

# **VLSI FRONT END**

**A CERTIFICATE COURSE CONDUCTED**

**BY**



**THE SURE TRUST**

**Skill Upgradation for Rural-youth Empowerment – TRUST**  
**[www.suretrustforruralyouth.com](http://www.suretrustforruralyouth.com)**

**COURSE TRAINING ATTENDED**

**BY**

**HARSHA N**  
**August 2023**

**Declaration**

This is to certify that Mr. HARSHA N has successfully completed the four months training given in “VLSI FRONT END conducted by The SURETRUST” during the period from May- August 2023.

Mr. Naga Sitaram M  
Trainer-SURE TRUST

Prof. Ch. Radha Kumari  
Executive Director & Founder  
SURE TRUST

Mrs. Vandana Nagesh  
Trustee & Advisor  
SURE TRUST

## **TABLE OF CONTENTS**

### **1. The SURE TRUST**

### **2. Course Content**

### **3. Conduct of the Course**

- **Student byelaws**
- **Written Tests**
- **Assignments**

### **4. Student Feedback**

### **5. Uniqueness of the Course according to student**

### **6. Concluding Remarks**

## **Introduction to The SURE TRUST**

The SURE TRUST is born to enhance the employability of educated unemployed rural youth. It is observed that there is a wide gap between the skills acquired by students from the academic institutions and the skills required by the industry to employ them. Employability enhancement is done through giving one on one training in emerging technologies, completely through online mode.

The mission of the SURE TRUST is to bridge the gap between the skills acquired and the skills required by training them in the most emerging technologies such as Cyber Security(CS), Artificial Intelligence (AI), Python Program, Machine Learning (ML), Deep Learning (DL), Data Science & data analytics block chain Technology, Robotic Process Automation (RPA), Project Management, Excel for Business Application, Statistical tools & Applications, Spoken English and Business Communication etc., that will enhance their employability. After completion of four months training in the course, the trainees will get live projects from industries as internship activity to get experience in applying to real time situation what they have learnt during the course. These projects will give them hands on experience which is much sought after by the prospective industry employing them. Currently students from all over India are enrolling for various courses offered by the SURE TRUST.

The SURE TRUST offers every course free of cost with no financial burden of any kind to students. This initiative is purely a service-oriented one aiming to guide the rural youth who are educated but unemployed due to lack of upgradation in their skill sets. The birth of SURE TRUST is a God given boon to rural youth who could reach great heights either in employment or in entrepreneurship once they receive the training offered followed by the company internship. Many companies are coming forward to join their hands with us by offering internship projects to hand hold and lead the rural youth in their career settlement.

## **Vision of the SURE TRUST**

The vision of the SURE TRUST is to enhance the employability of educated unemployed youth, particularly living in rural areas, through skill upgradation, with no cost to the students.

## **Mission of the SURE TRUST**

The mission is to bridge the gap between the skills acquired in the academic institutions and the skills required in industries as a pre-condition for employment.

## **Functioning of the SURE TRUST**

There are three dedicated, committed, and hard-working women on the board of management of the SURE TRUST who will look into the various administrative and other matters relating to the enrolment of students, organizing trainers, entering into agreements with companies for getting live projects to students as internship programs, and so on. All the three women on the board are all the alumni from Sri Sathya Sai Institute of Higher Learning, Anantapur Campus, deemed to be the University. The women board is supported by five eminent advisories who are from different walks of life and have made outstanding mark in career in their respective fields. For more details about SURE TRUST visit the website:

<http://www.suretrustforruralyouth.co.in/>

## **1. Course Content**

The SURE TRUST conducts a four months training for every course on a uniform basis. A session spanning across one to one & half hour is taken by the trainers for every major course. Sessions are conducted to complete the predesigned course structure within the fixed time period. Course content is designed to suit the current requirement of the Industry and validated by the industry experts. The course content of all these courses is so dynamic that any changed condition noticed in the industry will automatically get reflected in the content of the respective course.

# VLSI FRONT END

## Course Objective:

The dynamic curriculum of advance vlsi design and verification course fits perfectly with career aim of fresh engineering graduates and helps them to ‘future proof ‘ themselves and remain relevant for the rapidly evolving semiconductor technology space.

## 1. Course content :

**Module 1:** Combinational circuit

**Module 2:** Sequential circuit

**Module 3:** Verilog PROJECT

## 2. Conduct of the course:

a) The modalities for the conduct of all the courses are fixed by the SURETRUST which are uniformly followed across the courses.

- Mode of Training ----- Online
- Period of Training ----- Four months
- Sessions per week ----- 3 to 6
- Length of the session ----- 1 to 2 hours
- Tests to be taken ----- 2 per month
- Assignments ----- 2 per month
- Last 15 days ----- Final practice and preparing the course report

## b) Student byelaws:

Students enrolling for the courses under SURE TRUST are strictly required to follow the following byelaws set for them.

**Byelaws for students to become eligible for certificate at the end of the course**

- **Minimum Attendance:**
- Every student must put in a minimum of 85% attendance in attending the classes for getting the eligibility to receive the

certificates.

▪ **Assignment submissions:**

- Ten exercises constituting one assignment for every two to three new functions/topics taught, resulting in minimum seven such assignments are to be submitted during the four months period.

**Preparing the final course report in the prescribed format:**

During the last fifteen days in the fourth month, students, many be asked to consolidate and compile all the assignments submitted in a word document along with the other chapters which will constitute a course report for each student. This report will be the unique contribution a student carries from the trust to showcase the rigorous training, he/she received during the four month period. Besides, the report will stand as a testimony for the detailed learning a student has acquired in the chosen area. This will facilitate the industry in hand picking the required student for the job.

**Attend the full class:**

All the students are expected to attend each class for the full duration. Some students are observed moving out of classes after logging in which does not go well with the learning objective of students.

**Ensure discipline in the group:**

All the students are advised strictly to follow group etiquette and restrain from posting in the group any unethical messages or teasing messages or personal interactive messages. This group is purely created for academic purpose and hence only academic interactions should go on.

# **1. ADDER**

## **1.1 HALF ADDER**

### **TRUTH TABLE**

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

### **LOGIC EQUATION**

**SUM=A^B**

**CARRY=AB.**

**Verilog Code:**

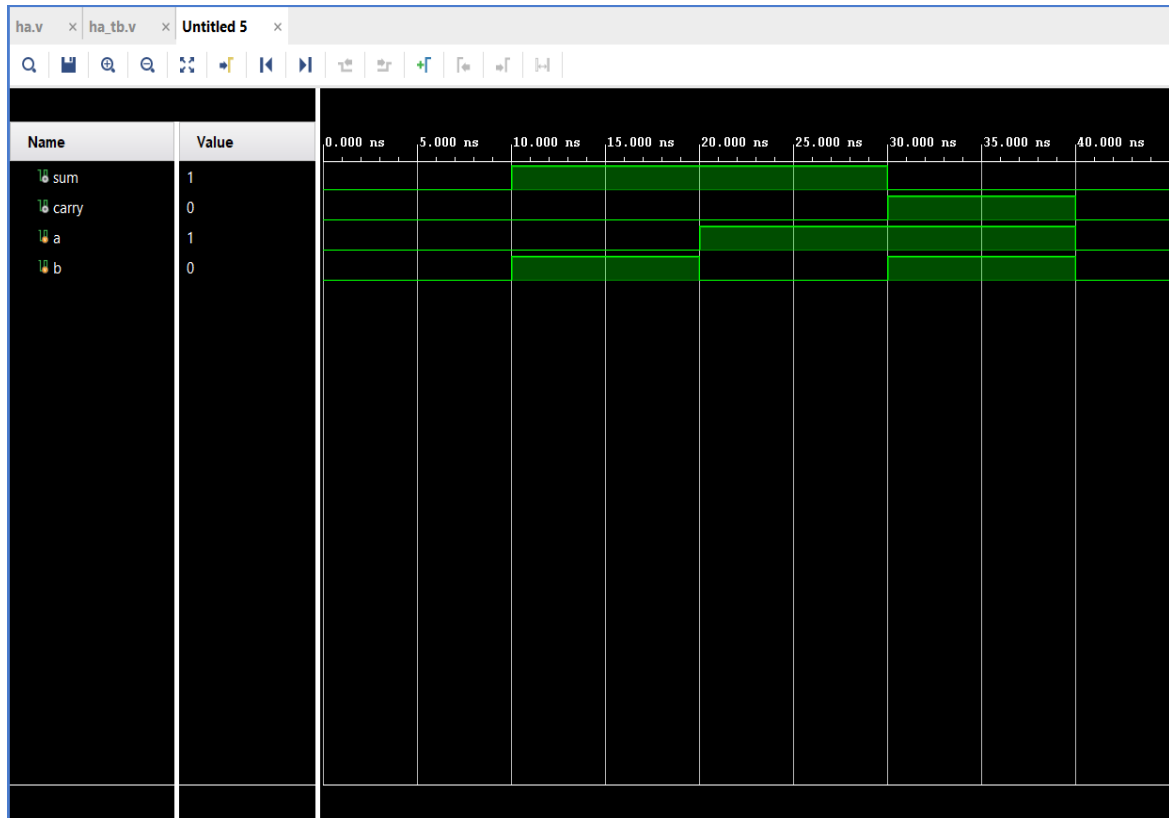
```
module ha(input a,b, output sum,carry);  
xor s1(sum,a,b);  
and s2(carry,a,b);  
endmodule
```

**Testbench:**

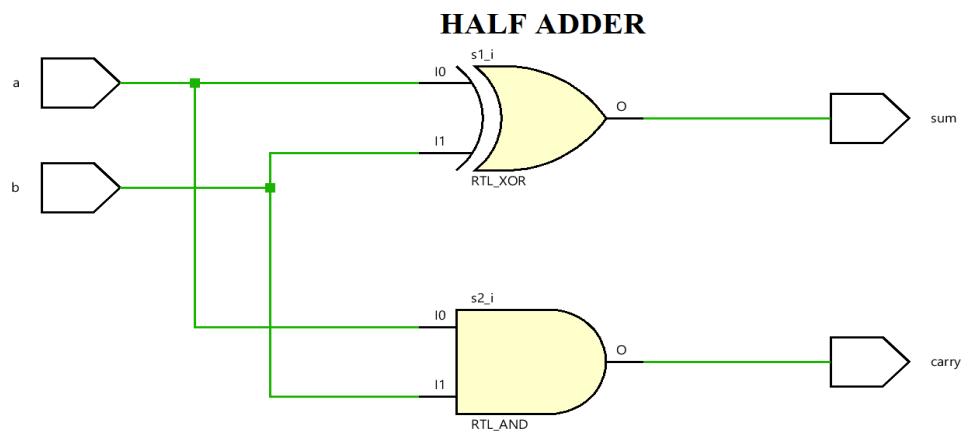
```
module ha_tb();  
wire sum,carry;  
reg a,b;  
ha a1(a,b,sum,carry);  
initial  
begin  
a=0;  
b=0;  
end  
always #20 a=~a;  
always #10 b=~b;  
endmodule
```



## WAVEFORM



## RTL SCHEMATIC FOR THE CODE



## **1.2 HALF SUBTRACTOR**

### **TRUTH TABLE**

A	B	DIFFERENCE	BORROW
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

### **LOGIC EQUATION**

**DIFFERENCE=  $A \oplus B$**

**BORROW=  $A'B$**

### **Verilog code**

```
module hs(diff,bor,a,b);  
output diff,bor;  
input a,b;  
wire na;  
not n1(na,a);  
xor s1(diff,a,b);  
and s2(bor,na,b);  
endmodule
```

### **Testbench**

```
module hs_tb();  
wire diff,bor;  
reg a,b;  
hs n1(diff,bor,a,b);  
initial  
begin  
a=0;
```

```
b=0;

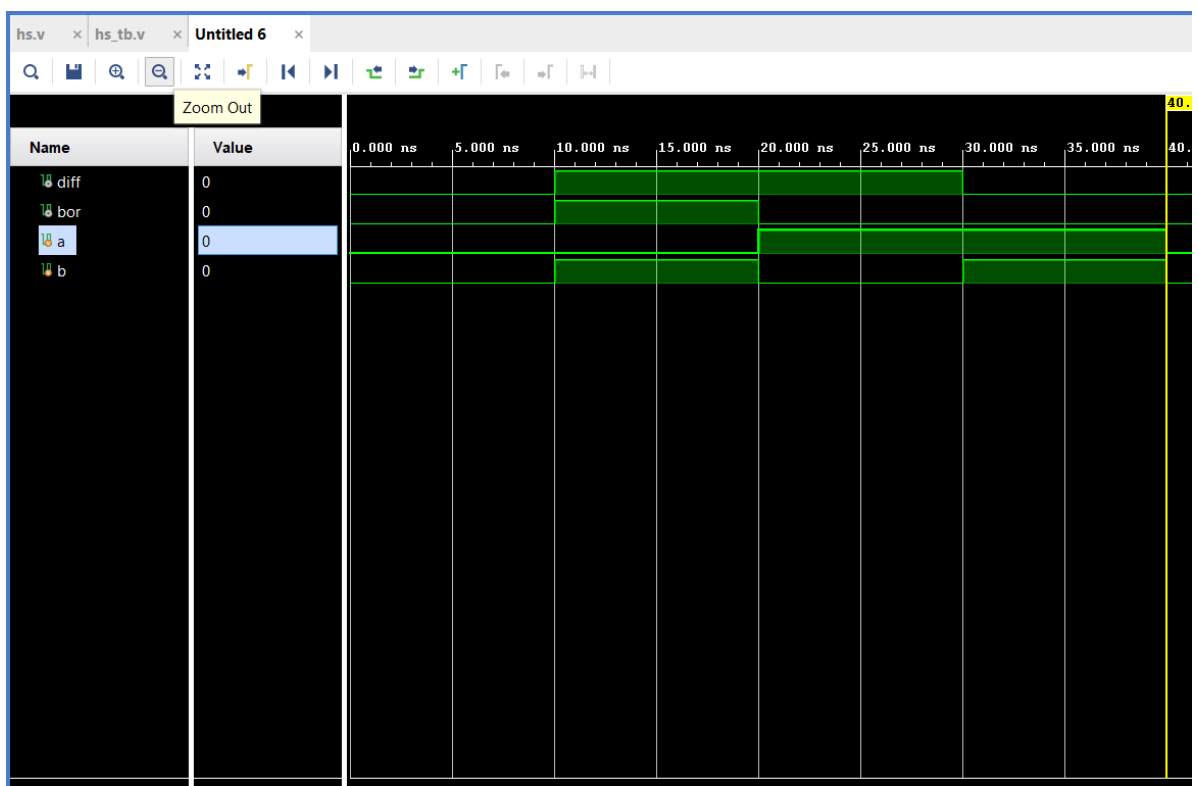
end

always #20 a=~a;

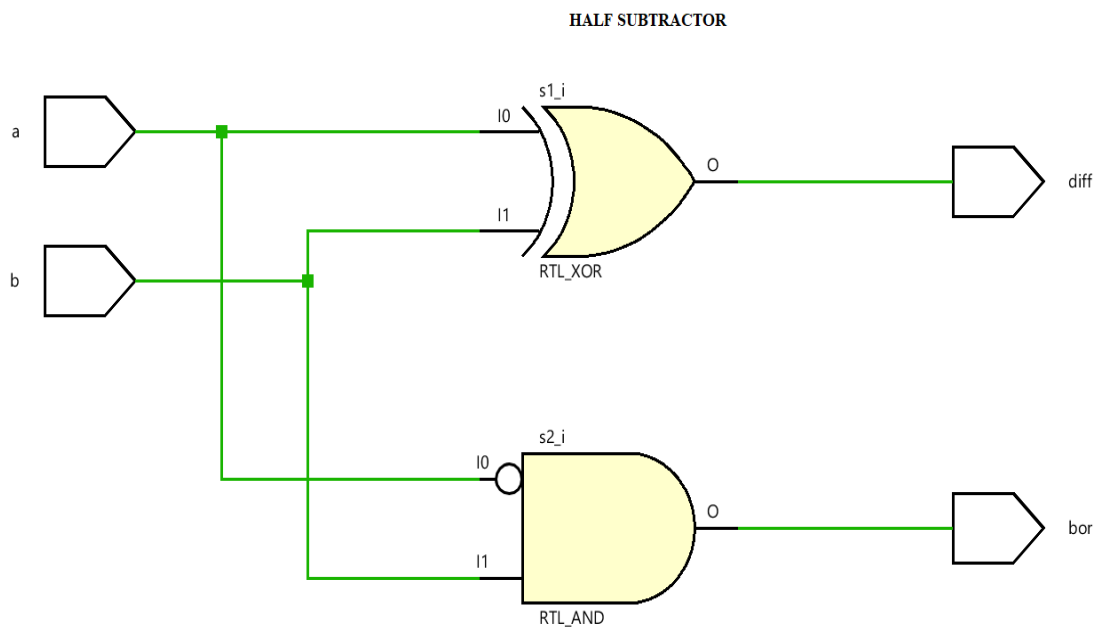
always #10 b=~b;

endmodule
```

## WAVEFORM



## RTL SCHEMATIC FOR THE CODE



## 1.3 FULL ADDER

### TRUTH TABLE

A	B	Cin	SUM	COUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

### LOGIC EQUATION

$$\text{SUM} = A \oplus B \oplus \text{Cin}$$

$$\text{COUT} = AB + (A \oplus B) \text{Cin}$$

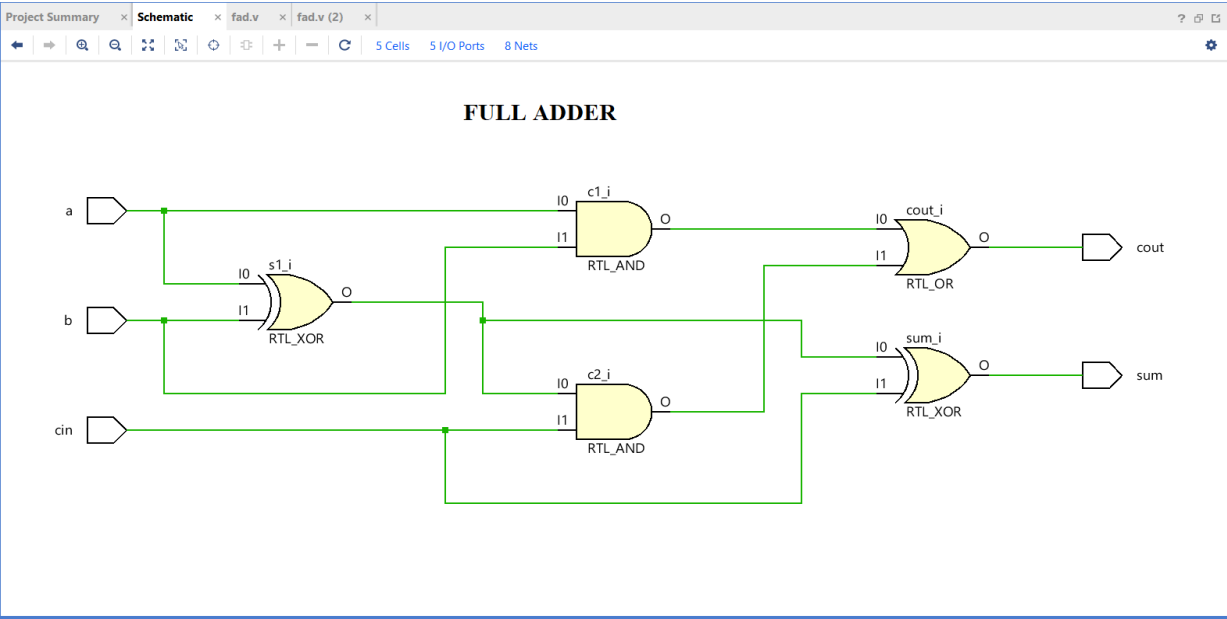
### **Verilog code**

```
module fad(input a,b,cin,output sum,cout);  
wire s1,c1,c2;  
xor(s1,a,b);  
xor(sum,s1,cin);  
and(c1,a,b);  
and(c2,s1,cin);  
or(cout,c1,c2);  
endmodule
```

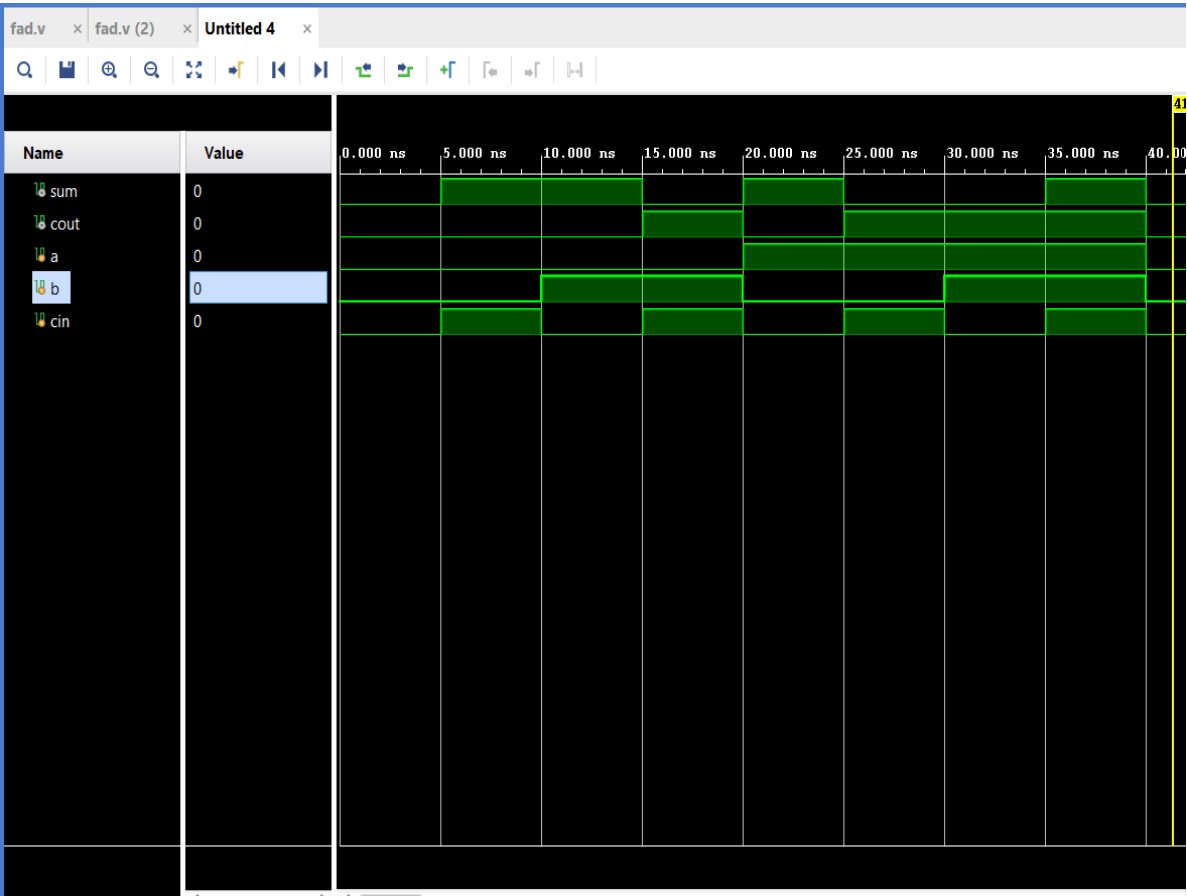
### **Testbench**

```
module fad_tb();  
wire sum,cout;  
reg a,b,cin;  
fad a1(a,b,cin,sum,cout);  
initial  
begin  
a=0;  
b=0;  
cin=0;  
end  
always #20 a=~a;  
always #10 b=~b;  
always #5 cin=~cin;  
endmodule
```

RTL SCHEMATIC FOR THE CODE



WAVEFORM



## **1.4 FULL SUBTRACTOR**

### **TRUTH TABLE**

A	B	C	DIFFERENCE	BORROW
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

### **LOGIC EQUATION**

$$\text{DIFFERENCE} = A \oplus B \oplus C$$

$$\text{BORROW} = A'B + (A + B)C$$

#### **Verilog code**

```
module fs(output diff,bor,input a,b,c);
wire na,d1,b1,b2;
xor a1(d1,a,b);
xor a2(diff,d1,c);
not n1(na,a);
or a3(b1,na,b);
and a4(b2,b1,c);
and a5(b3,na,b);
or a6(bor,b2,b3);
endmodule
```

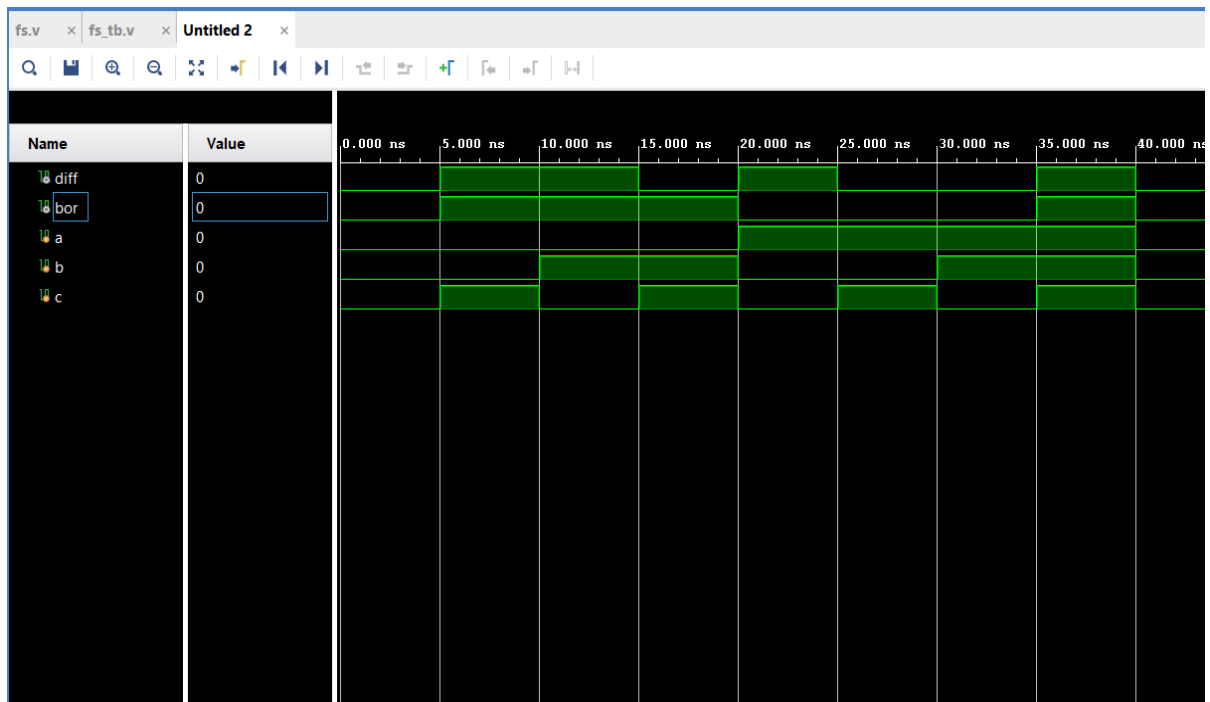
## Testbench

```

module fs_tb();
wire diff,bor;
reg a,b,c;
fs a1(diff,bor,a,b,c);
initial
begin
a=0;
b=0;
c=0;
end
always #20a=~a;
always #10 b=~b;
always #5 c=~c;
endmodule

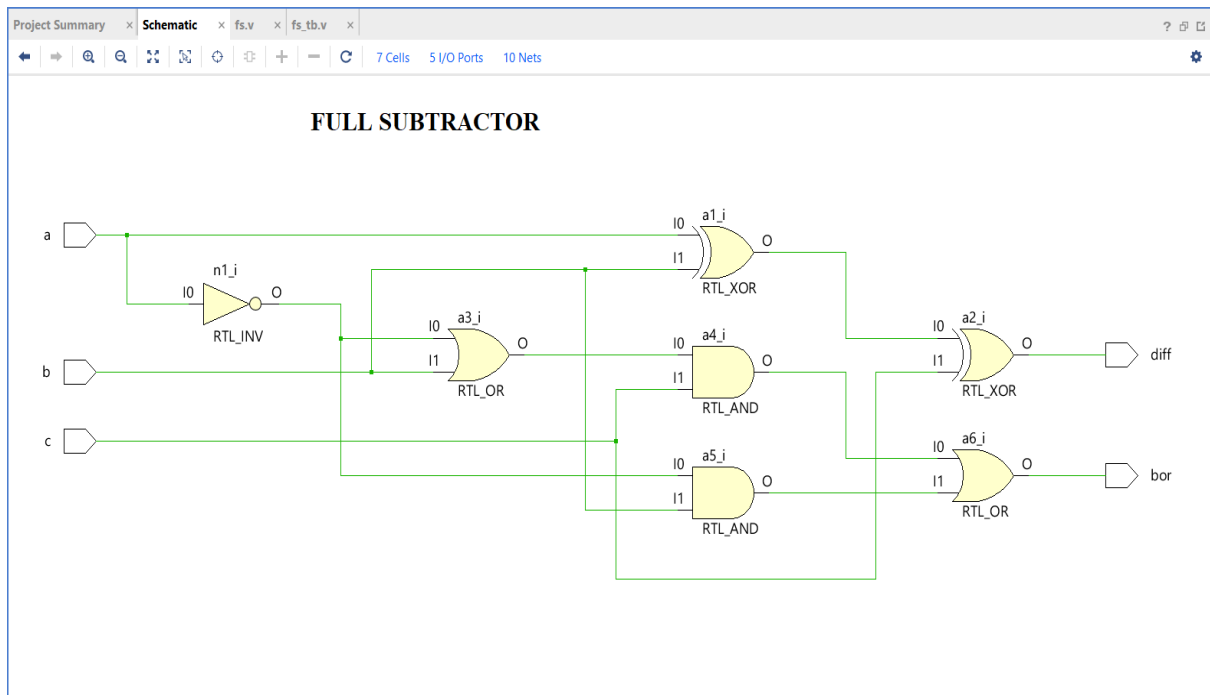
```

