore10011;
7
8          // Inputs
9          reg clk;
10         reg reset;
11         reg din;
12
13         // Outputs
14         wire seq_detected;
15
16         // Instantiate the Unit Under Test (UUT)
17         non0verlappingMoore10011 uut (
18                 .clk(clk),
19                 .reset(reset),
20                 .din(din),
21                 .seq_detected(seq_detected)
22         );
23
24         initial begin
25     clk = 0;
26     forever #5 clk = ~clk;
27   end
28
29   initial begin
30     reset = 1;
31     din = 0;
32     #12;
33     reset = 0;
34
35     din = 1; #10;
36     din = 0; #10;
37     din = 0; #10;
38     din = 1; #10;
39     din = 1; #10;
40     din = 0; #10;
41     din = 0; #10;
42     din = 1; #10;
43     din = 1; #10;
44     din = 1; #10;
45
46     #20;
47     $stop;
48   end
49
50 endmodule
```

Simulation :

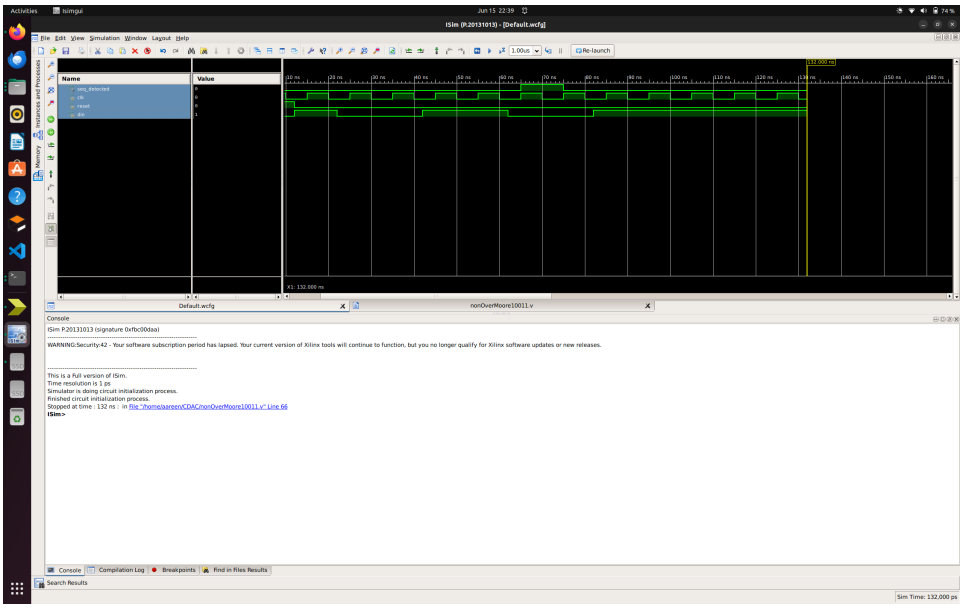


Fig. 53: Simulation for the Moore Non - Overlapping case

1.12.4 Moore Machine 10011 : Overlapping Case

Code:

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    22:17:04 06/15/2025
7  // Design Name:
8  // Module Name:    mooreover10011
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module mooreover10011(
22     input clk,
23     input reset,
24     input din,
25     output reg seq_detected
26 );
27
28   localparam S0 = 3'b000,
29              S1 = 3'b001,
```

```
30              S2 = 3'b010,
31              S3 = 3'b011,
32              S4 = 3'b100,
33              S5 = 3'b101;
34
35  reg [2:0] state;
36
37  always @(posedge clk or posedge reset) begin
38    if (reset) begin
39      state <= S0;
40      seq_detected <= 1'b0;
41    end
42    else begin
43      case (state)
44        S0: begin
45          seq_detected <= 1'b0;
46          if (din)
47            state <= S1;
48          else
49            state <= S0;
50        end
51
52        S1: begin
53          seq_detected <= 1'b0;
54          if (~din)
55            state <= S2;
56          else
57            state <= S1;
58        end
59
60        S2: begin
61          seq_detected <= 1'b0;
62          if (~din)
63            state <= S3;
64          else
65            state <= S1;
66        end
67
68        S3: begin
69          seq_detected <= 1'b0;
70          if (din)
71            state <= S4;
72          else
73            state <= S0;
74        end
75
76        S4: begin
77          seq_detected <= 1'b0;
78          if (din)
79            state <= S5;
80          else
81            state <= S2;  // allows overlap like 1001_001
82        end
83
84        S5: begin
85          seq_detected <= 1'b1;
86          if (din)
87            state <= S1;  // allows overlap if last 1 can start new match
88          else
89            state <= S2;  // because 0 could be part of 10...
```

```
        end

        default: begin
          state <= S0;
          seq_detected <= 1'b0;
        end
      endcase
    end
  end



endmodule
```

