VLSI PROJECT

BY :

SHIVANG SHARMA

ABSTRACT :

Basic circuits written in Verilog HDL, simulated and implemented on the FPGA.

OBJECTIVE :

To implement and observe the output waveforms using Verilog and Xilinx vivado

of the following circuits:

- Half adder
- Full adder
- Half subtracter
- Full subtracter
- Encoder
- Decoder
- Multiplexer
- Demultiplexer

INTRODUCTION :

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

We will use Verilog to implement the circuits on FPGA board using Xilinx vivado. Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks.

Here we do not have a FPGA board so we will implement the circuits in Xilinx vivado itself.

METHODOLOGY :

First we need to make truth table of the circuit, then using this truth table derive the Boolean expression of the output using simplification or K-Map. In Verilog, we have to benches namely design bench and test bench.

DESIGN BENCH : Give the name of the module and specify all the input and output port.

Then, with the help of the Boolean expression derived above, design circuit by any of the modelling techniques namely Gate level, Dataflow and behavioral modelling.

TEST BENCH : Mention all the input and output of the circuit as register , wire ,etc.

Set initial values of input under INITIAL block. Now to verify the circuit, check the output with different values of input which can be done under ALWAYS block.

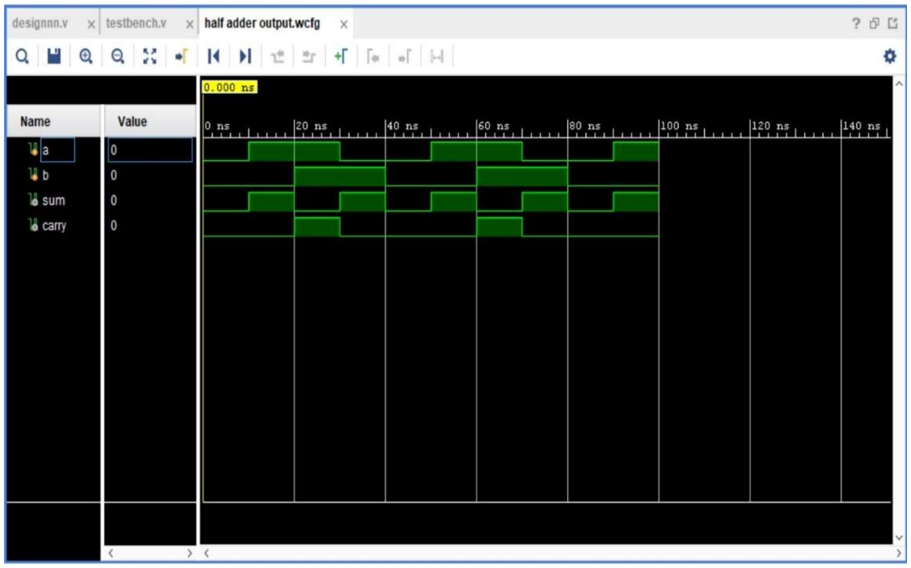
Finally run stimulation and observe the waveform to verify the working of the circuit.

CODE :

➢ HALF ADDER

TEST BENCH	DESIGN BENCH

module testbench();
reg a;
reg b;
wire sum;
wire carry;
half_adder uut(.a(a) ,   .b(b),   .sum(sum), .carry(carry));
initial
    begin
    a=0;
    b=0;
    end
always
    begin
    #10 a=~a;
    #10 b=~b;
    end
always
    #100 $finish;
endmodule
	module half_adder(
	input a,
	input b,
	output sum,
	output carry
	    );
	 xor(sum,a,b);
	and(carry,a,b);
	endmodule



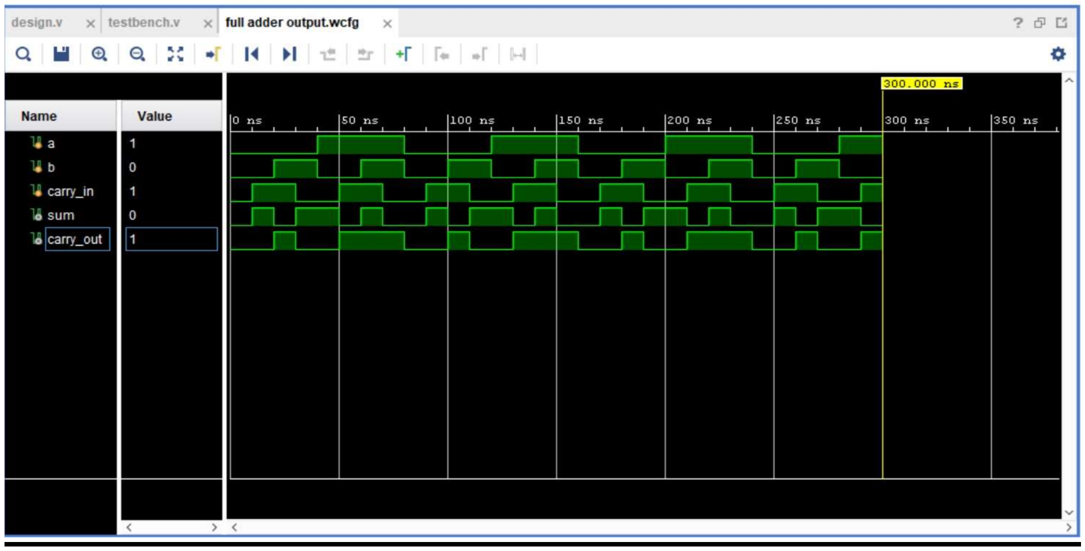
➢ FULL ADDER

TEST BENCH

```
module testbench();
reg a;
reg b;
reg carry_in;
wire sum;
wire carry_out;
full_adder
uut(.a(a),.b(b),.carry_in(carry_in),.sum(sum),
.carry_out(carry_out));
initial
    begin
    a=0;
    b=0;
    carry_in=0;
    end
always
    begin
    #10 carry_in=~carry_in;
    #10 b=~b;
    end
always
    #40 a=~a;
always
    #300 $finish;
endmodule
```

