3C07 Assignment 1

MINI ARITHMETIC LOGIC UNIT (MINI-ALU)

Ion Alin Spiridon ***15/03/2017***

Abstract

The present report describes the design, functioning and testing of a mini ALU. It presents the hierarchy and how Verilog modules are integrated and connected. The unit designed performed all its functions satisfactorily under the test. All the functional diagrams, Verilog code for the respective modules and their test vectors have been included at the end for future reference.

Introduction

The objective of this exercise was to implement a Mini ALU (Arithmetic Logic Unit). ALU is an integrating part of the CPU in modern processor units, used, as the name suggests, for performing arithmetic operations and combinatorial logic. The unit to be designed had to make use of the Verilog modules already created in class on previous exercises. The inputs A, B were to be two six bits binary numbers in 2' complement form. It was to be designed as a high level unit, which would make use of a four-bit selecting code "Sel" and incorporate the following functions:

- 0000 Output: 6'b000000
- 0001 Output: A
- 0010 Output: B
- 0011 Output: A>=B (is A greater than or equal to B)
- 0100 Output: -A 0101 -B
- 0110 Output: A (rotated by 3bits to the right)
- 0111 Output: B (rotated by 3bits to the right)
- 1000 Output: A^B (Bitwise exclusive OR)
- 1001 Output: A' (not A)
- 1010 Output: B' (not B)
- 1011 Output: A-B
- 1100 Output: A+B
- 1111 Output: 6'b111111

Each one of the most complex functions like the arithmetic operations, bit shifting, logic operators were to be implemented using a distinct Verilog module. Each module was to be tested separately in order to ensure it was performing adequately. The test vectors were to be carefully selected so that the corner conditions were included and tested for.

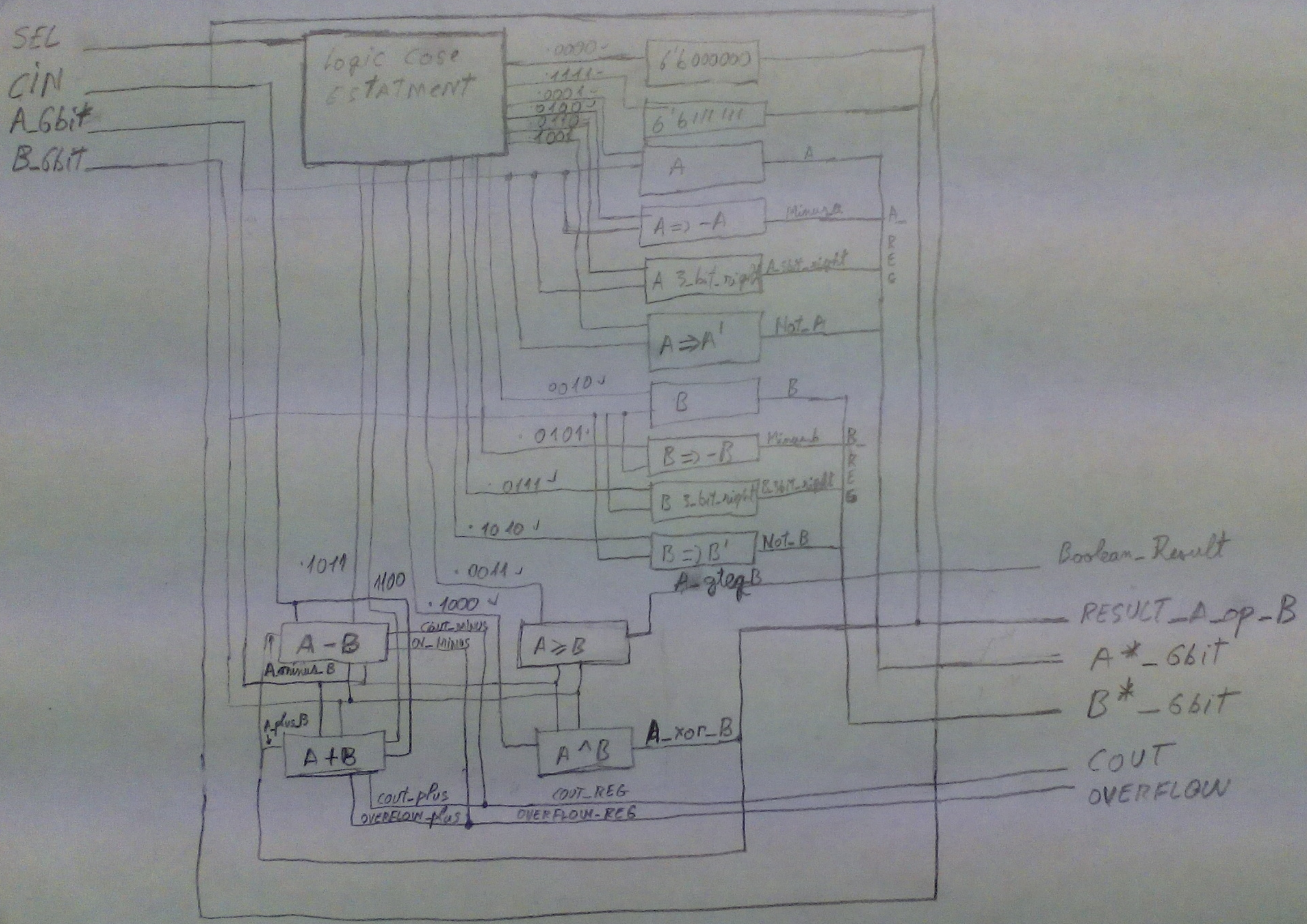
<u>Design</u>

The chosen design is graphically represented on figure 1 below. The high level ALU module is composed of 7 sub modules as follows:

- A_minus_b module performs the subtraction of the inputs A and B. It works by emploing a minus_a_b module to obtain "-B" and then an sixbit_carry_ripple_adder to add "A" to "-B".
- Six_bit_gt_eq compares "A" and "B" and outputs a boolean 1 is the first is greater or equal to the second.
- Barrel_shifter_3bits module is used to perform a logic shift to the right by 3 bits on either "A" or "B". I works by creating a new 12 bits binary number formed by two copies of the imput and then the bits are manipulated so that the output is formed by 6 bits starting from the 3rd position.
- Inverter module is used to obtain a bitwise not output of the input by using the logic operator "~" in Verilog code.
- Bitwise6_xor module performes a logic XOR bitwise operation between "A" and "B". It uses a submodule called bitwise_xor that performs the XOR operation for one bit at the time.
- Minus _a_b module is used to obtain the invers of the input "A" or "B" . It works by using an inverted module to invert the bits of the input and then uses a sixbit_carry_ripple adder to add one to it.
- Sixbit _ carry_ripple adder module uses a full_adder sub module to perform binary addition on a single bit taking into account any carry in and also outputting any carry out and overflow situation.