Testbench: (To verify cprrect operation I have used the same testbench for both Overlapping and Non Overlapping cases)

```
`timescale 1ns / 1ps

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


module nonOverMoore10011;

        // Inputs
        reg clk;
        reg reset;
        reg din;

        // Outputs
        wire seq_detected;

        // Instantiate the Unit Under Test (UUT)
        nonOverlappingMoore10011 uut (
                .clk(clk),
                .reset(reset),
                .din(din),
                .seq_detected(seq_detected)
        );

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

  initial begin
    reset = 1;
    din = 0;
    #12;
    reset = 0;

    din = 1; #10;
    din = 0; #10;
    din = 0; #10;
    din = 1; #10;
    din = 1; #10;
    din = 0; #10;
    din = 0; #10;
    din = 1; #10;
```

```
43      din = 1; #10;
44      din = 1; #10;
45
46      #20;
47      $stop;
48    end
49
50  endmodule
```

Simulation :

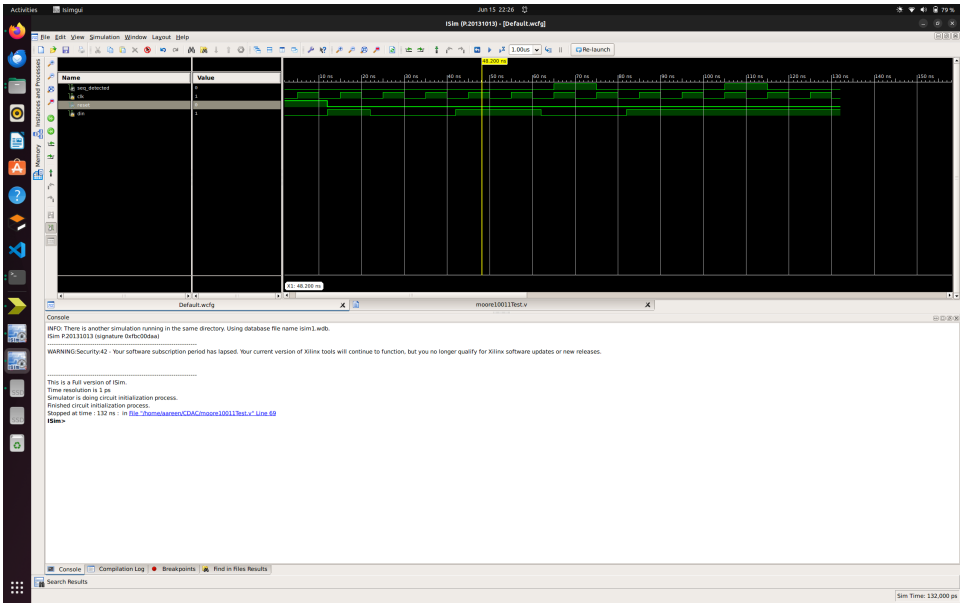


Fig. 54: Moore Overlapping Case

state Diagram

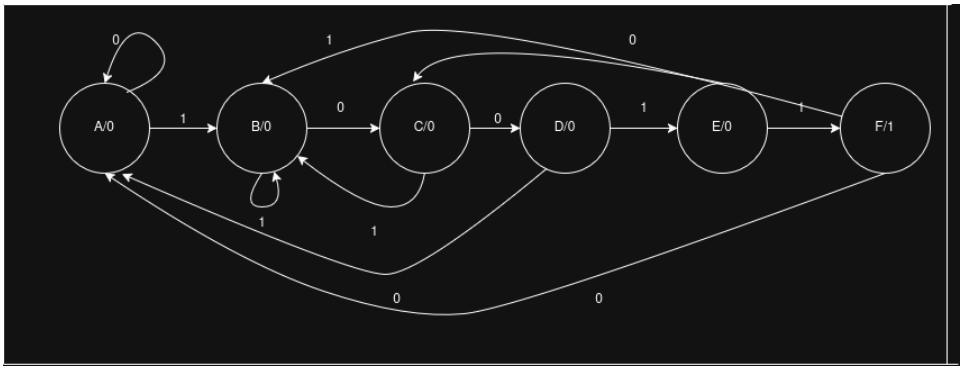


Fig. 55: State Diagram for the Moore Non Overlapping case

1.12.5 Mealy Machine 10101 : Overlapping Case

Code :

```
1  `timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////////////////
module MealyOver10101(
input clk,
    input reset,
    input din,
    output reg seq_detected
    );
localparam S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3, S4 = 3'd4;
  reg [2:0] state;

  always @(posedge clk or posedge reset) begin
    if (reset) begin
      state <= S0;
      seq_detected <= 0;
    end else begin
      case (state)
        S0: begin
          seq_detected <= 0;
          state <= din ? S1 : S0;
        end
        S1: begin
          seq_detected <= 0;
          state <= din ? S1 : S2;
        end
        S2: begin
          seq_detected <= 0;
          state <= din ? S3 : S0;
        end
        S3: begin
          seq_detected <= 0;
          state <= din ? S1 : S4;
        end
        S4: begin
          seq_detected <= din ? 1'b1 : 1'b0;
          state <= din ? S3: S0;
        end
        default: begin
          state <= S0;
          seq_detected <= 0;
        end
      endcase
    end
  end


endmodule
```

Testbench :