## WAVEFORM





## RTL SCHEMATIC FOR THE CODE



## 2. 4-BIT COMPARATOR

### TRUTH TABLE

[illegible]

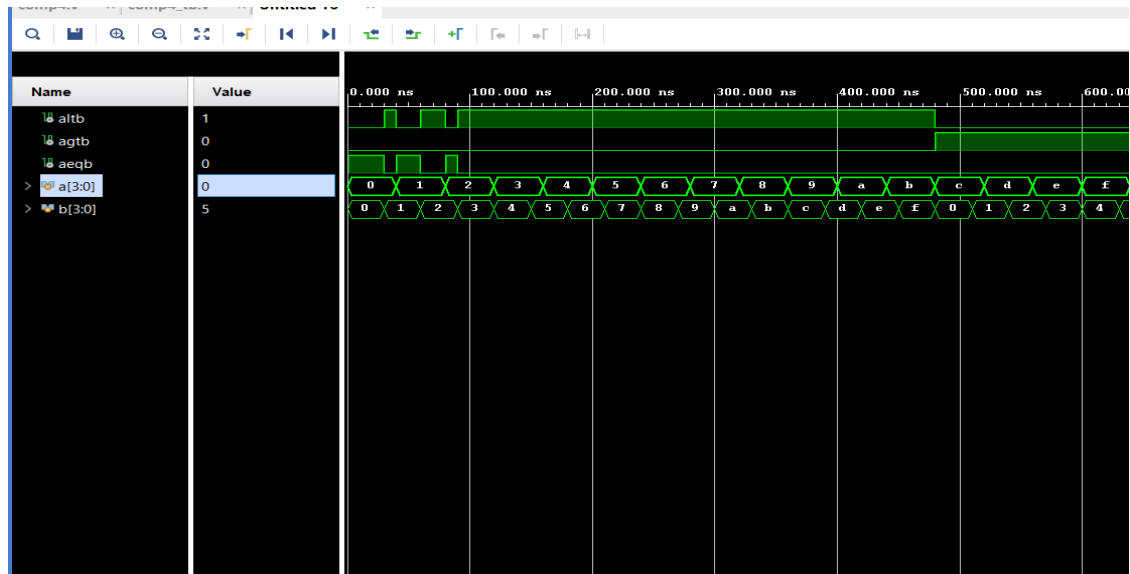
### **Verilog Code**

```
module comp4(input [3:0] a,[3:0]b,output altb,agtb,aeqb);  
assign altb=(a<b);  
assign agtb=(a>b);  
assign aeqb=(a==b);  
endmodule
```

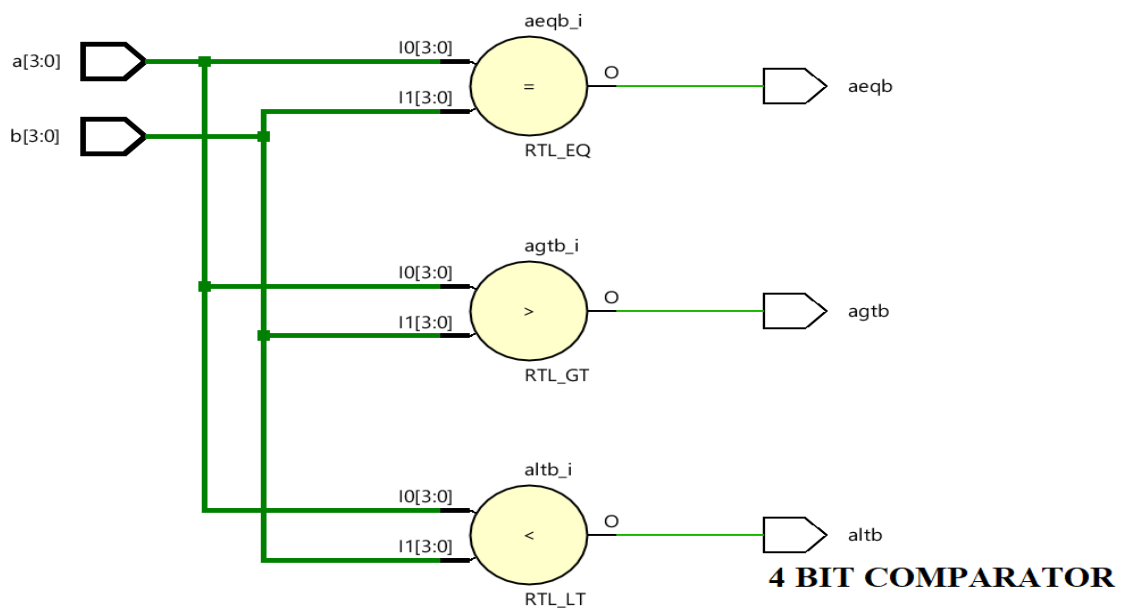
### **Testbench**

```
module comp4_tb();  
wire altb,agtb,aeqb;  
reg [3:0]a;  
reg [3:0]b;  
comp4 a1(a,b,altb,agtb,aeqb);  
initial  
begin  
a=0;b=0;  
end  
always #40 a[0]=~a[0];  
always #80 a[1]=~a[1];  
always #160 a[2]=~a[2];  
always #320 a[3]=~a[3];  
always #30 b[0]=~b[0];  
always #60 b[1]=~b[1];  
always #120 b[2]=~b[2];  
always #240 b[3]=~b[3];  
endmodule
```

## WAVEFORM



## RTL SCHEMATIC FOR THE CODE



### **3.CODE CONVERTER**

#### **3.1 GRAY TO BINARY CONVERTER**

##### **TRUTH TABLE**

GRAY[7:0]	BIN[7:0]
00	00
01	01
02	03
03	02
04	06
05	07
06	05
07	04
08	0C
09	0D
0A	0F
0B	0E
0C	0A
0D	0B
0E	09
0F	08
10	18
11	19
12	1B
13	1A
14	1E
15	1F
16	1D
17	1C

##### **Verilog code**

```
module graytobin_8bit(input [7:0] gray,output [7:0] bin);  
assign bin[7]=gray[7];  
assign bin[6]=gray[7]^gray[6];  
assign bin[5]=gray[6]^gray[5];  
assign bin[4]=gray[5]^gray[4];  
assign bin[3]=gray[4]^gray[3];  
assign bin[2]=gray[3]^gray[2];  
assign bin[1]=gray[2]^gray[1];
```

endmodule

## Testbench

```

module graytobin_8bit_tb();
wire [7:0]bin;
reg [7:0]gray;

graytobin_8bit g1(gray,bin);

initial

begin

gray=0;

#800 $finish;

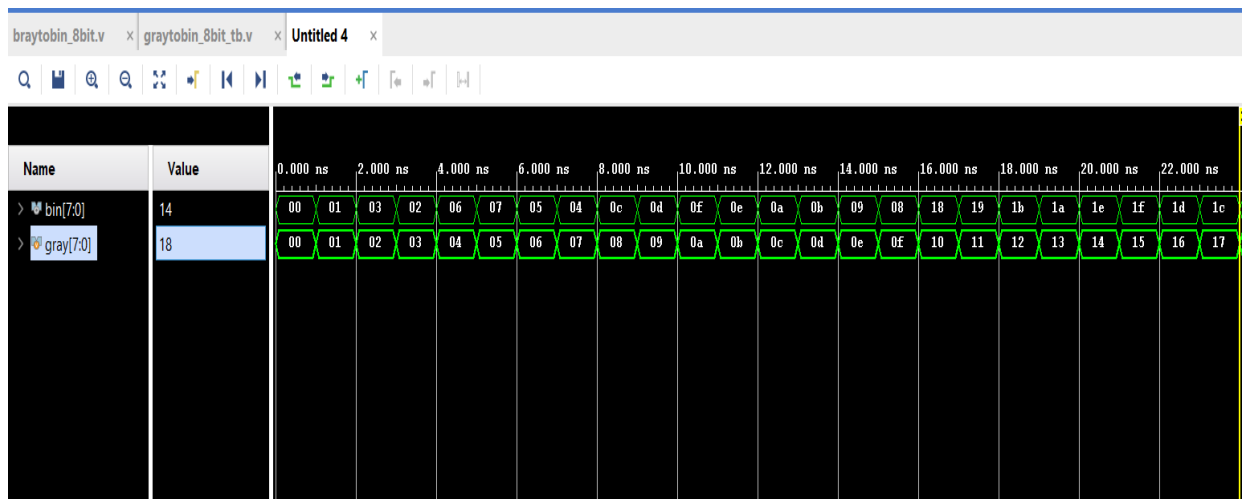
end

always #1 gray[0]=~gray[0];
always #2 gray[1]=~gray[1];
always #4 gray[2]=~gray[2];
always #8 gray[3]=~gray[3];
always #16 gray[4]=~gray[4];
always #32 gray[5]=~gray[5];
always #64 gray[6]=~gray[6];
always #128 gray[7]=~gray[7];

endmodule

```

## Waveform

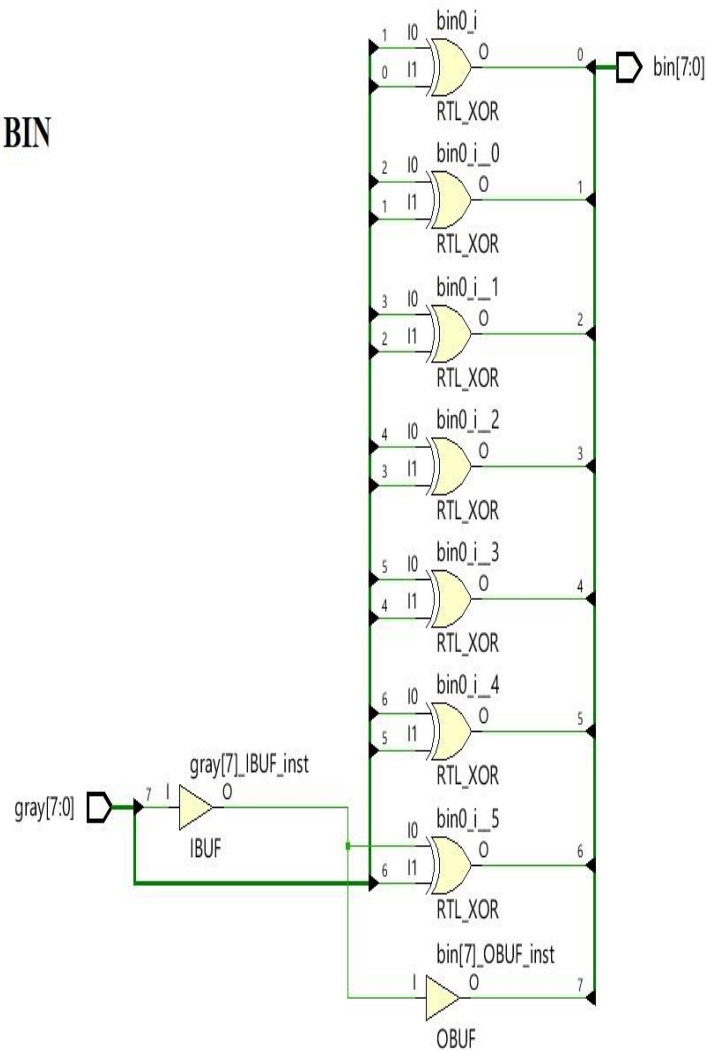


## RTL SCHEMATIC FOR THE CODE

Project Summary x Schematic x braytobin\_8bit.v x graytobin\_8bit\_tb.v x

9 Cells 16 I/O Ports 17 Nets

### GRAY TO BIN



### **3.2 BINARY TO GRAY CONVERTER**

#### **TRUTH TABLE**

<b>BIN[7:0]</b>	<b>GRAY[7:0]</b>
<b>00</b>	<b>00</b>
<b>01</b>	<b>01</b>
<b>02</b>	<b>03</b>
<b>03</b>	<b>02</b>
<b>04</b>	<b>06</b>
<b>05</b>	<b>07</b>
<b>06</b>	<b>05</b>
<b>07</b>	<b>04</b>
<b>08</b>	<b>0C</b>
<b>09</b>	<b>0D</b>
<b>0A</b>	<b>0F</b>
<b>0B</b>	<b>0E</b>
<b>0C</b>	<b>0A</b>
<b>0D</b>	<b>0B</b>
<b>0E</b>	<b>09</b>
<b>0F</b>	<b>08</b>
<b>10</b>	<b>18</b>
<b>11</b>	<b>19</b>
<b>12</b>	<b>1B</b>
<b>13</b>	<b>1A</b>
<b>14</b>	<b>1E</b>
<b>15</b>	<b>1F</b>
<b>16</b>	<b>1D</b>
<b>17</b>	<b>1C</b>

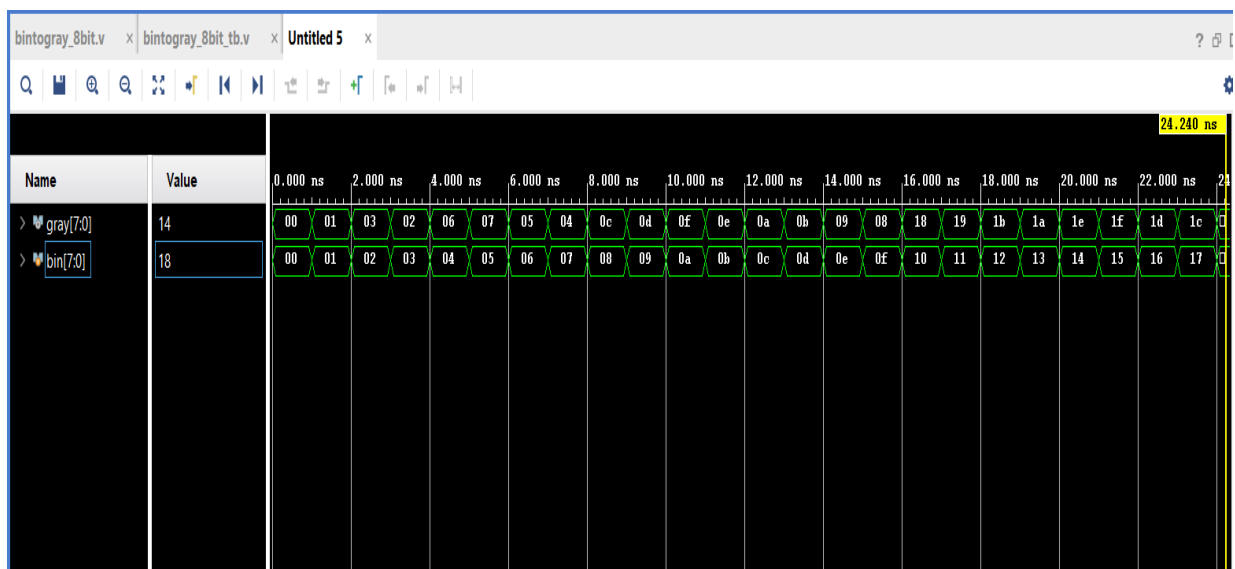
#### **Verilog code**

```
module bintogray_8bit(input [7:0] bin,output [7:0] gray);  
assign gray[7]=bin[7];  
assign gray[6]=bin[7]^bin[6];  
assign gray[5]=bin[6]^bin[5];  
assign gray[4]=bin[5]^bin[4];  
assign gray[3]=bin[4]^bin[3];  
assign gray[2]=bin[3]^bin[2];  
assign gray[1]=bin[2]^bin[1];  
assign gray[0]=bin[1]^bin[0];  
endmodule
```

## Testbench

```
module bintogray_8bit_tb();  
  
wire [7:0]gray;  
  
reg [7:0]bin;  
  
bintogray_8bit b1(bin,gray);  
  
initial  
  
begin  
  
bin=0;  
  
#800 $finish;  
  
end  
  
always #1 bin[0]=~bin[0];  
always #2 bin[1]=~bin[1];  
always #4 bin[2]=~bin[2];  
always #8 bin[3]=~bin[3];  
always #16 bin[4]=~bin[4];  
always #32 bin[5]=~bin[5];  
always #64 bin[6]=~bin[6];  
always #128 bin[7]=~bin[7];  
  
endmodule
```

## Waveform



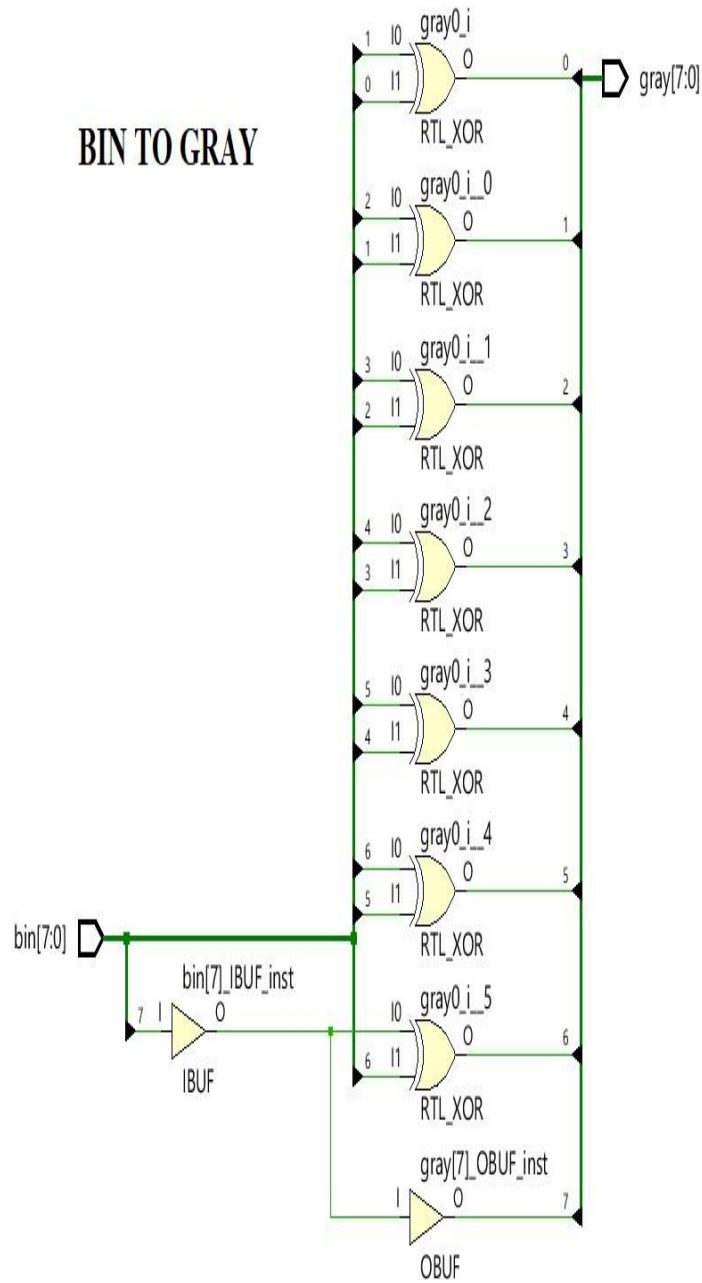


## RTL SCHEMATIC FOR THE CODE

x Schematic x bintogray\_8bit.v x bintogray\_8bit\_tb.v x ?

9 Cells 16 I/O Ports 17 Nets

### BIN TO GRAY



## **4.RIPPLE CARRY ADDER/SUBTRACTOR**

### **TRUTH TABLE**

A3	A2	A1	A0	B3	B2	B1	B0	Cin	Sum3	Sum2	Sum1	Sum0	Cout
0	0	1	1	0	1	0	0	0	0	1	1	1	0
1	0	1	1	0	0	1	0	0	1	1	0	1	0
1	1	0	0	1	1	0	0	0	1	0	0	0	1
0	0	1	1	0	1	1	1	1	1	0	1	1	0

### **Verilog code**

```
module rippleadder(output [3:0] s,output cout,input [3:0] a,b,input cin);
```

```
    wire [2:0] c;
```

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

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

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

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

```
endmodule
```

```
module fa(input a,b,cin,output s,cout);
```

```
    wire s1,c1,c2;
```

```
    xor(s1,a,b);
```

```
    xor(s,s1,cin);
```

```
    and(c1,a,b);
```

```
    and(c2,s1,cin);
```

```
    or(cout,c1,c2);
```

```
endmodule
```

### **Testbench**

```
module ripplecarryadder_tb;
```

```
    wire [3:0] s;
```

```
    wire cout;
```

```
    reg [3:0] a;
```

```
    reg [3:0] b;
```

```

reg cin;

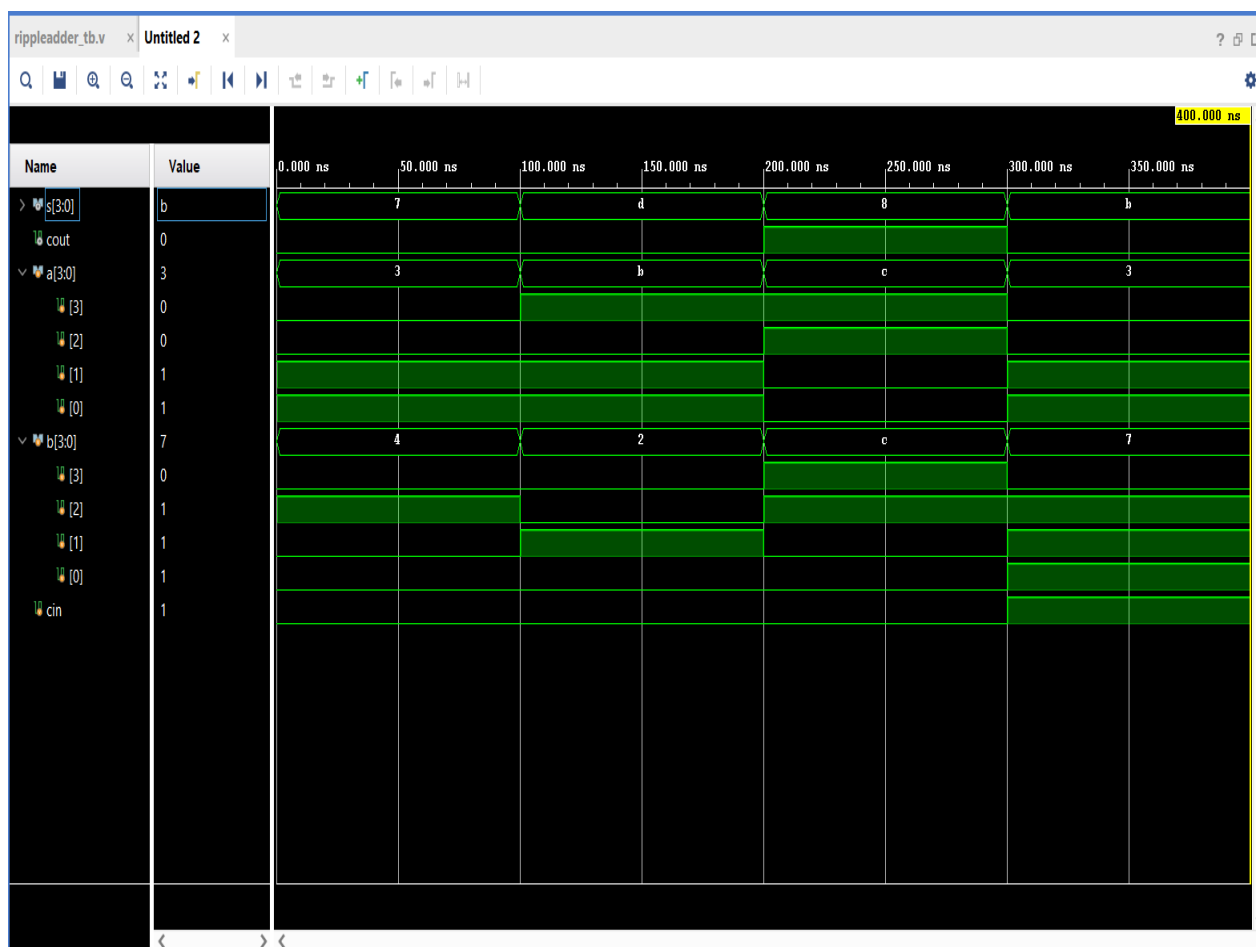
rippleadder ra(s,cout,a,b,cin);

initial begin
a = 4'b0011;b = 4'b0100;cin=1'b0;#100;
a = 4'b1011;b = 4'b0010;cin=1'b0;#100;
a = 4'b1100;b = 4'b1100;cin=1'b0;#100;
a = 4'b0011;b = 4'b0111;cin=1'b1;#100;$finish;
end

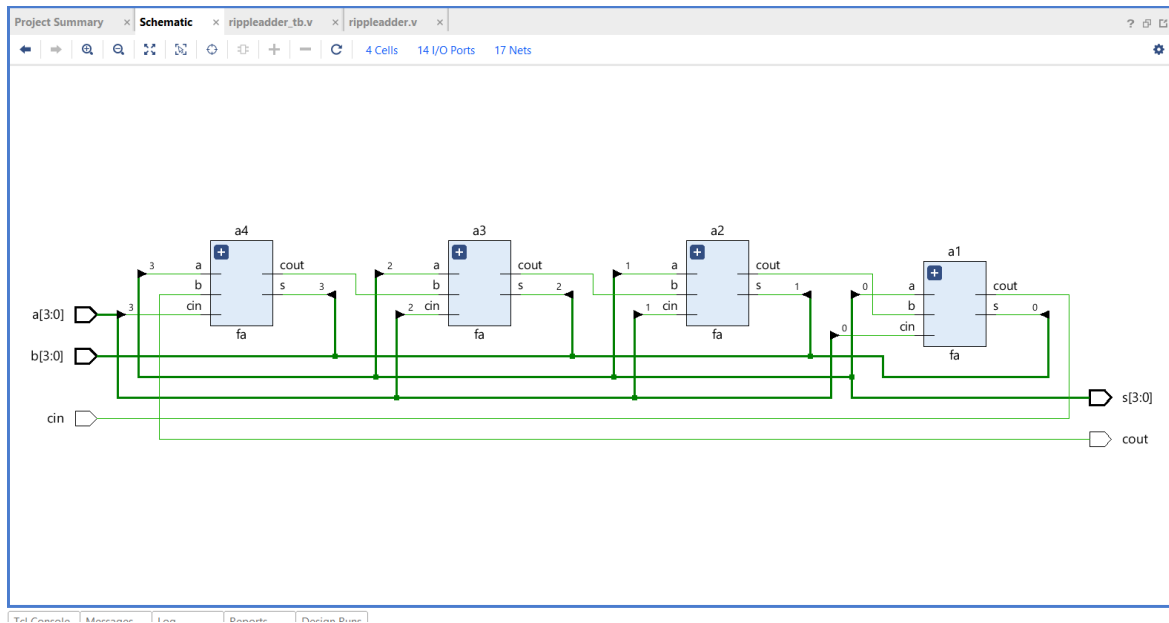
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## RIPPLE SUBTRACTOR

### TRUTH TABLE

A3	A2	A1	A0	B3	B2	B1	B0	C	Diff 3	Diff 2	Diff 1	Diff 0	Bor
0	0	1	1	0	1	0	0	0	1	1	1	1	1
1	0	1	1	0	0	1	0	0	1	0	0	1	0
1	1	0	0	1	1	0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	1	1	1	0	1	1	1

### Verilog code

```
module ripplesub(output [3:0] diff,output bor,input [3:0] a,b,input c);
```

```
    wire [2:0] d;
```

```
    fs a1(diff[0], d[0], a[0], b[0], c);
```

```
    fs a2(diff[1], d[1], a[1], b[1], d[0]);
```

```
    fs a3(diff[2], d[2], a[2], b[2], d[1]);
```

```
    fs a4(diff[3], bor, a[3], b[3], d[2]);
```

```
endmodule
```

```
module fs(output diff,bor,input a,b,c);
```

```
    wire na,d1,nd1,b1,b2;
```

```
    xor a1(d1,a,b);
```

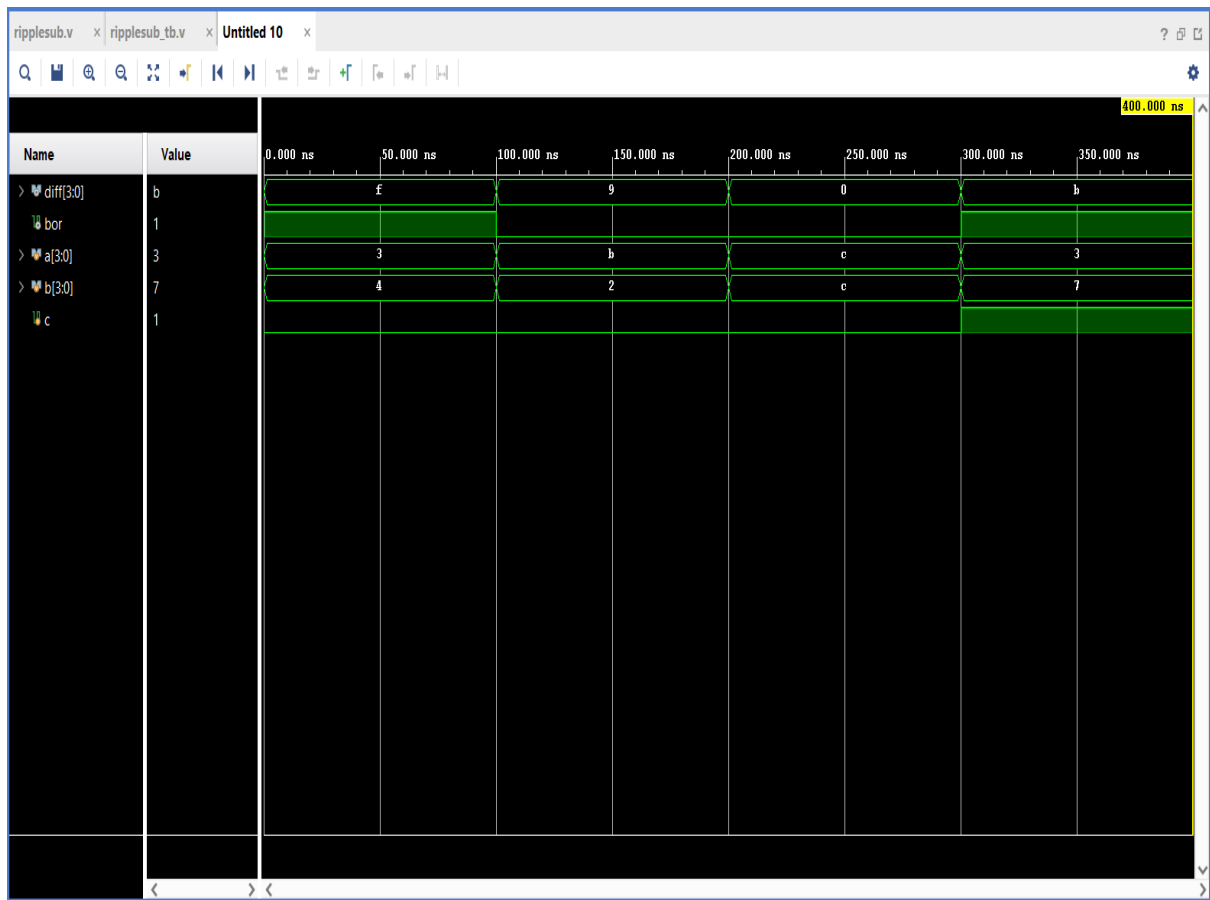
```
    xor a2(diff,d1,c);
```

