DIGITAL SYSTEMS DESIGN
ASSIGNMENT - 1

REPORT SUBMISSION FOR ASSIGNMENT-1

NAME: SWETANG KRISHNA

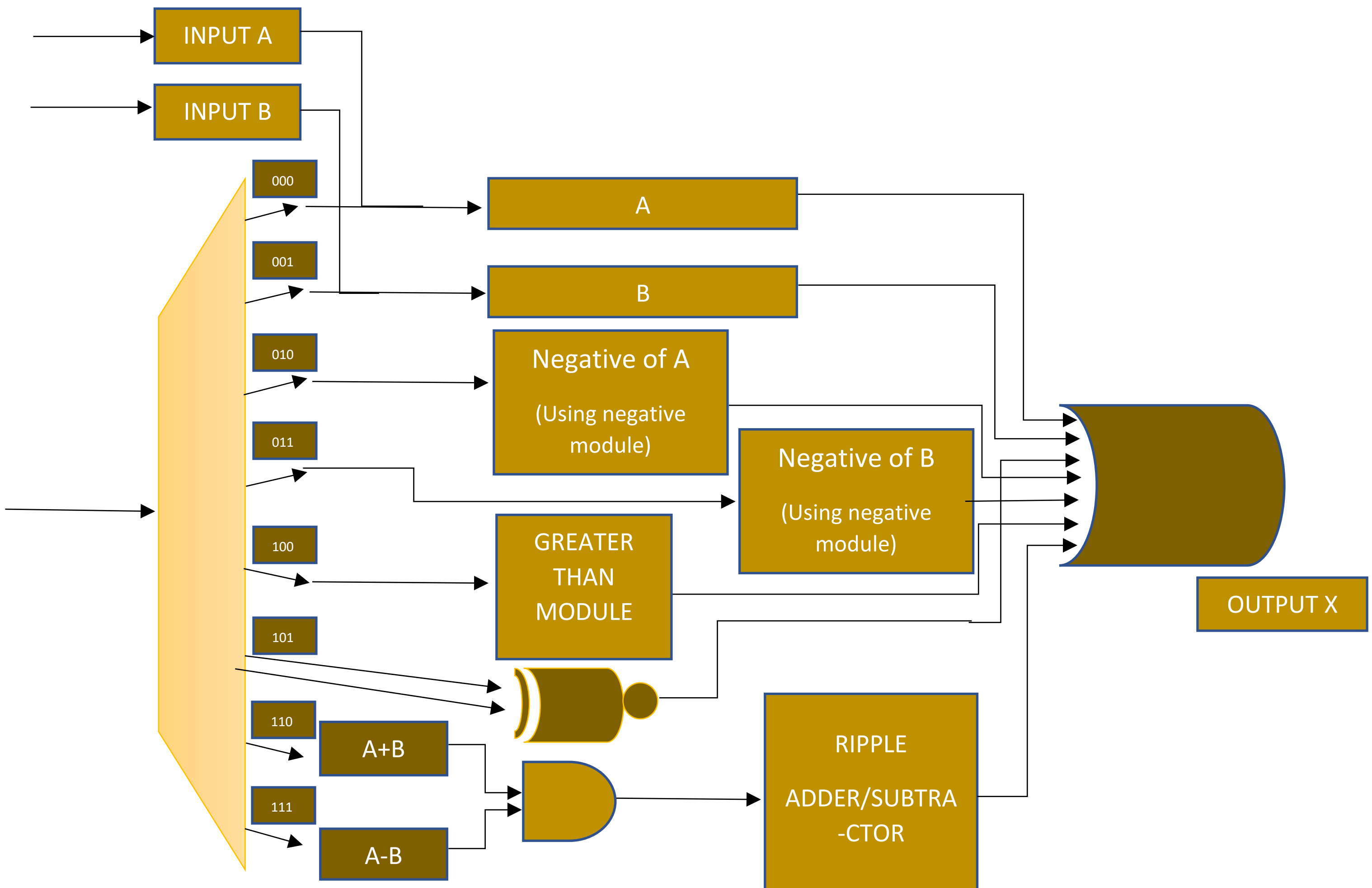
STUDENT ID: 21355087

CONTENT	PAGE
2'S COMPLIMENT FORM FOR 6-BIT BINARY NUMBER	2
MAIN CIRCUIT DESIGN (FIG-1)	3
SUB CIRCUIT DESIGN FOR MODULE NEGATIVE(FIG-3) & GREATER THAN MODULE	4
CIRCUIR DESIGN FOR RIPPLE ADDER MODULE(FIG-5)	5
CODE FOR TOP LEVEL MODULE (main_ALU_KRISHNA_code)	6
CODE FOR TOP LEVEL MODULE (main_ALU_KRISHNA_code)	7
CODE FOR MODULE NEGATIVE	8
CODE FOR MODULE GR1	9
CODE FOR MODULE GR6	10
CODE FOR MODULE XNOR1 AND XNOR6	11
CODE FOR MODULE FULL_ADDER AND SIX_BIT_ADDER	12-13
VIVADO SCREENSHOTS for different functions	14-18
CODE FOR TESTBENCH	19
TESTBENCH SCREENSHOT	20
TESTBENCH RESULTS and file directory screenshot	21
DEMO	22
PINS ASSIGNED IN FPGA BOARD	23-24
XDC FILE CODE	25-28
OUTPUTS SHOWN ON FPGA BOARD	29
OUTPUTS FOR UNEXPECTED INPUTS	30

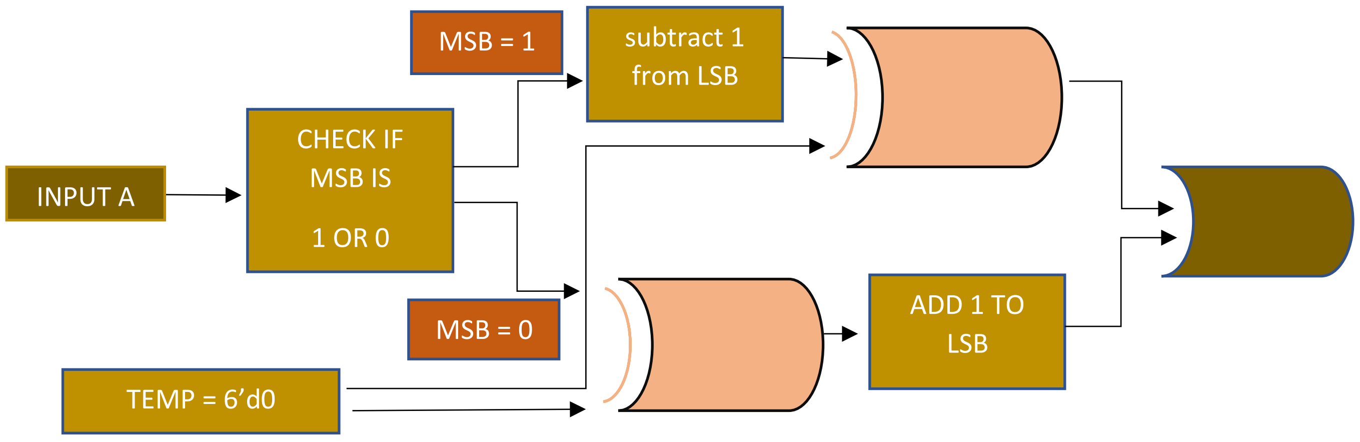
DECIMAL	BINARY	DECIMAL	BINARY
-32	100000	0	000000
-31	100001	1	000001
-30	100010	2	000010
-29	100011	3	000011
-28	100100	4	000100
-27	100101	5	000101
-26	100110	6	000110
-25	100111	7	000111
-24	101000	8	001000
-23	101001	9	001001
-22	101010	10	001010
-21	101011	11	001011
-20	101100	12	001100
-19	101101	13	001101
-18	101110	14	001110
-17	101111	15	001111
-16	110000	16	010000
-15	110001	17	010001
-14	110010	18	010010
-13	110011	19	010011
-12	110100	20	010100
-11	110101	21	010101
-10	110110	22	010110
-9	110111	23	010111
-8	111000	24	011000
-7	111001	25	011001
-6	111010	26	011010
-5	111011	27	011011
-4	111100	28	011100
-3	111101	29	011101
-2	111110	30	011110
-1	111111	31	011111

CIRCUIT DESIGN:

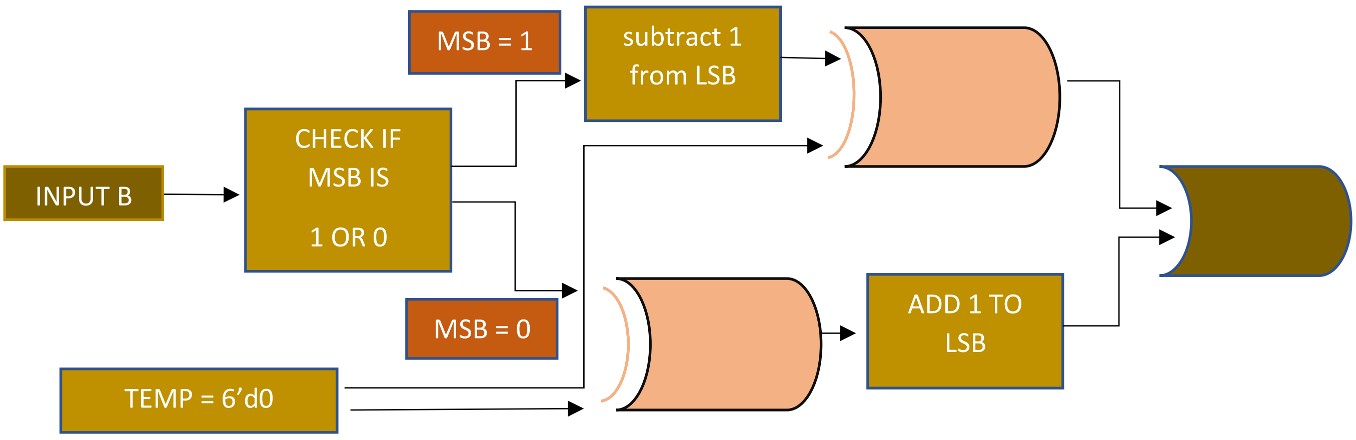
MAIN CIRCUIT-> (fig 1)


INPUT A
INPUT B
000
001
010
011
100
101
110
111
A
B
Negative of A (Using negative module)
Negative of B (Using negative module)
GREATER THAN MODULE
A+B
A-B
RIPPLE ADDER/SUBTRA -CTOR
OUTPUT X