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   23:48:02 06/15/2025
// Design Name:   MealyNonOver10101
// Module Name:   /home/aareen/CDAC/mealyNonover10101.v
// Project Name:  CDAC
// Target Device:
```

```
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: MealyNonOver10101
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module mealyNonover10101;

        // Inputs
        reg clk;
        reg reset;
        reg din;

        // Outputs
        wire seq_detected;

        // Instantiate the Unit Under Test (UUT)
        MealyNonOver10101 uut (
                .clk(clk),
                .reset(reset),
                .din(din),
                .seq_detected(seq_detected)
        );

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

  initial begin
    reset = 1;
    din = 0;
    #10 reset = 0;
#2;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
         din = 0; #10;
         din = 1;

    #20;
  end

endmodule
```

Simulation :

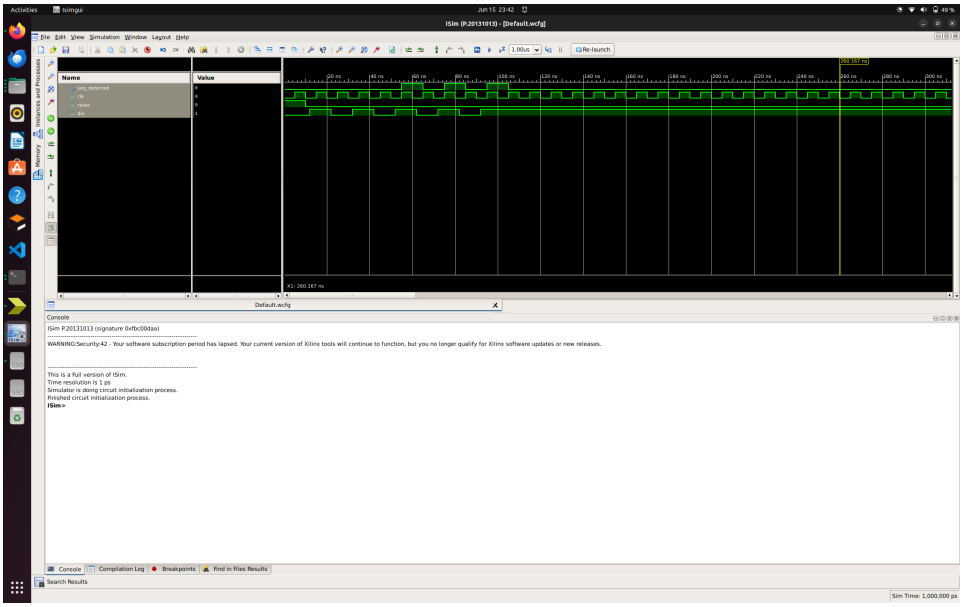


Fig. 56: Mealy Overlapping Case 10101

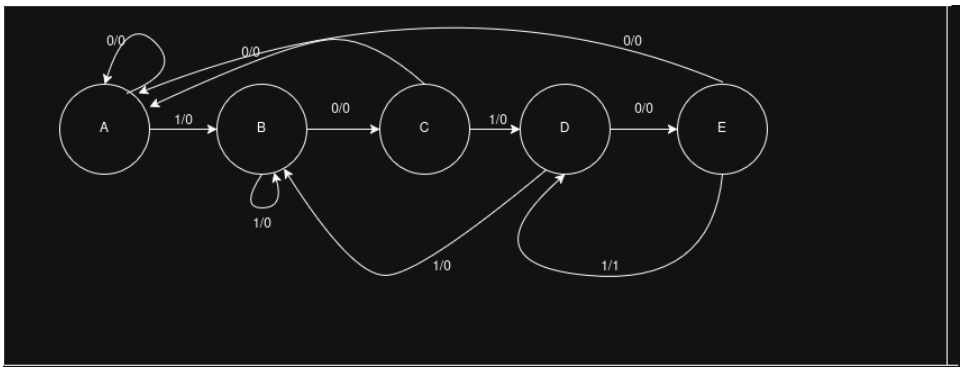


Fig. 57: State Diagram for the Mealy Overlapping case 10101

1.12.6 Mealy Machine 10101 : Non Overlapping Case

Code :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module MealyNonOver10101(
input clk,
    input reset,
    input din,
    output reg seq_detected
    );
         localparam S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3, S4 = 3'd4;
  reg [2:0] state;

  always @(posedge clk or posedge reset) begin
    if (reset) begin
      state <= S0;
      seq_detected <= 0;
    end else begin
```

```
17        case (state)
18          S0: begin
19            seq_detected <= 0;
20            state <= din ? S1 : S0;
21          end
22          S1: begin
23            seq_detected <= 0;
24            state <= din ? S1 : S2;
25          end
26          S2: begin
27            seq_detected <= 0;
28            state <= din ? S3 : S0;
29          end
30          S3: begin
31            seq_detected <= 0;
32            state <= din ? S1 : S4;
33          end
34          S4: begin
35            if (din) begin
36              seq_detected <= 1'b1;
37              state <= S0;
38            end else begin
39              seq_detected <= 0;
40              state <= S0;
41            end
42          end
43          default: begin
44            state <= S0;
45            seq_detected <= 0;
46          end
47        endcase
48      end
49    end
50
51
52
53  endmodule
```