The simpler operation like outputting the input or a string of "0" or "1" was hard coded into the ALU module itself. In order to deal with the "Sel" function a case statement was used. It checked the "Sel" input and for each possible value assigned the corresponding output. In order to protect against unexpected inputs a default statement was written and design to output a string of five "0". This error code generated can then be used to troubleshoot the system.

There were no modifications required to the exiting modules as they had been previously tested and working according


SEL
CIN
A_6bit*
B_6bit
logic case statment
0000
1111
0001
0100
0110
1001
6'b000000
6'b111111
A
A
A=) -A
Minus_A
A 3_bit_right
A_3bit_right
A ⇒ A'
Not_A
A_REG
0010
B
B
0101
B =) -B
Minus_B
0111
B 3_bit_right
B_3bit_right
1010
B =) B'
Not_B
B_REG
Boolean_Result
1011
1100
0011
1000
A_gteq_B
A - B
cout_minus
ov_minus
A_minus_B
A ≥ B
RESULT_A_op_B
A*_6bit
B*_6bit
A_plus_B
A + B
A ^ B
A_xor_B
cout_plus
OVERFLOW_plus
COUT_REG
OVERFLOW_REG
COUT
OVERFLOW

Testing

As the individual sub modules had been tested already, the only test bench designed was for the high level ALU module. For each function one test vector was created. Below waveforms for every test vector and the value of the inputs and corresponding outputs are presented.

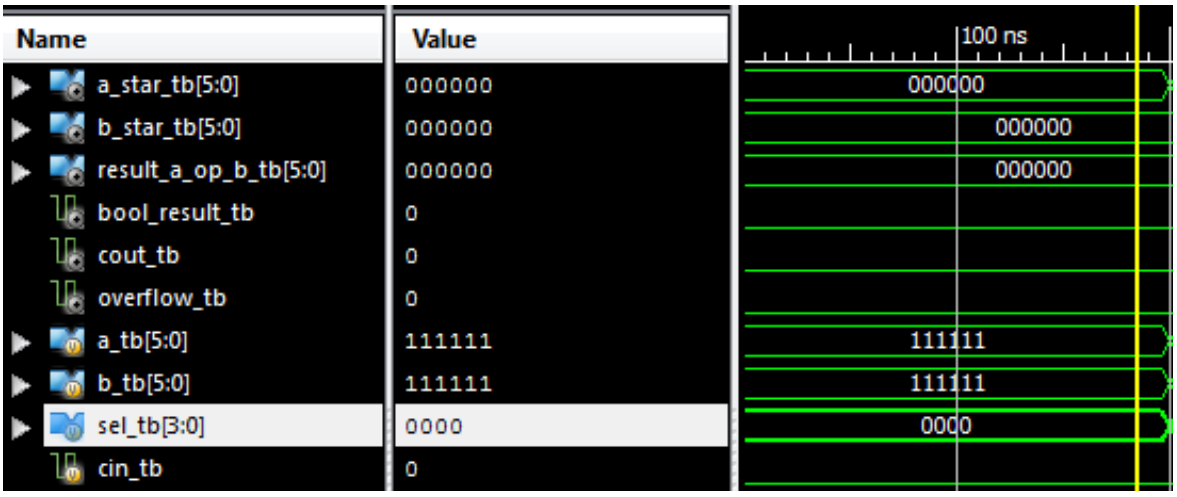
The variables are as follows:

- A_tb, B_tb represent the inputs A and B respectively
- A_star, B_star represent the output that is the input A or B after having an operant applied to it.
- Sel_tb represents the code of the operation to be performed
- Cin_tb represents a carry in from a previous operation for addition and subtraction
- Cout_tb represents a carry out from resultant from an operation
- Overflow_tb indicates that overflow has occurred
- Bool_result _tb represents the output of the greater or equal to comparison
- Result_a_op_b represents the result of the operation between the inputs A and B

A shortcoming to this test strategy is that, although the values for the test vectors was picked as to represent some of the corner conditions there is only one test vector for each function. More than one test vectors would have increased the certainty of a correct behaviour under a wider set of parameters.

0000:

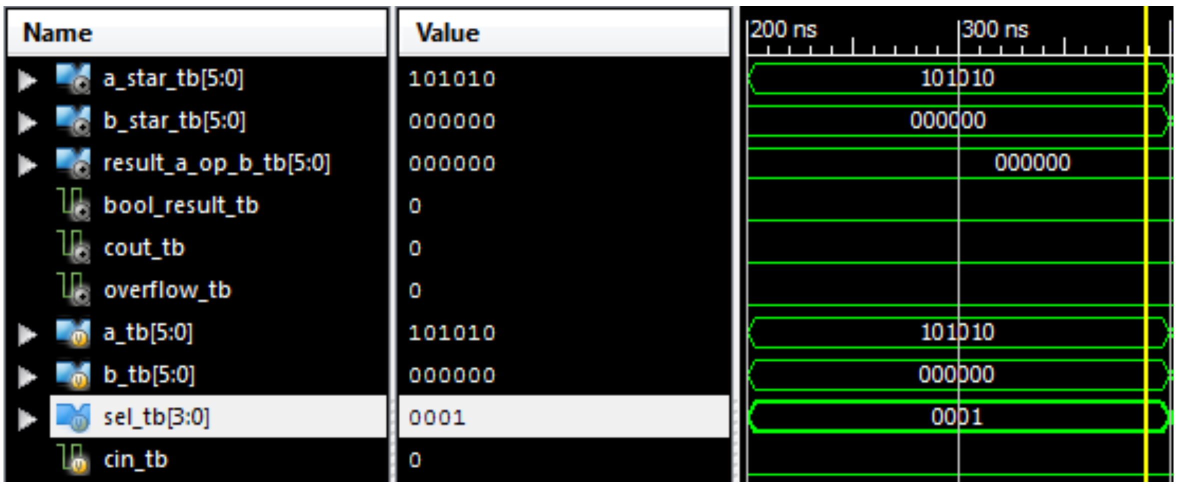
The expected output for the SEL = 0000 was as string of 6 zeros (000000) no matter what the values of inputs A or B were. The waveform below the result is 000000 as expected.