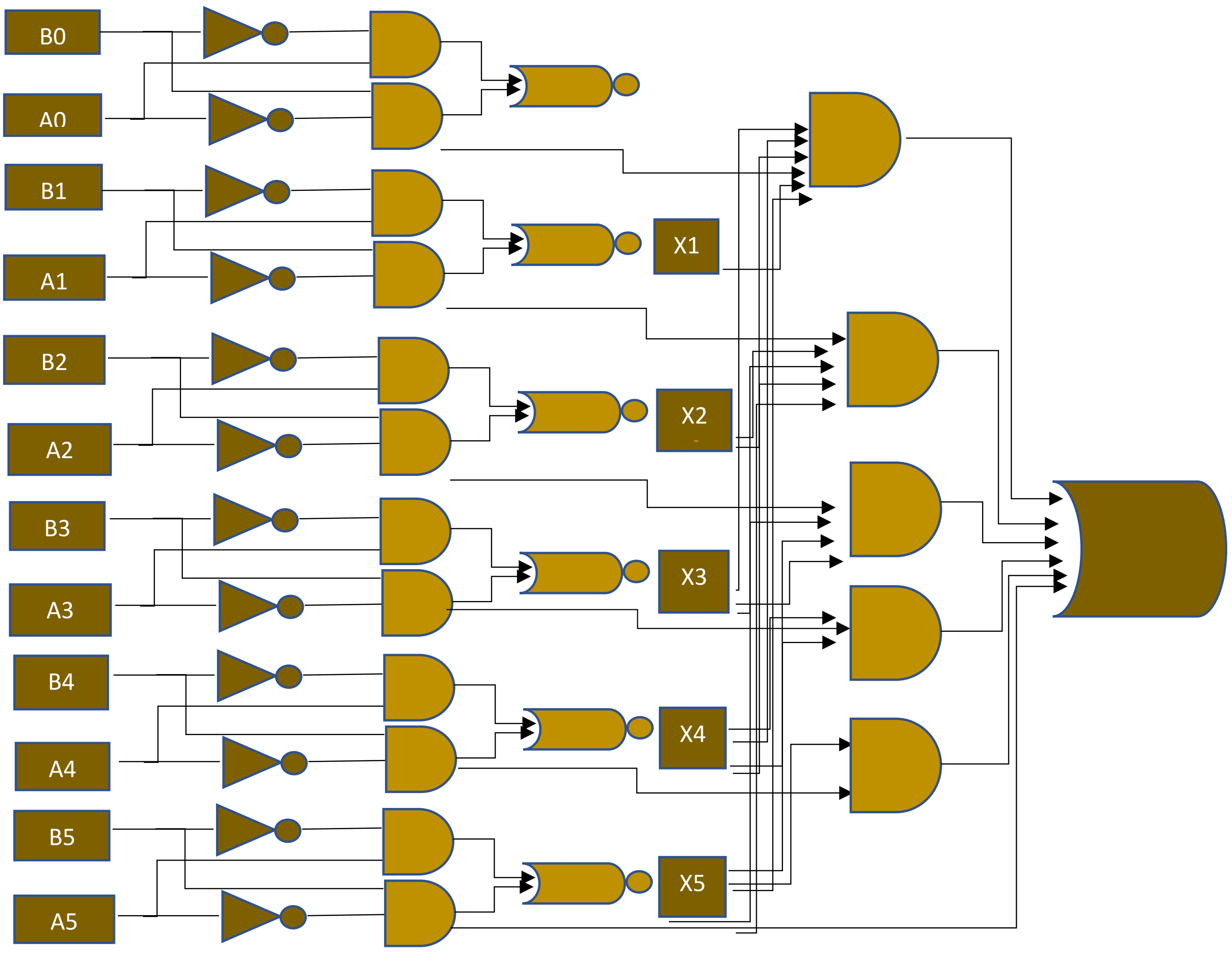
negative of A-> (fig2)



negative of B-> (fig 3)



Greater than module-> (fig4)

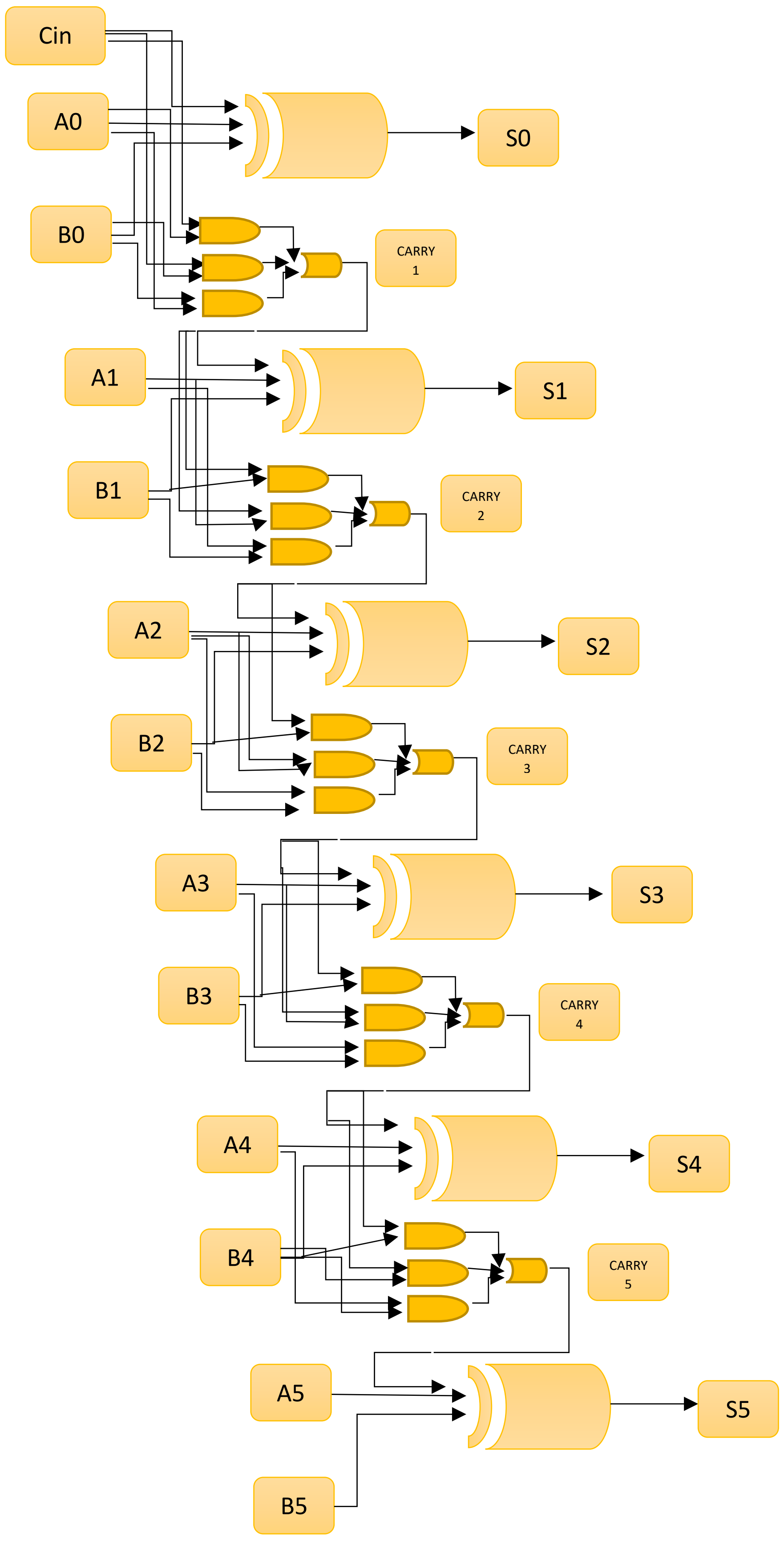


FOR A < B the Boolean expression is:

B[5]A[5] + X5.(B[4].A[4]) + X5.X4.(B[3].A[3]) + X5.X4.X3(B[2].A[2]) + X5.X4.X3.X2.(B[1].A[1]) + X5.X4.X3.X2.X1.(B[0].A[0])

NOTE: Text in red depicts compliment of the number

RIPPLE ADDER MODULE->(fig 5)



And for subtraction we can use:
X = A – B.

<u>CODE (top level – ALU module)</u>

<u>//this module is the main module.</u>

```
Module ALU_KRISHNA(
Input wire[5:0] A,B,                 //inputs ‘A’ and ‘B’ are 6-bit numbers.
Output wire[5:0]X                    //output is a 6-bit number as well.
);
Wire[5:0] NEG_A,NEG_B,               // ‘NEG_A’ & ‘NEG_B’ stores the negative
XNOR_A_B;                            //values of A & B respectively.
                                     //’XOR_A_B’ stores the value of ‘~(A ^ B)’.
Wire agrb;                           //agrb stores the value 0 or 1 depending
                                     //upon whether ‘a<b’ or ‘a>b’.
Wire add_a_b                         //’add_a_b’ stores the value of ‘A+B’.


Negative neg_A(.m(A),                //using module negative we get negative
.nega(NEG_A));                       //of number ‘A’ and storing it in ‘NEG_A’.
Negative neg_B(.m(B),                // using module negative we get negative
.nega(NEG_B));                       //of number ‘B’ and storing it in ‘NEG_B’.
Gr6                                  //using ‘Gr6’ module to check whether
greater(.x(A),.y(B),.gr(agrb));      //‘A<B’and storing the output in ‘agrb’.
XNOR6
constraints_XNOR(.j0(A),.j1(B),      //using ‘XNOR’ module to find ‘~(a ^ b)’
.k(XNOR_A_B));                       //and storing the value in ‘XNOR_A_B’.
```

```
Adder_6_bit                                   //using 'adder_6_bit' module to find 'a+b'
contraints_adder(.x(A),.y(B),                 //and storing the output in 'add_a_b'.
.z(add_a_b));


Always@ (*)                                   //using demux to find what operation
Begin                                         //must be performed.

        Case(fxn)

        3'b000:{X}= {A};                      //if fxn = 000 then output is 'A'.

        3'b001:{X}= {B};                      // if fxn = 001 then output is 'B'.

        3'b010:{X}= {NEG_A};                  // if fxn = 010 then output is '-A'.

        3'b011:{X}= {NEG_B};                  // if fxn = 011 then output is '-B'.

        3'b100:{X}= {agrb};                   // if fxn = 100 then output is 'agrb'.

        3'b101:{X}= {XNOR_A_B};               // if fxn = 101 then output is 'XNOR_A_B'.

        3'b110:{X}= {add_a_b};                // if fxn = 110 then output is 'add_a_b'.

        3'b111:{X}= {A-B};                    // if fxn = 111 then output is 'A-B'.

        Default: {X} = {0};                   // if no input is given the output will be 0.

        Endcase

end

Endmodule
```