Testbench :

```
1   `timescale 1ns / 1ps
2
3   ////////////////////////////////////////////////////////////////////////////////
4   // Company:
5   // Engineer:
6   //
7   // Create Date:   23:48:02 06/15/2025
8   // Design Name:   MealyNonOver10101
9   // Module Name:   /home/aareen/CDAC/mealyNonover10101.v
10  // Project Name:  CDAC
11  // Target Device:
12  // Tool versions:
13  // Description:
14  //
15  // Verilog Test Fixture created by ISE for module: MealyNonOver10101
16  //
17  // Dependencies:
18  //
19  // Revision:
20  // Revision 0.01 - File Created
```

```
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module mealyNonover10101;

        // Inputs
        reg clk;
        reg reset;
        reg din;

        // Outputs
        wire seq_detected;

        // Instantiate the Unit Under Test (UUT)
        MealyNonOver10101 uut (
                .clk(clk),
                .reset(reset),
                .din(din),
                .seq_detected(seq_detected)
        );

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

  initial begin
    reset = 1;
    din = 0;
    #10 reset = 0;
#2;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
         din = 0; #10;
         din = 1;

    #20;
  end

endmodule
```

Simulation :

State Diagram

1.12.7 Moore Machine 10101 : Overlapping Case

Code :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
```

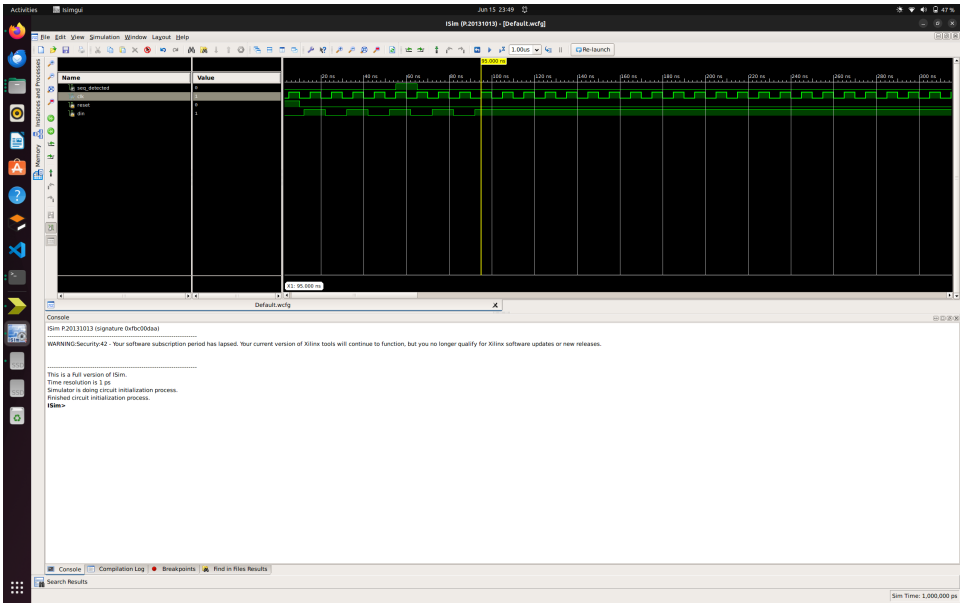


Fig. 58: Mealy Non Overlapping Case 10101

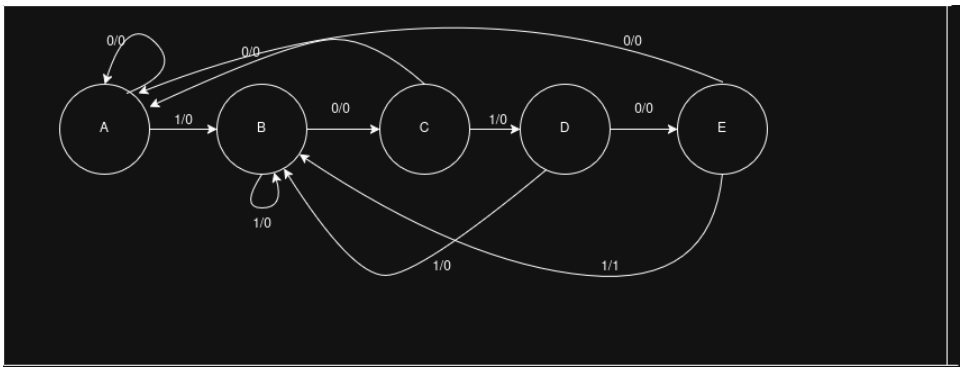


Fig. 59: Mealy Non Overlapping Case 10101

```
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    23:10:37 06/15/2025
7  // Design Name:
8  // Module Name:    MooreOver10101
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module MooreOver10101(
22 input clk,
```