Name	Value
a_star_tb[5:0]	000000
b_star_tb[5:0]	000000
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	111111
b_tb[5:0]	111111
sel_tb[3:0]	0000
cin_tb	0

<u>0001:</u>

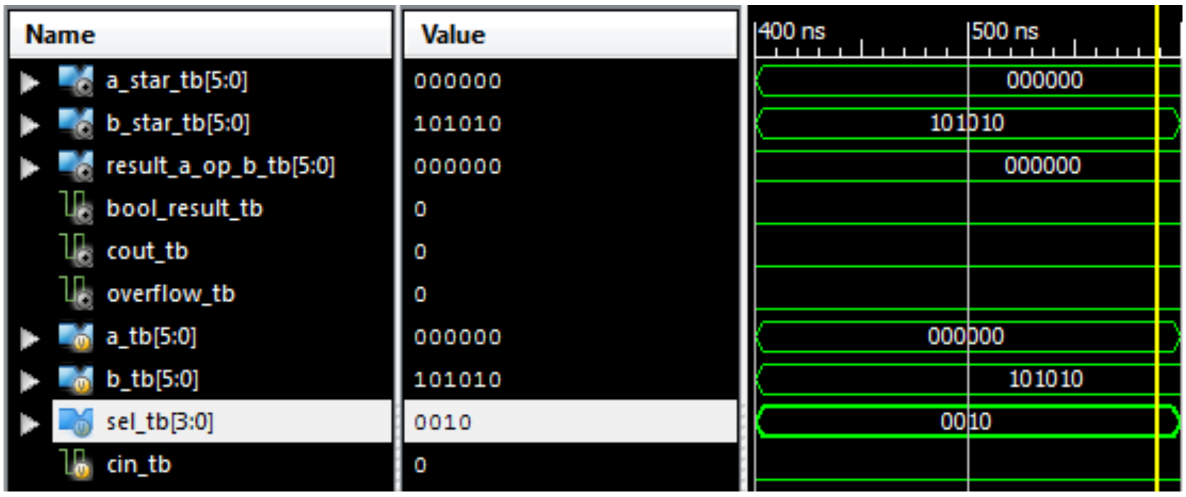
The expected output is the value of the input A. The waveform shows A_tb and A_star having the same value.



Name	Value
a_star_tb[5:0]	101010
b_star_tb[5:0]	000000
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	101010
b_tb[5:0]	000000
sel_tb[3:0]	0001
cin_tb	0

<u>0010:</u>

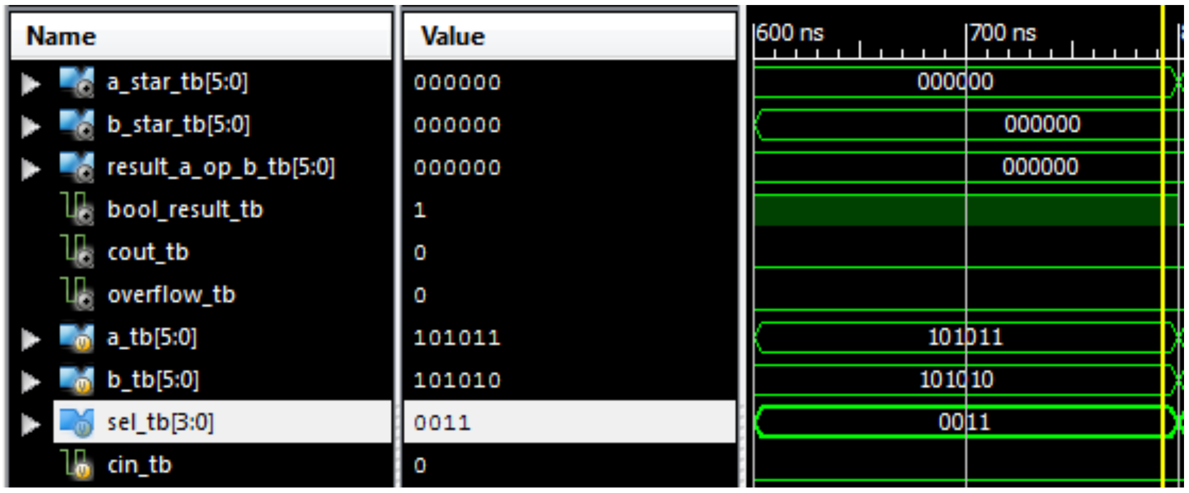
The expected output for the SEL = 0010 is the value of the input B. The waveform shows B_tb and B_star having the same value.