Other modules Used->

```
Module negative                     //this module finds the negative of a 6-bit
                                    //number in 2's complement form.
(input wire[5:0] m,                 //'m' is the number whose negative is to
                                    //be found.
Output wire[5:0] nega               //the negative of the number will be
                                    //stored in 'nega'.
);
Wire[5:0] temp = 6'd0;              //'temp' is a 000000 to help us find the 1's
                                    //compliment.

always@ (*)                         //in order to find the negative, we first
nega = ~((m - 1) ^ temp);           //need to subtract 1 from the LSB then
Endmodule                           //find //its XOR with temp and then find
                                    //the negation of the result.
```

```
Module gr1                            //using gr1 module to find ‘X_i’ where,
(                                     //’X_i’ = ~A[0].B[0] + A[0].~B[0] which we
                                      //will further use to find if A<B.
input wire i0,i1,                     //’i0’ and ‘i1’ are the input single bits.
output wire gr                        //’gr’ is the final output which is true for
                                      A<B.
);


Wire p0,p1;
Assign gr = p0 | p1;                  //’gr’ is true for either ‘p0’ or ‘p1’.


                                      //’p0’ and ‘p1’ will be true if ‘i0’ and ‘i1’
Assign p0 = ~i0 & ~i1;                //are equal.
Assign p1 = i0 & i1;


Endmodule
```

```
Module gr6(                                   //module gr6 is used to find if a 6-bit
                                              //number is greater than the other.
Input wire[5:0] x,y,                          //numbers x and y are to be compared.

Output wire final_bit                         //the final out put gr will be a single bit
                                              number.
);

Wire [5:0] gr_out;

Wire[5:0] out;                                //out is a 6-bit number which will be 1 if

                                              //the digits compared are equal.
Gr1
gr_bit5_unit(.i0(x[5]),.i1(y[5]),.gr(ou       //using gr1 module to find equality for
t[5]));                                       //each digit.

Gr1
gr_bit4_unit(.i0(x[4]),.i1(y[4]),.gr(ou       //further following this process for every
t[4]));                                       //digit.

Gr1
gr_bit3_unit(.i0(x[3]),.i1(y[3]),.gr(ou
t[3]));

Gr1
gr_bit2_unit(.i0(x[2]),.i1(y[2]),.gr(ou
t[2]));

Gr1
gr_bit1_unit(.i0(x[1]),.i1(y[1]),.gr(ou
t[1]));                                       //The final output is the sum of products,
                                              //that is described in circuit diagram
Assign gr = x[5]&~y[5] |                      //(fig-4).
out[5]&(y[4]&~x[4]) |
out[5]&out[4]&(y[3]&~x[3]) |
out[5]&out[4]&out[3](y[2]&~x[2]) |
out[5]&out[4]&out[3]&out[2]&(y[1]
&~x[1]) |
out[5]&out[4]&out[3]&out[2]&out[
1]&(y[0]&~x[0]);

endmodule
```

```
Module XNOR1(                                   //module XNOR1 will find the answer to
                                                //the expression X = ~(A ^ B);
Input wire j0,j1,

Output wire k

);

Assign k = ~(j0 ^ j1);

Endmodule;


Module XNOR6(                                   //module XNOR6 will find A XNOR B for
                                                //every digit using module XNOR1.
Input wire[5:0] f,g,

Output wire[5:0] h

);

XNOR1                                           //using XNOR1 module to get ~(A ^ B) for
ans_bit0_unit(.j0(f[0]),.j1(g[0]),..k(h[        //each digit.
0]));

XNOR1
ans_bit1_unit(.j0(f[1]),.j1(g[1]),.k(h[
1]));

XNOR1
ans_bit2_unit(.j0(f[2]),.j1(g[2]),.k(h[
2]));

XNOR1
ans_bit3_unit(.j0(f[3]),.j1(g[3]),.k(h[
3]));

XNOR1
ans_bit4_unit(.j0(f[4]),.j1(g[4]),.k(h[
4]));

XNOR1
ans_bit5_unit(.j0(f[5]),.j1(g[5]),..k(h[
5]));

Endmodule
```

```
module
full_Adder(a,b,cin,s,cout);
```

```
//full adder adds a single bit digit and
//stores the sum in ‘s’ and the carry digit
//in cout.
```

```
   input wire a, b, cin;
```

```
//cin is the initial carry which is set to 0.
```

```
        output wire s, cout;
```

```
//assigning 's' as : cin XOR a XOR b
//assigning 'cout'
```

```
        assign s = cin ^ a ^ b;
        assign cout = (b & cin) | (a
& cin) | (a & b);
endmodule
```

```
// module ‘six_bit_adder’ adds up the six
//bit inputs and store the result in z.
```

```
module six_bit_adder(x,y,z);
```

```
//as the numbers are in 2’s complement
//form we can set the result bit size to 6.
```

```
        input wire[5:0] x,y;
        output wire[5:0] z;
   wire[5:0] cinr;
```

```
//using full_adder module to get the sum
//and carry for each digit.
```

```
//the carry for LSB is set to 0.
```

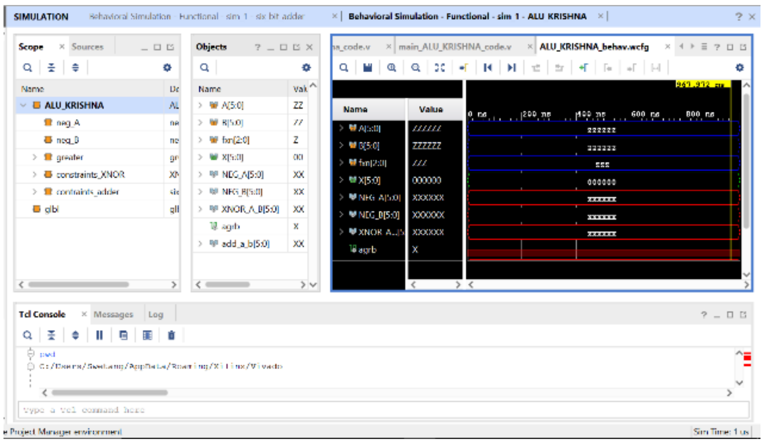
```
full_adder add_bit0(.a(x[0]),
.b(y[0]), .cin(1'd0), .s(z[0]),
.cout(cinr[0]));
```

```
//for further digits the ‘cout’ of the
//previous digit become the ‘cin’ for the
//following digit.
```

```
full_adder add_bit1(.a(x[1]),
.b(y[1]), .cin(cinr[0]), .s(z[1]),
.cout(cinr[1]));
full_adder add_bit2(.a(x[2]),
.b(y[2]), .cin(cinr[1]), .s(z[2]),
.cout(cinr[2]));
```

```
full_adder add_bit3(.a(x[3]),
.b(y[3]), .cin(cinr[2]), .s(z[3]),
.cout(cinr[3]));

full_adder add_bit4(.a(x[4]),
.b(y[4]), .cin(cinr[3]), .s(z[4]),
.cout(cinr[4]));

full_adder add_bit5(.a(x[5]),
.b(y[5]), .cin(cinr[4]), .s(z[5]),
.cout(cinr[5]));



endmodule
```

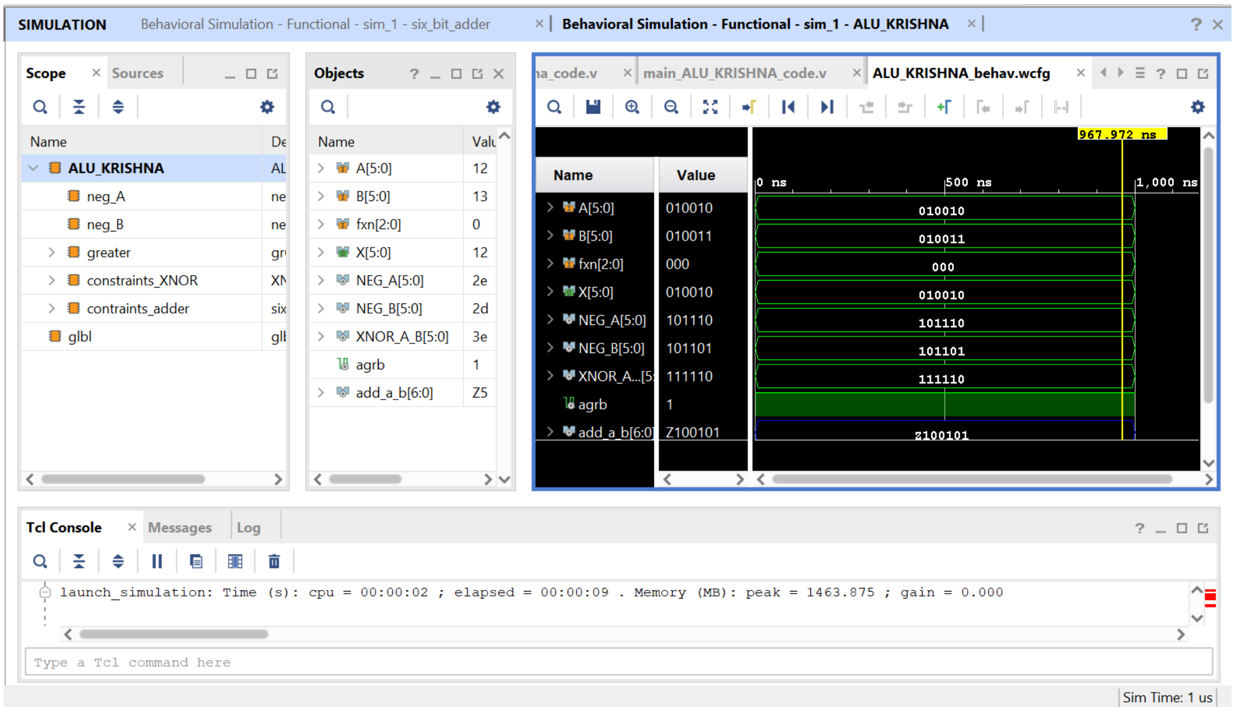
CODE COMPILED SUCCESSFULLY:



```
pwd
C:/Users/Seetang/AppData/Roaming/Xilinx/Vivado
```

All module

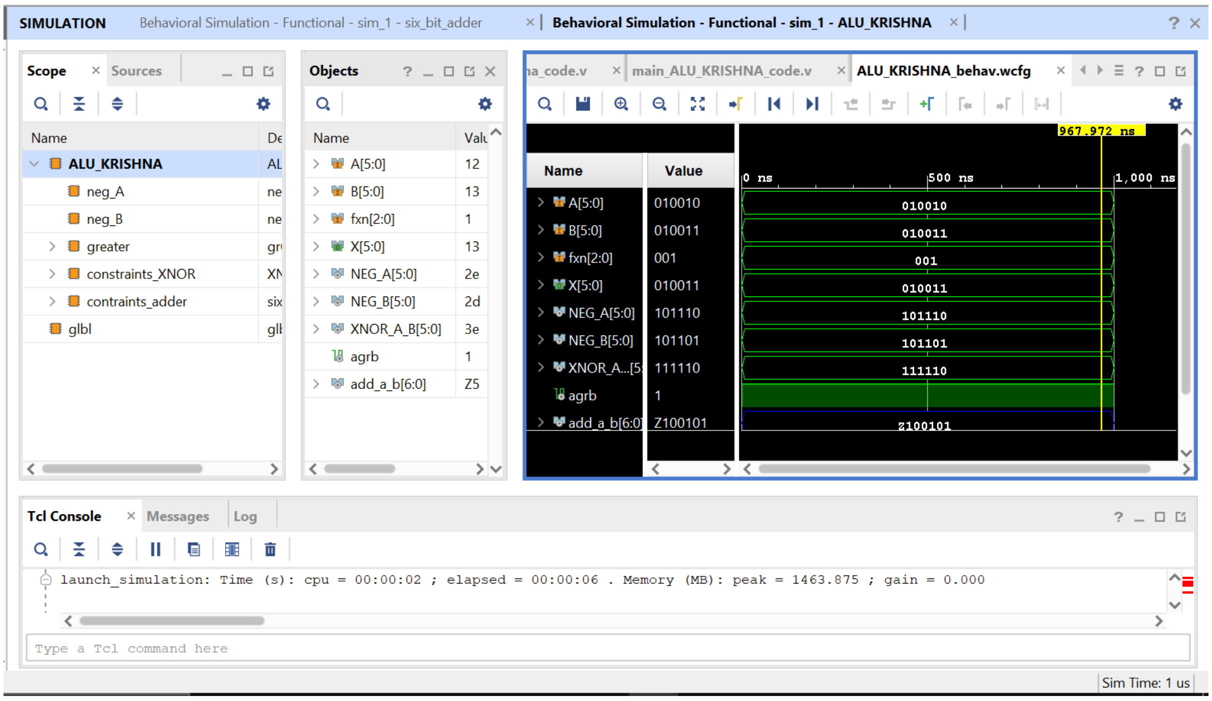
Fxn = 000->



Name	Value
A[5:0]	010010
B[5:0]	010011
fxn[2:0]	000
X[5:0]	010010
NEG_A[5:0]	101110
NEG_B[5:0]	101101
XNOR_A...[5	111110
agrb	1
add_a_b[6:0]	Z100101