```
23      input reset,
24      input din,
25      output reg seq_detected
26      );
27
28            localparam S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3, S4 = 3'd4, S5 = 3'd5;
29    reg [2:0] state;
30
31    always @(posedge clk or posedge reset) begin
32      if (reset) begin
33        state <= S0;
34        seq_detected <= 0;
35      end else begin
36        case (state)
37          S0: state <= din ? S1 : S0;
38          S1: state <= din ? S1 : S2;
39          S2: state <= din ? S3 : S0;
40          S3: state <= din ? S1 : S4;
41          S4: state <= din ? S5 : S0;
42          S5: state <= din ? S1 : S4;  // Overlapping restarts
43          default: state <= S0;
44        endcase
45
46        seq_detected = (state == S5);
47      end
48    end
49
50
51  endmodule
```

Testbench :

```
1   `timescale 1ns / 1ps
2
3   ////////////////////////////////////////////////////////////////////////////////
4   // Company:
5   // Engineer:
6   //
7   // Create Date:   23:48:02 06/15/2025
8   // Design Name:   MealyNonOver10101
9   // Module Name:   /home/aareen/CDAC/mealyNonover10101.v
10  // Project Name:  CDAC
11  // Target Device:
12  // Tool versions:
13  // Description:
14  //
15  // Verilog Test Fixture created by ISE for module: MealyNonOver10101
16  //
17  // Dependencies:
18  //
19  // Revision:
20  // Revision 0.01 - File Created
21  // Additional Comments:
22  //
23  ////////////////////////////////////////////////////////////////////////////////
24
25  module mealyNonover10101;
26
27          // Inputs
28          reg clk;
```

```
        reg reset;
        reg din;

        // Outputs
        wire seq_detected;

        // Instantiate the Unit Under Test (UUT)
        MealyNonOver10101 uut (
                .clk(clk),
                .reset(reset),
                .din(din),
                .seq_detected(seq_detected)
        );

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

  initial begin
    reset = 1;
    din = 0;
    #10 reset = 0;
#2;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
    din = 0; #10;
    din = 1; #10;
         din = 0; #10;
         din = 1;

    #20;
  end

endmodule
```

Simulation :

State Diagram :

1.12.8 Moore Machine 10101 : Non Overlapping Case

Code :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:11:15 06/15/2025
// Design Name:
// Module Name:    MooreNonOver10101
// Project Name:
// Target Devices:
```