```
not n1(na,a);
not n2(nd1,d1);
and a3(b1,na,b);
and a4(b2,nd1,c);
or a5(bor,b1,b2);
endmodule
```

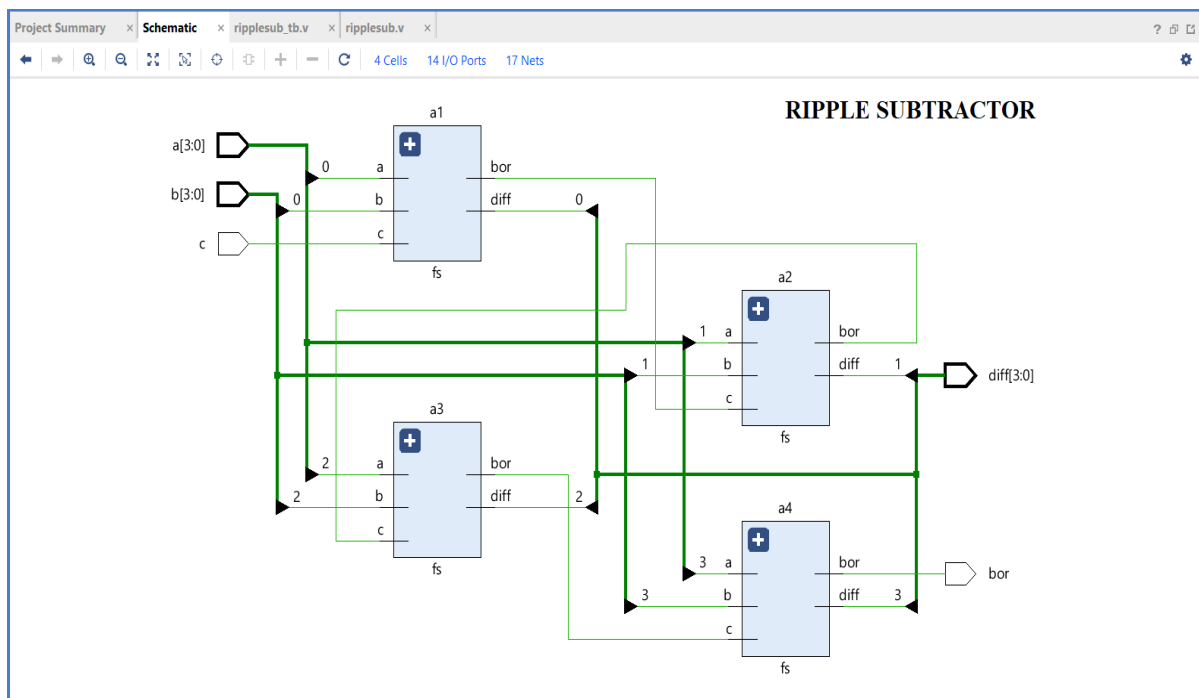
### **Testbench**

```
module ripplesub_tb();
  wire [3:0] diff;
  wire bor;
  reg [3:0] a;
  reg [3:0] b;
  reg c;
  ripplesub rs(diff,bor,a,b,c);
  initial
  begin
    a = 4'b0011;b = 4'b0100;c=1'b0;#100;
    a = 4'b1011;b = 4'b0010;c=1'b0;#100;
    a = 4'b1100;b = 4'b1100;c=1'b0;#100;
    a = 4'b0011;b = 4'b0111;c=1'b1;#100;
    $finish;
  end
endmodule
```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## **5.MULTIPLEXER**

### **5.1 4 TO 1 MUX**

**TRUTH TABLE**

S0	S1	Y
0	0	A
0	1	B
1	0	C
1	1	D

### **LOGIC EQUATION**

$$Y=S0'S1'A+S0'S1B+S0S1'C+S0S1C$$

### **Verilog code**

```
module mux4_1(input a,b,c,d,s1,s0,output out);  
    reg out;  
    always @ (*)  
    begin  
        case ({s0,s1})  
            2'b00 : out = a;  
            2'b01 : out = b;  
            2'b10 : out = c;  
            2'b11 : out = d;  
            default:out = 1'b0;  
        endcase  
    end  
endmodule
```

## Testbench

```
module mux4_1_tb;

wire out;

reg a;

reg b;

reg c;

reg d;

reg s0, s1;

mux4_1 mux1( a,b,c,d,s1,s0,out);

initial
begin
a=1'b0; b=1'b0; c=1'b0; d=1'b0;

s0=1'b0; s1=1'b0;

#500 $finish;

end

always #5 a=~a;

always #10 b=~b;

always #20 c=~c;

always #40 d=~d;

always #160 s0=~s0;

always #80 s1=~s1;

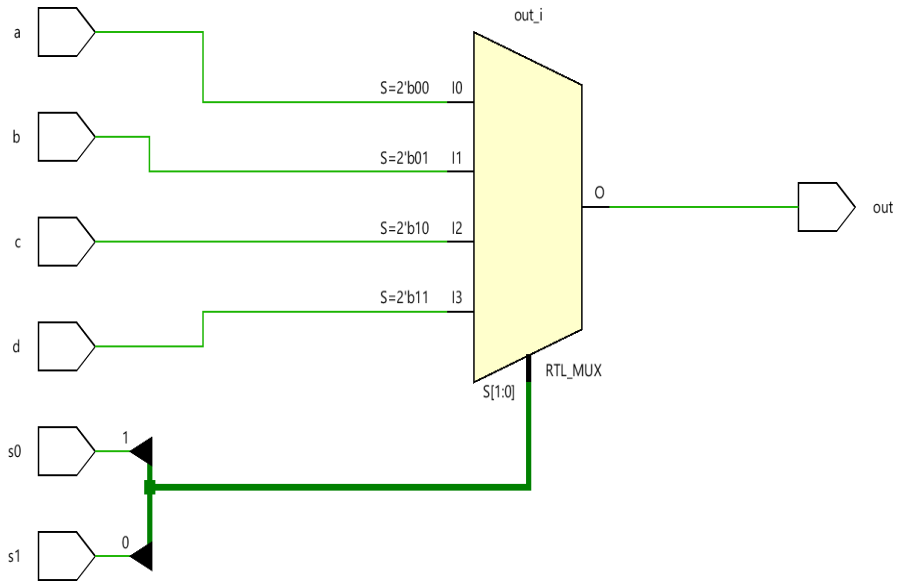
endmodule
```

## Waveform





## RTL SCHEMATIC FOR THE CODE



## 5.2 8 TO 1 MUX

### TRUTH TABLE

S[2]	S[1]	S[0]	Y
0	0	0	I[0]
0	0	1	I[1]
0	1	0	I[2]
0	1	1	I[3]
1	0	0	I[4]
1	0	1	I[5]
1	1	0	I[6]
1	1	1	I[7]

### LOGIC EQUATION

$$Y = S_2'S_1'S_0'I_0 + S_2'S_1'S_0I_1 + S_2'S_1S_0'I_2 + S_2'S_1S_0I_3 + S_2S_1'S_0'I_4 + S_2S_1'S_0I_5 + S_2S_1S_0'I_6 + S_2S_1S_0I_7$$

### Verilog code

```
module mux8to1(y,i,s);  
    output y;  
    input[2:0]s;  
    input[7:0]i;  
    wire[2:0]ns;  
    wire[7:0]w;  
    not g1(ns[0],s[0]);  
    not g2(ns[1],s[1]);  
    not g3(ns[2],s[2]);  
    and g4(w[0],i[0],ns[2],ns[1],ns[0]);  
    and g5(w[1],i[1],ns[2],ns[1],s[0]);  
    and g6(w[2],i[2],ns[2],s[1],ns[0]);  
    and g7(w[3],i[3],ns[2],s[1],s[0]);  
    and g8(w[4],i[4],s[2],ns[1],ns[0]);  
    and g9(w[5],i[5],s[2],ns[1],s[0]);  
    and g10(w[6],i[6],s[2],s[1],ns[0]);  
    and g11(w[7],i[7],s[2],s[1],s[0]);  
    or g12(y,w[0],w[1],w[2],w[3],w[4],w[5],w[6],w[7]);  
endmodule
```

### Testbench

```
module mux8to1_tb();  
    wire y;  
    reg [7:0]i;  
    reg [2:0]s;  
    mux8to1 a1(y,i,s);  
    initial  
    begin
```

```

i=0;s=0;

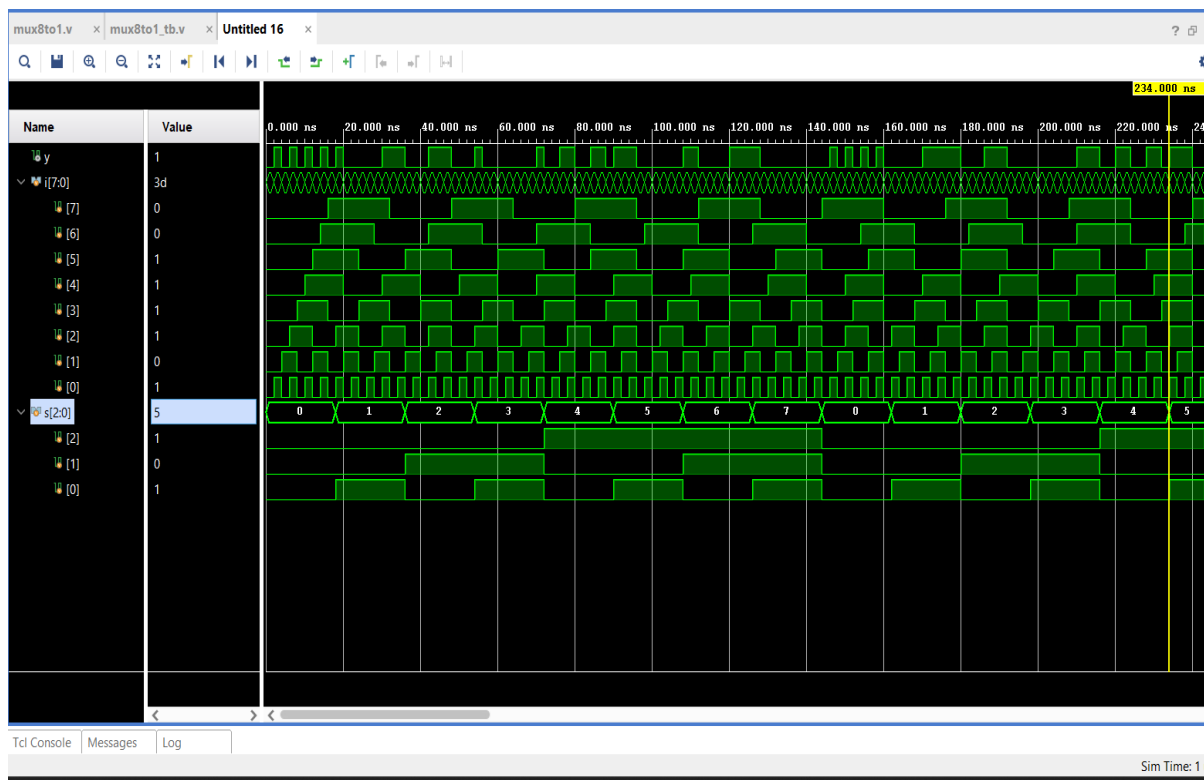
end

always #2 i[0]=~i[0];
always #4 i[1]=~i[1];
always #6 i[2]=~i[2];
always #8 i[3]=~i[3];
always #10 i[4]=~i[4];
always #12 i[5]=~i[5];
always #14 i[6]=~i[6];
always #16 i[7]=~i[7];
always #18 s[0]=~s[0];
always #36 s[1]=~s[1];
always #72 s[2]=~s[2];

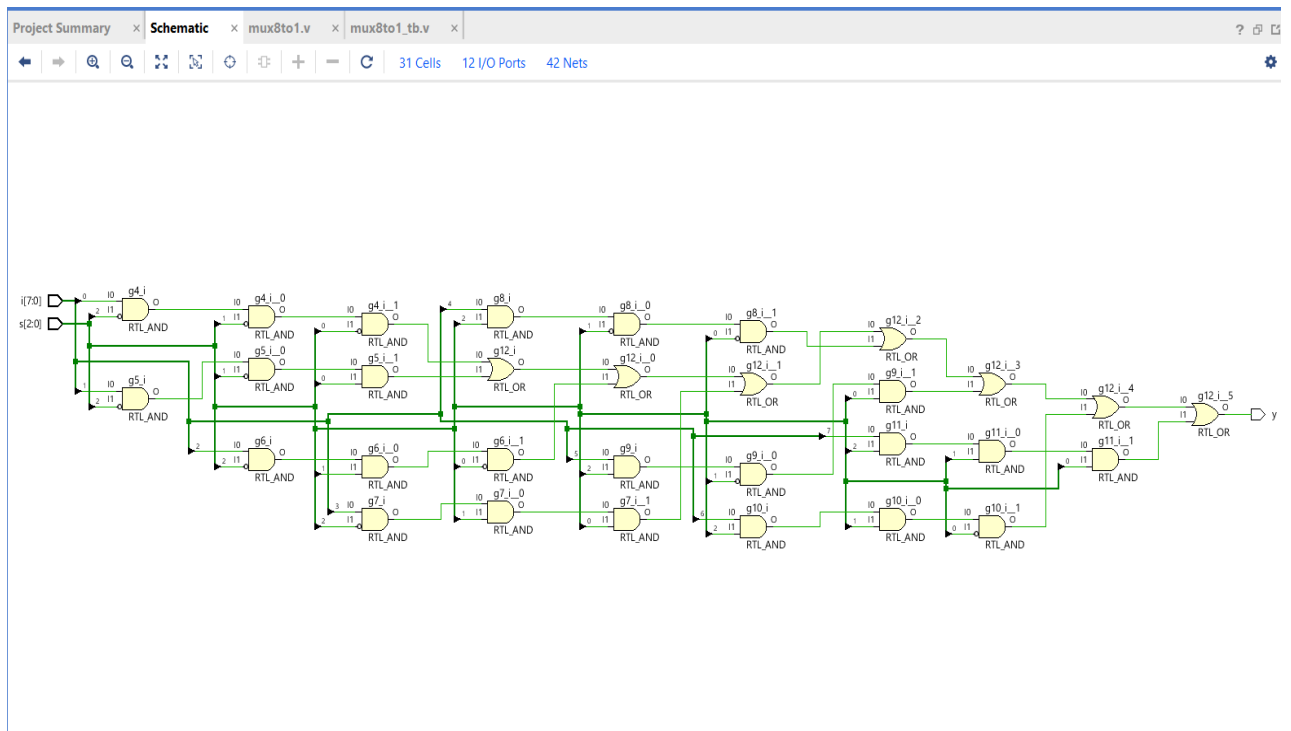
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## 6. ENCODER WITH PRIORITY

### TRUTH TABLE

EN	I[7]	I[6]	I[5]	I[4]	I[3]	I[2]	I[1]	I[0]	OUT[2]	OUT[1]	OUT[0]
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	X	0	0	1
1	0	0	0	0	0	1	X	X	0	1	0
1	0	0	0	0	1	X	X	X	0	1	1
1	0	0	0	1	X	X	X	X	1	0	0
1	0	0	1	X	X	X	X	X	1	0	1
1	0	1	X	X	X	X	X	X	1	1	0
1	1	X	X	X	X	X	X	X	1	1	1

### Verilog code

```

module priencoder8to3(input en,input [7:0] in,output [2:0] out);
    reg [2:0]out;
    always@(*)
    begin
        if(en==0)
    
```

```

out=0;

else

begin
  casex(in)
    8'b00000001:out=3'b000;
    8'b0000001x:out=3'b001;
    8'b000001xx:out=3'b010;
    8'b00001xxx:out=3'b011;
    8'b0001xxxx:out=3'b100;
    8'b001xxxxx:out=3'b101;
    8'b01xxxxxx:out=3'b110;
    8'b1xxxxxxx:out=3'b111;
    default:out=3'b000;
  endcase
end

end

endmodule

```

### **Testbench**

```

module priencode8to3_tb();
  wire [2:0]out;
  reg [7:0]in;
  reg en;

  priencoder8to3 p1(en,in,out);

  initial
  begin
    en=0;
    in=0;
    #10 en=1;
  end

```

```

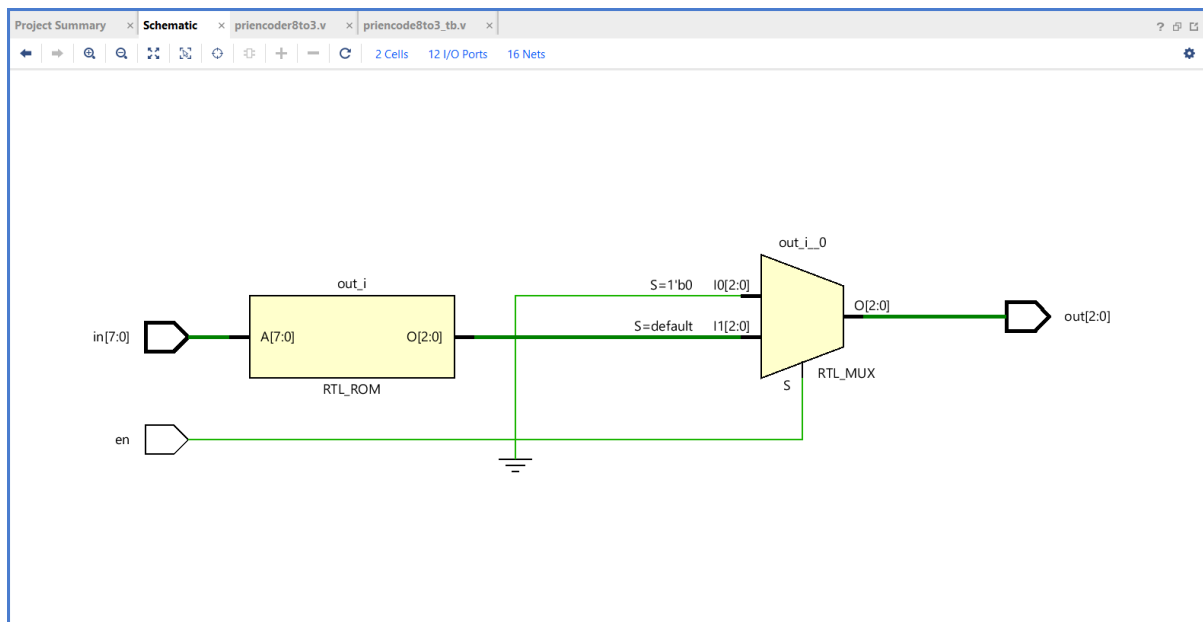
initial
begin
in=8'b000000001;
#20;
in=8'b00000001x;
#10;
in=8'b0000001xx;
#10;
in=8'b000001xxx;
#10;
in=8'b00001xxxx;
#10;
in=8'b0001xxxxxx;
#10;
in=8'b001xxxxxxx;
#10;
in=8'b01xxxxxxx;
#10;
in=8'b1xxxxxxxx;
#10;
$finish;
end
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## **7.DECODERS**

### 7.1 2 TO 4 DECODER

## TRUTH TABLE

EN	D[1]	D[0]	Y[3]	Y[2]	Y[1]	Y[0]
0	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

## Verilog code

```

module deco2_4(y,d,en);
output [3:0]y;
input [1:0]d;
input en;
wire [1:0]nd;
not n1(nd[0],d[0]);
not n2(nd[1],d[1]);
and a1(y[0],nd[1],nd[0],en);

```

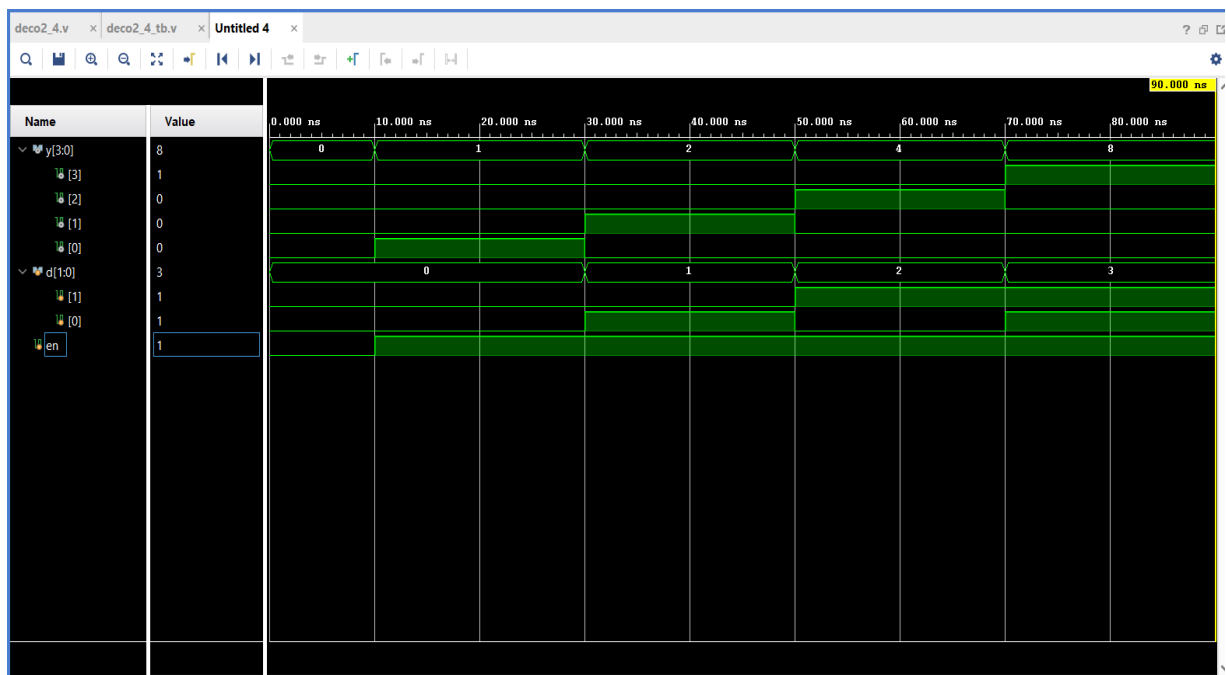
```
and a2(y[1],nd[1],d[0],en);
and a3(y[2],d[1],nd[0],en);
and a4(y[3],d[1],d[0],en);
endmodule
```

### **Testbench**

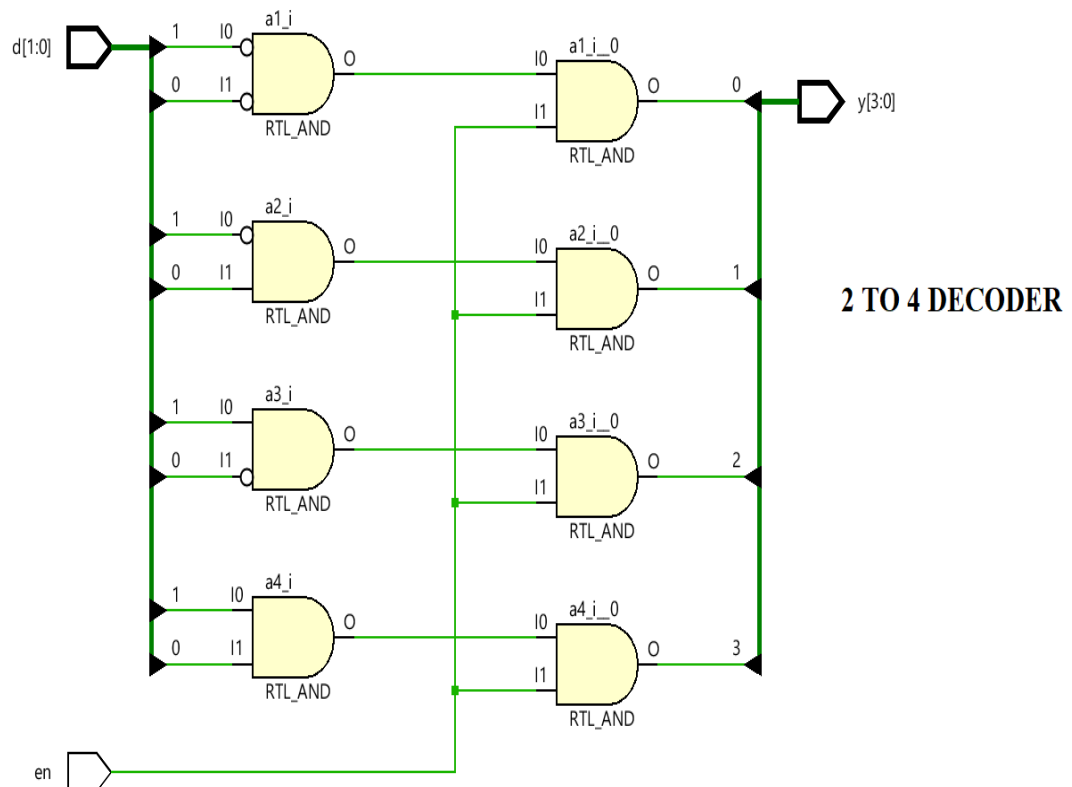
```
module deco2_4_tb();
wire [3:0]y;
reg [1:0]d;
reg en;
deco2_4 a1(y,d,en);
initial
begin
d=0;
en=0;
#10 en=1;
end
initial
begin
d[1]=0;d[0]=0;
#30;
d[1]=0;d[0]=1;;
#20;
d[1]=1;d[0]=0;
#20;
d[1]=1;d[0]=1;
#20;
$finish;
end
endmodule
```



## Waveform



## RTL SCHEMATIC FOR THE CODE



## 7.2 3 TO 8 DECODER

### TRUTH TABLE

EN	D[2]	D[1]	D[0]	Y[7]	Y[6]	Y[5]	Y[4]	Y[3]	Y[2]	Y[1]	Y[0]
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

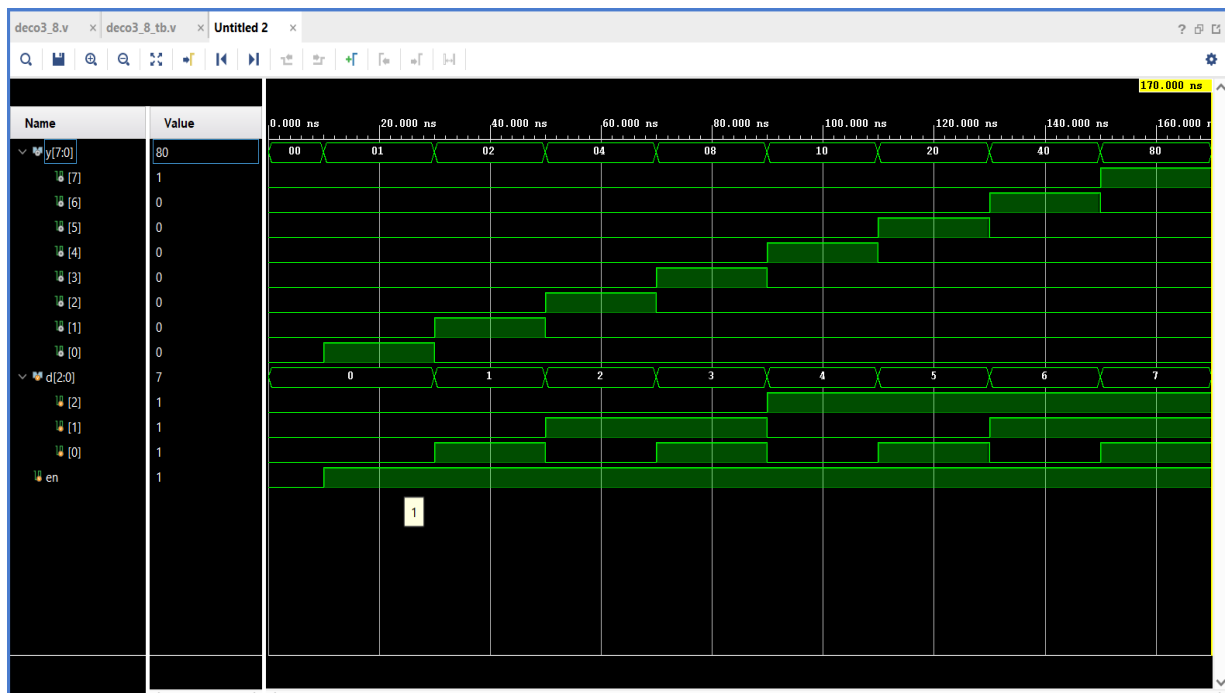
### Verilog code