DESIGN BENCH

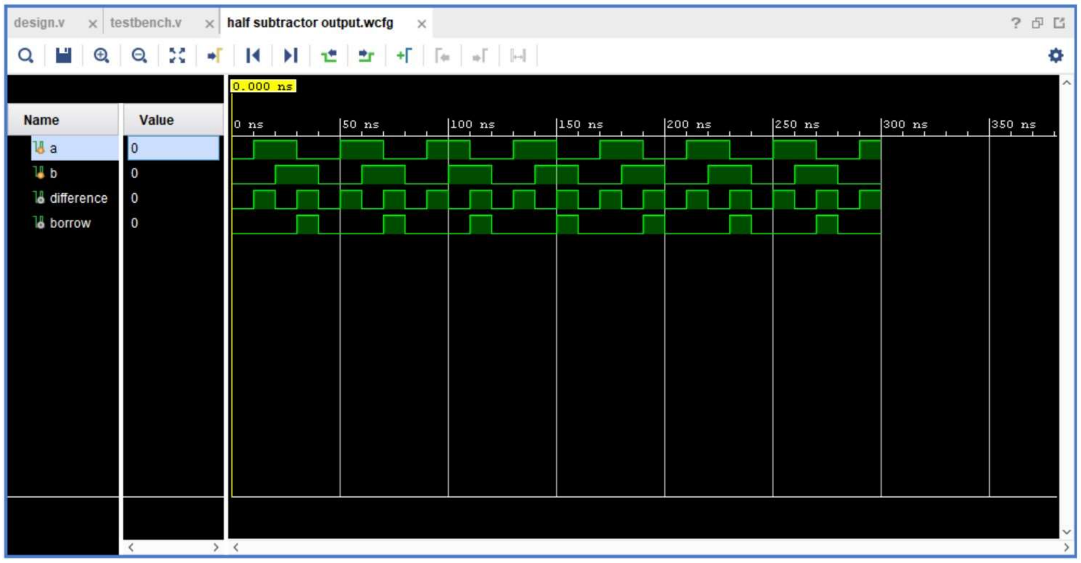
```
module full_adder(
input a,
input b,
input carry_in,
output sum,
output carry_out
    );
assign sum = a^b^carry_in;
assign carry_out = (a&b)+((carry_in)&(a^b));
endmodule
```



➢ HALF SUBTRACTER

TEST BENCH	DESIGN BENCH

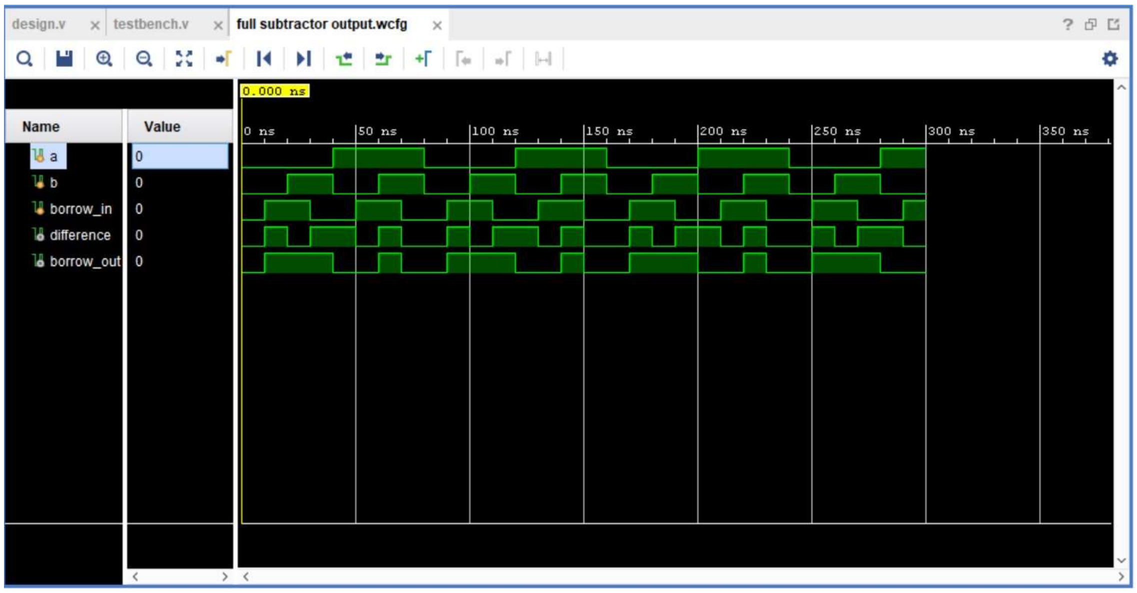
module testbench();
reg a;
reg b;
wire difference;
wire borrow;
half_subtractor
uut(.a(a),.b(b),.difference(difference),
.borrow(borrow));
initial
    begin
    a=0;
    b=0;
    end
always
    begin
    #10 a=~a;
    #10 b=~b;
    end
always
    #300 $finish;
endmodule
	module half_subtractor(
	input a,
	input b,
	output difference,
	output borrow
	    );
	assign difference = a^b;
	assign borrow =(~a)&b;
	endmodule



➢ **FULL SUBTRACTER**

TEST BENCH	DESIGN BENCH

module testbench();
reg a;
reg b;
reg borrow_in;
wire difference;
wire borrow_out;
full_subtractor
uut(.a(a),.b(b),.borrow_in(borrow_in),
.difference(difference),.borrow_out(borrow_out));
initial
    begin
    a=0;
    b=0;
    borrow_in=0;
    end
always
    begin
    #10 borrow_in=~borrow_in;
    #10 b=~b;
    end
always
    #40 a=~a;
always
    #300 $finish;
endmodule
	module full_subtractor(
	input a,
	input b,
	input borrow_in,
	output difference,
	output borrow_out
	    );
	assign difference = a^b^borrow_in;
	assign borrow_out =
	((~a)&b)+((borrow_in)&(~(a^b)));
	endmodule