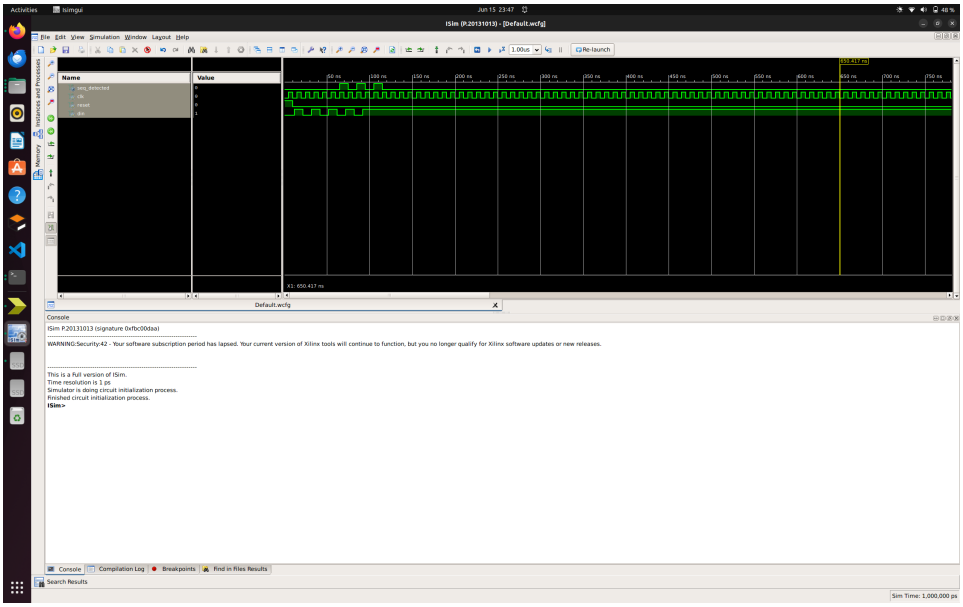


Fig. 60: Moore Overlapping Case 10101

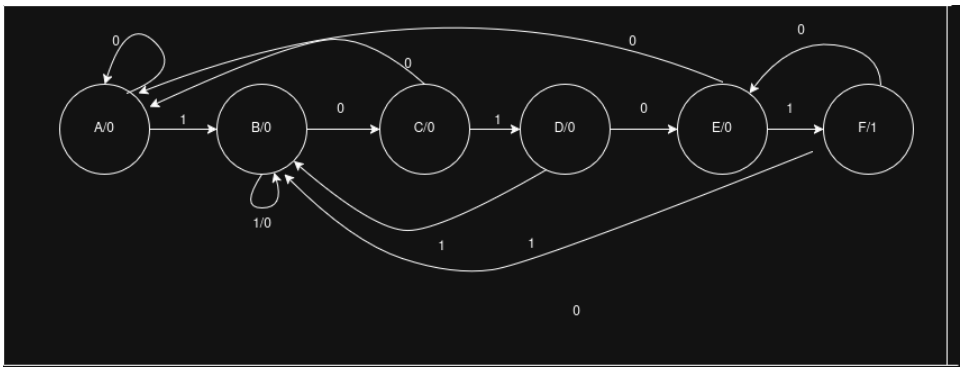


Fig. 61: State Diagram for the Moore Overlapping case 10101

```
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module MooreNonOver10101(
22  input clk,
23      input reset,
24      input din,
25      output reg seq_detected
26      );
27            localparam S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3, S4 = 3'd4, S5 = 3'd5;
28    reg [2:0] state;
29
30    always @(posedge clk or posedge reset) begin
```

```
    if (reset) begin
      state <= S0;
      seq_detected <= 0;
    end else begin
      case (state)
        S0: state <= din ? S1 : S0;
        S1: state <= din ? S1 : S2;
        S2: state <= din ? S3 : S0;
        S3: state <= din ? S1 : S4;
        S4: state <= din ? S5 : S0;
        S5: state <= S0;  // Reset after detection (non-overlap)
        default: state <= S0;
      endcase

      seq_detected <= (state == S5);
    end
  end


endmodule
```

Testbench :

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   23:48:02 06/15/2025
// Design Name:   MealyNonOver10101
// Module Name:   /home/aareen/CDAC/mealyNonover10101.v
// Project Name:  CDAC
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: MealyNonOver10101
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module mealyNonover10101;

        // Inputs
        reg clk;
        reg reset;
        reg din;

        // Outputs
        wire seq_detected;

        // Instantiate the Unit Under Test (UUT)
        MealyNonOver10101 uut (
                .clk(clk),
```

```
38                  .reset(reset),
39                  .din(din),
40                  .seq_detected(seq_detected)
41          );
42
43                  initial begin
44      clk = 0;
45      forever #5 clk = ~clk;
46    end
47
48    initial begin
49      reset = 1;
50      din = 0;
51      #10 reset = 0;
52  #2;
53      din = 1; #10;
54      din = 0; #10;
55      din = 1; #10;
56      din = 0; #10;
57      din = 1; #10;
58      din = 0; #10;
59      din = 1; #10;
60           din = 0; #10;
61           din = 1;
62
63      #20;
64    end
65
66  endmodule
```

Simulation :

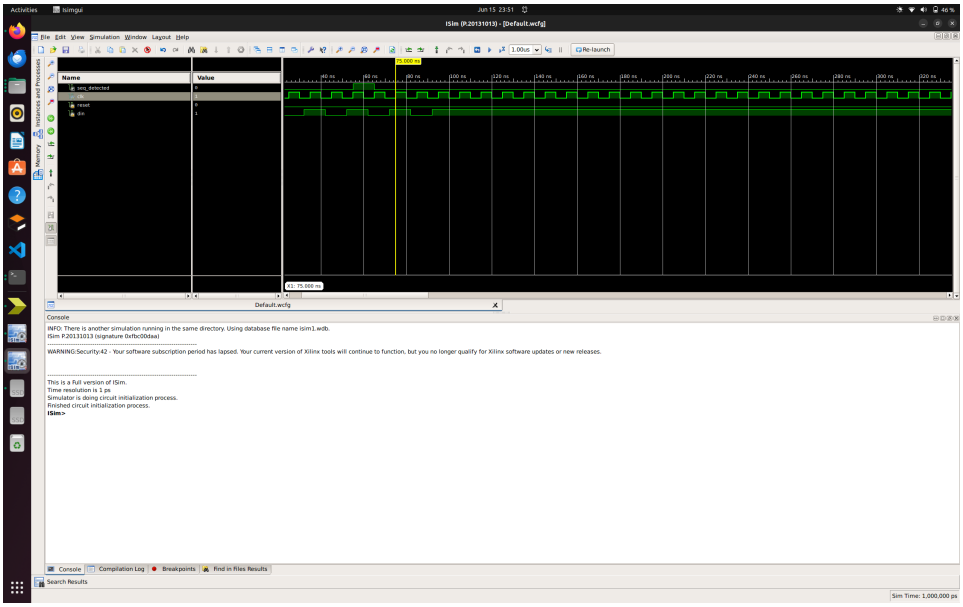


Fig. 62: Moore Non Overlapping Case 10101

State Diagram :

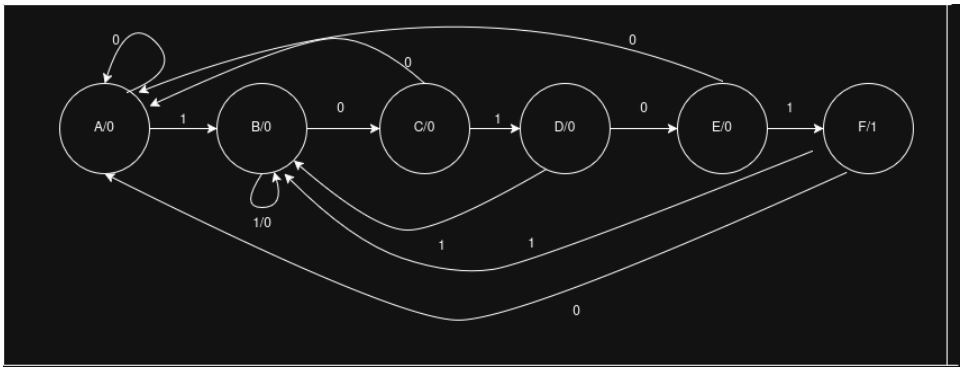


Fig. 63: Moore Non Overlapping Case 10101

1.13 Serial Adder

Both the adders have been implemented using Behavioral modeling . Module has 2 inputs a and b and a sum output . There's a storage register which Saves the carry .

The general block diagram for both the cirucits is provided below :

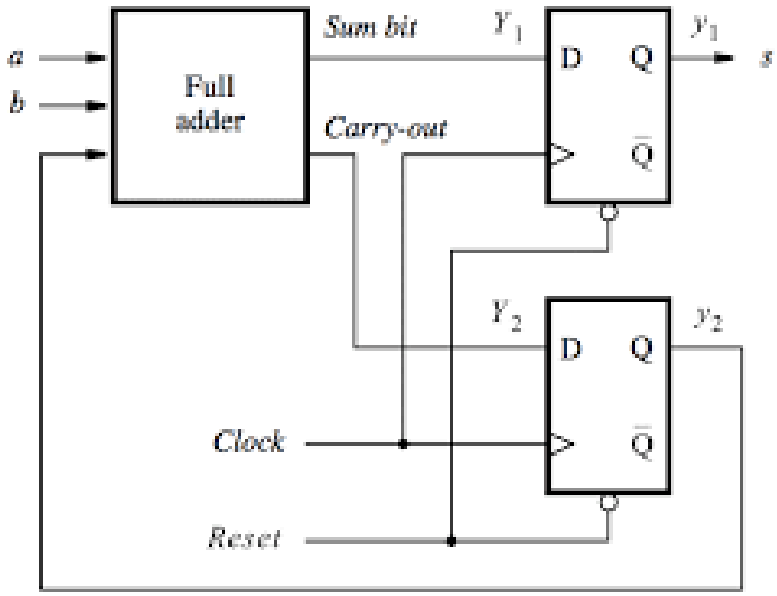


Fig. 64: Block Diagram of Univarsal Shift Register

Code for Mealy Design :