```
module deco3_8(output [7:0]y,input [2:0]d,input en);
wire [2:0] nd;
not n1(nd[0],d[0]);
not n2(nd[1],d[1]);
not n3(nd[2],d[2]);
and a1(y[0],nd[2],nd[1],nd[0],en);
and a2(y[1],nd[2],nd[1],d[0],en);
and a3(y[2],nd[2],d[1],nd[0],en);
and a4(y[3],nd[2],d[1],d[0],en);
and a5(y[4],d[2],nd[1],nd[0],en);
and a6(y[5],d[2],nd[1],d[0],en);
and a7(y[6],d[2],d[1],nd[0],en);
and a8(y[7],d[2],d[1],d[0],en);
endmodule
```

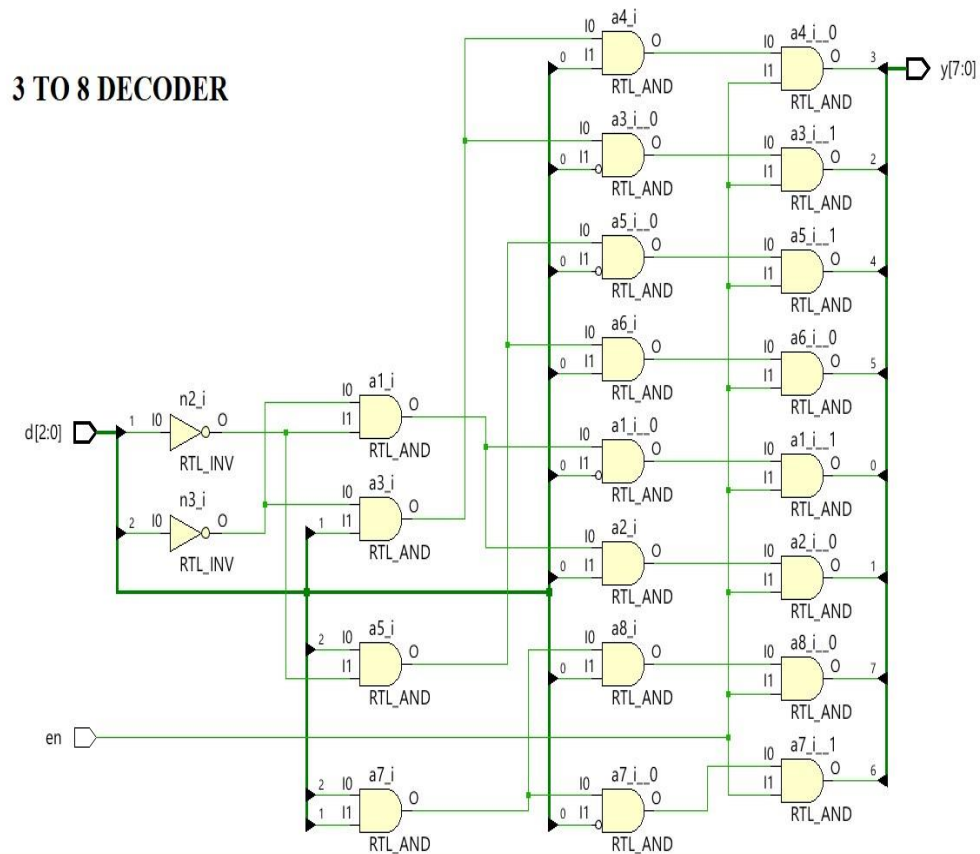
## Testbench

```
module deco3_8_tb();  
  
wire [7:0]y;  
reg [2:0]d;  
reg en;  
deco3_8 d1(y,d,en);  
initial  
begin  
d=0;en=0;  
#10 en=1;  
end  
initial  
begin  
d[2]=0;d[1]=0; d[0]=0;  
#30;  
d[2]=0;d[1]=0; d[0]=1;  
#20;  
d[2]=0;d[1]=1; d[0]=0;  
#20;  
d[2]=0;d[1]=1; d[0]=1;  
#20;  
d[2]=1; d[1]=0; d[0]=0;  
#20;  
d[2]=1;d[1]=0; d[0]=1;  
#20;  
d[2]=1;d[1]=1; d[0]=0;  
#20;  
d[2]=1;d[1]=1; d[0]=1;  
#20;  
$finish;  
end  
endmodule
```

## Waveform



## RTL SCHEMATIC FOR THE CODE



### 7.3 DECODER 4 TO 16

#### TRUTH TABLE

E N	D 3	D 2	D 1	D 0	Y 15	Y 14	Y 13	Y 12	Y 11	Y 10	Y 9	Y 8	Y 7	Y 6	Y 5	Y 4	Y 3	Y 2	Y 1	Y 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

#### Verilog code

```
module deco4_16(output [15:0] y,input [3:0] d,input en);  
wire [3:0]nd;  
not n1(nd[0],d[0]);  
not n2(nd[1],d[1]);  
not n3(nd[2],d[2]);  
not n4(nd[3],d[3]);  
and d1(y[0],nd[3],nd[2],nd[1],nd[0],en);  
and d2(y[1],nd[3],nd[2],nd[1],d[0],en);  
and d3(y[2],nd[3],nd[2],d[1],nd[0],en);  
and d4(y[3],nd[3],nd[2],d[1],d[0],en);  
and d5(y[4],nd[3],d[2],nd[1],nd[0],en);  
and d6(y[5],nd[3],d[2],nd[1],d[0],en);  
and d7(y[6],nd[3],d[2],d[1],nd[0],en);  
and d8(y[7],nd[3],d[2],d[1],d[0],en);  
and d9(y[8],d[3],nd[2],nd[1],nd[0],en);
```

```

and d10(y[9],d[3],nd[2],nd[1],d[0],en);
and d11(y[10],d[3],nd[2],d[1],nd[0],en);
and d12(y[11],d[3],nd[2],d[1],d[0],en);
and d13(y[12],d[3],d[2],nd[1],nd[0],en);
and d14(y[13],d[3],d[2],nd[1],d[0],en);
and d15(y[14],d[3],d[2],d[1],nd[0],en);
and d16(y[15],d[3],d[2],d[1],d[0],en);
endmodule

```

### **Testbench**

```

module deco4_16_tb();
wire [15:0]y;
reg [3:0]d;
reg en;
deco4_16 d1(y,d,en);
initial
begin
d=0;
en=0;
#10 en=1;
end
initial
begin
d[3]=0; d[2]=0; d[1]=0; d[0]=0;
#30;
d[3]=0; d[2]=0; d[1]=0; d[0]=1;
#20;
d[3]=0; d[2]=0; d[1]=1; d[0]=0;
#20;
d[3]=0; d[2]=0; d[1]=1; d[0]=1;
#20;
d[3]=0; d[2]=1; d[1]=0; d[0]=0;

```

```
#20;
d[3]=0; d[2]=1;d[1]=0; d[0]=1;
#20;
d[3]=0; d[2]=1;d[1]=1; d[0]=0;
#20;
d[3]=0; d[2]=1;d[1]=1; d[0]=1;
#20;
d[3]=1; d[2]=0;d[1]=0; d[0]=0;
#20;
d[3]=1; d[2]=0;d[1]=0; d[0]=1;
#20;
d[3]=1; d[2]=0;d[1]=1; d[0]=0;
#20;
d[3]=1; d[2]=0;d[1]=1; d[0]=1;
#20;
d[3]=1; d[2]=1;d[1]=0; d[0]=0;
#20;
d[3]=1; d[2]=1;d[1]=0; d[0]=1;
#20;
d[3]=1; d[2]=1;d[1]=1; d[0]=0;
#20;
d[3]=1; d[2]=1;d[1]=1; d[0]=1;
#20;
$finish;
end
endmodule
```

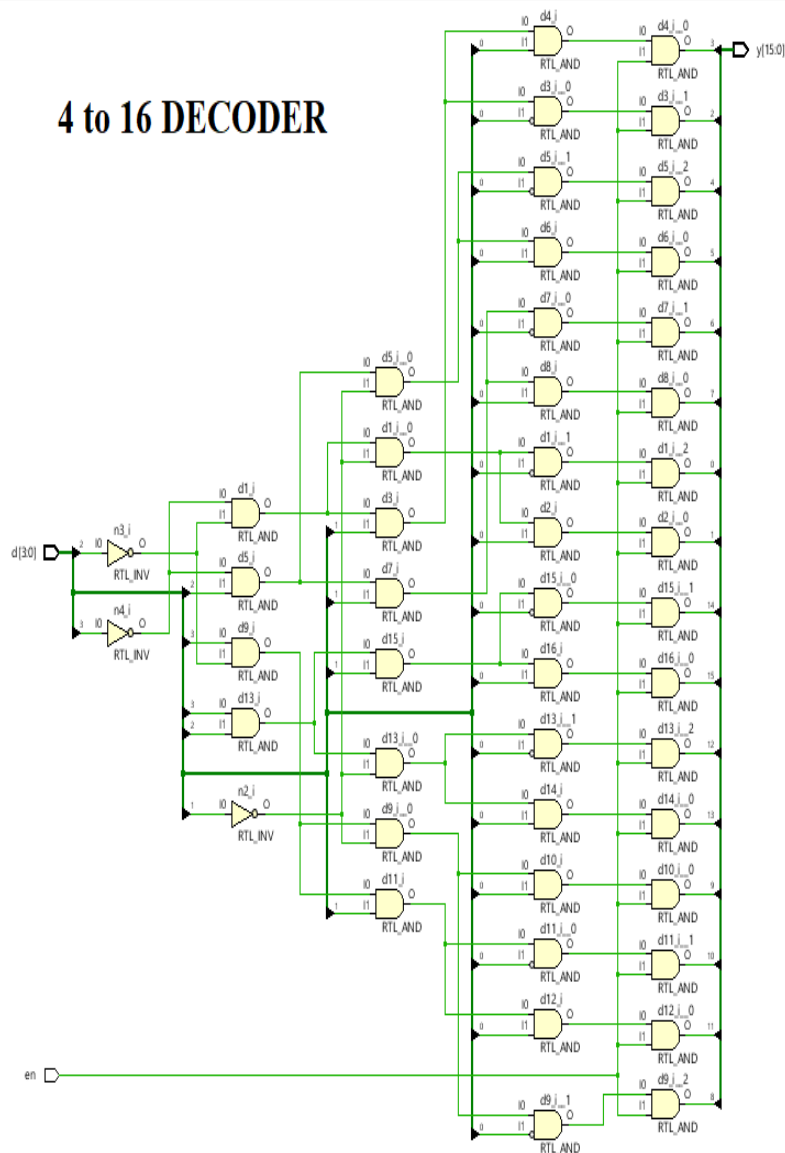
## Waveform



## RTL SCHEMATIC FOR THE CODE

7/0012 21/01/2023 02:19:03

### 4 to 16 DECODER





## 8. DEMUX

### TRUTH TABLE

S[2]	S[1]	S[0]	Y
0	0	0	IN
0	0	1	IN
0	1	0	IN
0	1	1	IN
1	0	0	IN
1	0	1	IN
1	1	0	IN
1	1	1	IN

### Verilog code

```
module demux1_8(input in, input [2:0]s,output [7:0]y);
reg [7:0]y;
always@(*)
begin
case({s[2],s[1],s[0]})
3'b000:y[0]=in;
3'b001:y[1]=in;
3'b010:y[2]=in;
3'b011:y[3]=in;
3'b100:y[4]=in;
3'b101:y[5]=in;
3'b110:y[6]=in;
3'b111:y[7]=in;
endcase
end
endmodule
```

### Testbench

```
module demux1_8tb;
reg in;
reg [2:0] s;
wire [7:0]y;
```

```

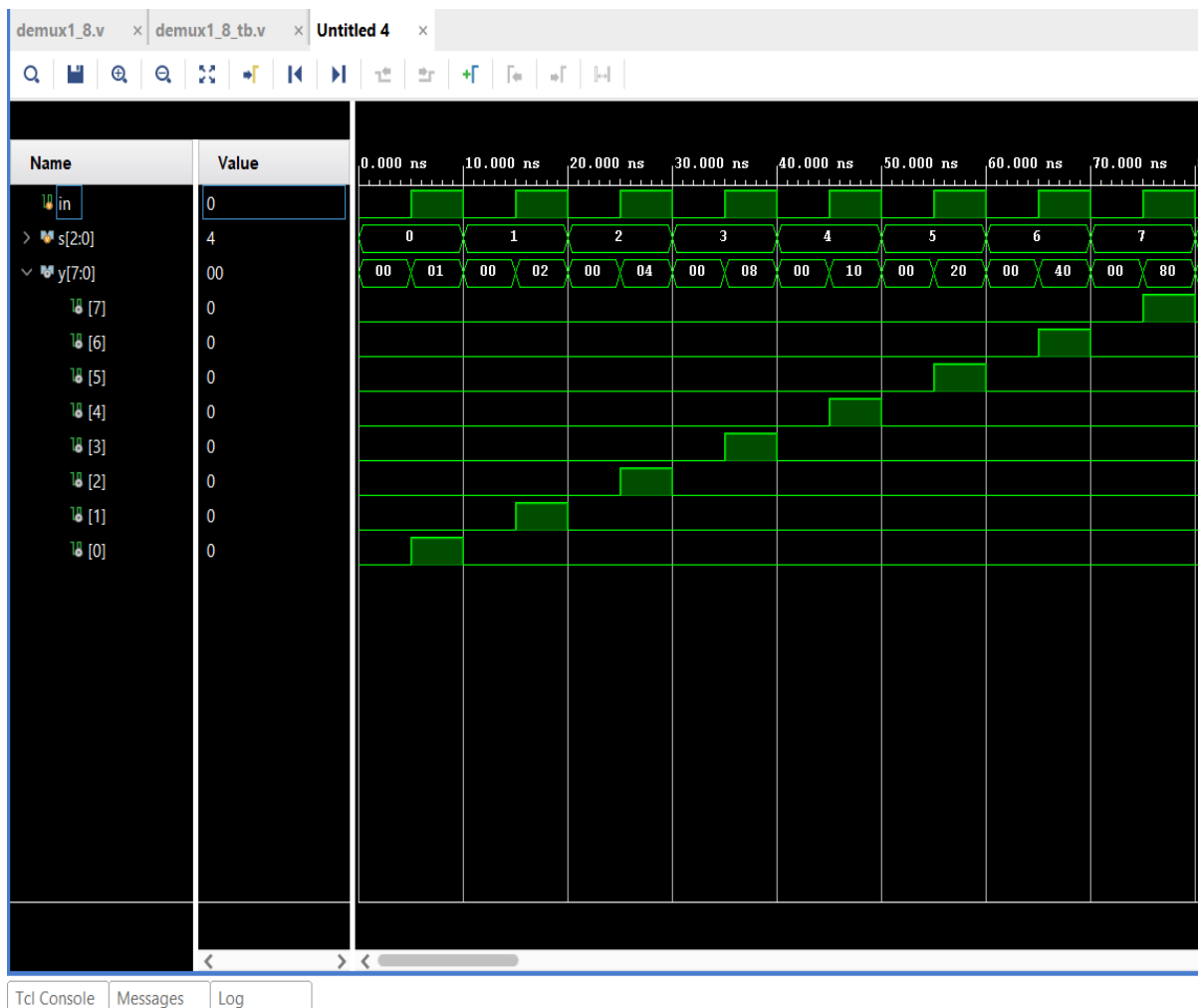
demux1_8 d1(in,s,y);

initial
begin
in=0;s=0;
$finish;
end

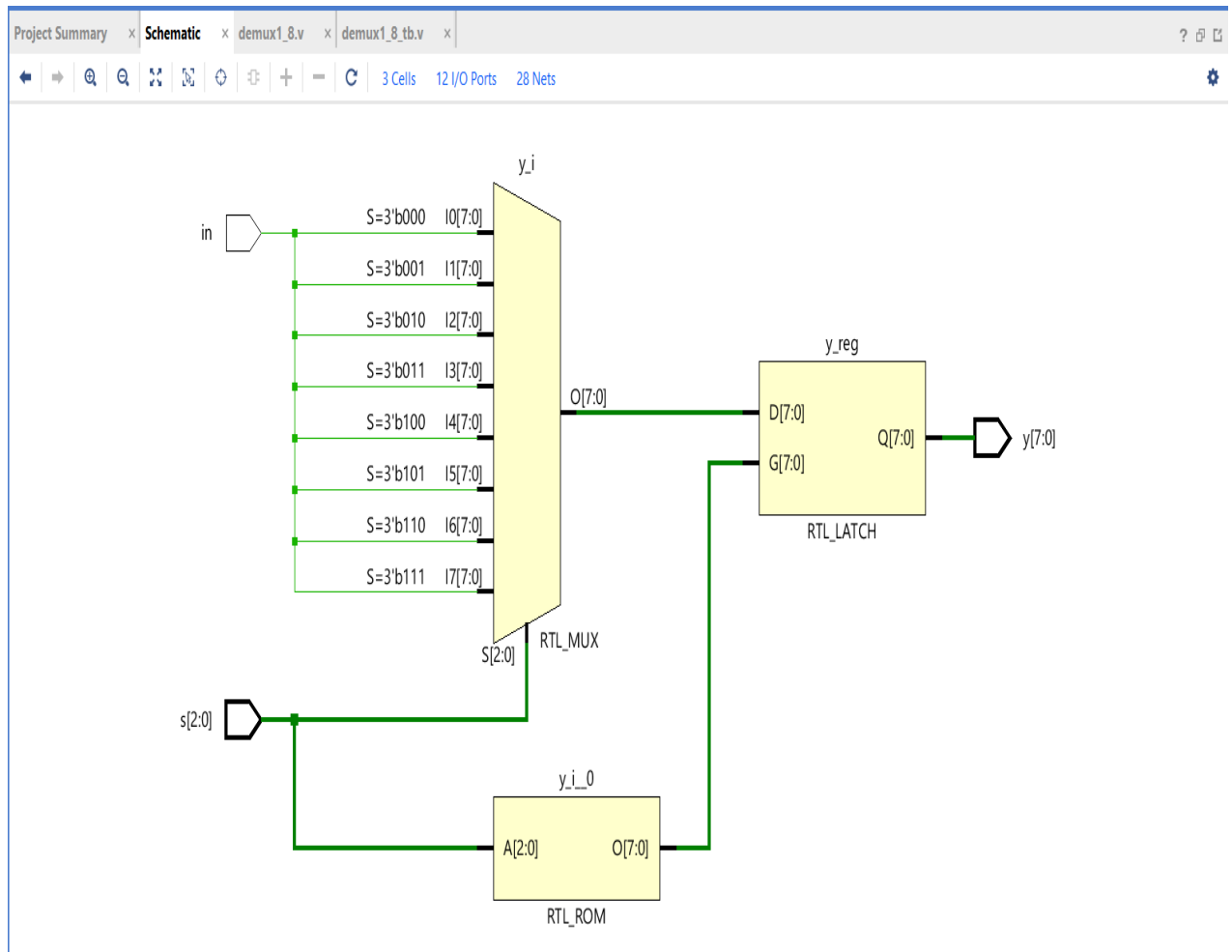
always #5 in=~in;
always #10 s[0]=~s[0];
always #20 s[1]=~s[1];
always #40 s[2]=~s[2];
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## 9. FLIP FLOP

### 9.1 D-FLIP FLOP

#### TRUTH TABLE

CLK	RST	D	Q <sub>n+1</sub>
0	0	0	NO CHANGE
1	1	0	0
1	1	1	1

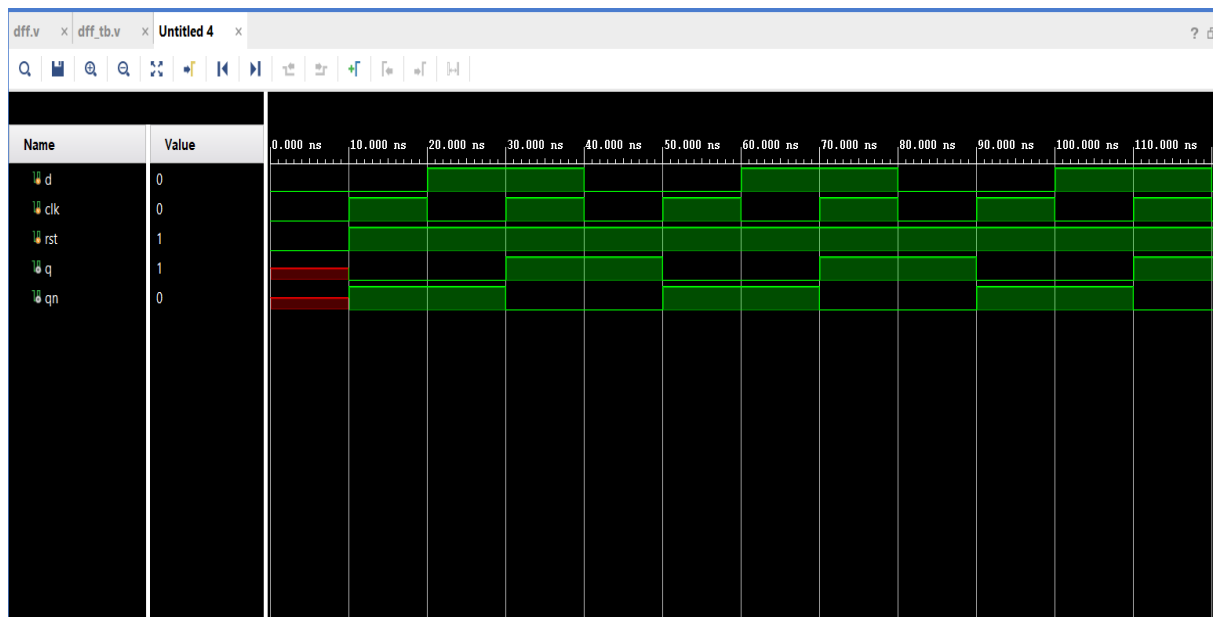
### **Verilog code**

```
module dff(input d,clk,rst,output q,qn);  
reg q;  
always@(posedge clk)  
begin  
if(rst==1'b0)  
q=1'b0;  
else  
q=d;  
end  
assign qn=~q;  
endmodule
```

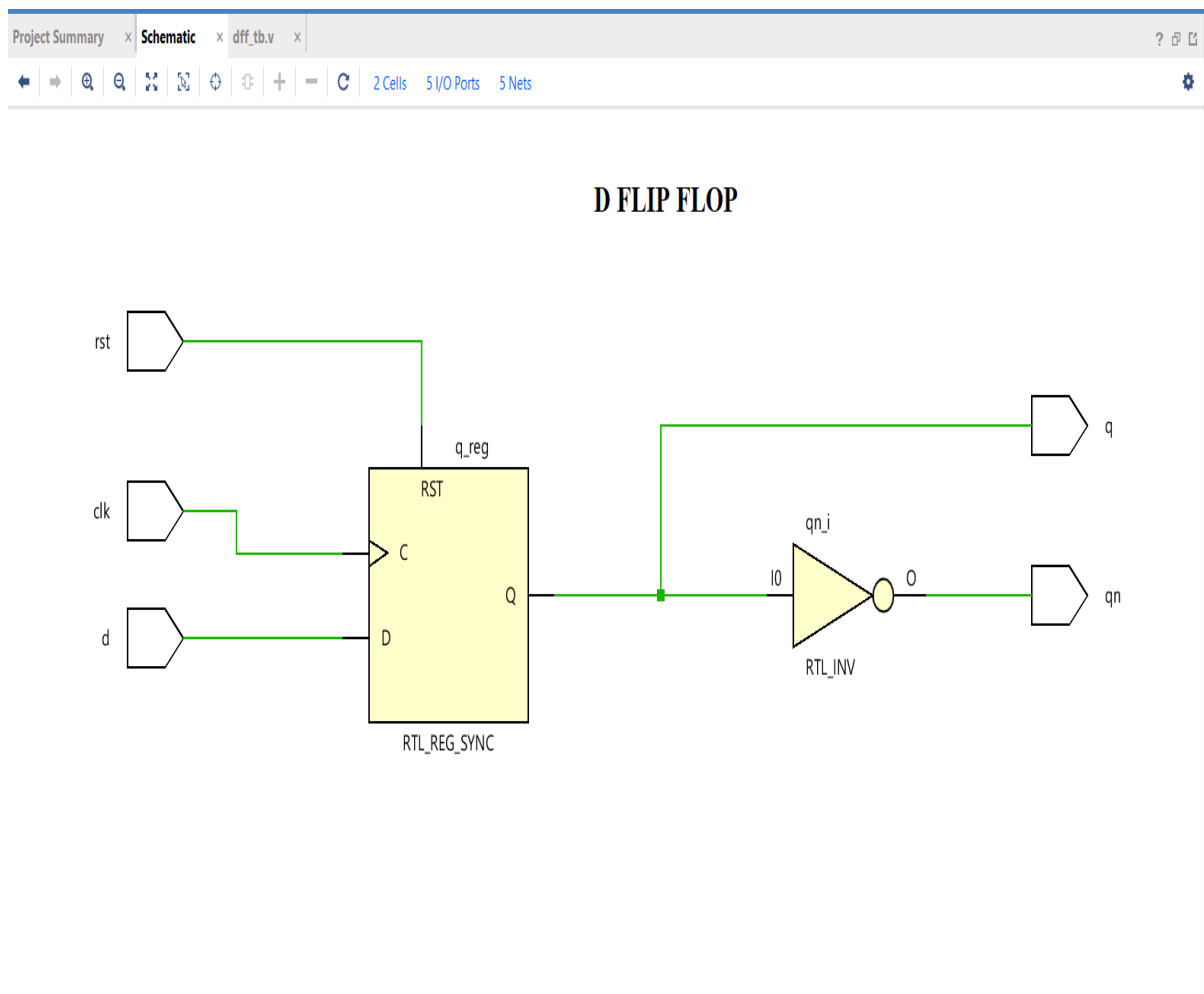
### **Testbench**

```
module dff_tb();  
reg d,clk,rst;  
wire q,qn;  
dff a1(d,clk,rst,q,qn);  
initial  
begin  
d=1'b0;  
clk=1'b0;  
rst=1'b0;  
#10 rst=1'b1;  
end  
always #10 clk=~clk;  
always #20 d=~d;  
endmodule
```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## 9.2 T FLIP FLOP

### TRUTH TABLE

CLK	RST	T	Q <sub>n+1</sub>
0	0	0	No change
1	1	0	Q <sub>n</sub>
1	1	1	Q <sub>n</sub> Bar

### Verilog code

```
module tff1(input t,clk,rst,output q,qb);  
reg q;  
always@(posedge clk or posedge rst)  
begin  
if(rst==0)  
q=1'b0;  
else  
if(t)  
q=q;  
else  
q=~q;  
end  
assign qb=~q;  
endmodule
```