Name	Value
a_star_tb[5:0]	000000
b_star_tb[5:0]	101010
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	000000
b_tb[5:0]	101010
sel_tb[3:0]	0010
cin_tb	0

0011:

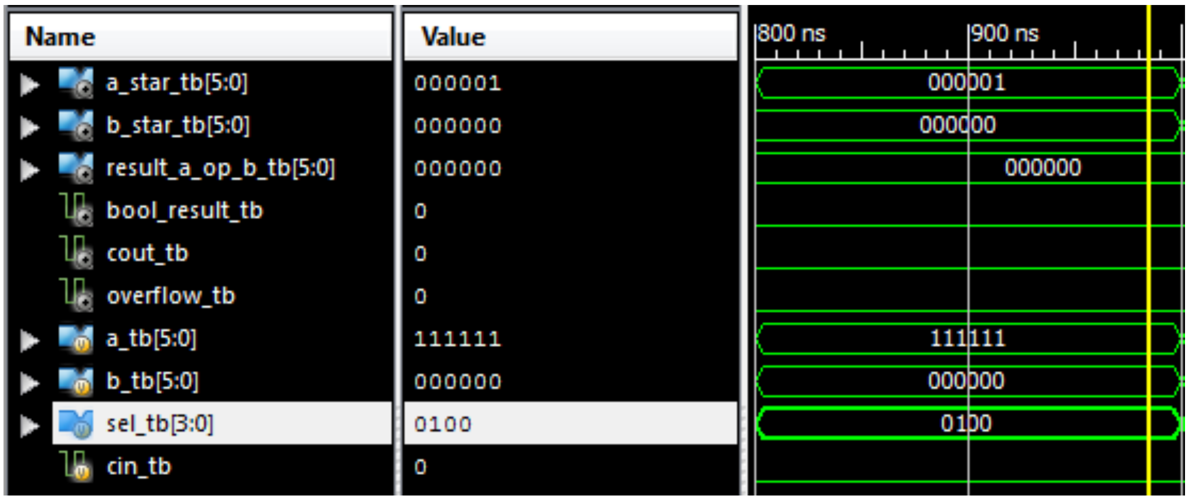
The expected output for the SEL = 0011 was 1 since the input A is greater than the input B. The bool_result_tb is high as expected.



Name	Value
a_star_tb[5:0]	000000
b_star_tb[5:0]	000000
result_a_op_b_tb[5:0]	000000
bool_result_tb	1
cout_tb	0
overflow_tb	0
a_tb[5:0]	101011
b_tb[5:0]	101010
sel_tb[3:0]	0011
cin_tb	0

0100:

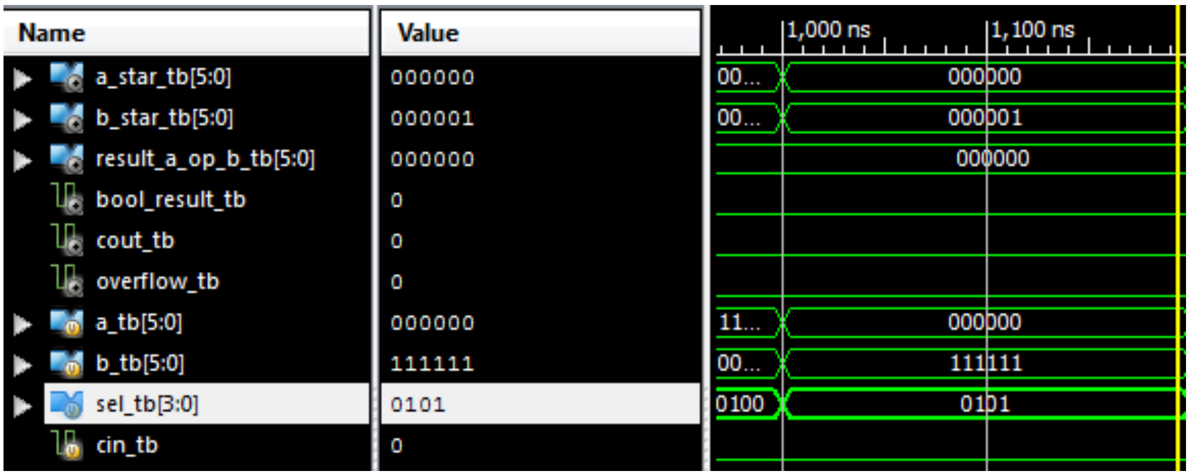
The expected output for the SEL = 0100 was the positive 1 in 6 bit bynary or 000001 since the input A was -1 in 2's complement notation. The output A_star is 000001 as expected.



Name	Value
a_star_tb[5:0]	000001
b_star_tb[5:0]	000000
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	111111
b_tb[5:0]	000000
sel_tb[3:0]	0100
cin_tb	0

0101:

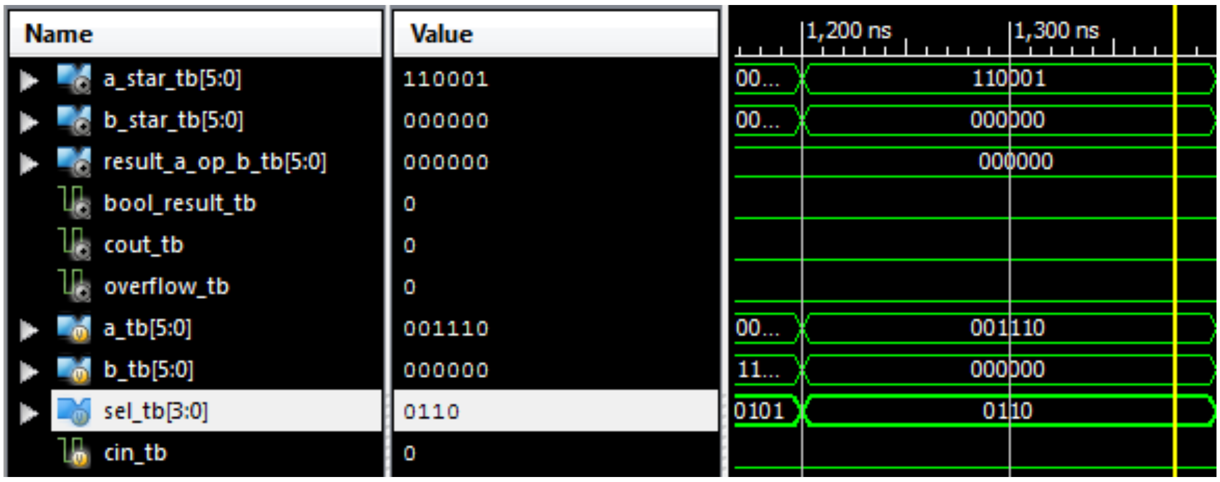
The expected output for the SEL = 0101 was the positive 1 in 6 bit bynary or 000001 since the input B was -1 in 2’s complement notation. The output B_star is 000001 as expected.