Name	Value
a	0
b	0
borrow_in	0
difference	0
borrow_out	0

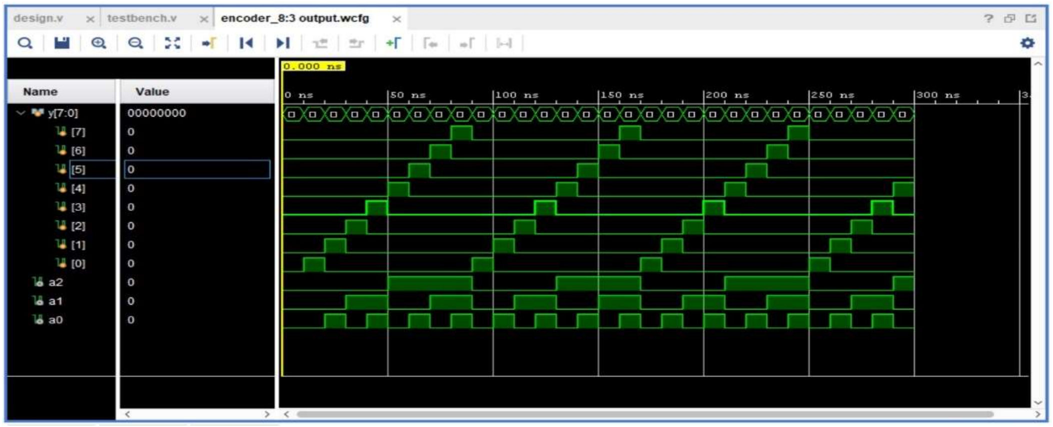
➢ **ENCODER**

TEST BENCH

```
module testbench();
reg [7:0]y;
wire a2;
wire a1;
wire a0;
encoder_8to3
uut(.y(y),.a2(a2),.a1(a1),.a0(a0));
initial
    begin
    y[0]=0;
    y[1]=0;
    y[2]=0;
    y[3]=0;
    y[4]=0;
    y[5]=0;
    y[6]=0;
    y[7]=0;
    end
always
    begin
    #10 y=8'b00000001;
    #10 y=8'b00000010;
    #10 y=8'b00000100;
    #10 y=8'b00001000;
    #10 y=8'b00010000;
    #10 y=8'b00100000;
    #10 y=8'b01000000;
    #10 y=8'b10000000;
    end
always
    #300 $finish;
endmodule
```

DESIGN BENCH

```
module encoder_8to3(
input [7:0]y,
output a2,
output a1,
output a0
    );
assign a2=y[7]+y[6]+y[5]+y[4];
assign a1=y[7]+y[6]+y[3]+y[2];
assign a0=y[7]+y[5]+y[3]+y[1];
endmodule
```



```
Name        Value
y[7:0]      00000000
  [7]       0
  [6]       0
  [5]       0
  [4]       0
  [3]       0
  [2]       0
  [1]       0
  [0]       0
a2          0
a1          0
a0          0
```

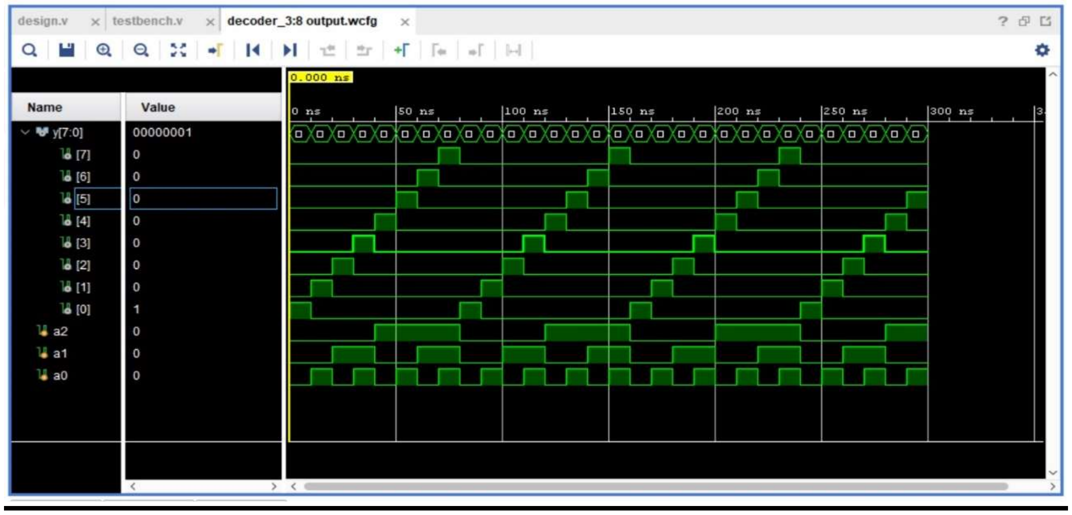
➢ DECODER

TEST BENCH

```
module testbench();
reg a0;
reg a1;
reg a2;
wire [7:0]y;
decoder_3to8
uut(.y(y),.a2(a2),.a1(a1),.a0(a0));
initial
    begin
    a0=0;
    a1=0;
    a2=0;
    end
always
    begin
   #10 a0=~a0;
   #10 a1=~a1;
       a0=~a0;
   #10 a0=~a0;
   #10 a2=~a2;
       a1=~a1;
       a0=~a0;
    end
always
    #300 $finish;
endmodule
```

DESIGN BENCH

```
module decoder_3to8(
input a0,
input a1,
input a2,
output [7:0]y
    );
assign y[7]=a0&a1&a2;
assign y[6]=(~a0)&a1&a2;
assign y[5]=a0&(~a1)&a2;
assign y[4]=(~a0)&(~a1)&a2;
assign y[3]=a0&a1&(~a2);
assign y[2]=(~a0)&a1&(~a2);
assign y[1]=a0&(~a1)&(~a2);
assign y[0]=(~a0)&(~a1)&(~a2);
endmodule
```



```
Name        Value
y[7:0]      00000001
  [7]       0
  [6]       0
  [5]       0
  [4]       0
  [3]       0
  [2]       0
  [1]       0
  [0]       1
a2          0
a1          0
a0          0
```