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:09 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Fxn = 001->



Name	De
ALU_KRISHNA	AL
neg_A	ne
neg_B	ne
greater	gr
constraints_XNOR	XN
contraints_adder	six
glbl	gl

Name	Valu
A[5:0]	12
B[5:0]	13
fxn[2:0]	1
X[5:0]	13
NEG_A[5:0]	2e
NEG_B[5:0]	2d
XNOR_A_B[5:0]	3e
agrb	1
add_a_b[6:0]	Z5

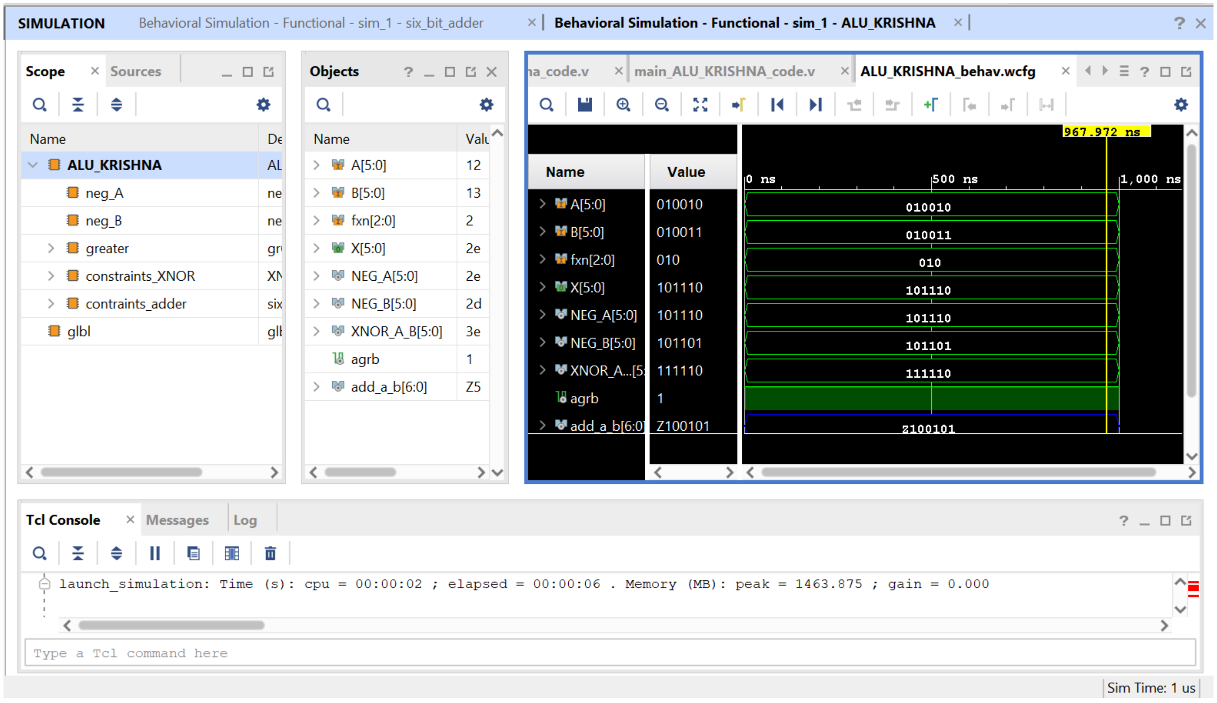
Name	Value
A[5:0]	010010
B[5:0]	010011
fxn[2:0]	001
X[5:0]	010011
NEG_A[5:0]	101110
NEG_B[5:0]	101101
XNOR_A...[5:0]	111110
agrb	1
add_a_b[6:0]	Z100101

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Type a Tcl command here

Sim Time: 1 us

Fxn = 010->



Name	De
ALU_KRISHNA	AL
neg_A	ne
neg_B	ne
greater	gr
constraints_XNOR	XN
contraints_adder	six
glbl	gl

Name	Valu
A[5:0]	12
B[5:0]	13
fxn[2:0]	2
X[5:0]	2e
NEG_A[5:0]	2e
NEG_B[5:0]	2d
XNOR_A_B[5:0]	3e
agrb	1
add_a_b[6:0]	Z5

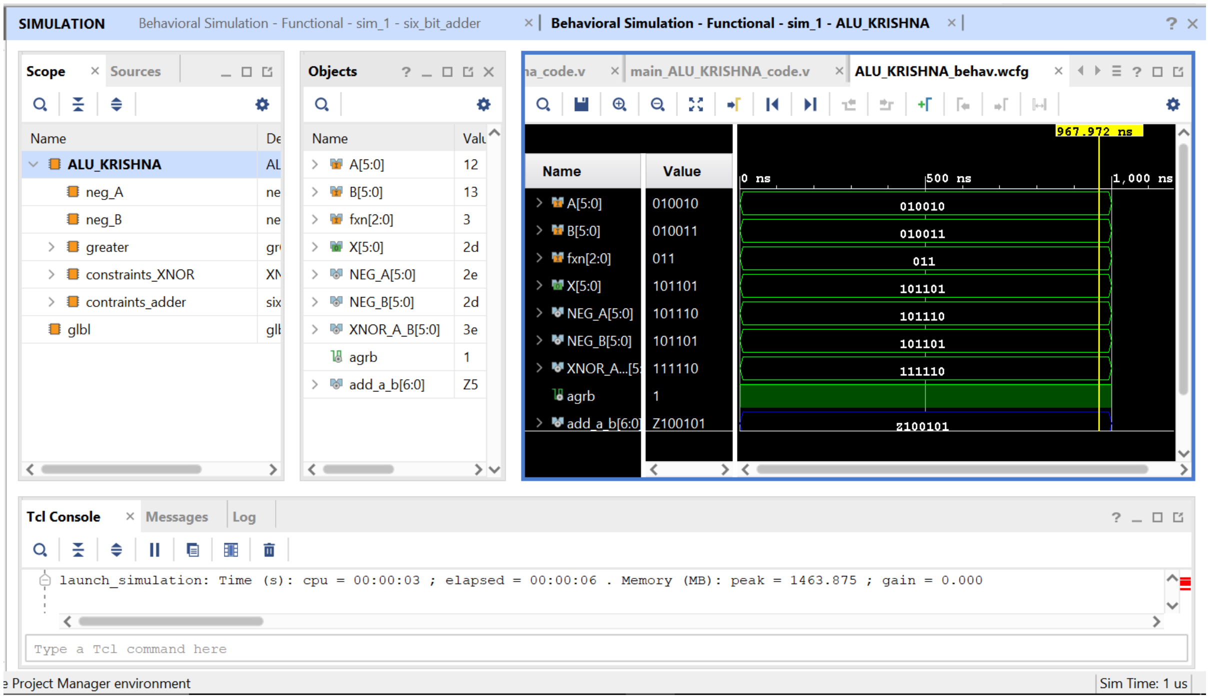
Name	Value
A[5:0]	010010
B[5:0]	010011
fxn[2:0]	010
X[5:0]	101110
NEG_A[5:0]	101110
NEG_B[5:0]	101101
XNOR_A...[5:0]	111110
agrb	1
add_a_b[6:0]	Z100101

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Type a Tcl command here

Sim Time: 1 us

Fxn = 011->



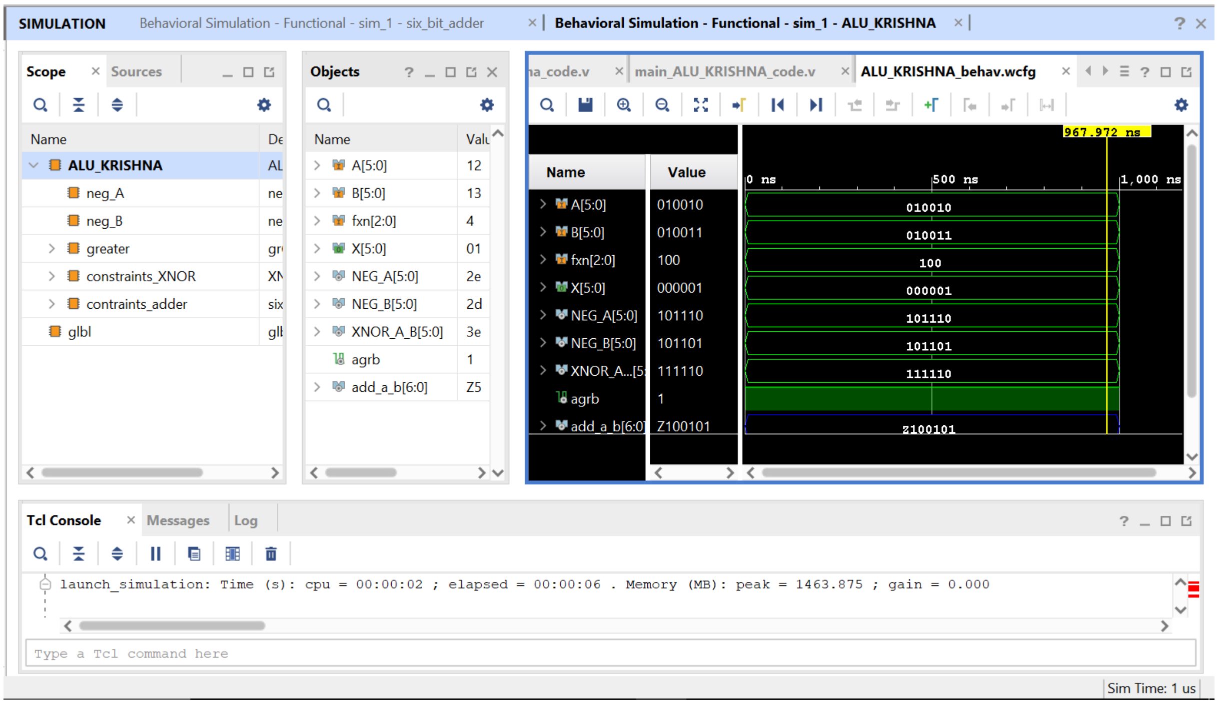
Name	De
ALU_KRISHNA	AL
neg_A	ne
neg_B	ne
greater	gr
constraints_XNOR	XN
contraints_adder	six
glbl	gl