Name	Value
a_star_tb[5:0]	000000
b_star_tb[5:0]	000001
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	000000
b_tb[5:0]	111111
sel_tb[3:0]	0101
cin_tb	0

0110:

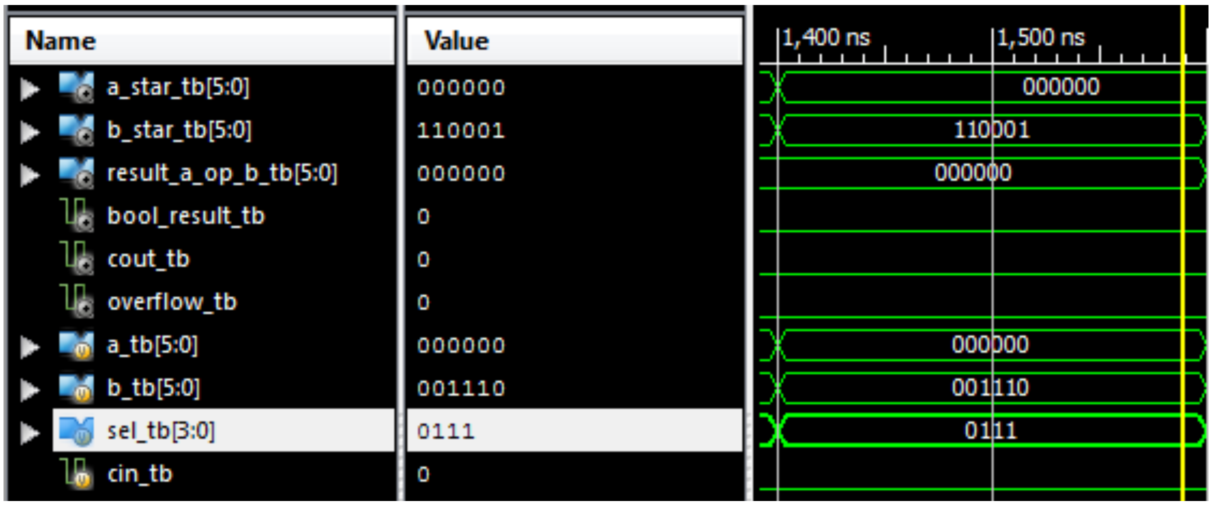
The expected output for the SEL = 0110 was 110001 or the value of the input A shifted right by 3 bits. The value of the output A_star is 110001 as expected.



Name	Value
a_star_tb[5:0]	110001
b_star_tb[5:0]	000000
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	001110
b_tb[5:0]	000000
sel_tb[3:0]	0110
cin_tb	0

0111:

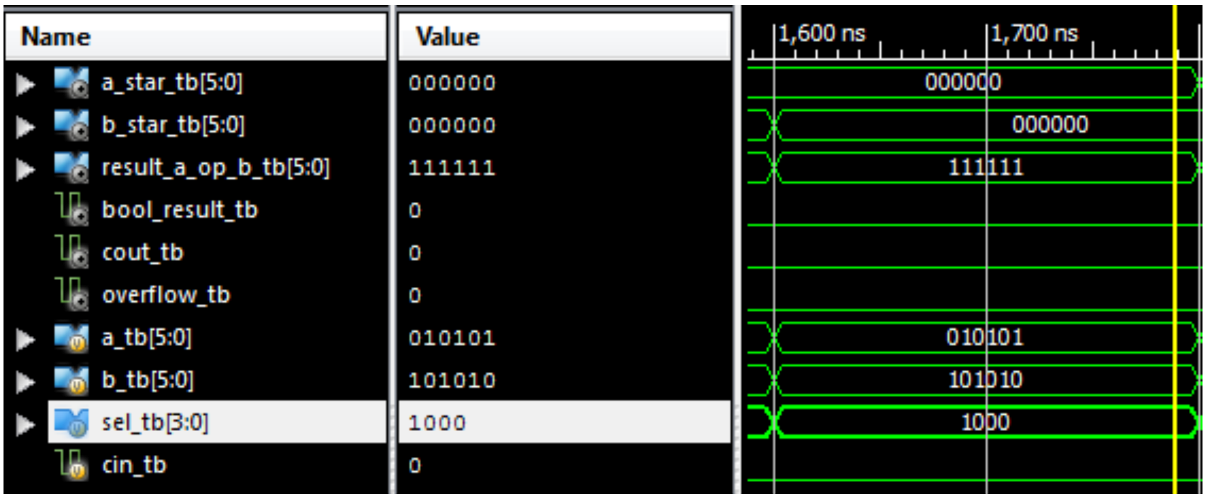
The expected output for the SEL = 0111 was 110001 or the value of the input B shifted right by 3 bits. The value of the output B_star is 110001 as expected.



```
Name                    Value
a_star_tb[5:0]          000000
b_star_tb[5:0]          110001
result_a_op_b_tb[5:0]   000000
bool_result_tb          0
cout_tb                 0
overflow_tb             0
a_tb[5:0]               000000
b_tb[5:0]               001110
sel_tb[3:0]             0111
cin_tb                  0
```

1000:

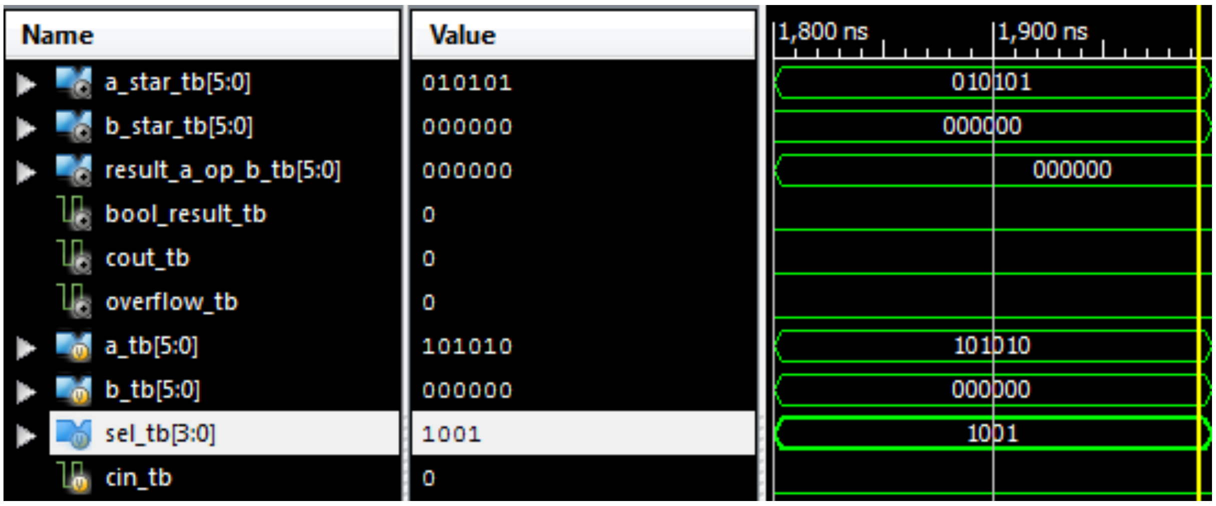
The expected result for the SEL = 1000 was 111111 since the inputs A and B have been XOR-ed. The result_a_op_b is 111111 as expected



```
Name                    Value
a_star_tb[5:0]          000000
b_star_tb[5:0]          000000
result_a_op_b_tb[5:0]   111111
bool_result_tb          0
cout_tb                 0
overflow_tb             0
a_tb[5:0]               010101
b_tb[5:0]               101010
sel_tb[3:0]             1000
cin_tb                  0
```

1001:

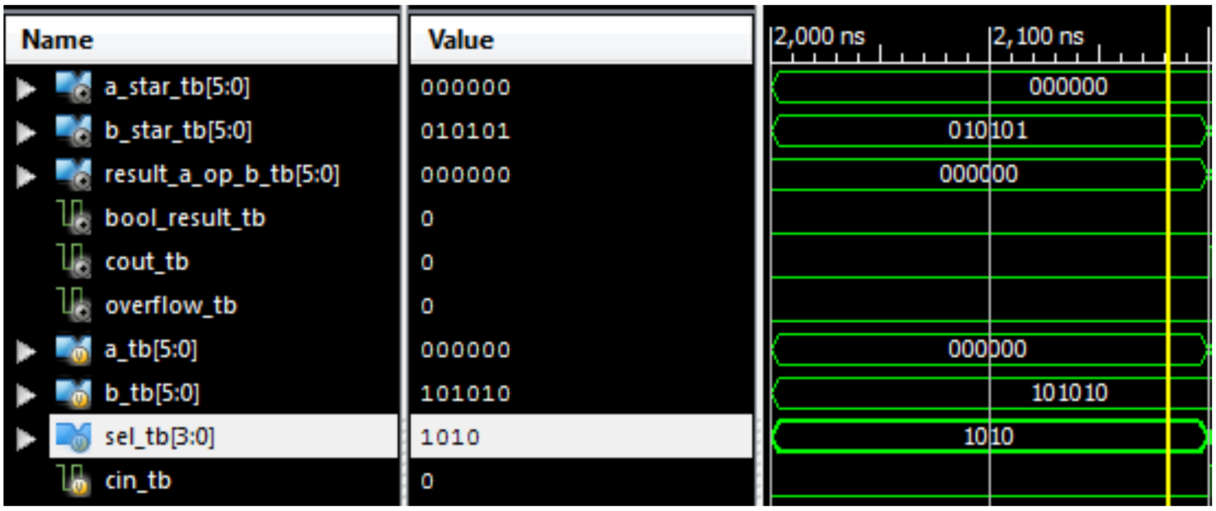
The expected output for the SEL = 1001 was 010101 which represent the bitvise inverted version of the input A. The output A_star is 010101 as expected.



Name	Value
a_star_tb[5:0]	010101
b_star_tb[5:0]	000000
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	101010
b_tb[5:0]	000000
sel_tb[3:0]	1001
cin_tb	0

1010:

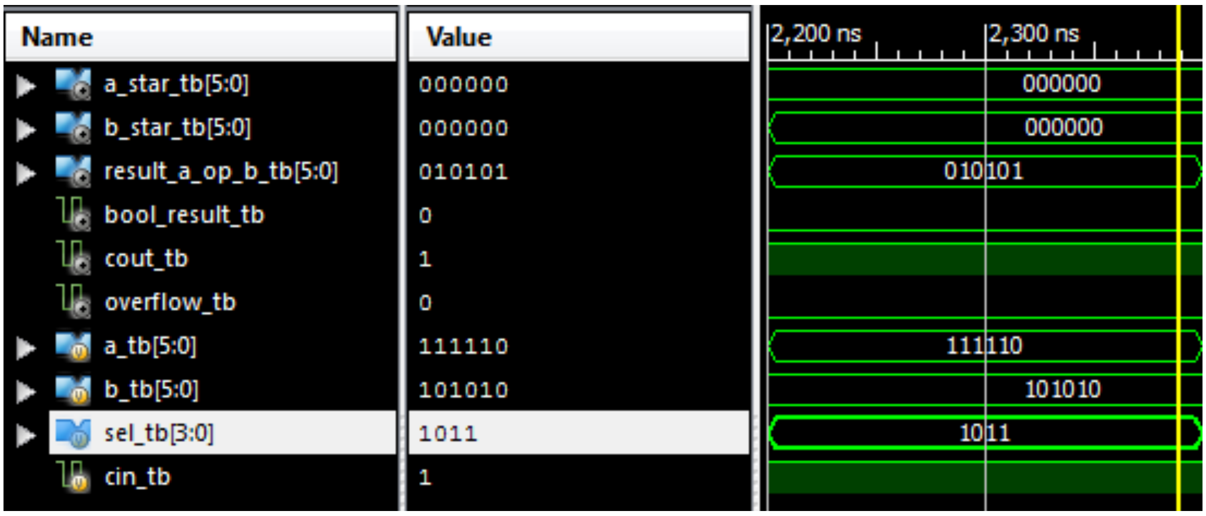
The expected output for the SEL = 1010 was 010101 which represent the bitvise inverted version of the input B. The output B_star is 010101 as expected.