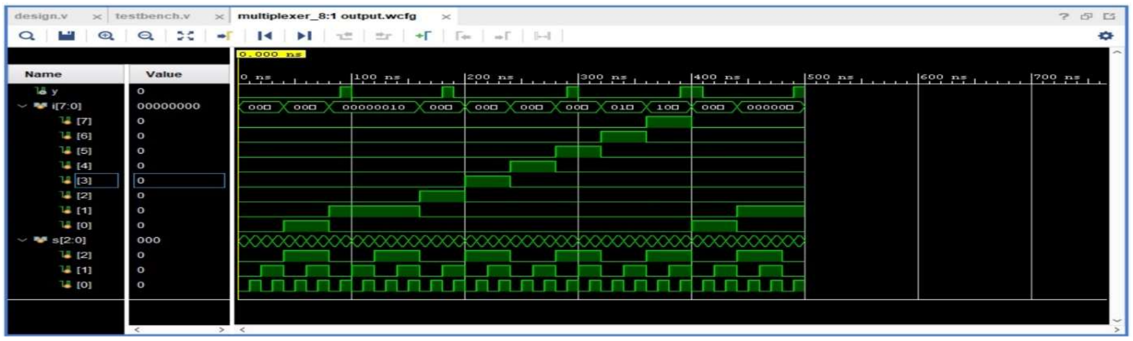
➢ **MULTIPLEXER**

TEST BENCH

```
module testbench();
reg [7:0]i;
reg [2:0]s;
wire y;
multiplexer_8to1 uut(.y(y),.s(s),.i(i));
initial
    begin
    s=3'b000;
    i=8'b00000000;
    end
always
    begin
   #10 s[0]=~s[0];
   #10 s[1]=~s[1];
       s[0]=~s[0];
   #10 s[0]=~s[0];
   #10 s[2]=~s[2];
       s[1]=~s[1];
       s[0]=~s[0];
    end
always
    begin
    #40 i=8'b0000_0001;
    #40 i=8'b0000_0010;
    #40 i=8'b0000_0010;
    #40 i=8'b0000_0100;
    #40 i=8'b0000_1000;
    #40 i=8'b0001_0000;
    #40 i=8'b0010_0000;
    #40 i=8'b0100_0000;
    #40 i=8'b1000_0000;
    end
always
    #500 $finish;
endmodule
```

DESIGN BENCH

```
module multiplexer_8to1(
input [7:0]i,
input [2:0]s,
output y
    );
reg y;
always@(s or i)
    begin
    if(s==3'b000)
    y=i[0];
    else if(s==3'b001)
    y=i[1];
    else if(s==3'b010)
    y=i[2];
    else if(s==3'b011)
    y=i[3];
    else if(s==3'b100)
    y=i[4];
    else if(s==3'b101)
    y=i[5];
    else if(s==3'b110)
    y=i[6];
    else if(s==3'b111)
    y=i[7];
    end
endmodule
```



Name	Value
y	0
i[7:0]	00000000
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0
s[2:0]	000
[2]	0
[1]	0
[0]	0

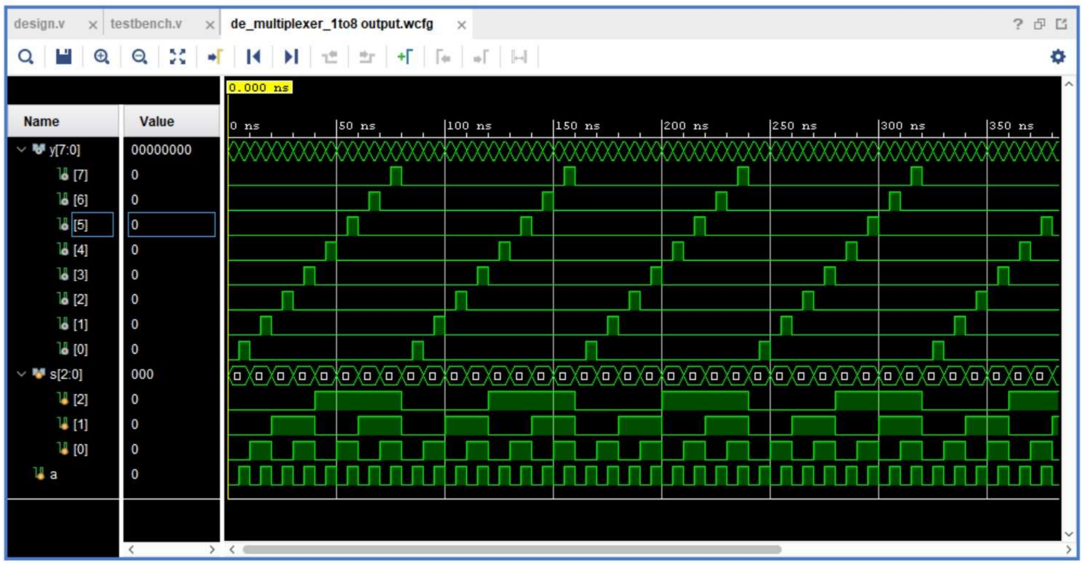
➢ **DEMULTIPLEXER**

TEST BENCH

```
module testbench();
reg a;
reg [2:0]s;
wire [7:0]y;
de_multiplexer_1to8 uut(.y(y),.s(s),.a(a));
initial
    begin
    s=3'b000;
    a=0;
    end
always
    begin
   #10 s[0]=~s[0];
   #10 s[1]=~s[1];
       s[0]=~s[0];
   #10 s[0]=~s[0];
   #10 s[2]=~s[2];
       s[1]=~s[1];
       s[0]=~s[0];
    end
always
    begin
    #5 a=~a;
    end
always
    #500 $finish;
endmodule
```

DESIGN BENCH

```
module de_multiplexer_1to8(
input a,
input [2:0]s,
output [7:0]y
    );
    assign y[0]=((~s[0])&(~s[1])&(~s[2]))&(a);
    assign y[1]=((s[0])&(~s[1])&(~s[2]))&(a);
    assign y[2]=((~s[0])&(s[1])&(~s[2]))&(a);
    assign y[3]=((s[0])&(s[1])&(~s[2]))&(a);
    assign y[4]=((~s[0])&(~s[1])&(s[2]))&(a);
    assign y[5]=((s[0])&(~s[1])&(s[2]))&(a);
    assign y[6]=((~s[0])&(s[1])&(s[2]))&(a);
    assign y[7]=((s[0])&(s[1])&(s[2]))&(a);
endmodule
```