Name	Valu
A[5:0]	12
B[5:0]	13
fxn[2:0]	3
X[5:0]	2d
NEG_A[5:0]	2e
NEG_B[5:0]	2d
XNOR_A_B[5:0]	3e
agrb	1
add_a_b[6:0]	Z5

Name	Value
A[5:0]	010010
B[5:0]	010011
fxn[2:0]	011
X[5:0]	101101
NEG_A[5:0]	101110
NEG_B[5:0]	101101
XNOR_A...[5:	111110
agrb	1
add_a_b[6:0]	Z100101

```
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:06 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Fxn = 100->



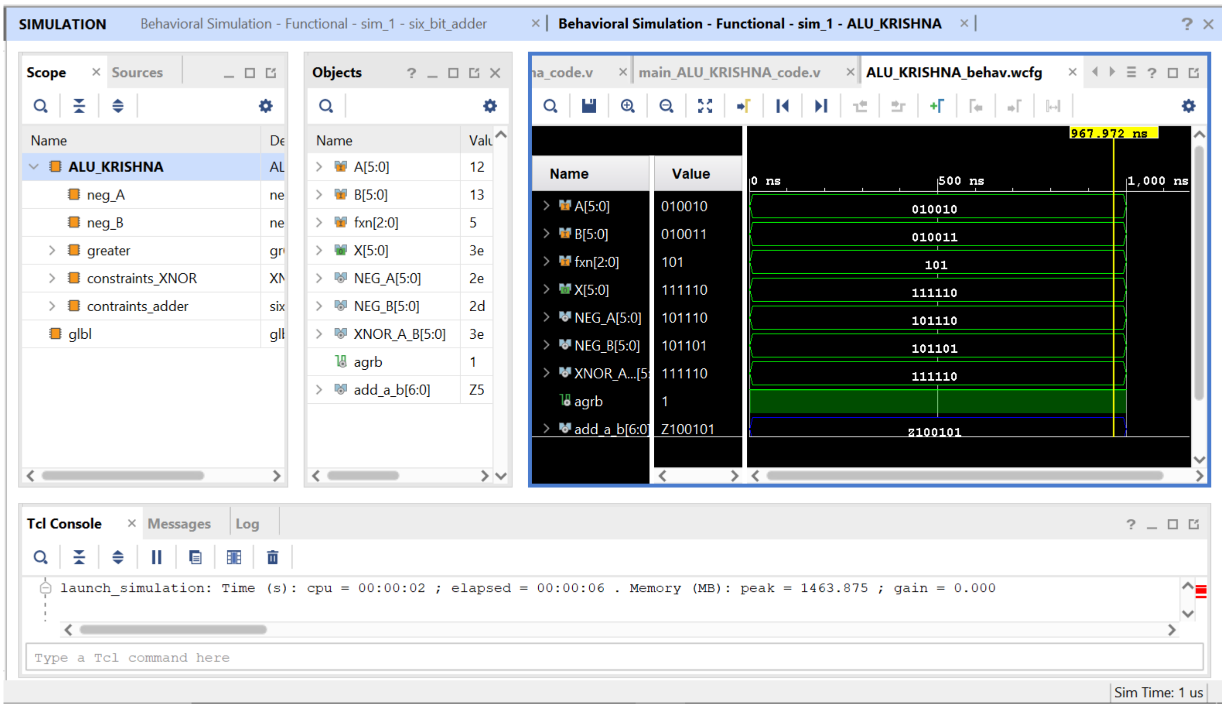
Name	De
ALU_KRISHNA	AL
neg_A	ne
neg_B	ne
greater	gr
constraints_XNOR	XN
contraints_adder	six
glbl	gl

Name	Valu
A[5:0]	12
B[5:0]	13
fxn[2:0]	4
X[5:0]	01
NEG_A[5:0]	2e
NEG_B[5:0]	2d
XNOR_A_B[5:0]	3e
agrb	1
add_a_b[6:0]	Z5

Name	Value
A[5:0]	010010
B[5:0]	010011
fxn[2:0]	100
X[5:0]	000001
NEG_A[5:0]	101110
NEG_B[5:0]	101101
XNOR_A...[5:	111110
agrb	1
add_a_b[6:0]	Z100101

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Fxn = 101->



Name	De
ALU_KRISHNA	AL
neg_A	ne
neg_B	ne
greater	gr
constraints_XNOR	XN
contraints_adder	six
glbl	gl

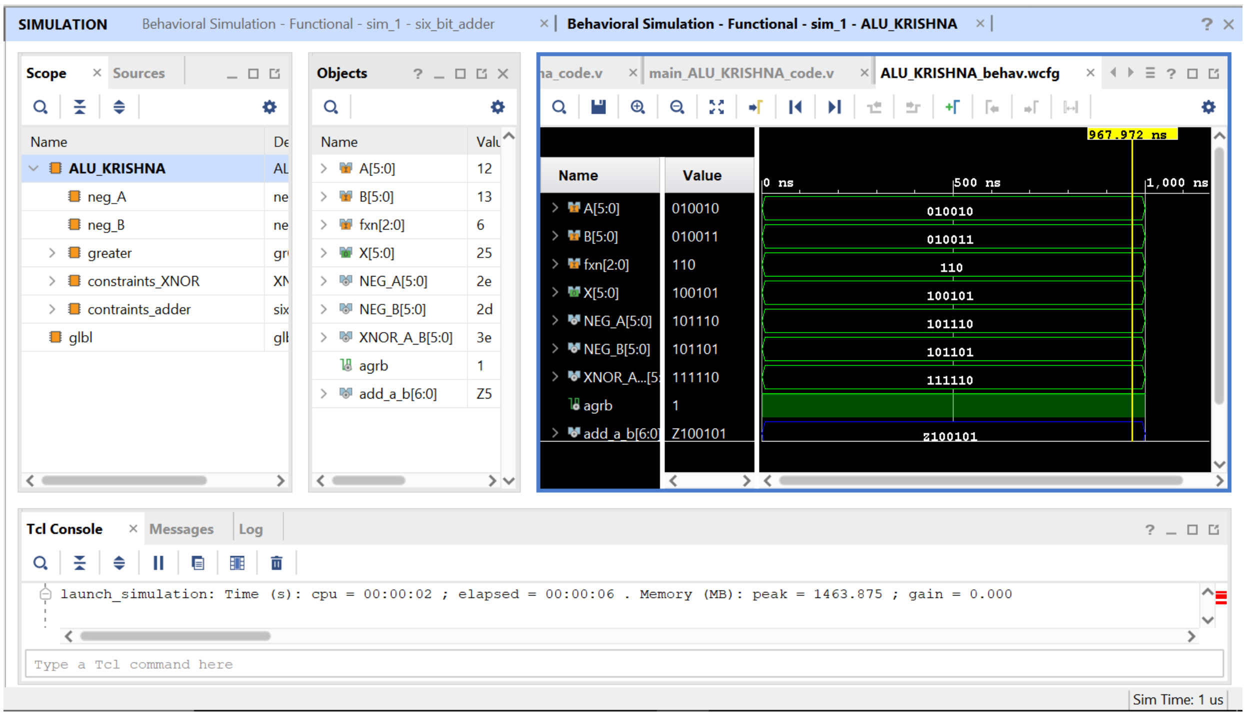
Name	Valu
A[5:0]	12
B[5:0]	13
fxn[2:0]	5
X[5:0]	3e
NEG_A[5:0]	2e
NEG_B[5:0]	2d
XNOR_A_B[5:0]	3e
agrb	1
add_a_b[6:0]	Z5

Name	Value	
A[5:0]	010010	010010
B[5:0]	010011	010011
fxn[2:0]	101	101
X[5:0]	111110	111110
NEG_A[5:0]	101110	101110
NEG_B[5:0]	101101	101101
XNOR_A...[5	111110	111110
agrb	1	
add_a_b[6:0]	Z100101	Z100101

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Sim Time: 1 us

Fxn = 110->



Name	De
ALU_KRISHNA	AL
neg_A	ne
neg_B	ne
greater	gr
constraints_XNOR	XN
contraints_adder	six
glbl	gl

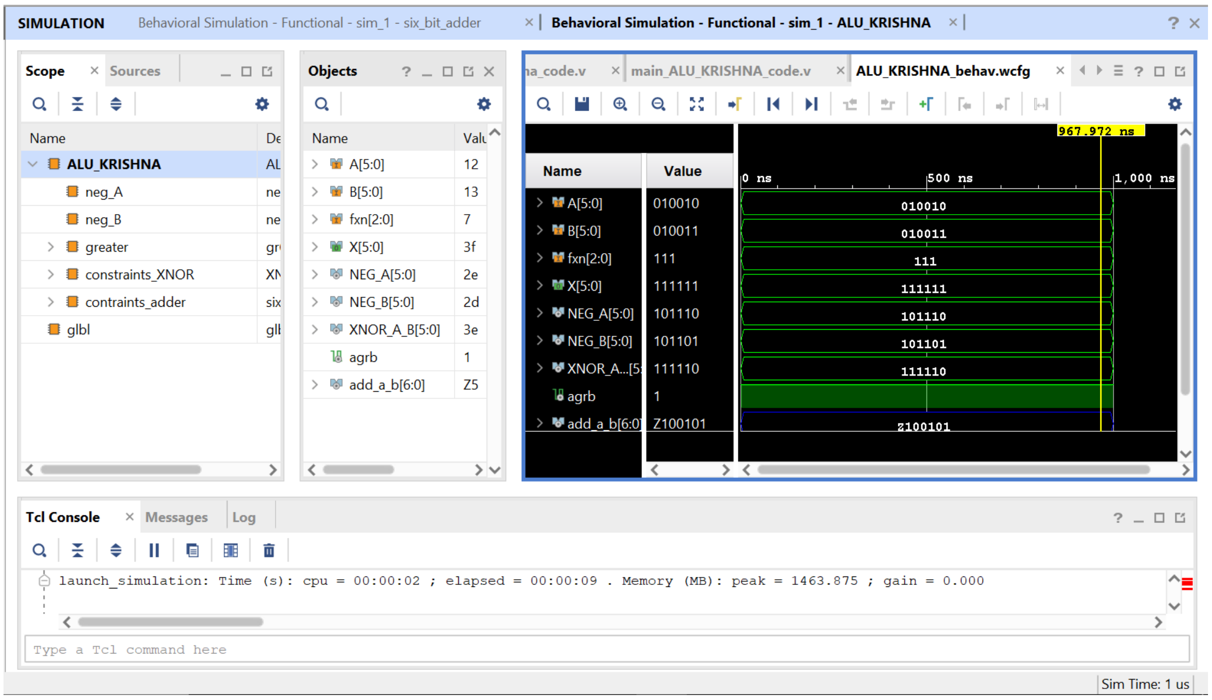
Name	Valu
A[5:0]	12
B[5:0]	13
fxn[2:0]	6
X[5:0]	25
NEG_A[5:0]	2e
NEG_B[5:0]	2d
XNOR_A_B[5:0]	3e
agrb	1
add_a_b[6:0]	Z5

Name	Value	
A[5:0]	010010	010010
B[5:0]	010011	010011
fxn[2:0]	110	110
X[5:0]	100101	100101
NEG_A[5:0]	101110	101110
NEG_B[5:0]	101101	101101
XNOR_A...[5	111110	111110
agrb	1	
add_a_b[6:0]	Z100101	Z100101

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Sim Time: 1 us

Fxn = 111->



Name	De
ALU_KRISHNA	AL
neg_A	ne
neg_B	ne
greater	gr
constraints_XNOR	XN
contraints_adder	six
glbl	gl

Name	Valu
A[5:0]	12
B[5:0]	13
fxn[2:0]	7
X[5:0]	3f
NEG_A[5:0]	2e
NEG_B[5:0]	2d
XNOR_A_B[5:0]	3e
agrb	1
add_a_b[6:0]	Z5

Name	Value	Waveform
A[5:0]	010010	010010
B[5:0]	010011	010011
fxn[2:0]	111	111
X[5:0]	111111	111111
NEG_A[5:0]	101110	101110
NEG_B[5:0]	101101	101101
XNOR_A...[5:0]	111110	111110
agrb	1	
add_a_b[6:0]	Z100101	Z100101