```
1 `timescale 1ns / 1ps
2
3 module SerialAdderMealy(
4     input a,
5     input b,
6     output reg y,
```

```
7       input reset,
8            input clk
9            ,output reg q
10      );
11           //reg q;//state of the adder ,i.e. whether carry is 0 or 1
12          always @(posedge clk)
13          begin
14          if(reset)
15          y=0;
16          else
17          begin
18          if(q)
19                  begin
20                  case ({a,b})
21                  2'b00:begin
22                          q=0;
23                          y=1;
24                          end
25                  2'b01:begin
26                          q=1;
27                          y=0;
28                          end
29                  2'b10:begin
30                          q=1;
31                          y=0;
32                          end
33                  2'b11:begin
34                          q=1;
35                          y=1;
36                          end
37
38                          endcase
39                  end
40
41          else
42          begin
43                  case({a,b})
44                  2'b00:begin
45                                  q=0;
46                                  y=0;
47                                  end
48                  2'b01:begin
49                                  q=0;
50                                  y=1;
51                                  end
52                  2'b10: begin
53                                          q=0;
54                                          y=1;
55                                          end
56                  2'b11: begin
57                          q=1;
58                          y=0;
59                  end
60                  endcase
61          end
62          end
63
64          end
65
66  endmodule
```

Code for Moore Design:

```
`timescale 1ns / 1ps
module SerialAdderMoore(
    input a,
    input b,
    input reset,
    output reg [1:0]q,
    output reg y,input clk
    );
         reg [1:0]nxtq;
         reg nxty;
         always @(posedge clk)
         begin
         if(reset)
         begin
         q=0;
         y=0;
         end

         else
         begin
         q[0]<=(~a&~b&~q[1]&q[0])|(~a&~b&q[1]&~q[0])|(~a&b&~q[1]&~q[0])|(b&q[1]&q[0])|(q[0]&
         q[1]<=(a&b)|(q[1]&q[0])|(b&q[0])|(b&q[1])|(a&q[0])|(a&q[1]);
         y<=nxty;
         end

         end

         always @(a,b,clk)
         begin
         nxtq[0]=(~a&~b&~q[1]&q[0])|(~a&~b&q[1]&~q[0])|(~a&b&~q[1]&~q[0])|(b&q[1]&q[0])|(q[0
         nxtq[1]=(a&b)|(q[1]&q[0])|(b&q[0])|(b&q[1])|(a&q[0])|(a&q[1]);
         nxty=nxtq[0];
         end


endmodule
```

Testbench for Mealy Machine:

```
`timescale 1ns / 1ps


module SerialAdderMealyTest;

        // Inputs
        reg a;
        reg b;
        reg reset;
        reg clk;

        // Outputs
        wire y;
        wire q;