➢ COMPARATOR

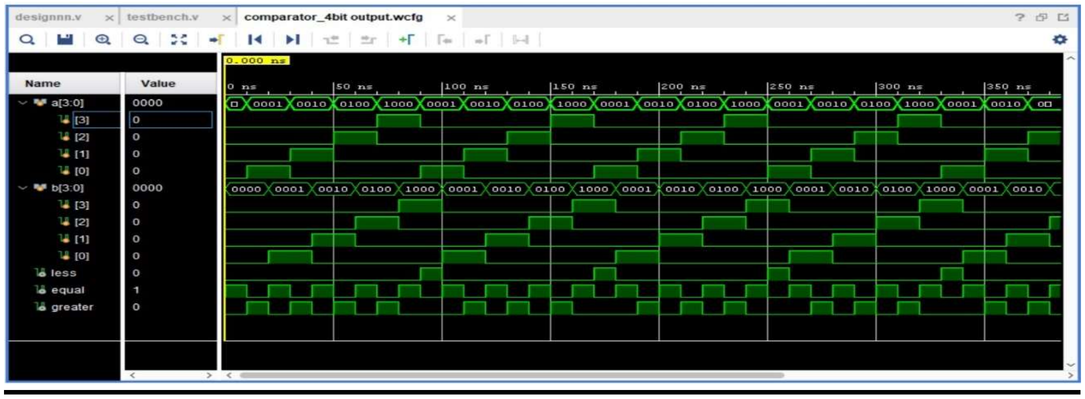
TEST BENCH

```
module testbench();
   reg [3:0] a;
   reg [3:0] b;
   wire less;
   wire equal;
   wire greater;
   comparator_4bit uut (
       .a(a),
       .b(b),
       .less(less),
       .equal(equal),
       .greater(greater)
   );

 initial
    begin
      a=4'b0000;
      b=4'b0000;
    end
 always
    begin
    #10 a=4'b0001;
    #10 b=4'b0001;
    #10 a=4'b0010;
    #10 b=4'b0010;
    #10 a=4'b0100;
    #10 b=4'b0100;
    #10 a=4'b1000;
    #10 b=4'b1000;
    end
 always
   #500 $finish;
endmodule
```

DESIGN BENCH

```
module comparator_4bit(
    a,b,less,equal,greater
    );
    input [3:0] a;
    input [3:0] b;
    output less;
     output equal;
     output greater;
     reg less;
     reg equal;
     reg greater;
    always @(a or b)
     begin
        if(a > b)
          begin
            less = 0;
            equal = 0;
            greater = 1;
          end
        else if(a == b)
          begin
            less = 0;
            equal = 1;
            greater = 0;
          end
        else
          begin
            less = 1;
            equal = 0;
            greater =0;
          end
    end
endmodule
```

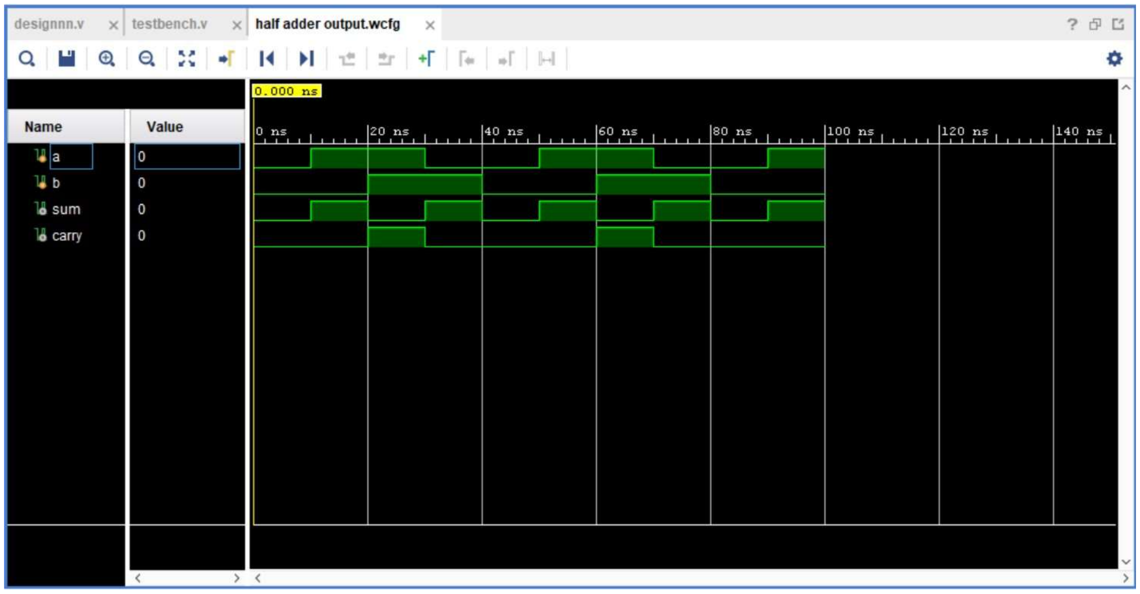


Name	Value
a[3:0]	0000
[3]	0
[2]	0
[1]	0
[0]	0
b[3:0]	0000
[3]	0
[2]	0
[1]	0
[0]	0
less	0
equal	1
greater	0