```
0 ns                500 ns              1,000 ns
967.972 ns
```

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:09 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

Type a Tcl command here

Testbench code:

```
module tb_Assign_Krishna();

reg[5:0] A,B;

reg[2:0] fxn;

wire[5:0] X;


ALU_KRISHNA A1(A,B,fxn,X);

initial

begin

        A=6'b000001; B=6'b111111;
fxn= 3'b000;

        #100;

        A=6'b000001; B=6'b111111;
fxn= 3'b001;

        #100;

        A=6'b000001; B=6'b111111;
fxn= 3'b010;

        #100;

        A=6'b000001; B=6'b111111;
fxn= 3'b011;

        #100;

        A=6'b000001; B=6'b111111;
fxn= 3'b100;

        #100;

        A=6'b000001; B=6'b111111;
fxn= 3'b101;

        #100;
```

//testbench module

//all the inputs will be taken as registers
//and the outputs as wire.

//using the main ALU module to test our
// design.

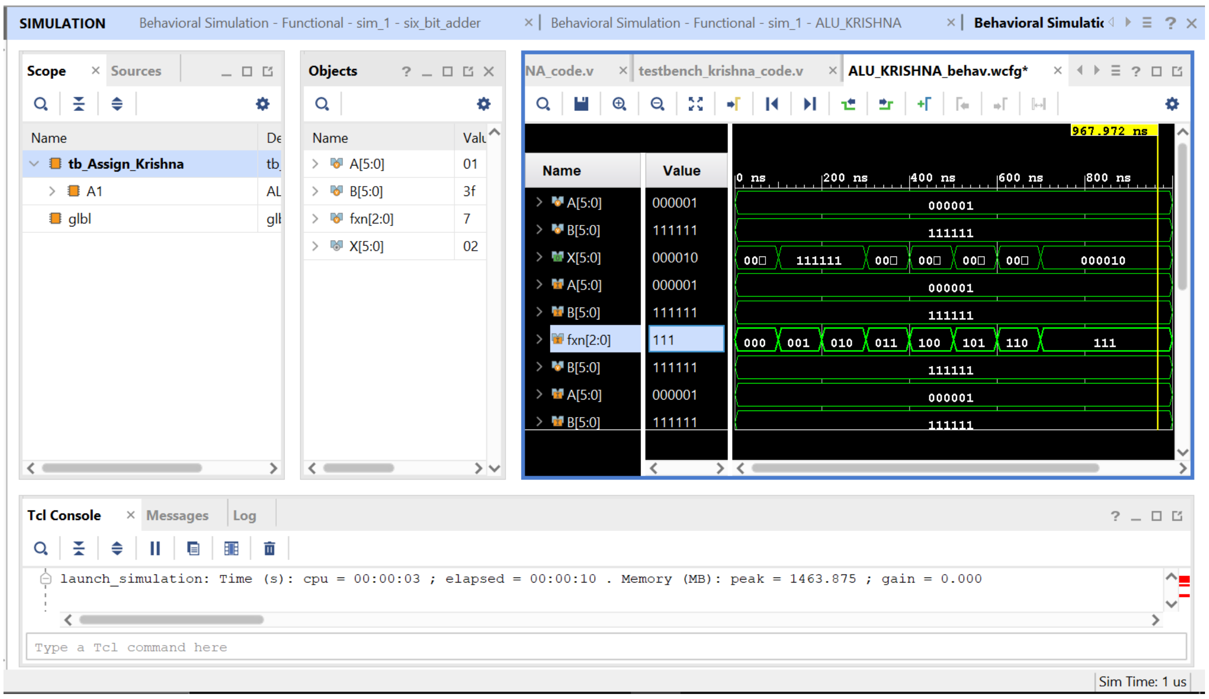
//using 6'bit binary input for testing //the
code.

//setting value of A as 1 and B as -1.

//following the tests for each 'fxn'.

```
        A=6'b000001; B=6'b111111;
fxn= 3'b110;

        #100;

        A=6'b000001; B=6'b111111;
fxn= 3'b111;

        #100;

end

endmodule
```

TESTBENCH RUNS SUCCESSFULLY
WITH ALL THE CORRECT OUTPUTS->



SIMULATION — Behavioral Simulation - Functional - sim_1 - six_bit_adder | Behavioral Simulation - Functional - sim_1 - ALU_KRISHNA | Behavioral Simulatic

Scope: tb_Assign_Krishna, A1, glbl

Name	Value
A[5:0]	01
B[5:0]	3f
fxn[2:0]	7
X[5:0]	02

ALU_KRISHNA_behav.wcfg*

Name	Value
A[5:0]	000001
B[5:0]	111111
X[5:0]	000010
A[5:0]	000001
B[5:0]	111111
fxn[2:0]	111
B[5:0]	111111
A[5:0]	000001
B[5:0]	111111

```
967.972 ns
0 ns    200 ns    400 ns    600 ns    800 ns
000001
111111
000  111111  000  000  000  000  000010
000001
111111
000  001  010  011  100  101  110  111
111111
000001
111111
```

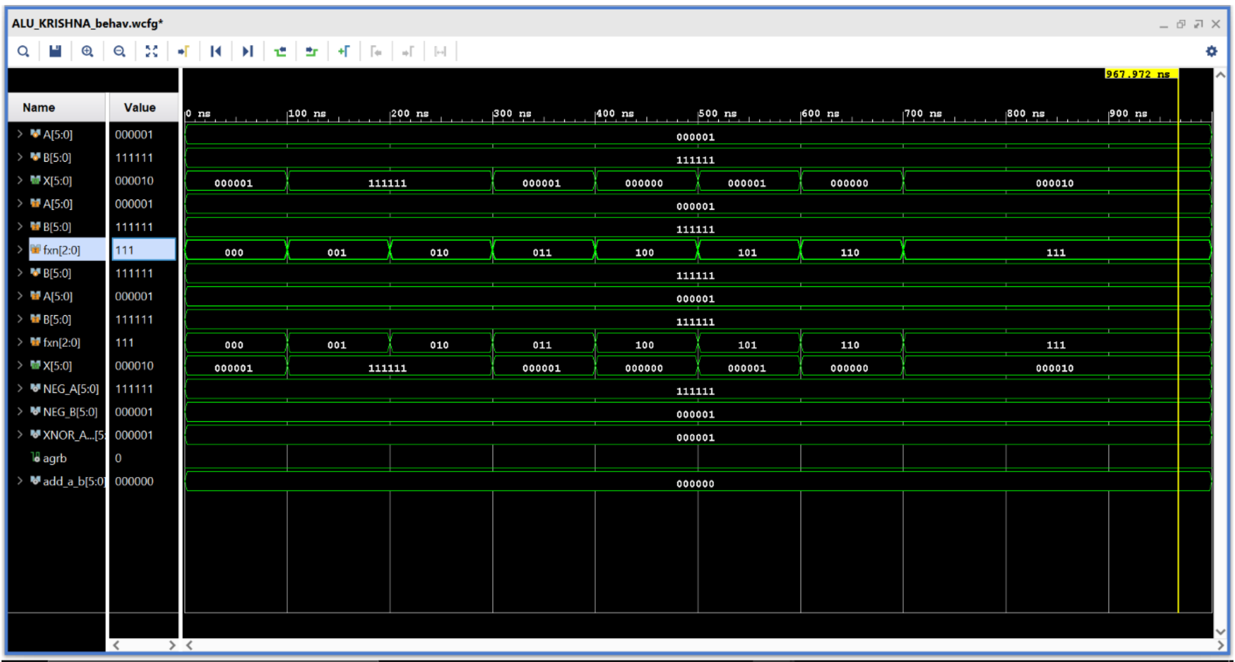
Tcl Console | Messages | Log