### Testbench

```
module tff1_tb();  
wire q,qb;  
reg t,clk,rst;  
tff1 a1(t,clk,rst,q,qb);  
initial  
begin  
clk=0;  
t=0;rst=0;#10;  
t=1;rst=1;#10;
```

```

t=0;rst=1;#10;

$finish;

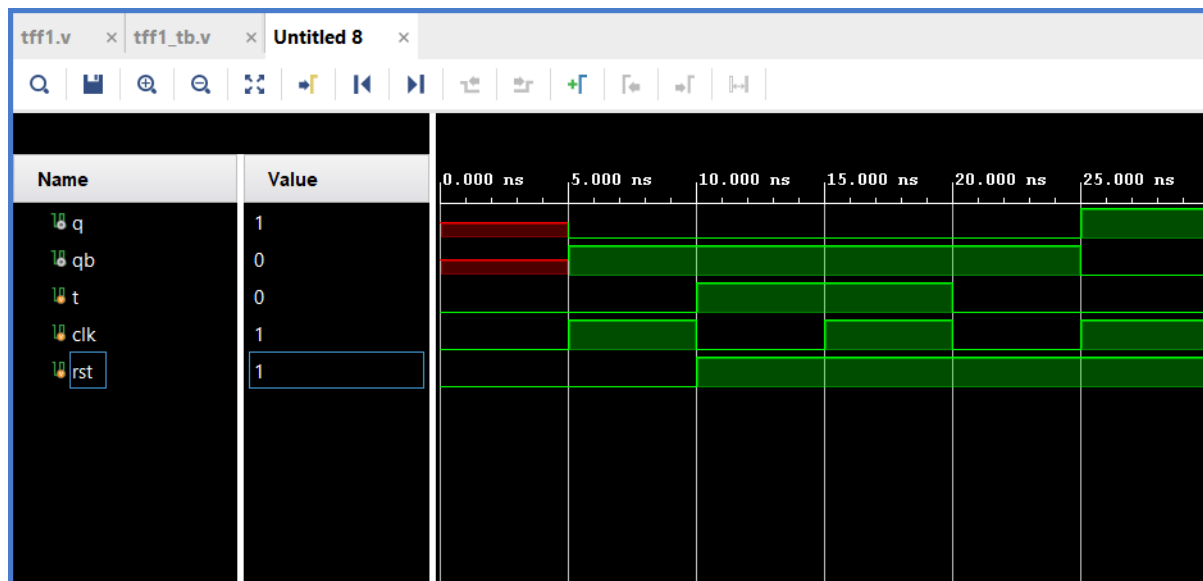
end

always #5 clk=~clk;

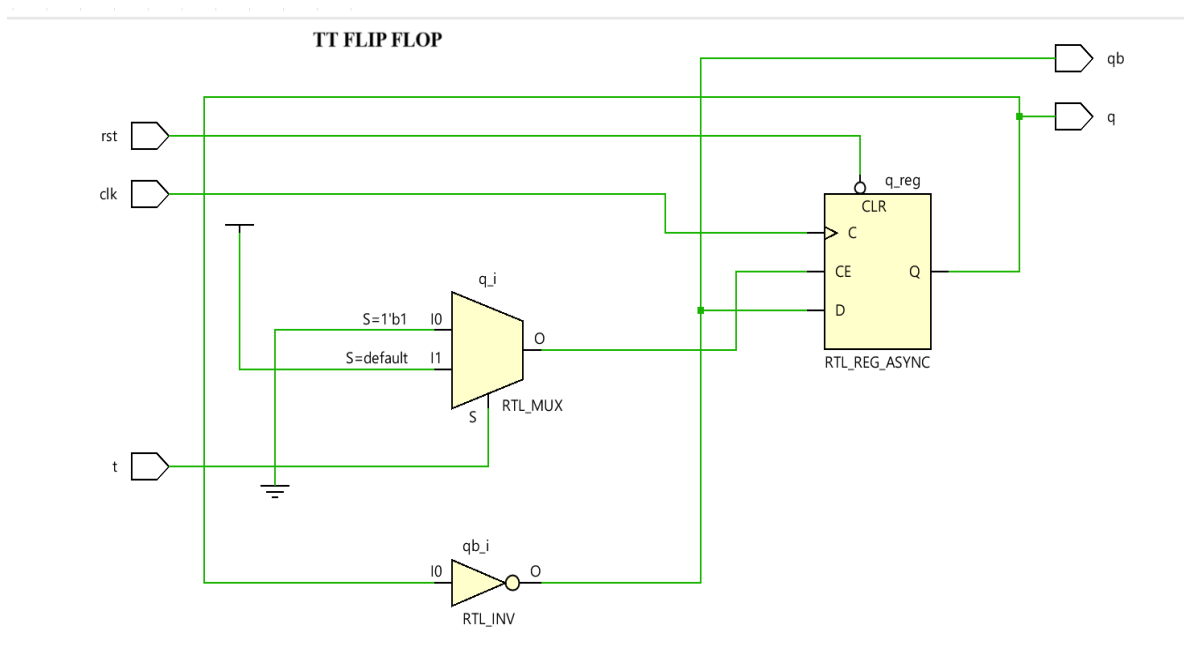
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE



### 9.3 JK FLIP FLOP

#### TRUTH TABLE

CLK	J	K	Q <sub>n+1</sub>
0	0	0	NO CHANGE
1	0	0	NO CHANGE
1	0	1	0
1	1	0	1
1	1	1	TOGGLE

#### Verilog code

```
module jkff(input j,k,clk,output q,qn);  
reg q;  
    always@(posedge clk)  
    begin  
        case({j,k})  
            2'b00:q<=q;  
            2'b01:q<=0;  
            2'b10:q<=1;  
            2'b11:q<=~q;  
        endcase  
    end  
    assign qn=~q;  
endmodule
```

#### Testbench

```
module jkff_tb();  
reg j,k,clk;  
wire q,qn;  
jkff a1(j,k,clk,q,qn);  
initial begin  
    clk=0; j=0; k=0;  
end  
initial
```

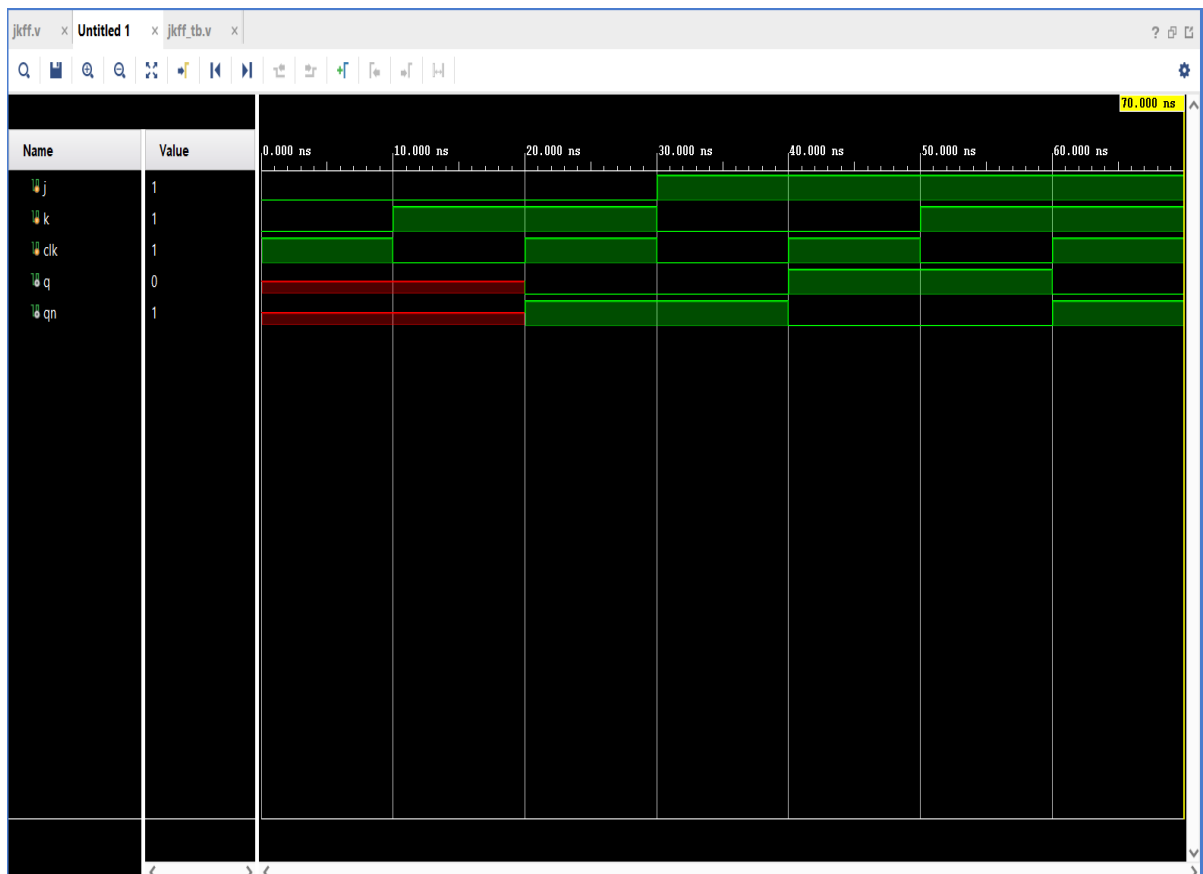


```

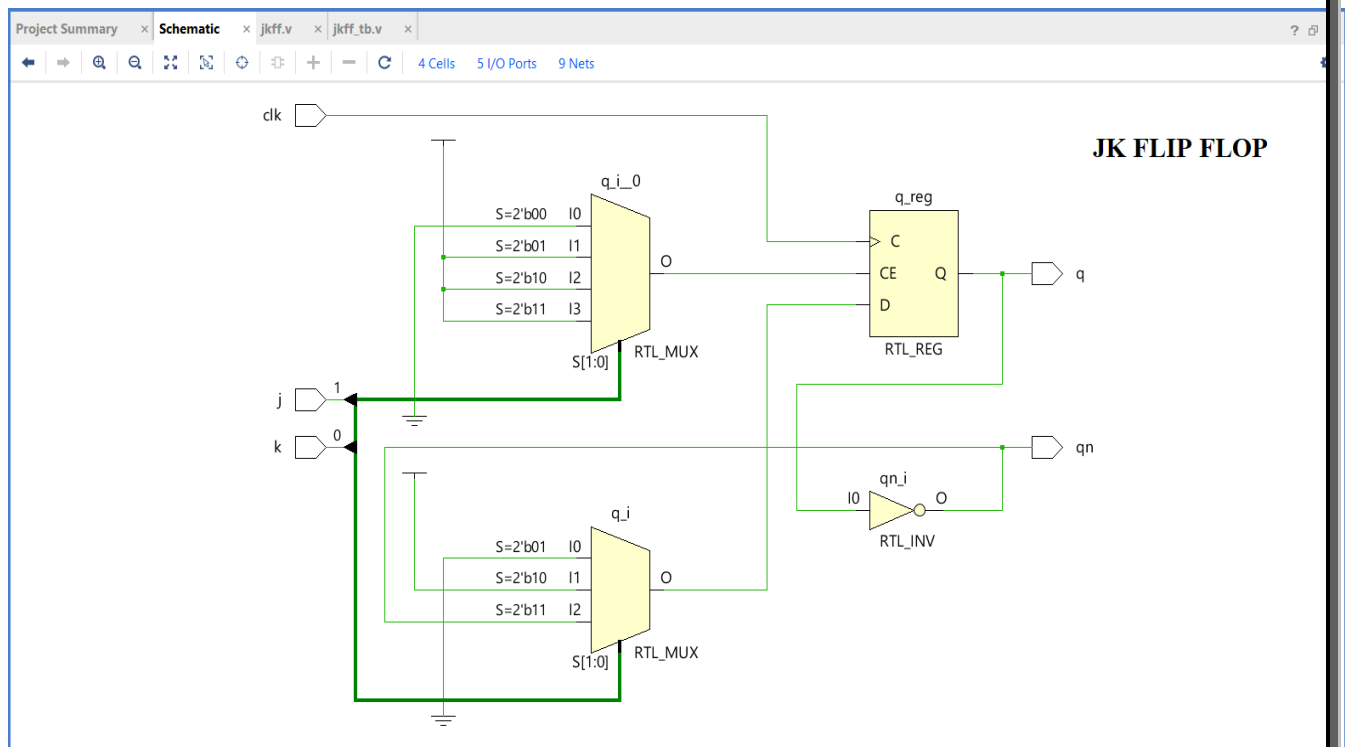
begin
clk=1; j=0; k=0; #10;
clk=0; j=0; k=1; #10;
clk=1; j=0; k=1; #10;
clk=0; j=1; k=0; #10;
clk=1; j=1; k=0; #10;
clk=0; j=1; k=1; #10;
clk=1; j=1; k=1; #10;
$finish;
end
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## 10. n BIT UP/DOWN COUNTER

### Verilog code

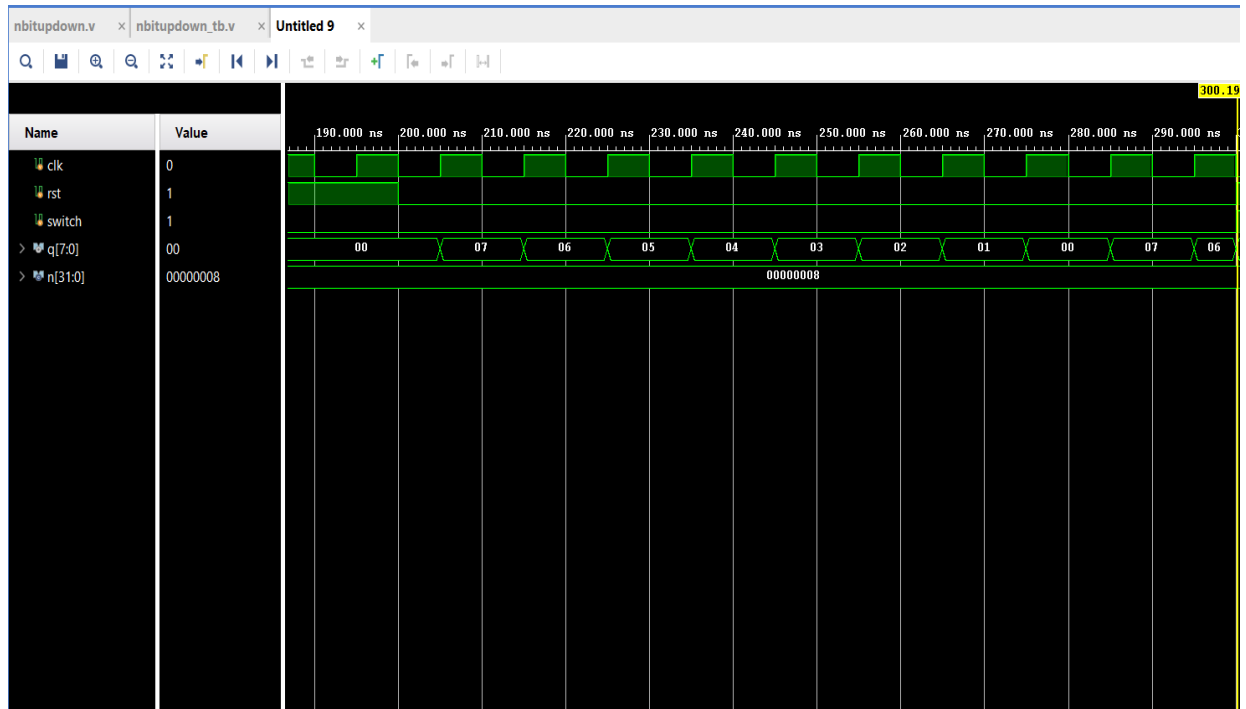
```
module nbitupdown(input clk,rst,switch,output[n-1:0]q);
parameter n=8;
reg[n-1:0]q;
always @(posedge clk or posedge rst)
begin
if(rst==1)
q<=0;
else
if(switch==1)
if(q==(n-1))
q<=0;
else
```

```
        q<=q+1;
    else
    if(q==0)
        q<=(n-1);
    else
        q<=q-1;
    end
endmodule
```

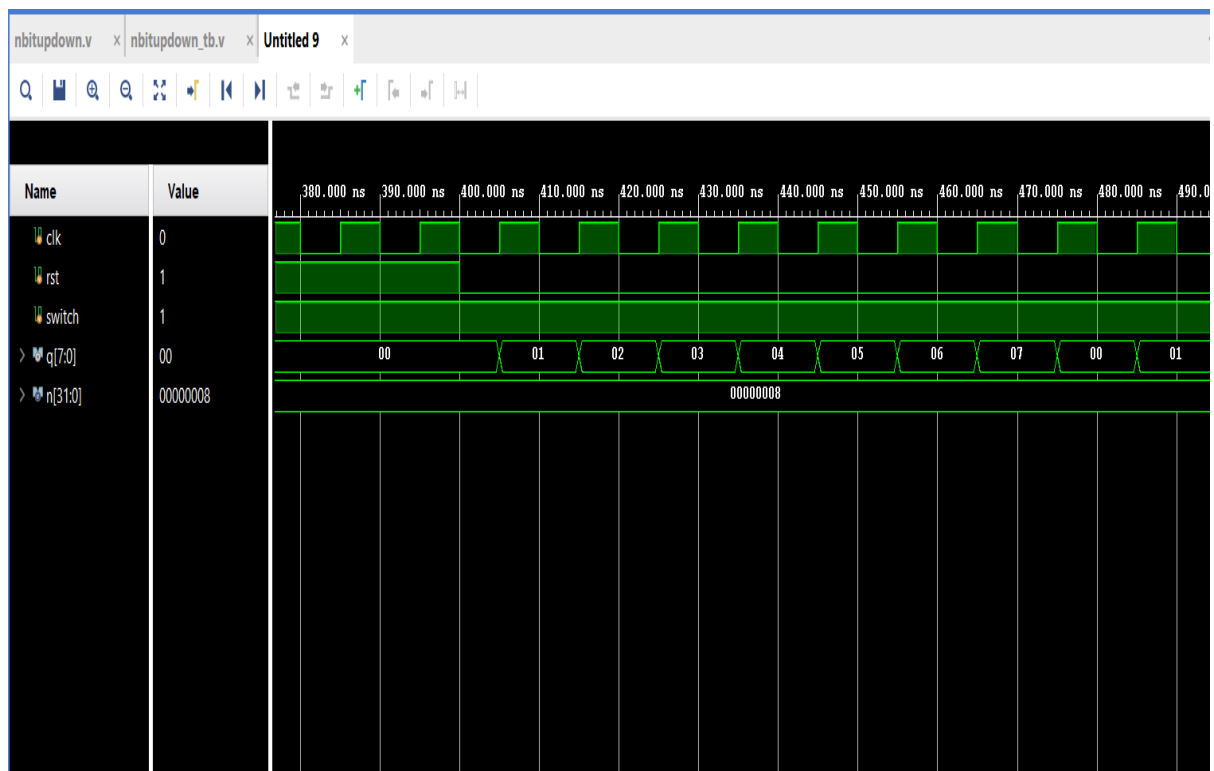
### **Testbench**

```
module nbitupdown_tb();
reg clk,rst,switch;
wire [n-1:0]q;
parameter n=8;
nbitupdown n1(clk,rst,switch,q);
initial
begin
    clk = 0;
    rst=0;
    switch=0;
end
always #5 clk = ~clk;
always #100 rst=~rst;
always #300 switch=~switch;
endmodule
```

## Waveform

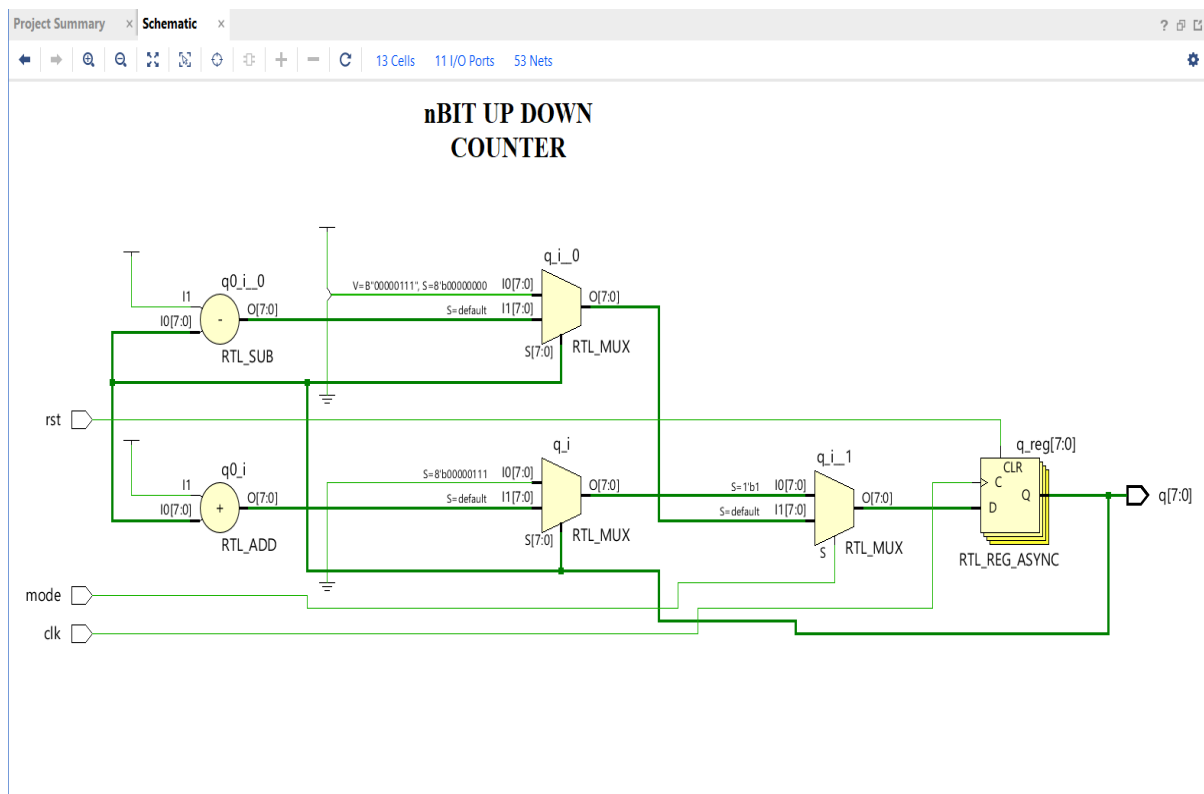


**n BIT DOWN COUNTER**



**n BIT UP COUNTER**

## RTL SCHEMATIC FOR THE CODE



## 11. MOD-n COUNTER

### Verilog code

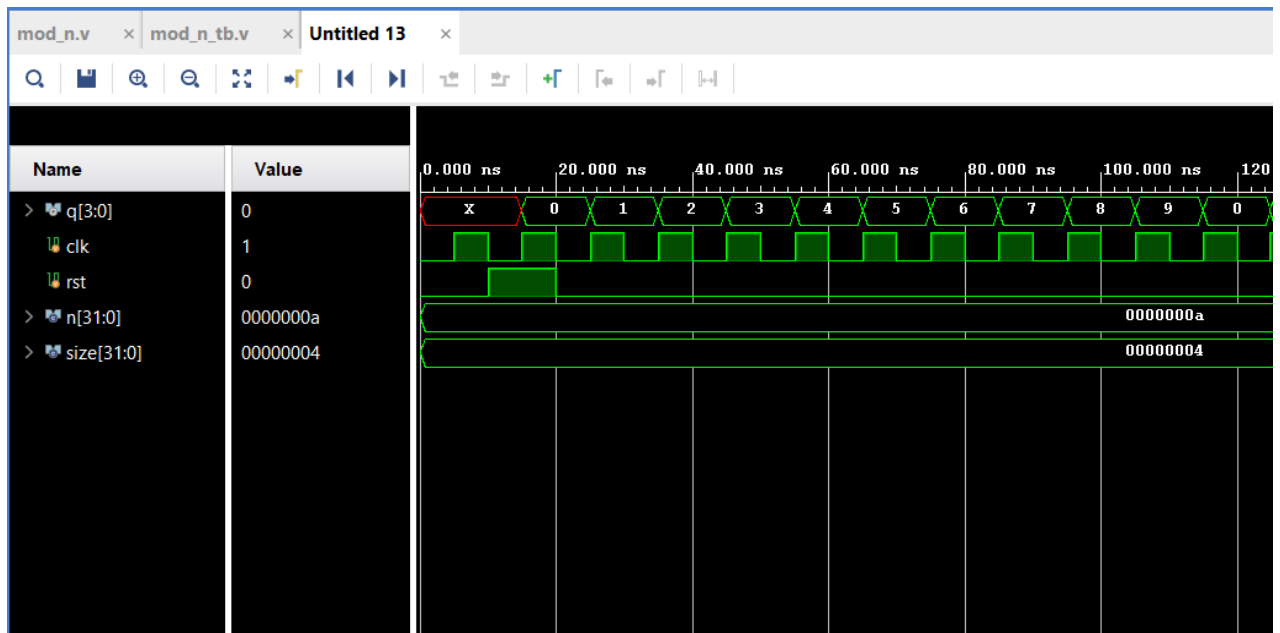
```
module mod_n(output [width-1:0]q,input clk,rst);  
    reg [size-1:0]q;  
    parameter n=10;  
    parameter size=4;  
    always @(posedge clk)  
    begin  
        if (rst)  
            begin  
                q <= 0;  
            end  
        else  
            begin
```

```
        if (q == n-1)
            q <= 0;
        else
            q <= q + 1;
        end
    end
endmodule
```

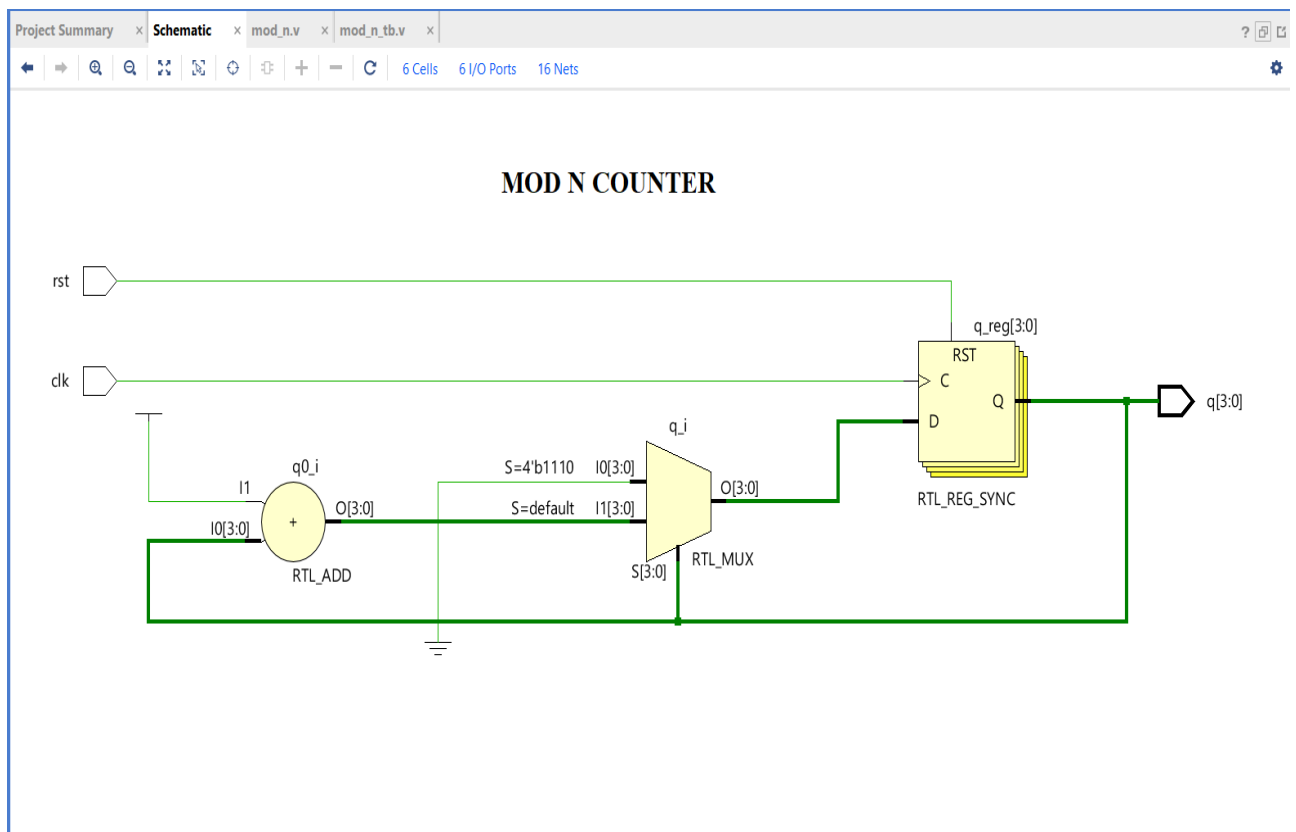
### **Testbench**

```
module mod_n_tb();
parameter n=10;
parameter size=4;
wire [size-1:0]q;
reg clk,rst;
mod_n m1(q,clk,rst);
    initial
    begin
        clk = 0;
        rst = 0;
        #10 rst = 1;
        #10 rst = 0;
        $finish;
    End
    always #5 clk = ~clk;
endmodule
```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## **12. GRAY CODE COUNTER**

### **Verilog code**

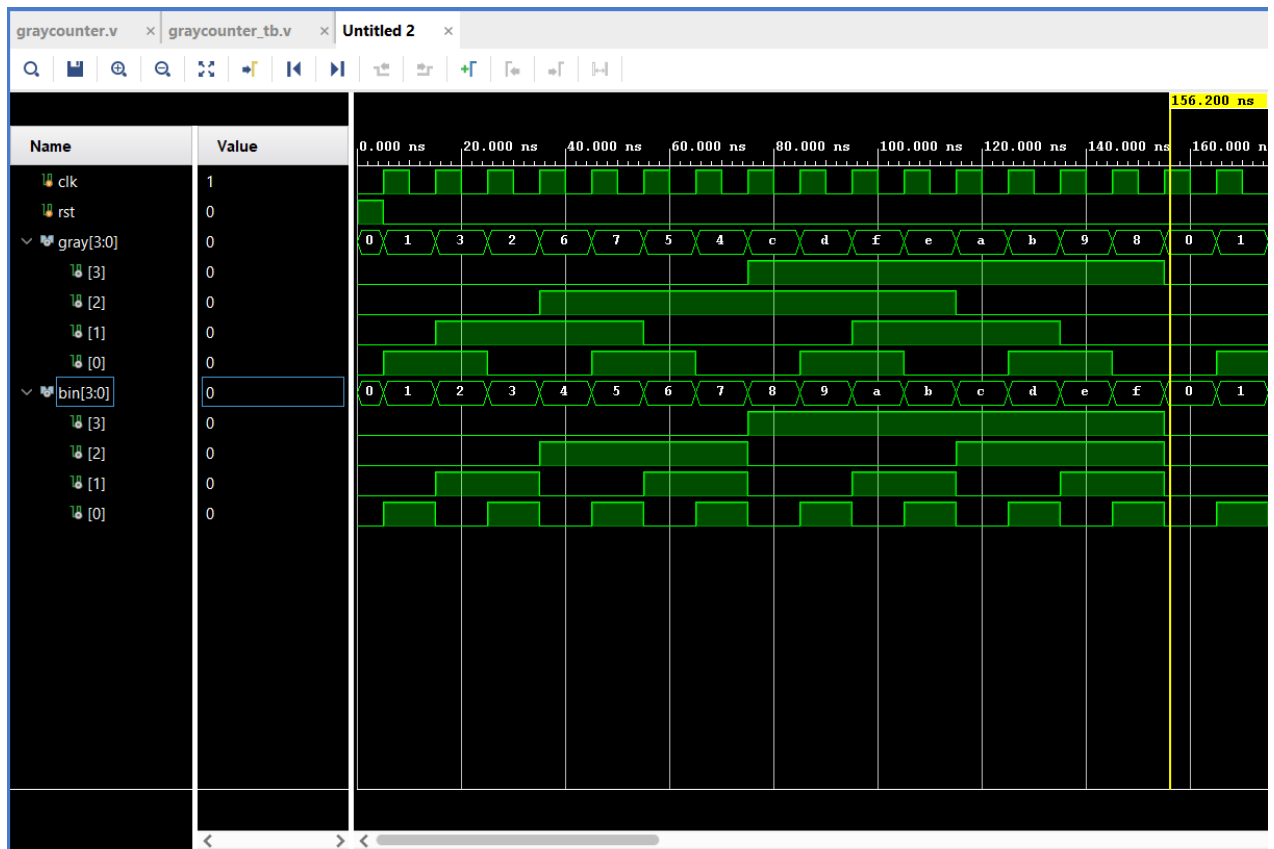
```
module graycounter (input clk,rst,output [3:0]gray,bin);
reg [3:0]bin;
always @(posedge clk or posedge rst)
    begin
        if (rst)
            bin <= 4'b0000;
        else
            bin <= bin + 1;
        end
    assign gray[3]=bin[3];
    assign gray[2]=bin[3]^bin[2];
    assign gray[1]=bin[2]^bin[1];
    assign gray[0]=bin[1]^bin[0];
endmodule
```