Name	Value
a_star_tb[5:0]	000000
b_star_tb[5:0]	010101
result_a_op_b_tb[5:0]	000000
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	000000
b_tb[5:0]	101010
sel_tb[3:0]	1010
cin_tb	0

1011:

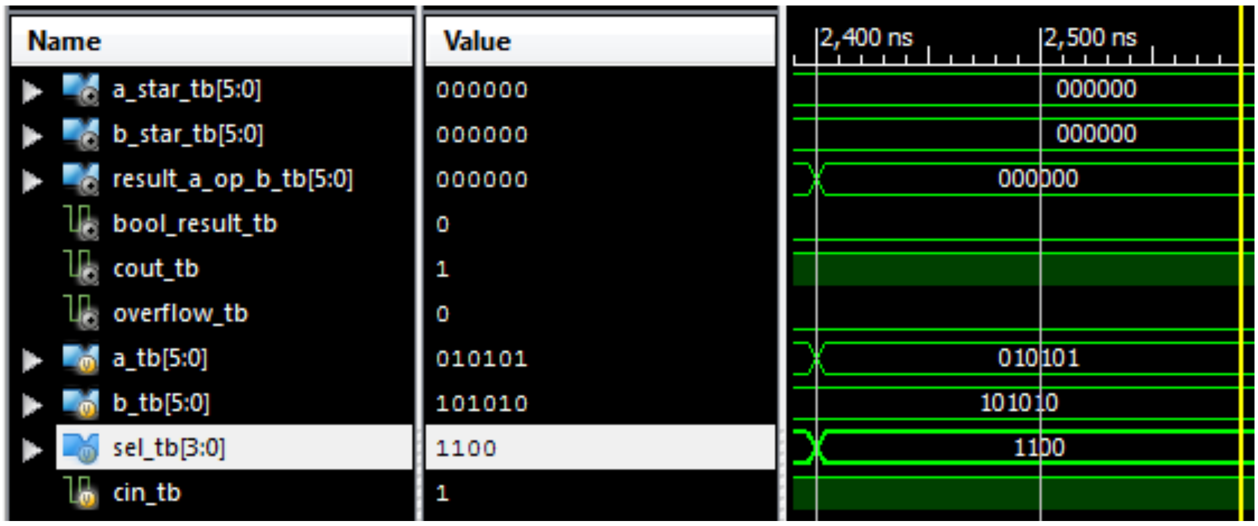
The expected result for the SEL = 1011 was 010101 representing the subtraction of the input A and B. the result_a_op_b is 010101 as expected.



```
Name                    Value
a_star_tb[5:0]          000000
b_star_tb[5:0]          000000
result_a_op_b_tb[5:0]   010101
bool_result_tb          0
cout_tb                 1
overflow_tb             0
a_tb[5:0]               111110
b_tb[5:0]               101010
sel_tb[3:0]             1011
cin_tb                  1
```

1100:

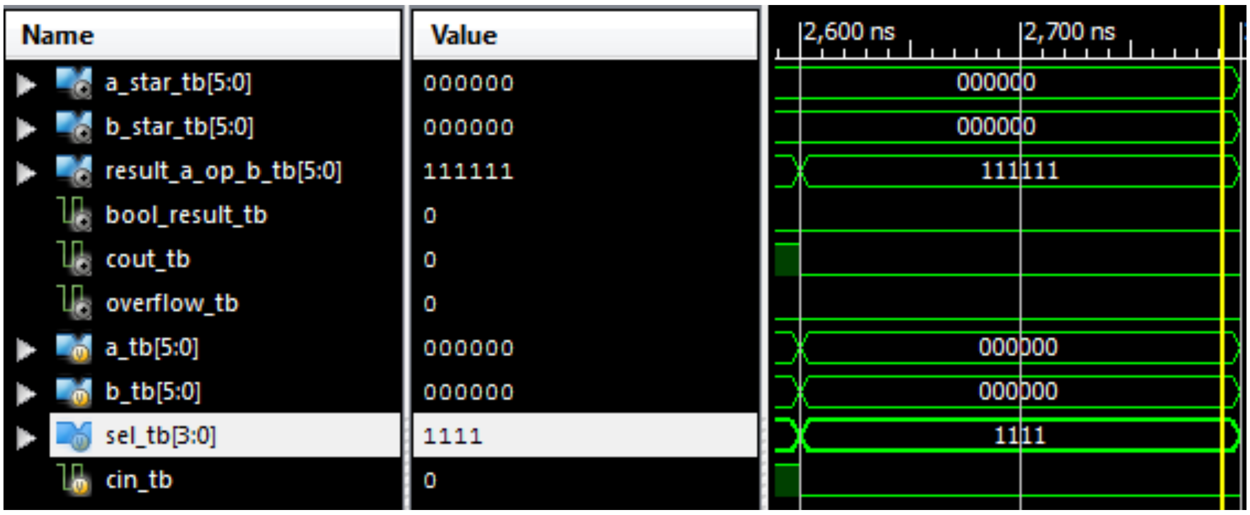
The expected result for the SEL = 1100 was 000000 representing the addition of the input A and B. the result_a_op_b is 000000 as expected. It is also worth noting the correct operation of module given that there was a carry in (cin_tb) as input and a carry out as (cou_tb) as outpu.



```
Name                    Value
a_star_tb[5:0]          000000
b_star_tb[5:0]          000000
result_a_op_b_tb[5:0]   000000
bool_result_tb          0
cout_tb                 1
overflow_tb             0
a_tb[5:0]               010101
b_tb[5:0]               101010
sel_tb[3:0]             1100
cin_tb                  1
```

1111:

The expected output for the SEL = 1111 was as string of 6 ones (111111) no matter what the values of inputs A or B were. The waveform below the result is 1111 as expected.



Name	Value
a_star_tb[5:0]	000000
b_star_tb[5:0]	000000
result_a_op_b_tb[5:0]	111111
bool_result_tb	0
cout_tb	0
overflow_tb	0
a_tb[5:0]	000000
b_tb[5:0]	000000
sel_tb[3:0]	1111
cin_tb	0

Conclusion

The module executed as expected all the operations for all the test vectors and showed stability to unexpected inputs.