```
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:10 . Memory (MB): peak = 1463.875 ; gain = 0.000
```

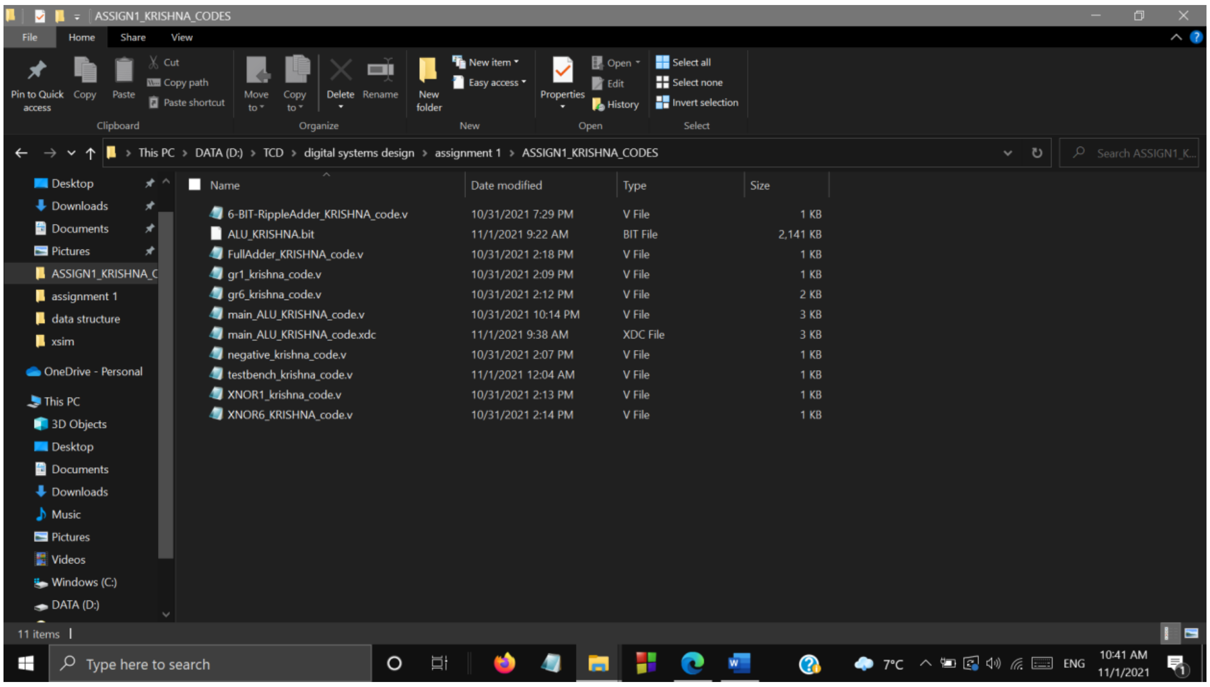
Type a Tcl command here

Sim Time: 1 us

Testbench results:

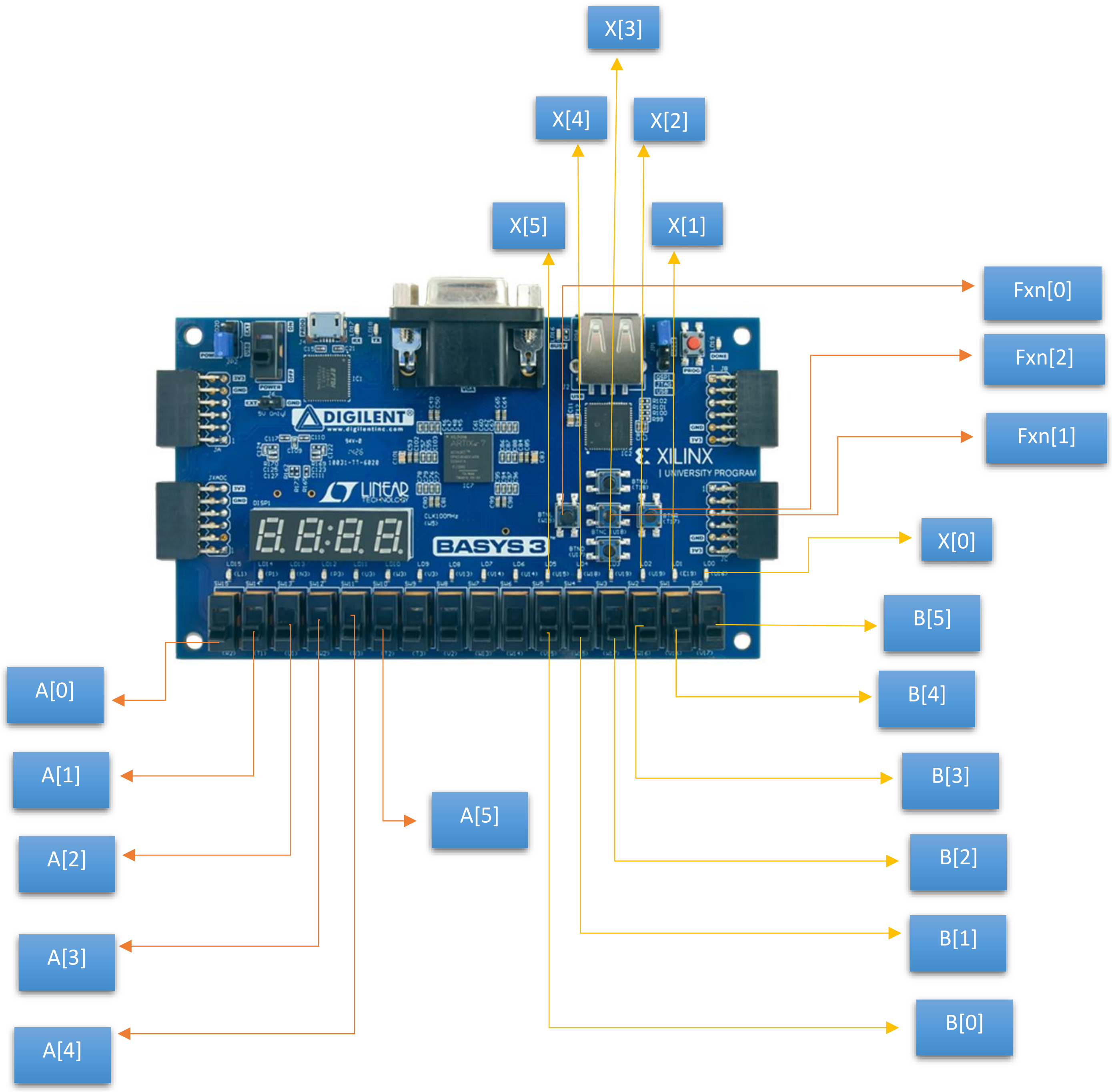


Screenshot (file directory structure):



Name	Date modified	Type	Size
6-BIT-RippleAdder_KRISHNA_code.v	10/31/2021 7:29 PM	V File	1 KB
ALU_KRISHNA.bit	11/1/2021 9:22 AM	BIT File	2,141 KB
FullAdder_KRISHNA_code.v	10/31/2021 2:18 PM	V File	1 KB
gr1_krishna_code.v	10/31/2021 2:09 PM	V File	1 KB
gr6_krishna_code.v	10/31/2021 2:12 PM	V File	2 KB
main_ALU_KRISHNA_code.v	10/31/2021 10:14 PM	V File	3 KB
main_ALU_KRISHNA_code.xdc	11/1/2021 9:38 AM	XDC File	3 KB
negative_krishna_code.v	10/31/2021 2:07 PM	V File	1 KB
testbench_krishna_code.v	11/1/2021 12:04 AM	V File	1 KB
XNOR1_krishna_code.v	10/31/2021 2:13 PM	V File	1 KB
XNOR6_KRISHNA_code.v	10/31/2021 2:14 PM	V File	1 KB

DEMO:



DEMO:

INPUT PINS:

##for inputs using the switches

##SWITCHES FOR A:

##A[0] = R2

##A[1] = T1

##A[2] = U1

##A[3] = W2

##A[4] = R3

##A[5] = T2

##SWITCHES FOR B:

##B[0] = V15

##B[1] = W15

##B[2] = W17

##B[3] = W16

##B[4] = V16

##B[5] = V17

##FOR fxn using buttons

BTNL for fxn[0] (pin - T17)

BTNC for fxn[1] (pin - V18)

BTNR for fxn[2] (pin - W19)

##LEDs for OUTPUT:

##X[0] =U16

##X[1] =E19

##X[2] =U19

##X[3] =V19

##X[4] =W18

##X[5] =U15

TO TEST THE PROGRAM ON BOARD, FOLLW THE STEPS:

The inputs A and B are assigned to switches so switch on sw5, sw4, sw3, sw2, sw1 and sw15 to give inputs 1 and -1 to A and B respectively.

The input fxn is assigned to buttons, press the button BTNR to give input value of 001 to fxn.

Now the expected output which must be shown in LEDs is 000001.

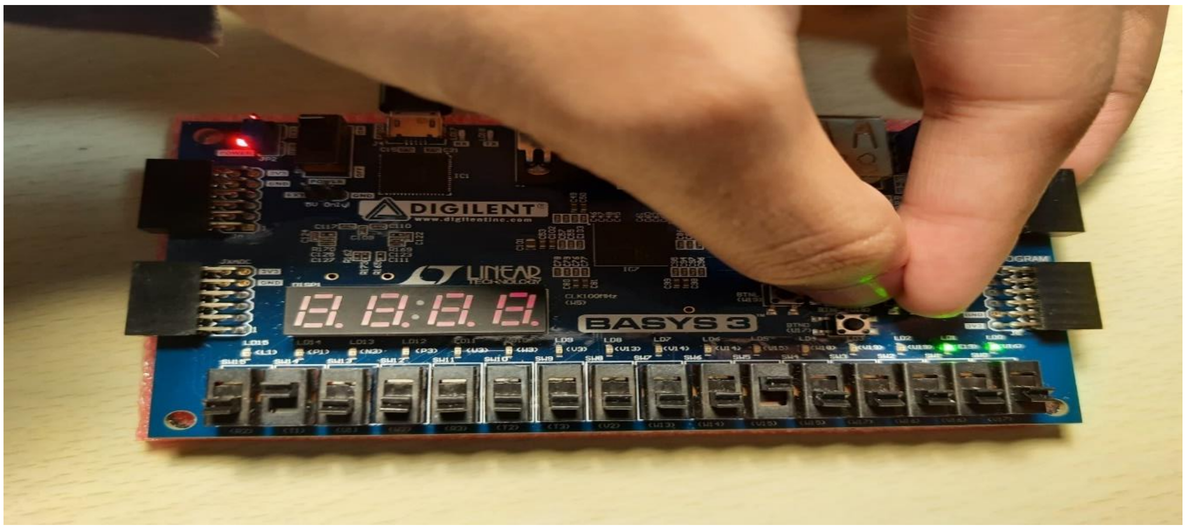
Similarly we can test it for the rest of the cases.

XDC FILE CODE:

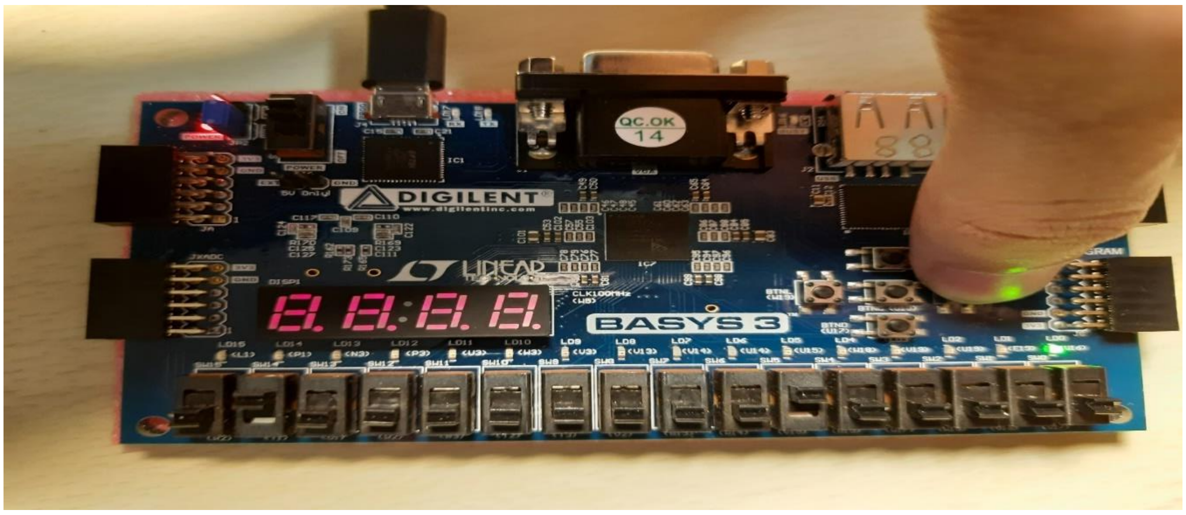
```
#set_property PACKAGE_PIN W5 [get_ports CCLK]
#set_property IOSTANDARD LVCMOS33 [get_ports CCLK]