### **Testbench**

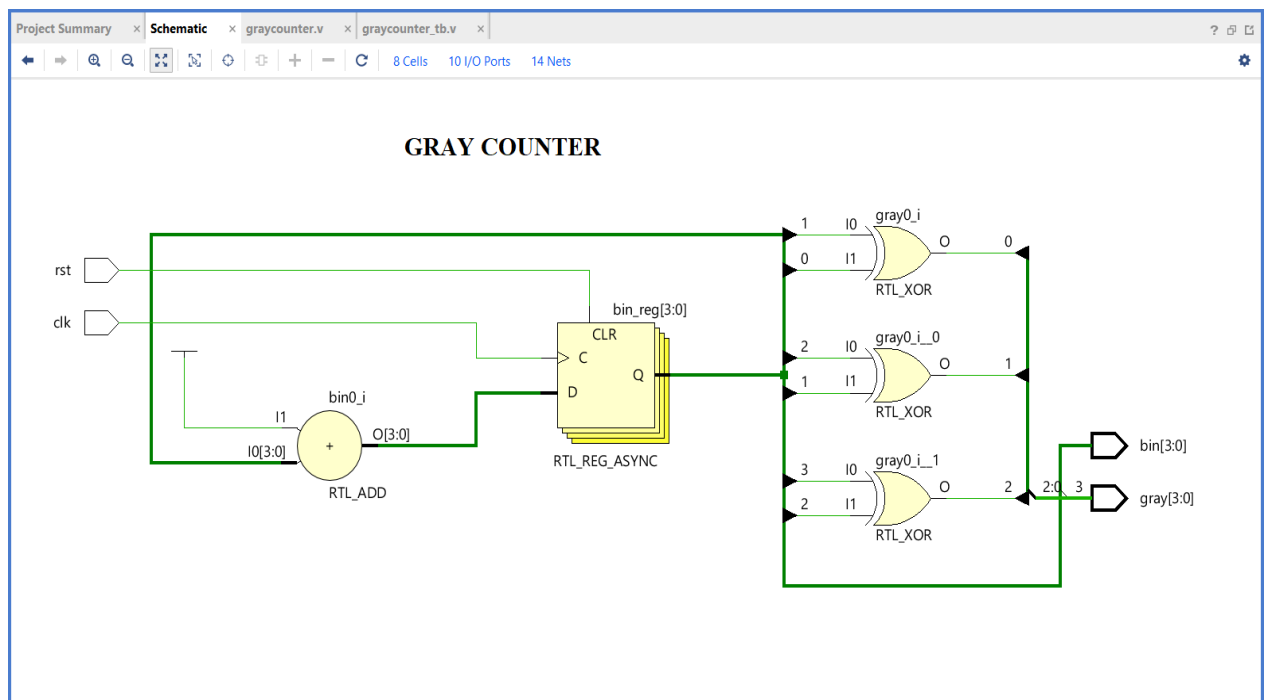
```
module graycounter_tb();
reg clk,rst;
wire [3:0]gray,bin;
graycounter g1(clk,rst,gray,bin);
initial
begin
    clk=0;
    rst=1;
    #5
    rst=0;
end
always #5 clk=~clk;
endmodule
```



## Waveform



## RTL SCHEMATIC FOR THE CODE



### **13. LINEAR FEEDBACK SHIFT REGISTER**

#### **Verilog code**

```
module lfsr(input clk,rst,output [n-1:0] q);
parameter n=4;
reg [n-1:0]q,qnxt;
wire shift;
always @(posedge clk or negedge rst)
begin
if(~rst)
q<=1;
else
q<=qnxt;
end
always @(shift,q)
qnxt={ shift,q[n-1:1]};
assign shift=q[3]^q[2];
endmodule
```

#### **Testbench**

```
module lfsr_tb();
parameter n=3;
reg clk,rst;
wire [n-1:0]q;
lfsr l1(clk,rst,q);
always
begin
clk=1'b0;
#5;
clk=1'b1;
```

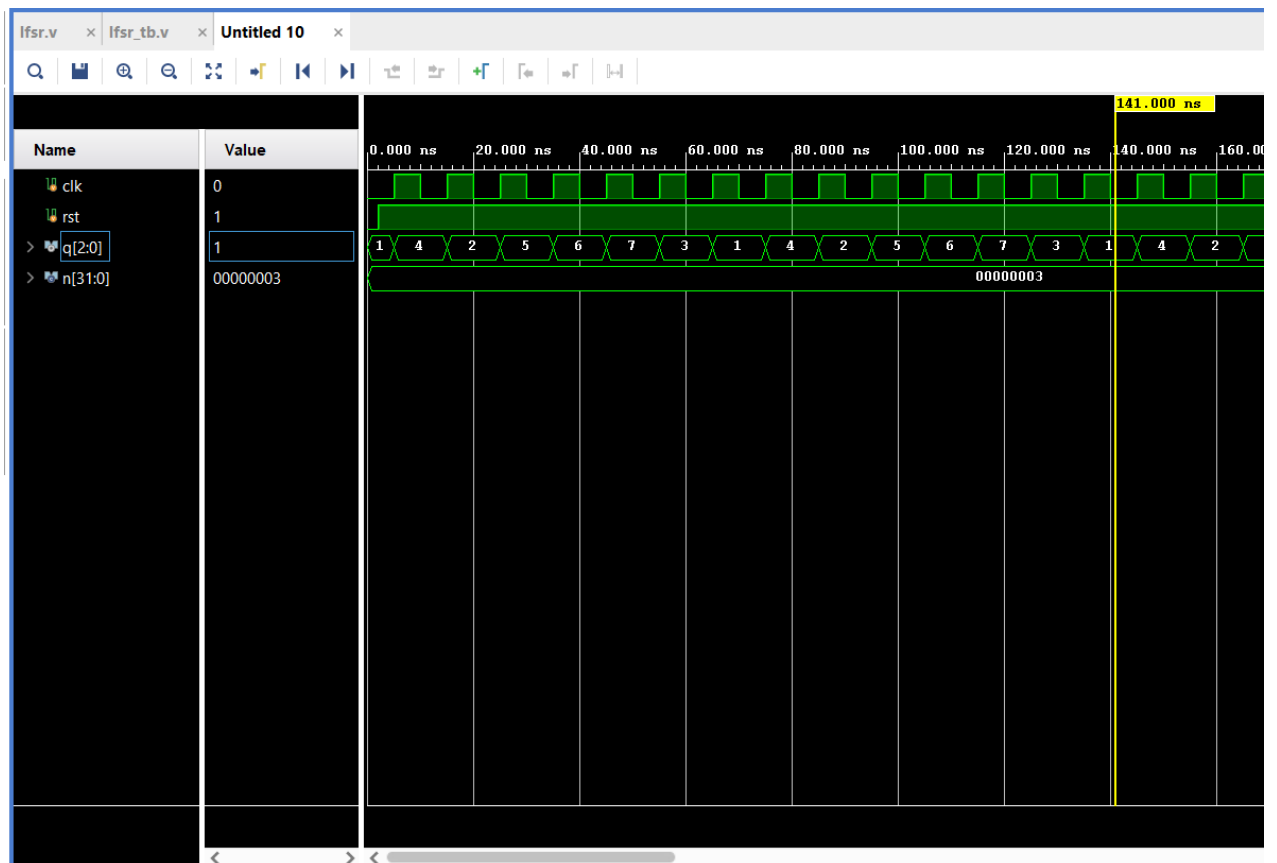
```

#5;
end

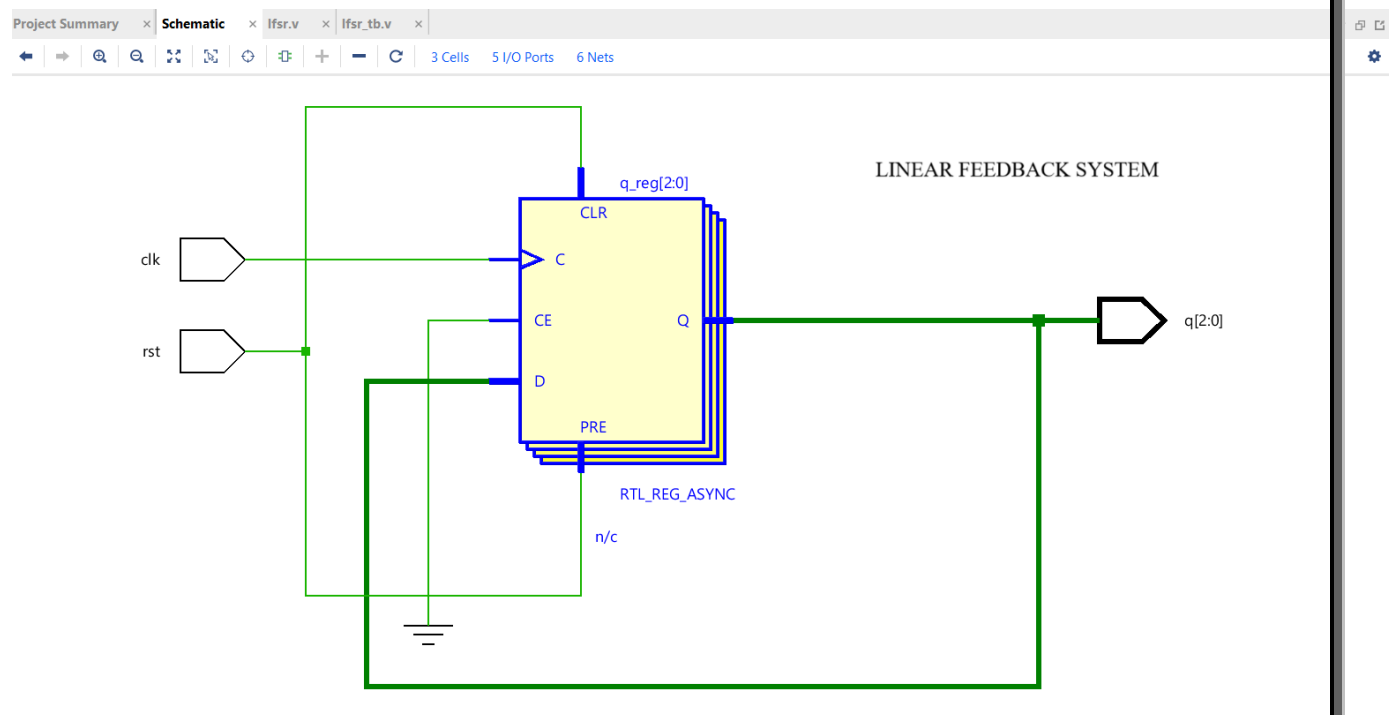
initial
begin
rst=1'b0;
#2;
rst=1'b1;
#10;
end
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## 14. RING COUNTER

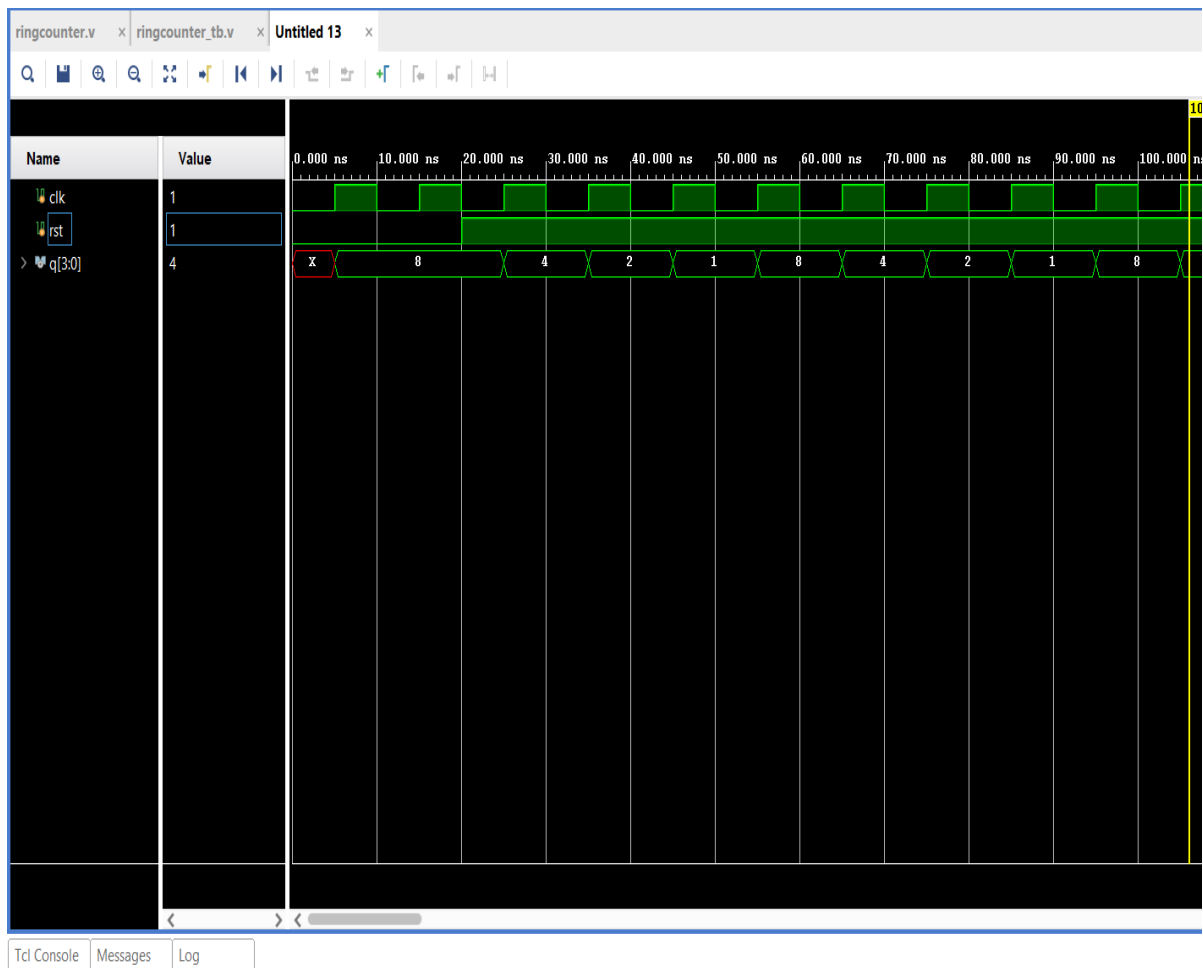
## Verilog code

```
module ringcounter(output [3:0] q,input clk,rst);
reg [3:0]q;
always @(posedge clk)
if(rst==0)
q<=4'b1000;
else
begin
q[3]<=q[0];
q[2]<=q[3];
q[1]<=q[2];
q[0]<=q[1];
end
endmodule
```

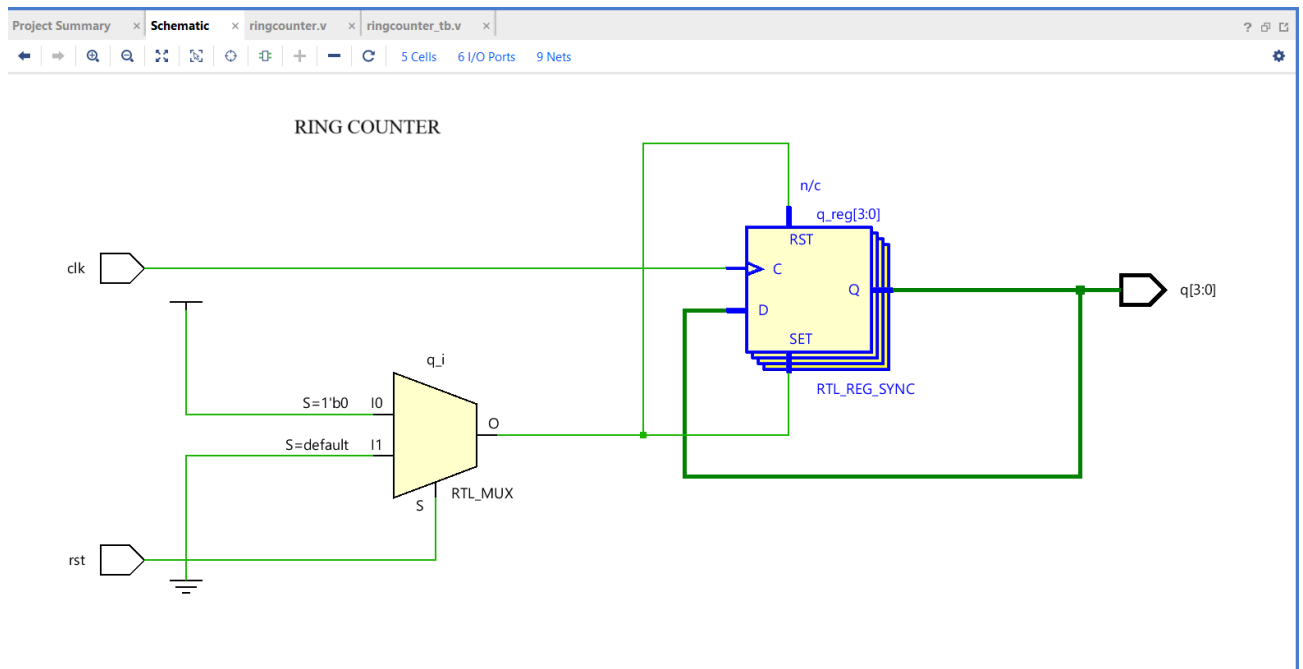
## Testbench

```
module ringcounter_tb();  
  
reg clk,rst;  
  
wire [3:0]q;  
  
ringcounter r1(q,clk,rst);  
  
initial  
  
begin  
  
clk=0;  
  
rst=0;  
  
#20 rst=1'b1;  
  
end  
  
always #5 clk=~clk;  
  
endmodule
```

## Waveform



## RTL SCHEMATIC FOR THE CODE



## 15. TWISTED RING COUNTER

### Verilog code

```
module twistedringcounter(output [3:0] q, input clk,rst);
reg [3:0]q;
always @(posedge clk)
begin
if(rst)
q<=4'b1000;
else
case (q)
4'b1000: q <= 4'b1100;
4'b1100: q <= 4'b1110;
4'b1110: q <= 4'b1111;
4'b1111: q <= 4'b0111;
4'b0111: q <= 4'b0011;
4'b0011: q <= 4'b0001;
```

```

4'b0001: q <= 4'b0000;

default: q <= 4'b0001;

endcase

end

endmodule

```

## Testbench

```

module twistedringcounter_tb();

wire [3:0] q;

reg clk,rst;

twistedringcounter a1 (q,clk,rst);

initial

begin

clk=0;

rst=1;

#20 rst=1'b0;

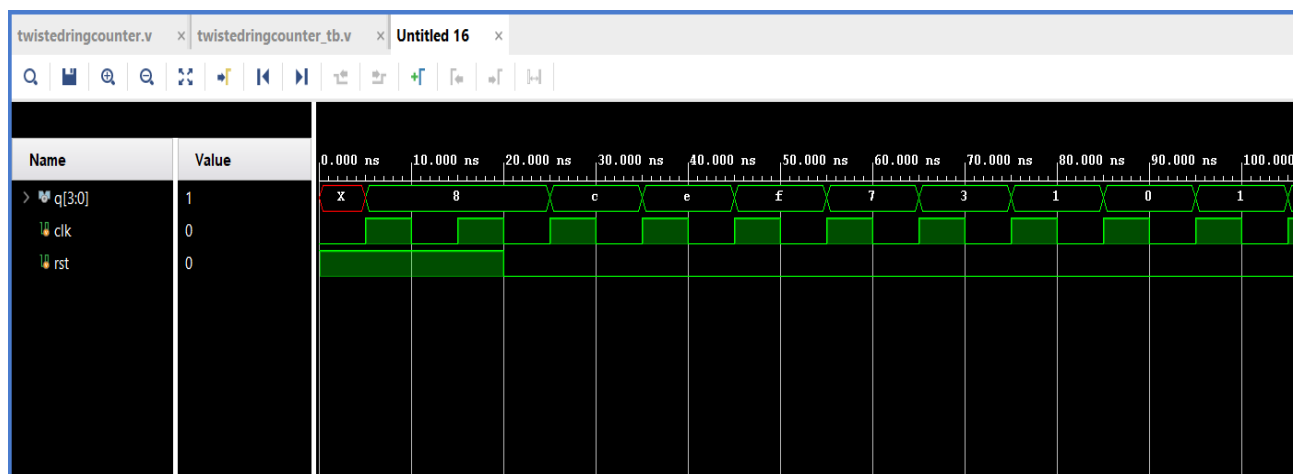
end

always #5 clk=~clk;

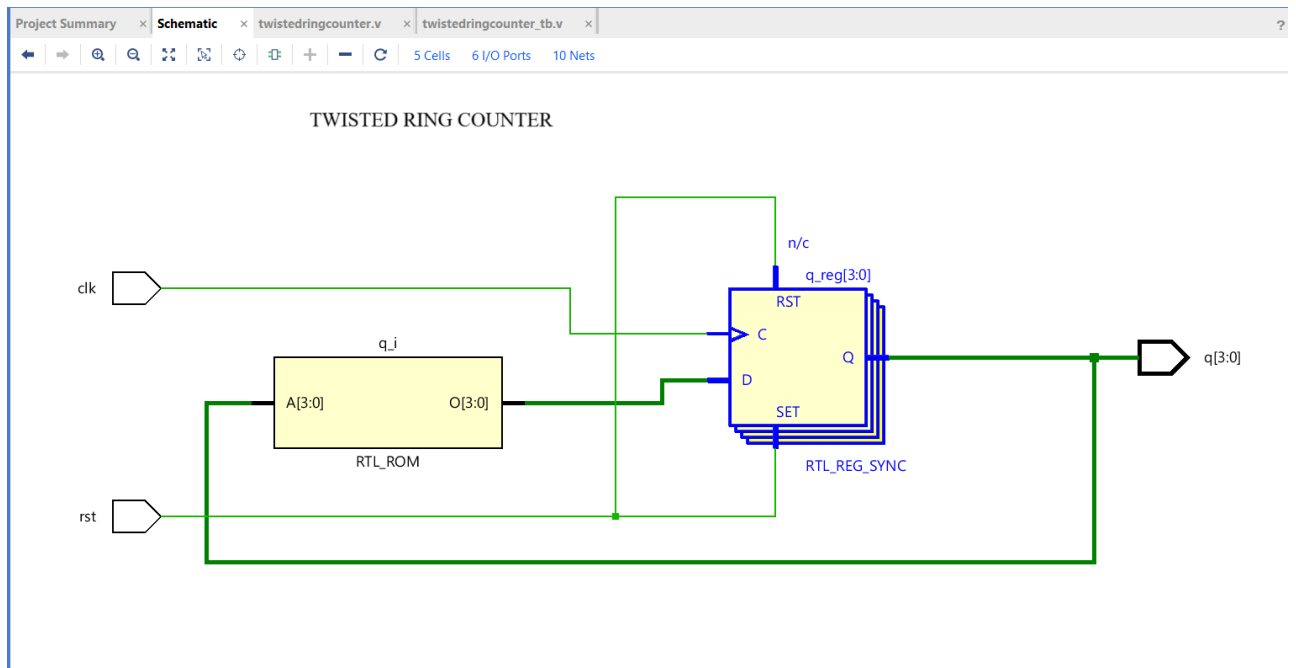
endmodule

```

## Waveform



## RTL SCHEMATIC FOR THE CODE





# PROJECT-1

## 32-BIT ALU

### 32- BIT ALU

A 32-bit ALU (Arithmetic Logic Unit) is a commonly used digital circuit used for various arithmetic and logic operations on 32-bit binary data.

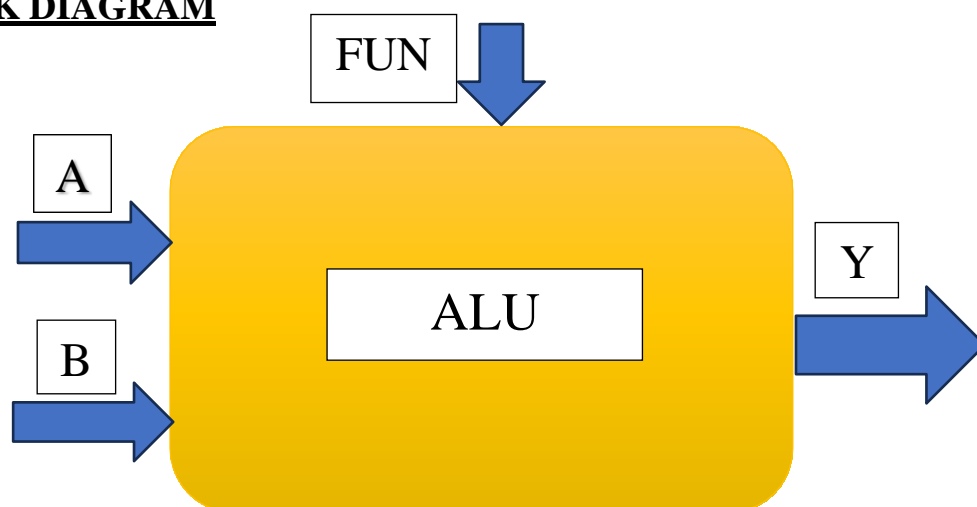
The ALU operates on binary data, which consists of 1s and 0s, and processes this data according to the instructions provided by user.

As per below Verilog code there will be some of the operations which will be happening like logical AND, OR, XOR, arithmetic addition, subtraction, multiplication, division and modulo and complementary operation.

### TRUTH TABLE

A[31:0]	B[31:0]	FUNCTION	OUTPUT
a=32'h00000010	b=32'h00000110	fun=3'b000	Y=32'h00000010
a=32'h00000110	b=32'h00001000	fun=3'b001	Y=32'h00001110
a=32'h00001010	b=32'h00000111	fun=3'b010	Y=32'h00000011
a=32'h000001011	b=32'h00001001	fun=3'b011	Y=32'h00002012
a=32'h00001001	b=32'h00000110	fun=3'b100	Y=32'h00000ef1
a=32'h00000010	b=32'h00000110	fun=3'b101	Y=32'h00001100
a=32'h00001000	b=32'h00001000	fun=3'b110	Y=32'hffffefff
a=32'h00000010	b=32'h00000010	fun=3'b111	Y=32'h00000000

### BLOCK DIAGRAM



## APPLICATION

- Microprocessor & CPU
- Control unit operations
- Signal processing
- Cryptographic operations
- Data compression
- Graphics processing

## VERILOG CODE