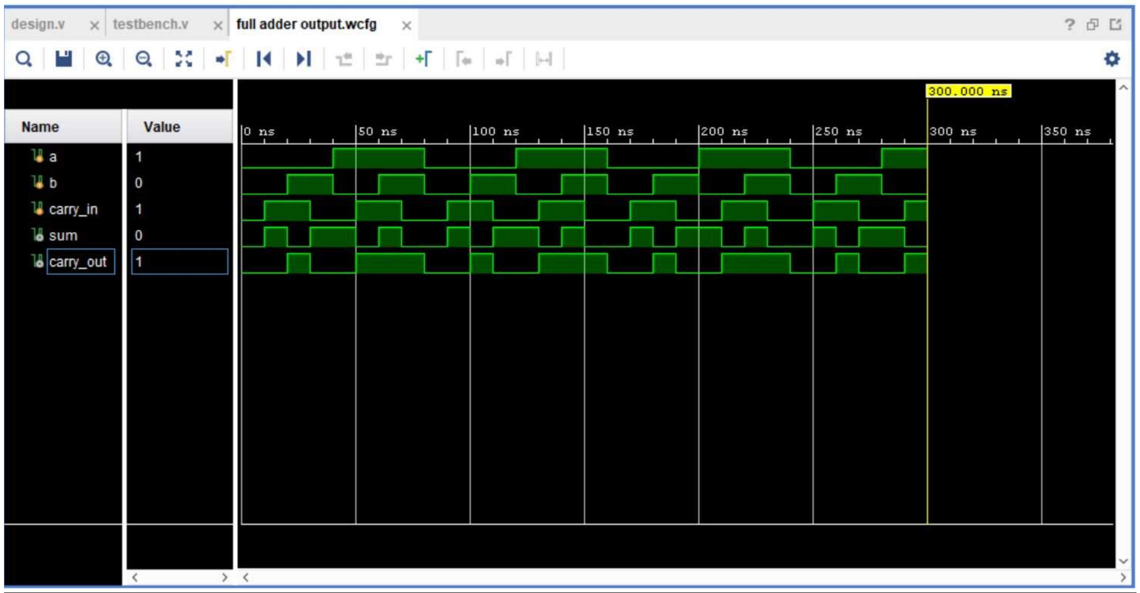
CONCLUSION :