##using the LEDs to get the output stored in X.
##from LSB(X[0]) to MSB(X[5]) using LEDs 0-5.
set_property PACKAGE_PIN U16 [get_ports {X[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {X[0]}]

set_property PACKAGE_PIN E19 [get_ports {X[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {X[1]}]

set_property PACKAGE_PIN U19 [get_ports {X[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {X[2]}]

set_property PACKAGE_PIN V19 [get_ports {X[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {X[3]}]

set_property PACKAGE_PIN W18 [get_ports {X[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {X[4]}]

set_property PACKAGE_PIN U15 [get_ports X[5]]
set_property IOSTANDARD LVCMOS33 [get_ports X[5]]
```

```
##for inputs using the switches

##SWITCHES FOR:

##A[0] = R2

##A[1] = T1

##A[2] = U1

##A[3] = W2

##A[4] = R3

##A[5] = T2



set_property PACKAGE_PIN R2 [get_ports {A[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]


set_property PACKAGE_PIN T1 [get_ports {A[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]


set_property PACKAGE_PIN U1 [get_ports {A[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]


set_property PACKAGE_PIN W2 [get_ports {A[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]


set_property PACKAGE_PIN R3 [get_ports {A[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {A[4]}]
set_property PACKAGE_PIN T2 [get_ports {A[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {A[5]}]


##SWITCHES FOR:

##B[0] = V15

##B[1] = W15

##B[2] = W17

##B[3] = W16

##B[4] = V16

##B[5] = V17


set_property PACKAGE_PIN V15 [get_ports {B[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]


set_property PACKAGE_PIN W15 [get_ports {B[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]


set_property PACKAGE_PIN W17 [get_ports {B[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]


set_property PACKAGE_PIN W16 [get_ports {B[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]


set_property PACKAGE_PIN V16 [get_ports {B[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {B[4]}]
set_property PACKAGE_PIN V17 [get_ports {B[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {B[5]}]


##FOR fxn using buttons

## BTNL for fxn[0] (pin - T17)

## BTNC for fxn[1] (pin - V18)

## BTNR for fxn[2] (pin - W19)



set_property PACKAGE_PIN T17 [get_ports fxn[0]]

set_property IOSTANDARD LVCMOS33 [get_ports fxn[0]]



set_property PACKAGE_PIN V18 [get_ports fxn[1]]

set_property IOSTANDARD LVCMOS33 [get_ports fxn[1]]


set_property PACKAGE_PIN W19 [get_ports fxn[2]]

set_property IOSTANDARD LVCMOS33 [get_ports fxn[2]]
```

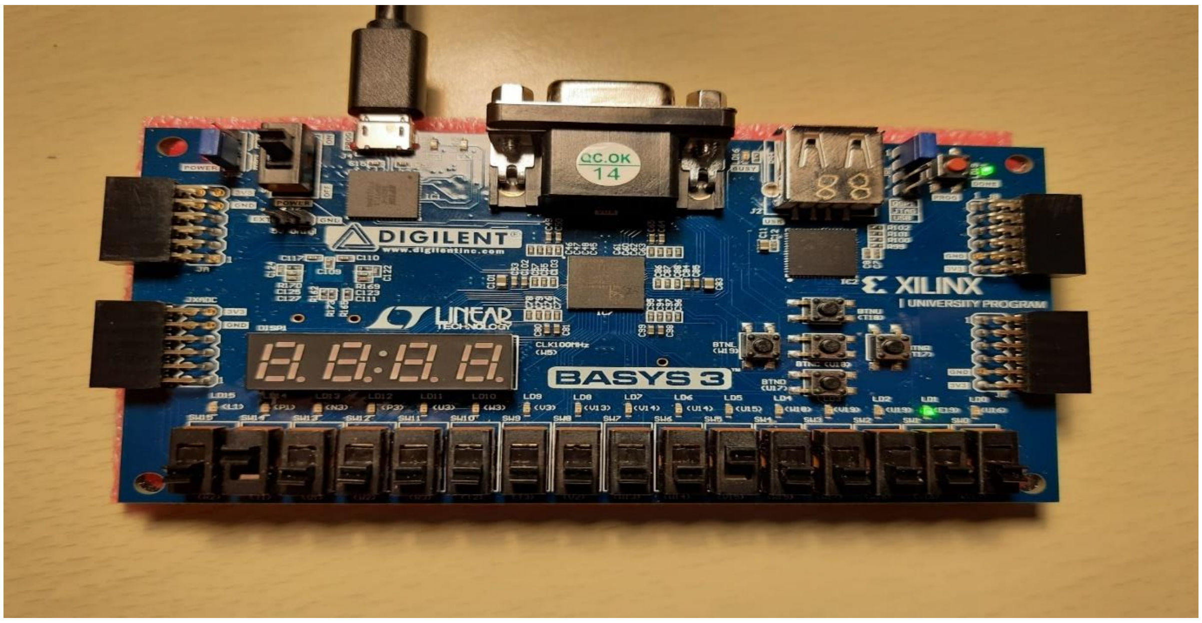
Outputs shown on board:



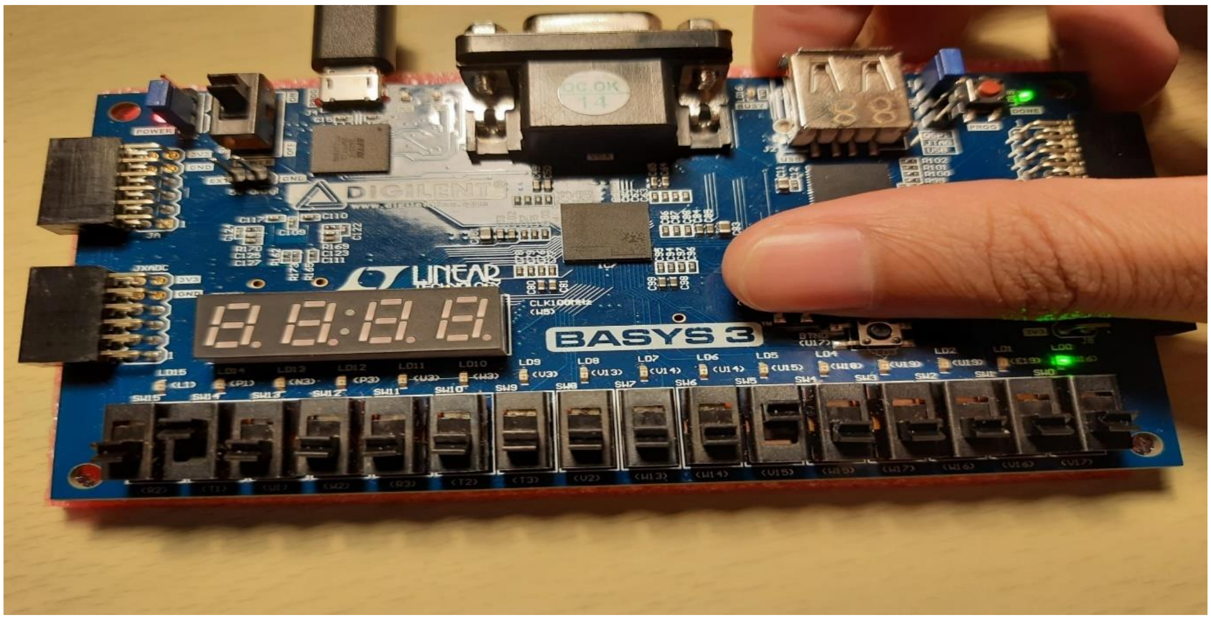
- inputs: OUTPUT:
- A = 2;
- fxn = 110; X = 000011 = 3;
- B = 1;



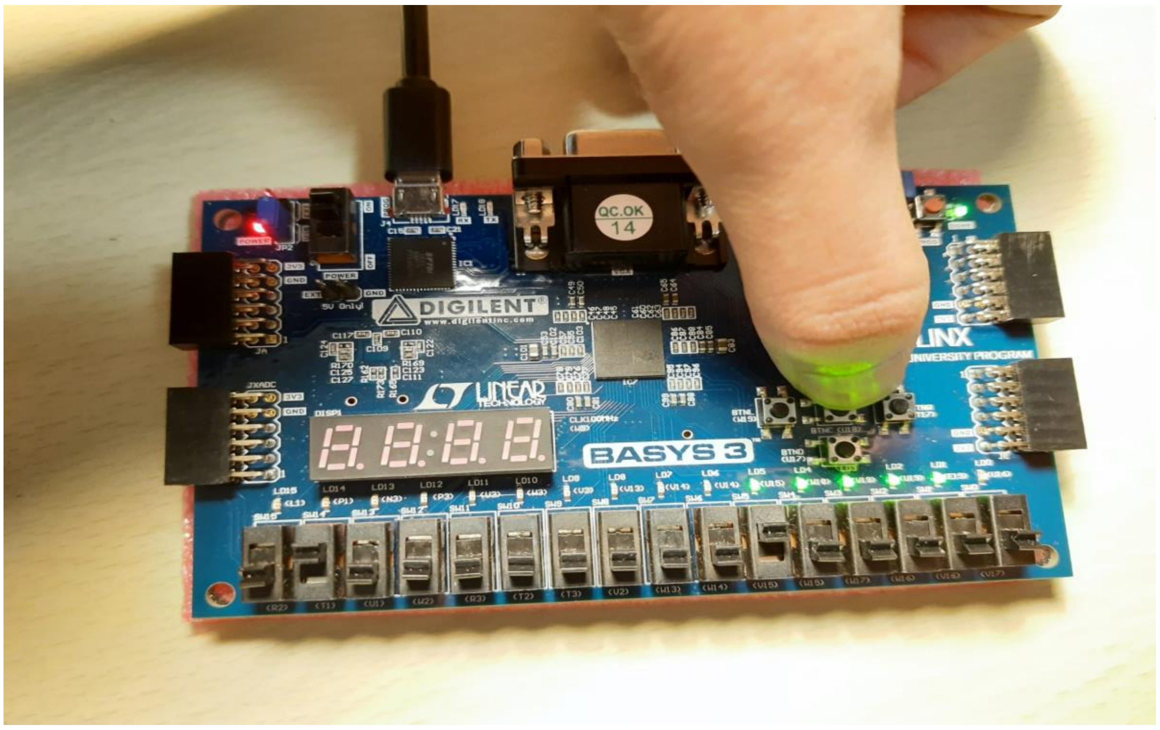
- inputs: OUTPUT:
- A = 2;
- fxn = 110; X = 000011 = 3;
- B = 1;



- inputs: OUTPUT:
- A = 2;
- fxn = 000; X = 000010 = 2;
- B = 1;



- inputs: OUTPUT:
- A = 2;
- Fxn = 111; X = 000001 = 1;
- B = 1;



- inputs: OUTPUT:
- A = 2;
- fxn = 010; X = -2;
- B = 1;

The design that I have created is using Multiplexer so if it gets any other unexpected input it would a default value output as 0.