```
module alu (input [31:0]a,input [31:0]b,input [2:0]fun,output [31:0]y);  
  reg [31:0]y;  
  ○ always@(*)  
  begin  
    ○ if(fun==3'b000)  
    ○ y=a&b;  
    ○ else if (fun==3'b001)  
    ○ y=a|b;  
    ○ else if (fun==3'b010)  
    ○ y=a&b;  
    ○ else if (fun==3'b011)  
    ○ y=a+b;  
    ○ else if (fun==3'b100)  
    ○ y=a-b;  
    ○ else if (fun==3'b101)  
    ○ y=a*b;  
    ○ else if (fun==3'b110)  
    ○ y=~a;  
    ○ else if (fun==3'b111)  
    ○ y=a^b;  
    else  
    ○ y=32'bz;  
  end  
endmodule
```

---

## TESTBENCH

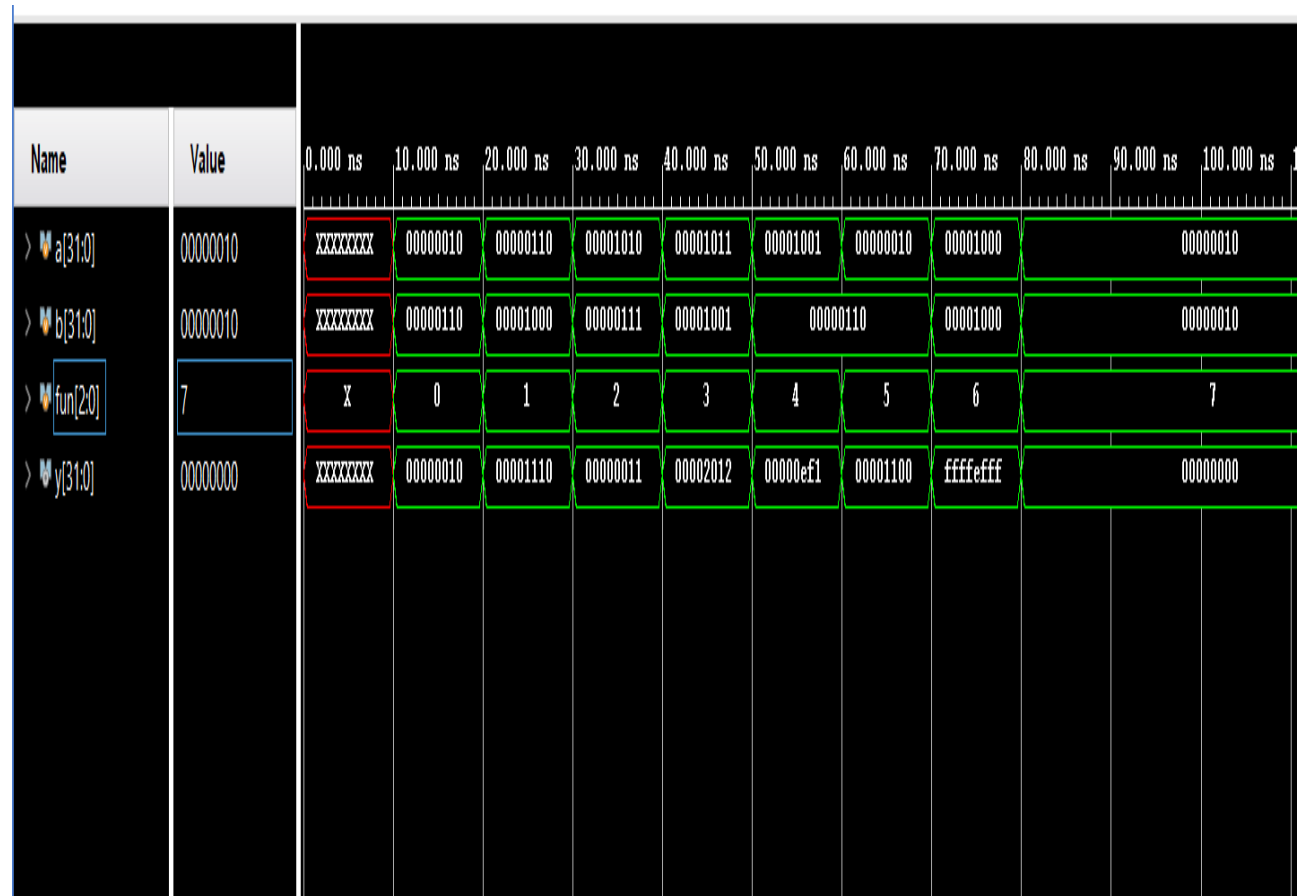
```

timescale 1ns / 1ps

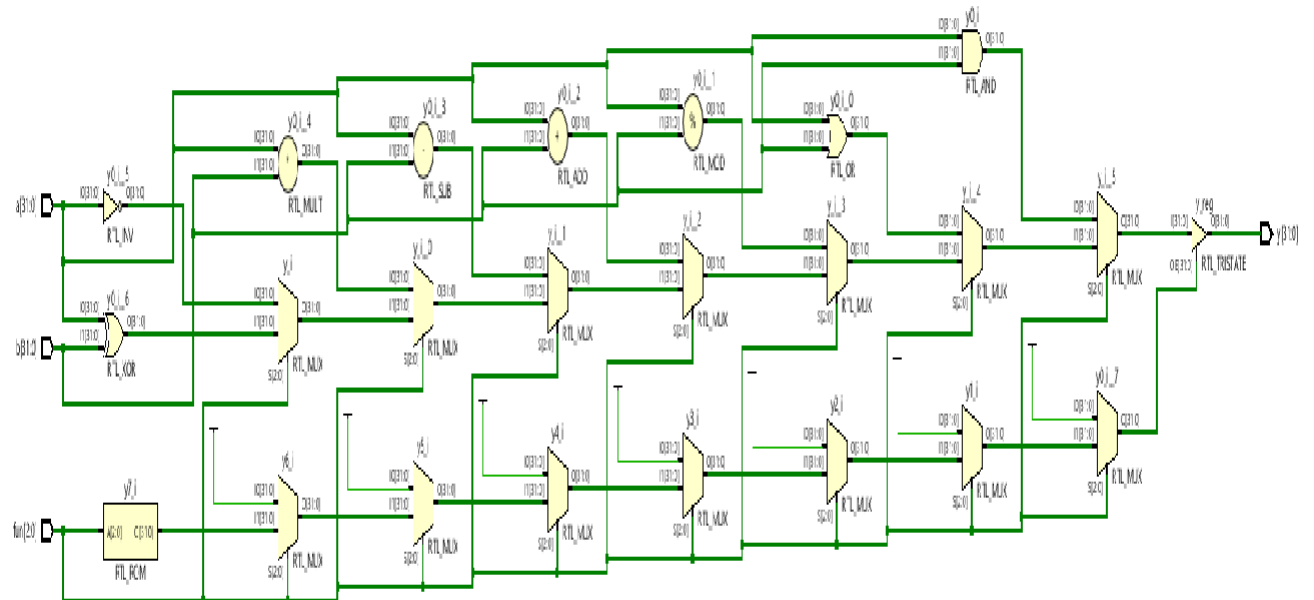
module alu_tb();
reg [31:0]a;
reg [31:0]b;
reg [2:0]fun;
wire [31:0]y;
alu m1 (a,b,fun,y);
initial
begin
  #10 a=32'h00000010;b=32'h00000110;fun=3'b000;
  #10 a=32'h00000110;b=32'h00001000;fun=3'b001;
  #10 a=32'h00001010;b=32'h00000111;fun=3'b010;
  #10 a=32'h000001011;b=32'h00001001;fun=3'b011;
  #10 a=32'h00001001;b=32'h00000110;fun=3'b100;
  #10 a=32'h00000010;fun=3'b101;
  #10 a=32'h00001000;b=32'h00001000;fun=3'b110;
  #10 a=32'h00000010;b=32'h00000010;fun=3'b111;
  #100 $finish;
end
endmodule

```

## WAVEFORM



## RTL SCHEMATIC FOR CODE

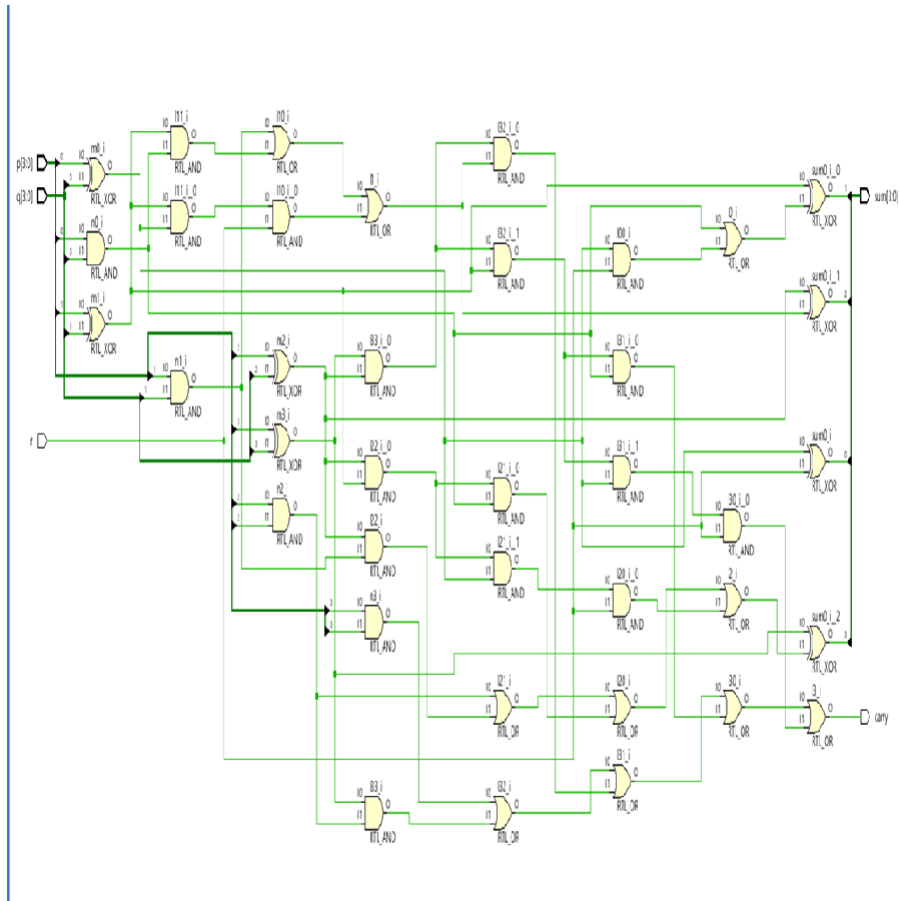


## **PROJECT-2**

### **CARRY LOOK AHEAD ADDER**

Carry look ahead adder is a adder which will add two binary numbers with fast and efficient method by using carry signals and it reduces propagation delays while calculating carry and which will end up into speed up adding in the circuit.

#### **BLOCK DIAGRAM**














#### **APPLICATIONS**

- Arithmetic logic units
- Digital signal processing
- Scientific computing
- Embedded systems
- Error detection and correction
- Digital design and verification

## VERILOG CODE

Project Summary x Schematic x carrylook.v x

D:/vivado program/carrylook.srscs/sources\_1/new/carrylook.v



```
1  `timescale 1ns / 1ps
2  module carrylook(input [3:0]p,q, input r,output [3:0]sum,output carry);
3      wire m0,m1,m2,m3,n0,n1,n2,n3,l0,l1,l2,l3;
4      assign m0=p[0]^q[0],
5              m1=p[1]^q[1],
6              m2=p[2]^q[2],
7              m3=p[3]^q[3];
8
9      assign n0=p[0]&q[0],
10             n1=p[1]&q[1],
11             n2=p[2]&p[2],
12             n3=p[3]&p[3];
13
14     assign l0=n0|(m0 & r),
15             l1=n1|(m1 & n0) | (m1 & m0 &r),
16             l2=n2|(m2 & n1) | (m2 & m1 & n0)| (m2 & m1 & m0 & r),
17             l3=n3| (m3 & n2) | (m3 & m2 &l1) | (m3 & m2 & m1 & n0) | (m3 & m2 & m1 & m0 & r);
18     assign sum[0]=m0^r,
19             sum[1]=m1^l0,
20             sum[2]=m2^l1,
21             sum[3]=m3^l2,
22             carry=l3;
23     endmodule
24
25
```

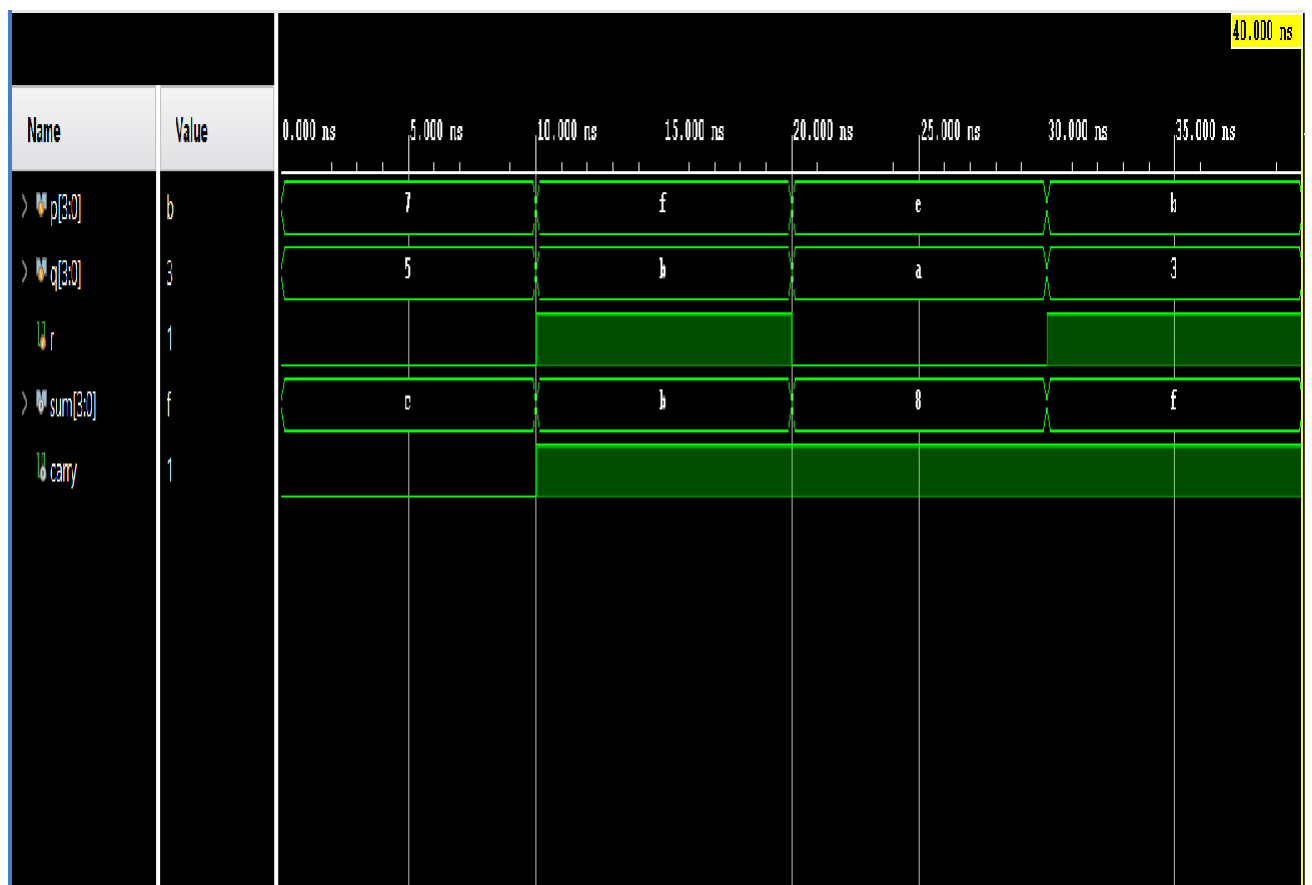
## TESTBENCH

```

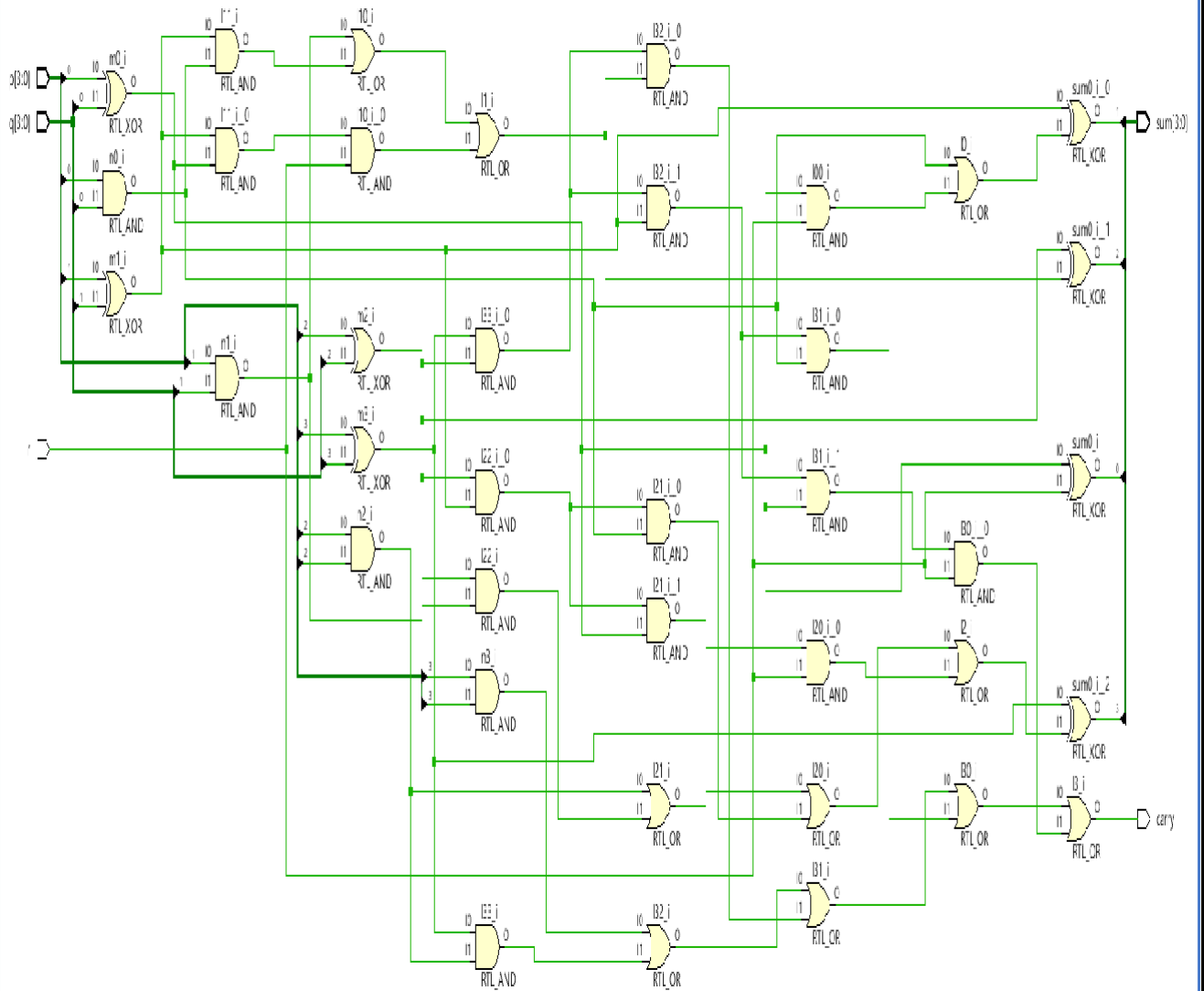
`timescale 1ns / 1ps
module carrylook_tb();
    reg [3:0]p,q;
    reg r;
    wire [3:0]sum;
    wire carry;
    carrylook_al(p,q,r,sum,carry);
    initial
    begin
        p = 4'b0111;q = 4'b0101;r=1'b0;#10;
        p = 4'b1111;q = 4'b1011;r=1'b1;#10;
        p = 4'b1110;q = 4'b1010;r=1'b0;#10;
        p = 4'b1011;q = 4'b0011;r=1'b1;#10;
    $finish;
    end
endmodule

```

## WAVEFORM



## RTL SCHEMATIC FOR CODE





## PROJECT-3

### UNIVERSAL SHIFT REGISTER

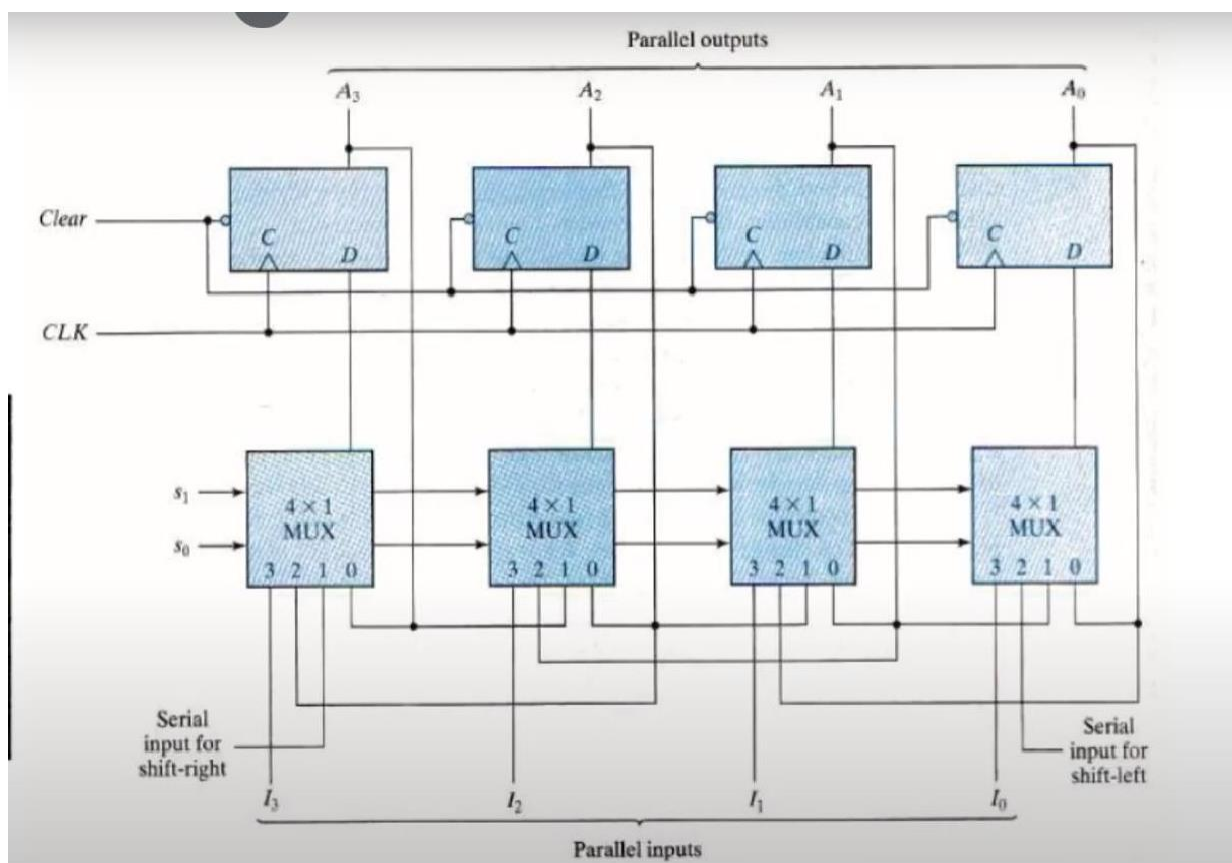
Universal Shift Register is register which is used that performs shifting operations based on the given inputs like shift left and shift right operations.

It is considered to be universal as it has the functionalities of different shift registers like SIPO, PIPO, SISO, PISO in a single device.

#### TRUTH TABLE

MODE S1	MODE S0	REGISTER OPERATION
0	0	No Change
0	1	Shift Right
1	0	Shift Left
1	1	Parallel Load

#### BLOCK DAIGRAM



## VERILOG CODE

```
`timescale 1ns / 1ps

module unisr(output [3:0] o,input [3:0] p_in,
input [1:0] s,input rst,input l_in,input r_in,
input clk);
    reg [3:0] o;
    always@(posedge clk)
    begin
        if(rst==1)
            o=0;
        else
            case(s)
                2'b00:o=o;
                2'b01:o=r_in;
                2'b10:o=l_in;
                2'b11:o=p_in;
            endcase
        end
    endmodule
```

## TESTBENCH

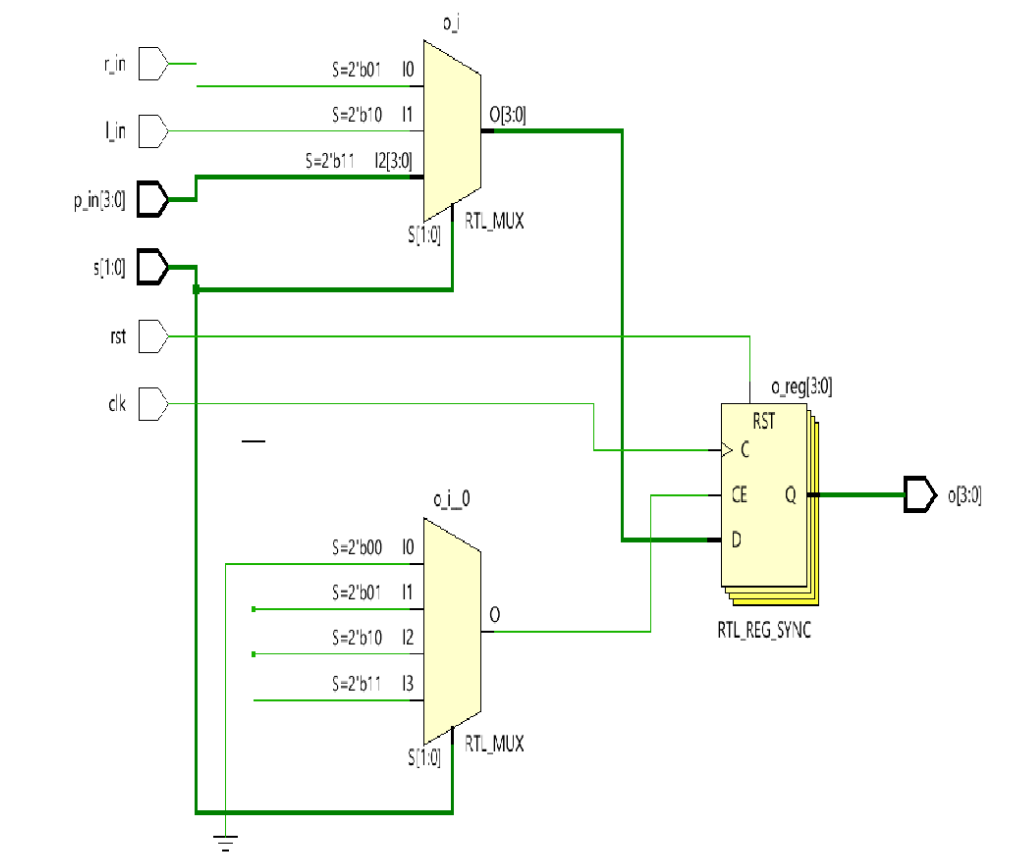
```
`timescale 1ns / 1ps
module unisr_tb();
    wire [3:0] o;
    reg [3:0] p_in;
    reg [1:0] s;
    reg rst;
    reg l_in;
    reg r_in;
    reg clk;
    unisr u1(o,p_in,s,rst,l_in,r_in,clk);
    initial
    begin
        ○ p_in=0;
        ○ s=0;
        ○ clk=0;
        ○ l_in=0;
        ○ r_in=0;

        ○ rst=1;
        ○ #10;
        ○ rst=0;#20;
        ○ r_in=1;s=2'b01;#10;
        ○ l_in=0;s=2'b10;#10;
        ○ p_in=4'b1010;s=2'b11;#10;
        ○ →$finish;
        end
        ○ always #5 clk=~clk;
    endmodule
```

## WAVEFORM



## RTL SCHEMATIC FOR CODE



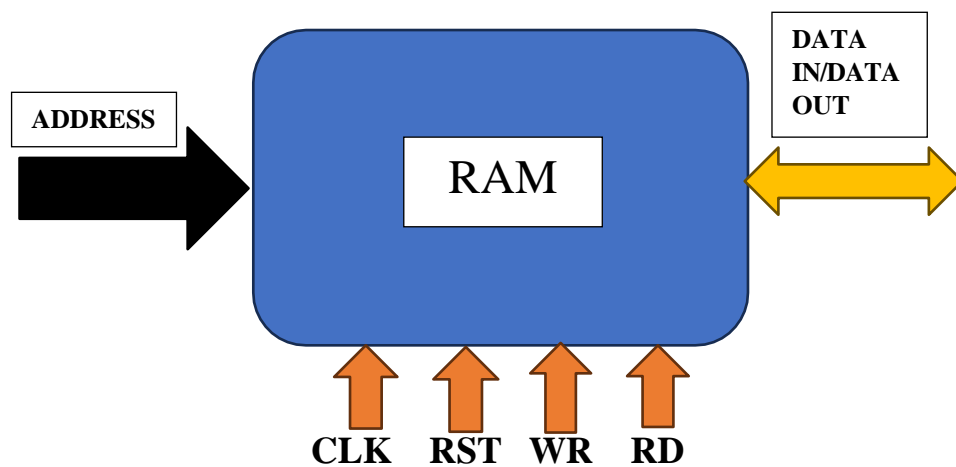
## **PROJECT-4**

### **RAM**

RAM stands for Random Access Memory is a computer memory that is allowed data's to be stored and can be retrieved from CPU.

Where data is stored in some location and some address from that particular location data can be stored and retrieved.

### **BLOCK DIAGRAM**














### **APPLICATIONS**

- System memory
- Web browsing
- Database management system
- Data analysis
- Data science
- Server operations

## VERILOG CODE

ram.v \* x ram\_tb.v x Untitled 10 x

D:/vivado program/ram.srscs/sources\_1/new/ram.v

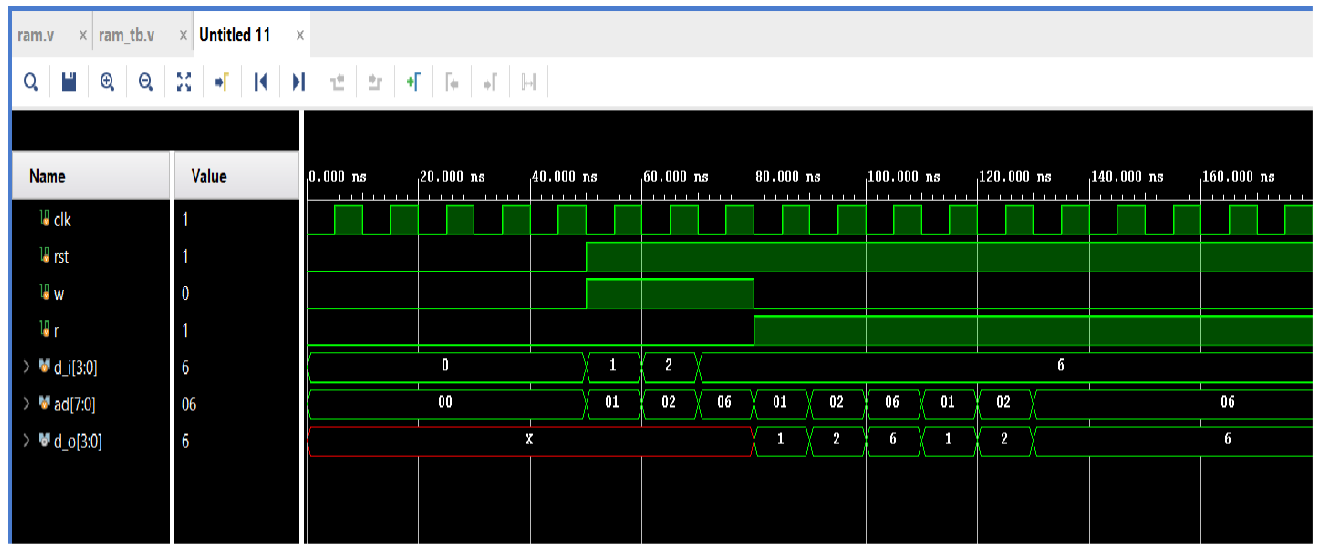
```
1      `timescale 1ns / 1ps
2
3      module ram(input clk,rst,w,r,
4      input [3:0] d_i,
5      input [7:0] ad,output [3:0] d_o);
6      reg [3:0]d_o;
7      ○ reg[3:0] mem[255:0];
8      always @(posedge clk)
9      ○ begin
10     ○ if(rst==0)
11         d_o=4'b×;
12     ○ else
13     ○     if(w)
14     ○         mem[ad]=d_i;
15     ○         if(r)
16         d_o=mem[ad];
17     end
18     endmodule
19
```

### **Testbench**

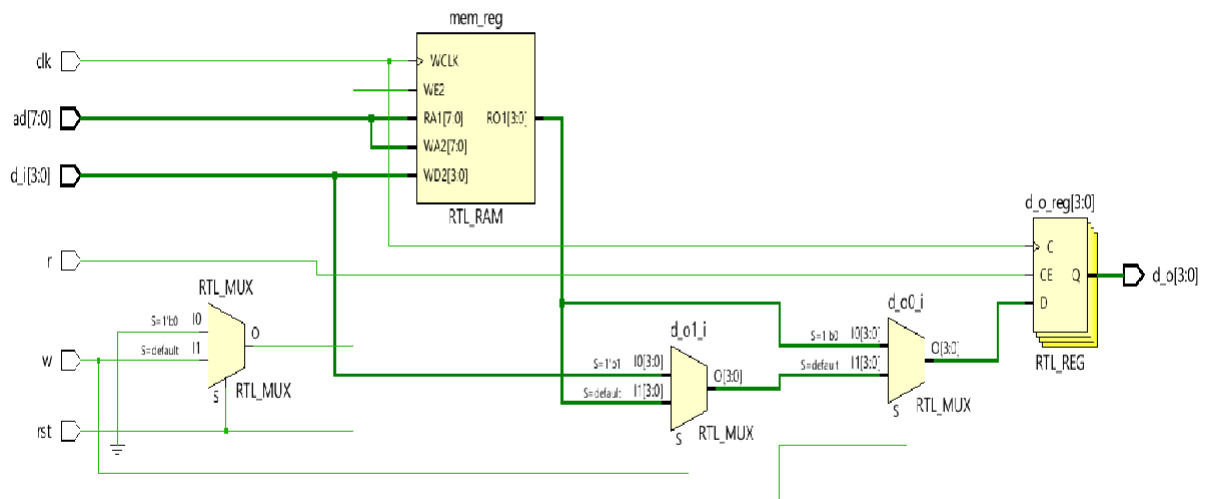
```
module ram_tb();  
reg clk,rst,w,r;  
reg [3:0] d_i;  
reg [7:0] ad;  
wire [3:0] d_o;  
ram m1(clk,rst,w,r,d_i,ad,d_o);  
  
initial  
begin  
clk=0;  
rst=0;  
w=0;  
r=0;  
ad=0;  
d_i=0;  
#50;  
rst=1;  
w=1;  
r=0;  
d_i=4'b0001;  
ad=4'b0001;  
#10;  
d_i=4'b0010;  
ad=4'b0010;  
#10;  
d_i=4'b0110;  
ad=4'b0110;
```