Here are the output waveforms of the circuits we implemented on Xilinx vivado

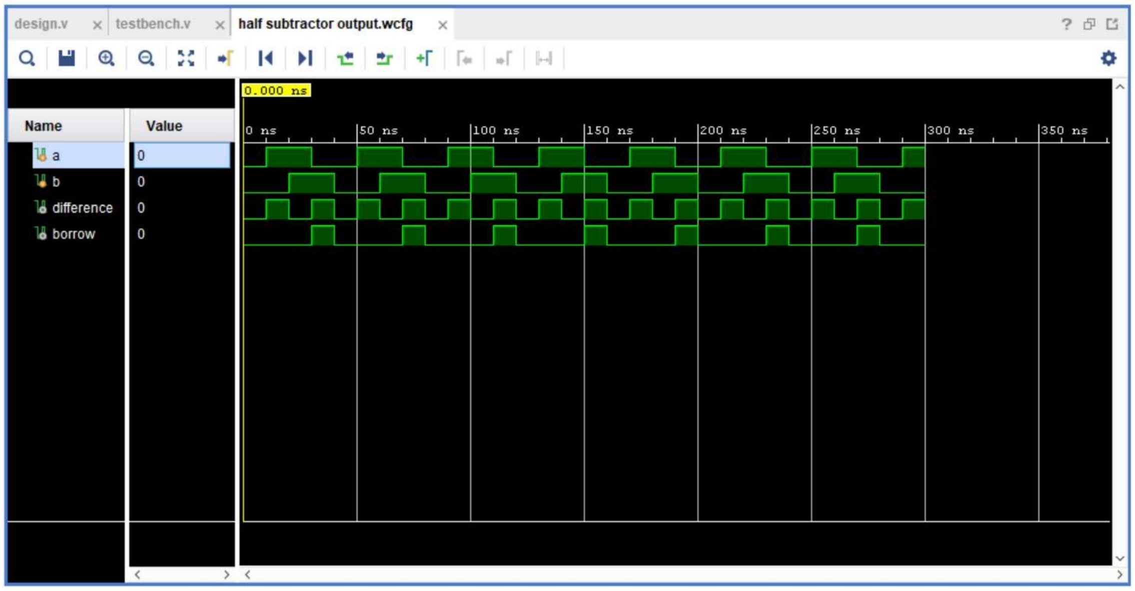
- HALF ADDER



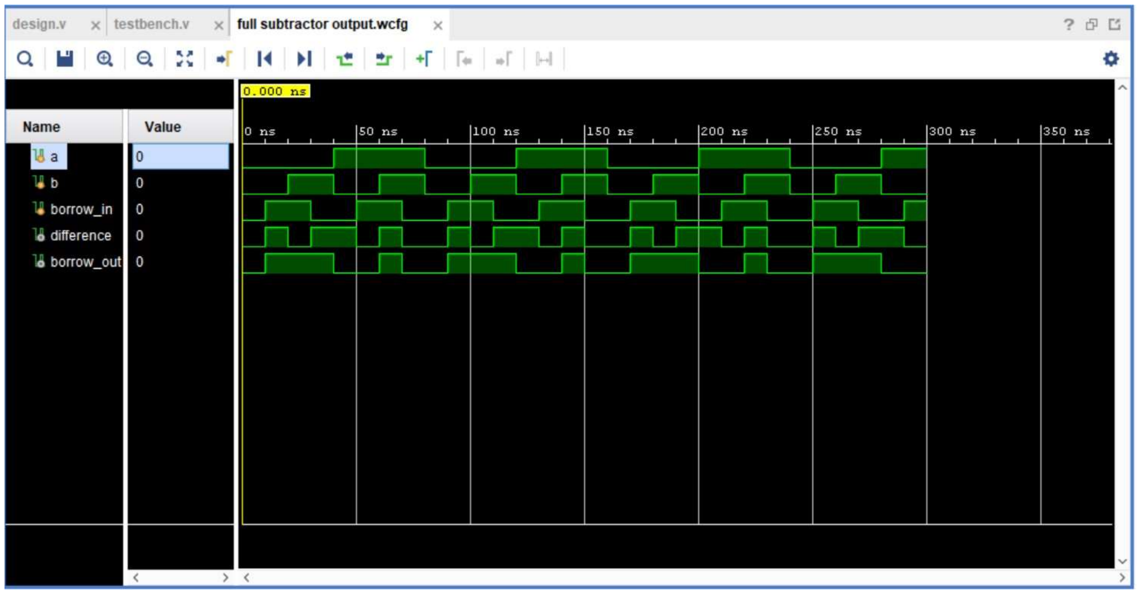
- FULL ADDER



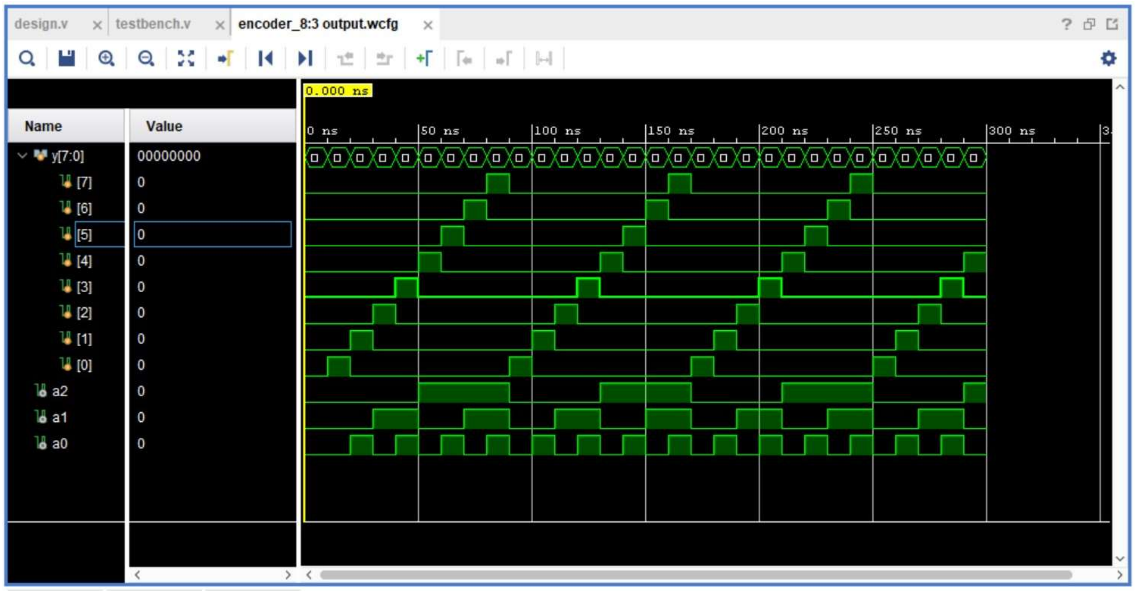
➢ HALF SUBTRACTER



➢ FULL SUBTRACTER

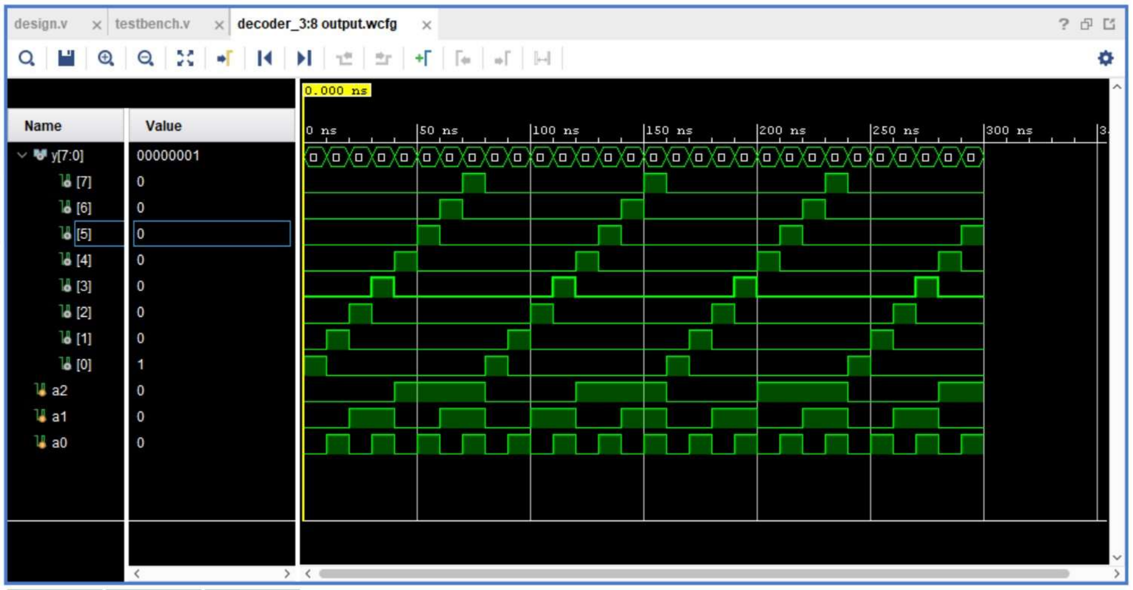


➢ <u>ENCODER</u>



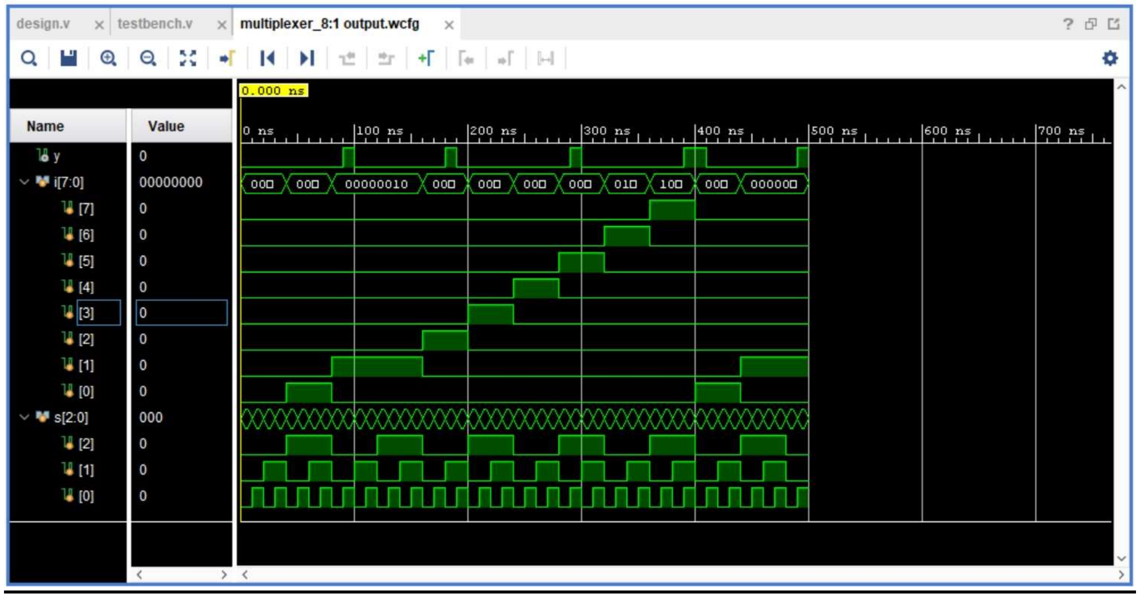
Name	Value
y[7:0]	00000000
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0
a2	0
a1	0
a0	0