        // Instantiate the Unit Under Test (UUT)
        SerialAdderMealy uut (
```

```
                .a(a),
                .b(b),
                .y(y),
                .reset(reset),
                .clk(clk),
                .q(q)
        );
initial begin
clk=0;
forever #5 clk=~clk;
end
        initial begin
                // Initialize Inputs
                a = 0;
                b = 0;
                reset = 1;
        //clk = 0;

                // Wait 100 ns for global reset to finish
                #30;
                reset=0;
                #22;
                a=1;
                b=1;
                #20;
                a=0;
                b=0;
                #10;
                a=1;
                #10;
                b=1;
                #20;
                $finish;


                // Add stimulus here

        end

endmodule
```

Testbench for Moore Machine:

```
`timescale 1ns / 1ps
module SerialAdderMoore(
    input a,
    input b,
    input reset,
    output reg [1:0]q,
    output reg y,input clk
    );
         reg [1:0]nxtq;
         reg nxty;
         always @(posedge clk)
         begin
         if(reset)
         begin
```

```
15              q=0;
16              y=0;
17              end
18
19              else
20              begin
21              q[0]<=(~a&~b&~q[1]&q[0])|(~a&~b&q[1]&~q[0])|(~a&b&~q[1]&~q[0])|(b&q[1]&q[0])|(q[0]&
22              q[1]<=(a&b)|(q[1]&q[0])|(b&q[0])|(b&q[1])|(a&q[0])|(a&q[1]);
23              y<=nxty;
24              end
25
26              end
27
28              always @(a,b,clk)
29              begin
30              nxtq[0]=(~a&~b&~q[1]&q[0])|(~a&~b&q[1]&~q[0])|(~a&b&~q[1]&~q[0])|(b&q[1]&q[0])|(q[0
31              nxtq[1]=(a&b)|(q[1]&q[0])|(b&q[0])|(b&q[1])|(a&q[0])|(a&q[1]);
32              nxty=nxtq[0];
33              end
34
35
36  endmodule
```

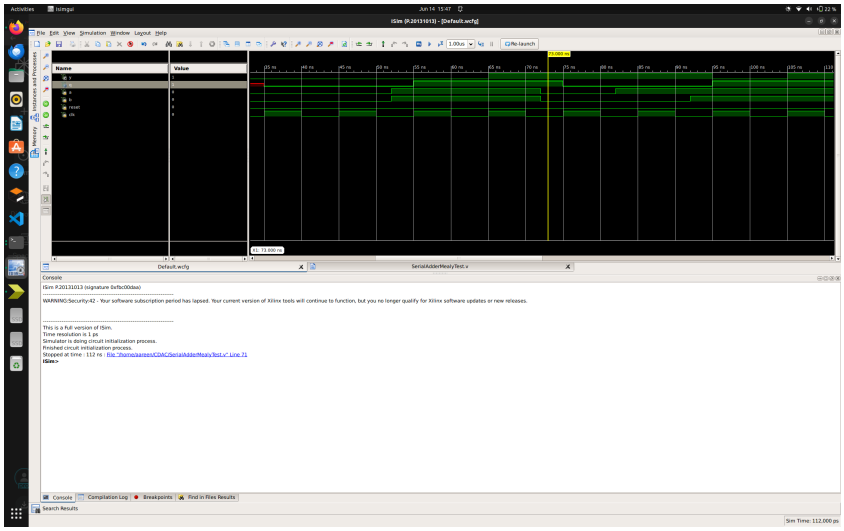


Fig. 65: Simulation of Mealy Serial Adder

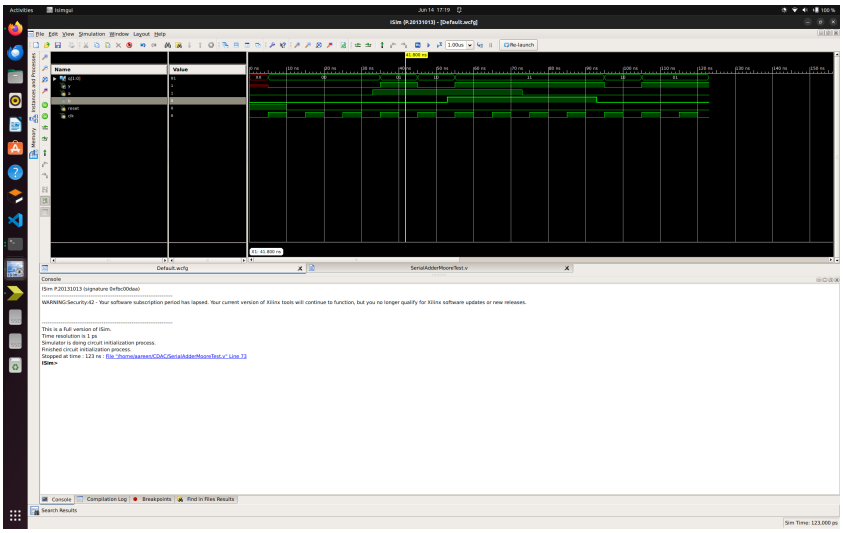


Fig. 66: Simulation of Moore Serial Adder