```
#10;
r=1;
w=0;
ad=4'b0001;
#10;
ad=4'b0010;
#10;
ad=4'b0110;
#10;
ad=4'b0001;
#10;
ad=4'b0010;
#10;
ad=4'b0110;
#10;
end
always #5 clk=~clk;
initial
begin
#200 $finish;
end
endmodule
```

## WAVEFORM



## RTL SCHEMATIC FOR CODE



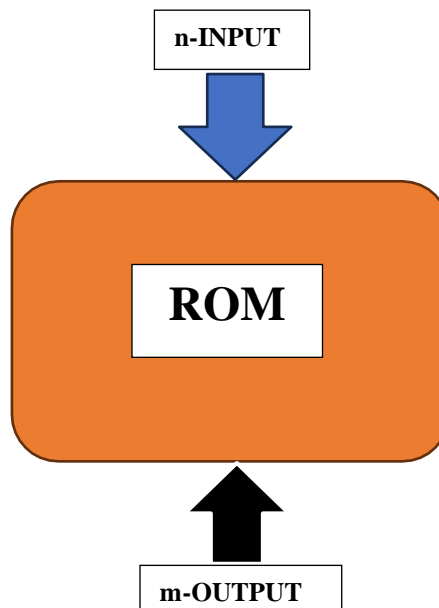


## **PROJECT-5**

### **ROM**

ROM stands for Read only memory which is used to store the permanently or temporarily and which can be accessed whenever required but not erasable. ROM can be used in many applications.

#### **BLOCK DIAGRAM**



#### **APPLICATIONS**

- Firmware
- Embedded systems
- Consumer products
- Smartphones and tablets
- Automotive electronics

## Verilog code

```
module rom(input [7:0] addr,


---


```

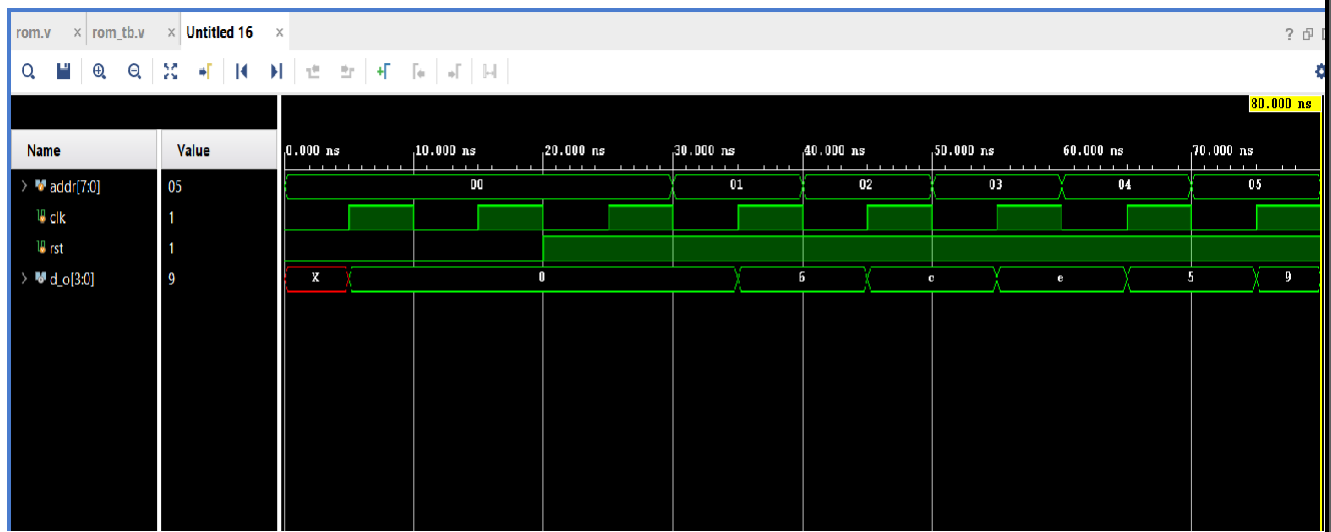
## Testbench

```
`timescale 1ns / 1ps
module rom_tb();
reg [7:0]addr;
reg clk,rst;
wire [3:0]d_o;
rom r1(addr,rst,clk,d_o);
initial
begin
clk=0;
addr=0;
rst=0;
#20;

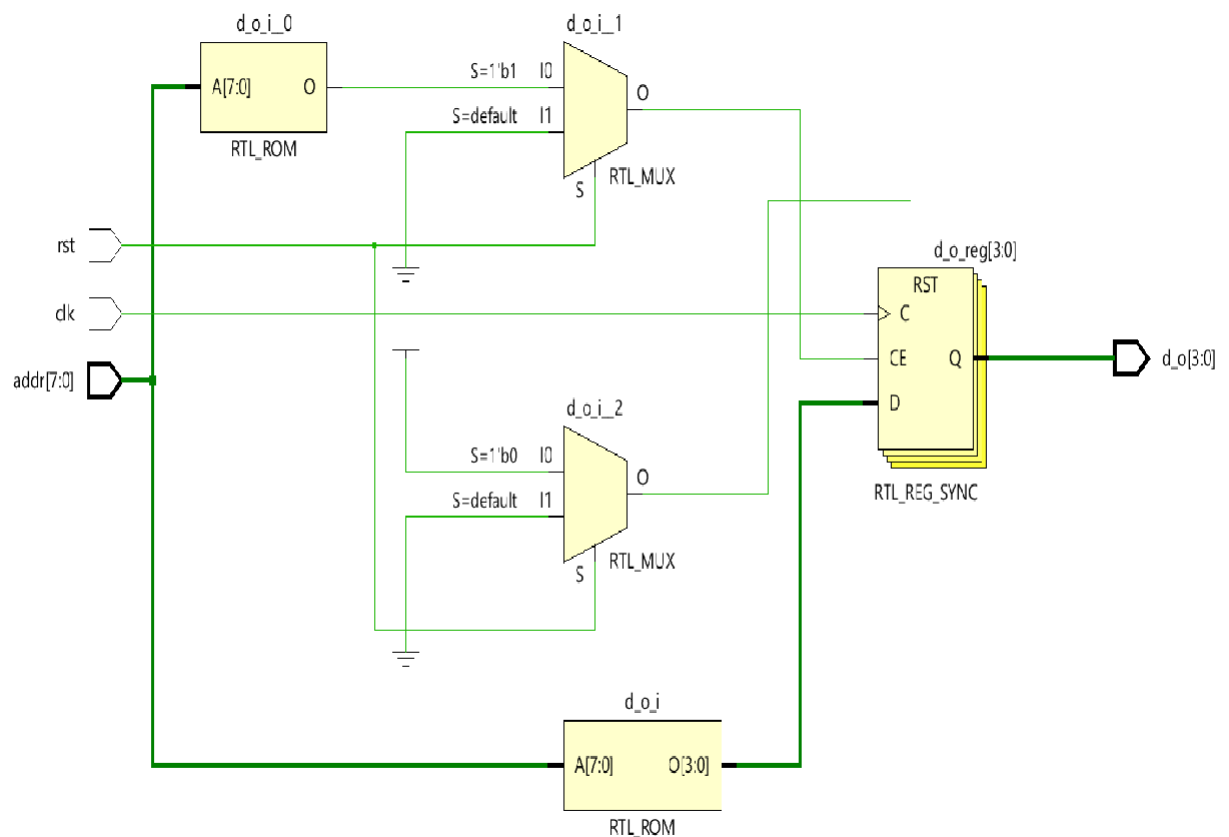
rst=1;#10;
addr= 8'b00000001;#10;
addr= 8'b00000010;#10;
addr=8'b00000011;#10;
addr=8'b00000100;#10;
addr=8'b00000101;#10;
$finish;
end
always #5 clk=~clk;
endmodule
```

---

## WAVEFORM



## RTL SCHEMATIC FOR CODE



## **PROJECT-6**

### **BCD 7-SEGMENT DISPLAY**

A BCD (Binary-Coded Decimal) 7-segment display is a type of digital display commonly used to represent decimal numbers in visual form. It combines the concept of BCD, a binary encoding of decimal digits, with a 7-segment display, which is a visual representation of digits using seven individually controlled segments.

#### **VERILOG CODE**

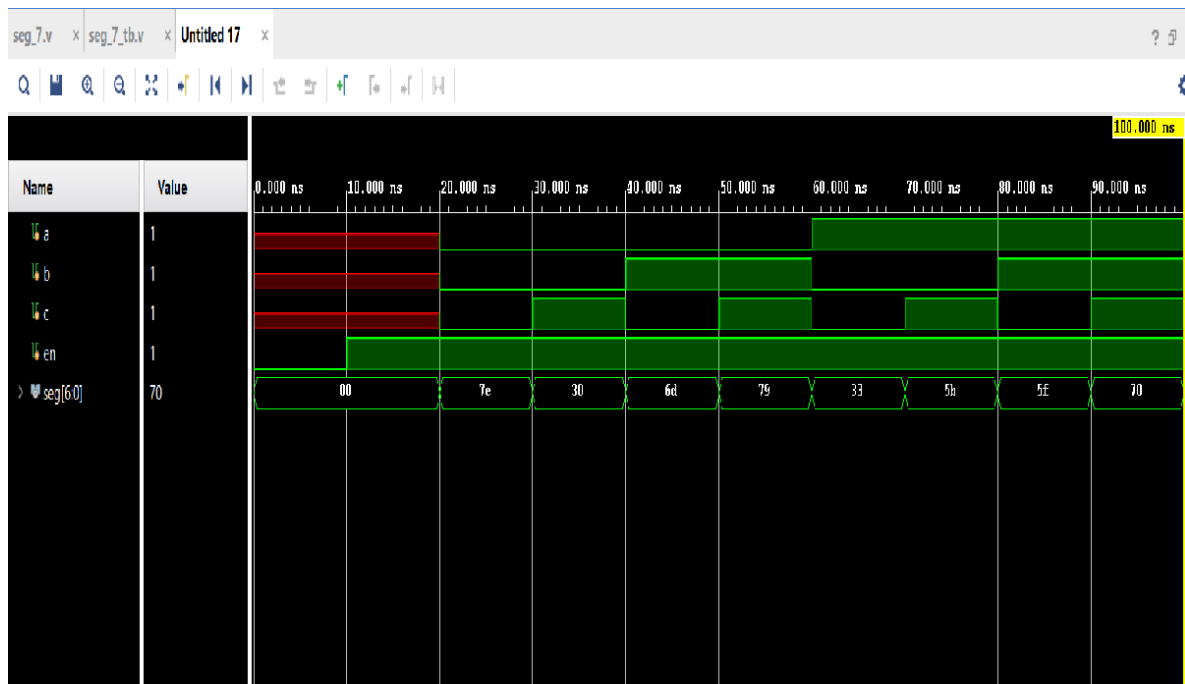
```
module seg_7(input a,b,c,en,output [6:0] seg);  
  reg [6:0]seg;  
  always@(*)  
  begin  
    if(en==0)  
      seg=0;  
    else  
      begin  
        case({a,b,c})  
          3'b000:seg=7'b1111110;  
          3'b001:seg=7'b0110000;  
          3'b010:seg=7'b1101101;  
          3'b011:seg=7'b1111001;  
          3'b100:seg=7'b0110011;  
          3'b101:seg=7'b1011011;  
          3'b110:seg=7'b1011111;  
          3'b111:seg=7'b1110000;  
        endcase  
      end  
    end  
  endmodule
```

---

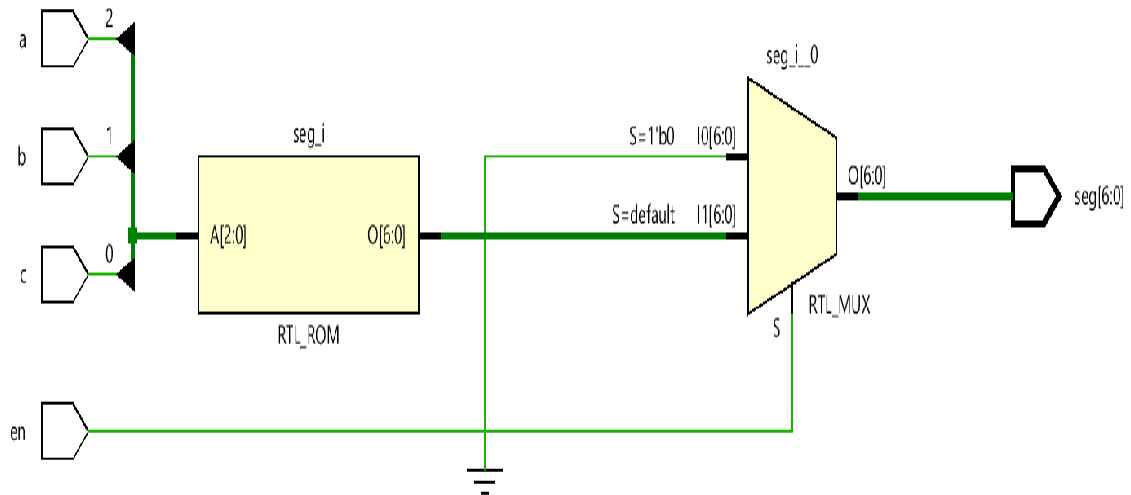
## TESTBENCH

```
timescale 1ns / 1ps
module seg_7_tb();
reg a,b,c,en;
wire [6:0]seg;
seg_7 s1(a,b,c,en,seg);
initial
begin
en=0;
#10;
en=1;
#10;
a=0;b=0;c=0;#10;
a=0;b=0;c=1;#10;
a=0;b=1;c=0;#10;
a=0;b=1;c=1;#10;
a=1;b=0;c=0;#10;
a=1;b=0;c=1;#10;
a=1;b=1;c=0;#10;
a=1;b=1;c=1;#10;
$finish;
end
endmodule
```

## WAVEFORM



## RTL SCHEMATIC FOR CODE



# MAJOR PROJECT

## UART

UART stands for Universal Asynchronous Receiver-Transmitter. It is a communication protocol used for serial communication between two devices. UART is commonly used in microcontrollers, embedded systems, and various communication interfaces. It's a simple and widely used method for transmitting and receiving data between devices that will not be having common clock but operates on particular delay.

Here are the key features and concepts of UART:

1. **Asynchronous Communication:** It is not synchronous communication, where devices will use a common clock signal, but UART is asynchronous. This means that there is no common clock signal between the transmitting and receiving devices. Instead, data is transmitted along with particular delay.
2. **Data Frame Structure:** A UART data frame consists of several components:
  - **Start Bit:** A low-level signal that indicates the beginning of a data frame.
  - **Data Bits:** The actual data being transmitted. Common configurations are 7 or 8 bits per data frame.
  - **Stop Bit:** One or more high-level signal that indicate the end of the data frame. The most common configuration is one stop bit.
3. **Baud Rate:** The baud rate (also known as the bit rate) determines the speed at which data is transmitted. It represents the number of bits transmitted per second. Both the transmitting and receiving devices must operate at the same baud rate for successful communication.
4. **Full-Duplex Communication:** UART supports full-duplex communication, meaning data can be transmitted and received simultaneously. This is achieved using separate transmit and receive lines.
5. **Serial Communication Pins:** UART requires two communication lines: one for transmitting data (TX) and one for receiving data (RX). These lines are connected between the transmitting and receiving devices.

UART is widely used for various applications, including communication between microcontrollers, sensors, displays, and other embedded devices. It's a straightforward protocol for transmitting data and commands, although it lacks some advanced features present in more complex communication protocols.

## VERILOG CODE

```
`timescale 1ns / 1ps
module uart (input clk, input rst, input tx_start, input [7:0] tx_data, output reg tx_busy,
  output reg tx_done, output reg tx_ack, output [7:0] rx_data, input rx_ready);
  reg tx_busy;
  reg tx_done;
  reg tx_ack;
  reg [7:0] rx_data;
  reg [3:0] tx_state;
  reg [3:0] rx_state;
  reg [2:0] bit_count;
  reg [9:0] tx_shift_reg;
  reg [9:0] rx_shift_reg;
  reg rx_data_ready;
  parameter IDLE = 4'b0000;
  parameter START = 4'b0001;
  parameter DATA = 4'b0010;
  parameter STOP = 4'b0011;

  always @(posedge clk or posedge rst) begin
    if (rst)
      tx_state <= IDLE;
    else
      case (tx_state)
        IDLE: tx_state <= tx_start ? START : IDLE;
        START: tx_state <= DATA;
        DATA: tx_state <= bit_count == 7 ? STOP : DATA;
        STOP: tx_state <= IDLE;
        default: tx_state <= IDLE;
      endcase
  end

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      bit_count <= 0;
      tx_shift_reg <= 10'b0;
      tx_busy <= 0;
      tx_done <= 0;
      tx_ack <= 0;
    end
  end
end
```



```

case (tx_state)
    IDLE: begin
        tx_busy <= 0;

        tx_done <= 0;
        tx_ack <= 0;
    end
    START: begin
        tx_busy <= 1;
        tx_shift_reg <= {1'b0, tx_data, 1'b1};
        bit_count <= 0;
        tx_ack <= 1;
    end
    DATA: begin
        tx_shift_reg <= {tx_shift_reg[8:0], tx_data[bit_count]};
        bit_count <= bit_count + 1;
        tx_ack <= 1;
    end
    STOP: begin
        tx_busy <= 0;
        tx_done <= 1;
        tx_ack <= 1;
    end
    default: begin
        tx_busy <= 0;
        tx_done <= 0;
        tx_ack <= 0;
    end
endcase
end
end

```

```

always @(posedge clk or posedge rst) begin
    if (rst) begin
        rx_state <= IDLE;
        rx_shift_reg <= 10'b0;
        rx_data_ready <= 0;
    end
else
begin
    case (rx_state)
        IDLE: begin
            rx_state <= rx_ready ? START : IDLE;
            rx_shift_reg <= 10'b0;
            rx_data_ready <= 0;
        end
    endcase
end

```

```

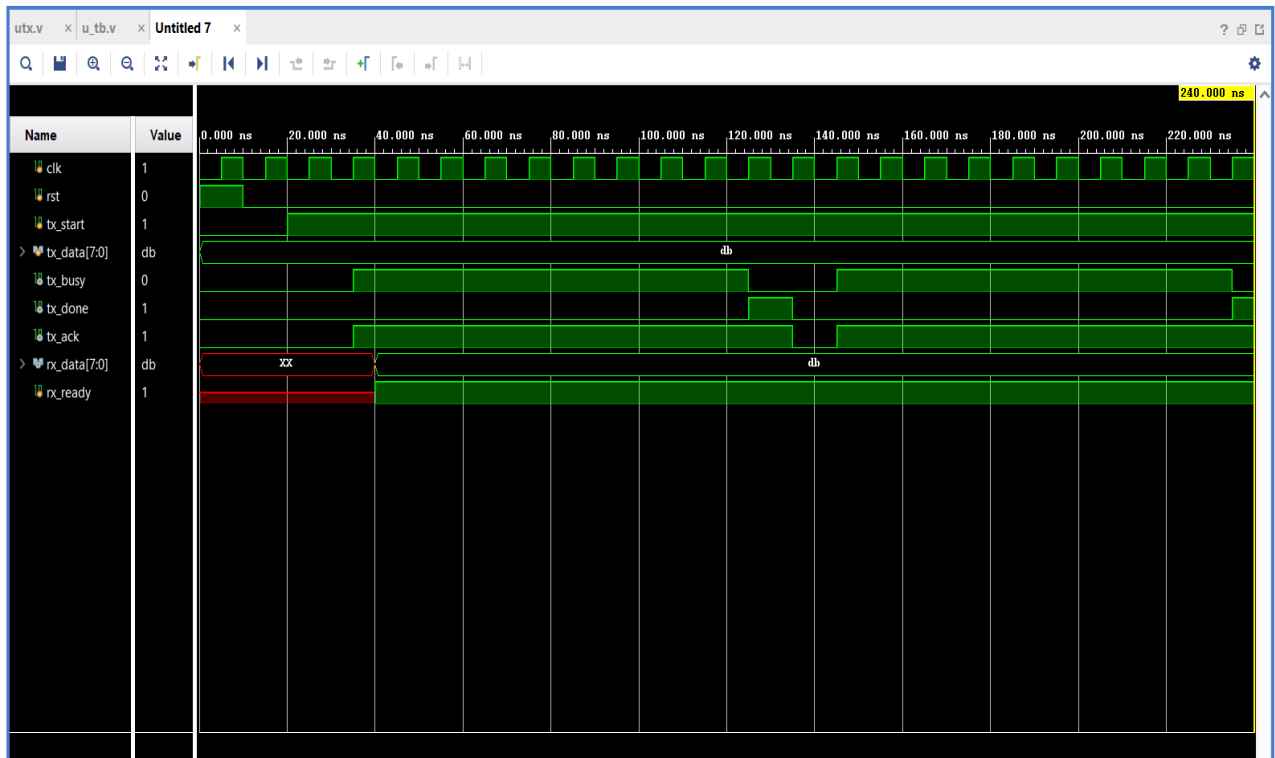
START: begin
    rx_state <= DATA;
    rx_data_ready <= 0;
end
DATA: begin
    rx_shift_reg <= {rx_shift_reg[8:0], rx_ready};
    bit_count <= bit_count + 1;
    rx_data_ready <= 1;
    rx_state <= bit_count == 7 ? STOP : DATA;
end
STOP: begin
    rx_data_ready <= 0;
    rx_state <= IDLE;
end
default: begin
    rx_state <= IDLE;
    rx_shift_reg <= 10'b0;
    rx_data_ready <= 1;
end
endcase
end
end
assign tx_data=rx_data;
endmodule

```

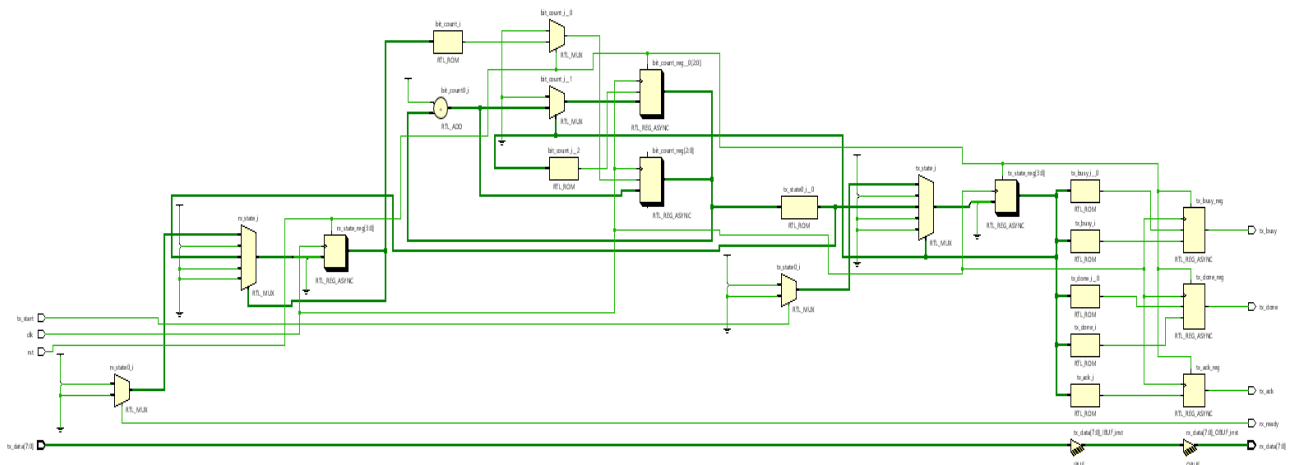
## **TESTBENCH**

```
`timescale 1ns / 1ps
module testbench;
    reg clk;
    reg rst;
    reg tx_start;
    reg [7:0] tx_data;
    wire tx_busy;
    wire tx_done;
    wire tx_ack;
    wire [7:0] rx_data;
    reg rx_ready;
    uart a1 (clk,rst,tx_start,,tx_data,tx_busy,tx_done,tx_ack,rx_data,rx_ready);
    initial begin
        clk = 0;
        rst = 1;
        tx_start = 0;
        tx_data = 8'b11011011;
        #10 rst = 0;
        #10 tx_start = 1;
        #20 rx_ready=1;
        #200 $finish;
    end
    always #5 clk=~clk;
endmodule
```

# WAVEFORM



# RTL SCHEMATIC



## **MY EXPERIENCE OF THE COURSE:**

It is one of the amazing journey, I have gone through, these four months helped me a lot to gain knowledge in VLSI FRONT END. I have learnt a lot from our trainer, Mr. Naga Sitaram M who is punctual, dedicated person. Sir teaching is understandable, sir taught each concept from the very basic that make us more interested in learning. Sir gave many suggestions to improve over understanding and I assure, I will follow them.

During these four months I somehow ace my life-skills by listening at great people like Radha Kumari mam. LST sessions which is one of the skill enhancement program that helped us to explore the ways to overcome various life challenges. It enriched the potential of rural students to built hope on their goals.

## **UNIQUENESS OF THE COURSE:**

Generally outside administrations only concentrate on skills which is related to academics but when it comes to SURE Trust, it concentrates on various factors like student discipline, punctuality and all valuable aspects

- SURE Trust brings every student performance in track.
- SURE Trust actually focuses mainly on quality of education rather than quantity.
- It engages many activities like plantation drive and life skill development programs to make students responsible for better society.
- With the slogan **EACH ONE PLANT TWO**, it let the students to perform actively on social activities.
- The organization monitors each student and their participation during the course

## **CONCLUDING REMARKS**

I am so happy that I have come across this greatest platform and I'm thankful to our trainer and the organizers and my friend who introduced me this wonderful platform. SURE Trust helps many student and also helped me to enhance my knowledge and skills which are needed for myself to upgrade. Organizers of SURE Trust are really inspiring. Here everyone has a discipline and time punctuality. Also, the LST sessions- which are conducted on alternate Sundays make students improved in soft skills and get motivated.

Thank you so much SURE Trust for this wonderful initiative for students like me.