➢ <u>DECODER</u>

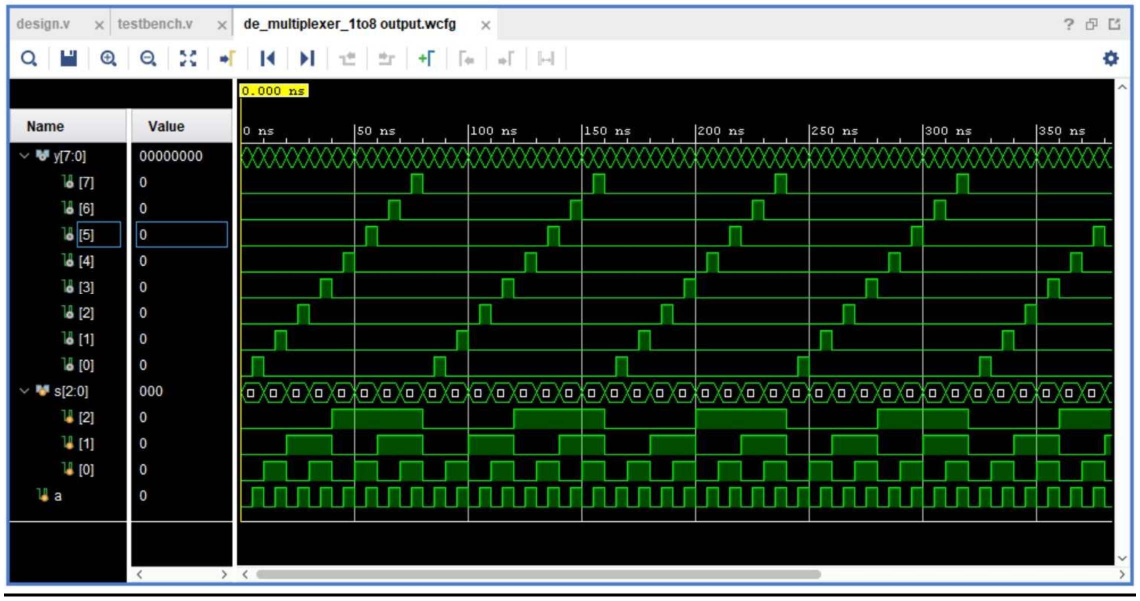


Name	Value
y[7:0]	00000001
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	1
a2	0
a1	0
a0	0

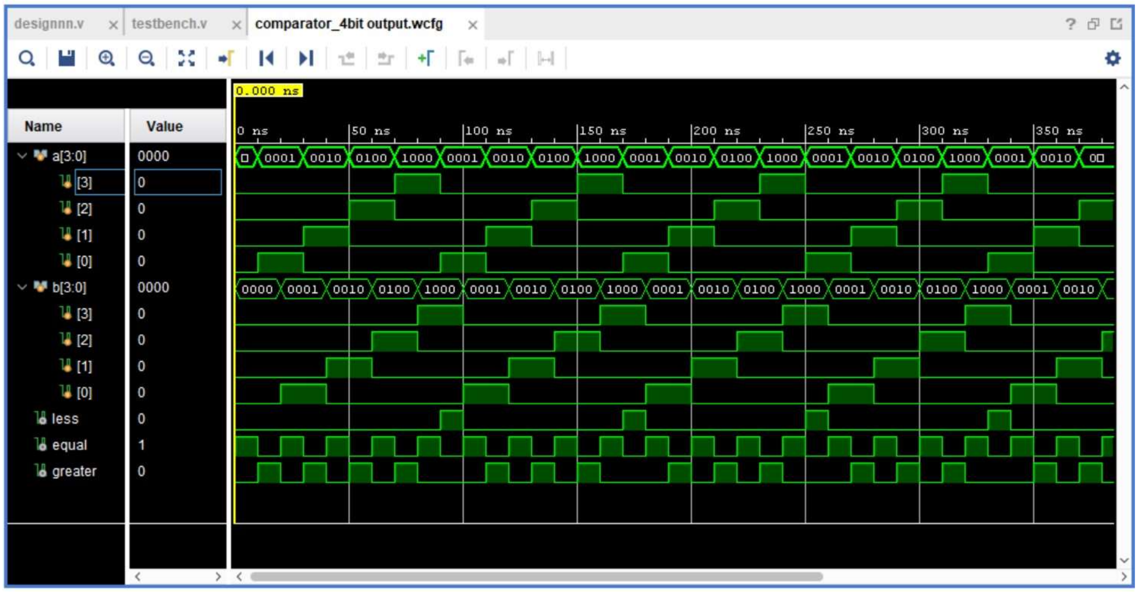
➢ <u>MULTIPLEXER</u>



➢ <u>DEMULTIPLEXER</u>



➢ COMPARATOR


designnn.v
testbench.v
comparator_4bit output.wcfg
0.000 ns
Name
Value
a[3:0]
[3]
[2]
[1]
[0]
b[3:0]
less
equal
greater
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns

THANK

YOU