The exercise was a very useful way to practice Verilog coding and also to deeper student's understanding of how digital circuits are designed, implemented and tested. The fact that it used other modules to form a more complex one was particularly important as it provided an opportunity for experiencing with a larger combinational design.

ALU module Code:

```
 `timescale 1ns / 1ps

module ALU

        (

                input wire [5:0] a, b,

                input wire [3:0] sel,

                input wire cin,

                output wire [5:0] a_star,b_star, result_a_op_b,

                output wire boolean_result, cout, overflow

        );


// local parameters needed for the case statement


localparam oooo =  4'b0000, oooi =  4'b0001, ooio =  4'b0010, ooii =  4'b0011, oioo =
4'b0100;

localparam oioi =  4'b0101, oiio =  4'b0110, oiii =  4'b0111, iooo =  4'b1000, iooi =  4'b1001;

localparam ioio =  4'b1010, ioii =  4'b1011, iioo =  4'b1100, iiii =  4'b1111, zero = 6'b000000,
one = 6'b111111;

// local wires

wire [5:0] minus_a, minus_b, a_3bit_right, b_3bit_right, not_a, not_b, a_xor_b, a_plus_b,
a_minus_b;

wire a_gteq_b, cout_plus,cout_minus, overflow_plus, overflow_minus;

reg [5:0] a_reg, b_reg, result_reg, bool_res_reg;

reg cout_reg,  overflow_reg;

// instantiating the required modules

        // module that returns - the given value

        minus_a_b Minus_a (.a_or_b (a), .minus_a_or_b ( minus_a));

        minus_a_b Minus_b (.a_or_b (b), .minus_a_or_b ( minus_b));

        // module that shifts right by 3 bits

        barrel_shifter_3bits A_3bit_right (. data (a), .result (a_3bit_right));
	barrel_shifter_3bits B_3bit_right (. data (b), .result (b_3bit_right));
	// inverter module
	inverter A_not ( . ininvert (a), . outinvert (not_a));
	inverter B_not ( . ininvert (b), . outinvert (not_b));
	// module that checks if A is greater or equal to B
	six_bit_gt_eq A_gteq_B ( . a (a), .  b( b), . agteqb (a_gteq_b) );
	// module that calculates bitwise XOR between A and B
	bitwise6_xor A_xor_B(. x (a), . y (b), . result (a_xor_b) );


	// module that calculates A+B
	bit6_ripple_adder A_plus_B ( . x (a), . y (b), . cin_6 (cin), . result(a_plus_b), . cout_6
(cout_plus), . overflow (overflow_plus));
	// module that calculates A-B
	a_minus_b A_minus_B  ( . a (a), . b (b) , . cin (cin), . cout (cout_minus), . overflow
(overflow_minus), . aminusb (a_minus_b));


//logic case statment that parses to the correct output in function of the given "sel" value
always @*
begin
	//default values
	a_reg = zero;
	b_reg = zero;
	result_reg = zero;
	bool_res_reg = 1'b0;
	cout_reg = 1'b0;
	overflow_reg = 1'b0;
	case (sel)
	oooo:
		begin
			result_reg = zero;
```

```
          end
iiii:
          begin
                    result_reg = one ;
          end
oooi:
          begin
                    a_reg = a;
          end


oioo:
          begin
                    a_reg = minus_a;
          end
oiio:
          begin
                    a_reg = a_3bit_right;
          end
iooi:
          begin
                    a_reg = not_a;
          end
ooio:
          begin
                    b_reg = b;
          end
oioi:
          begin
                    b_reg = minus_b;
```

```
        end
oiii:
        begin
                b_reg = b_3bit_right;
        end
ioio:
        begin
                b_reg = not_b;
        end
ooii:
        begin
                bool_res_reg = a_gteq_b;
        end
iooo:
        begin
                result_reg = a_xor_b;
        end
iioo:
        begin
                result_reg = a_plus_b;
                cout_reg = cout_plus;
                overflow_reg = overflow_plus;
        end
ioii:
        begin
                result_reg = a_minus_b;
                cout_reg = cout_minus;
                overflow_reg = overflow_minus;
        end
```

```
          default:
                    result_reg = 5'b11111;
          endcase
          end
assign a_star= a_reg;
assign b_star = b_reg;
assign result_a_op_b = result_reg;
assign boolean_result = bool_res_reg;
assign cout = cout_reg;
assign overflow = overflow_reg;
endmodule
```

ALU_tb module code:

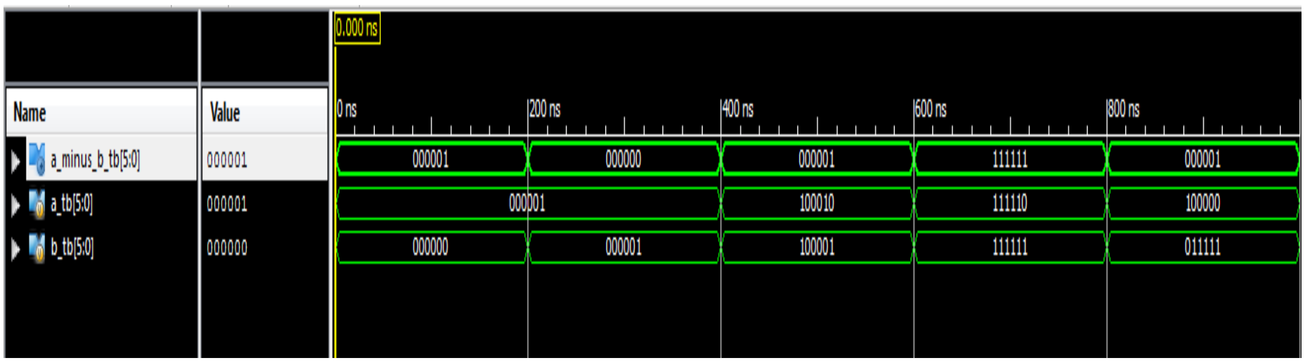
```
`timescale 1ns / 1ps

module ALU_tb();

                reg [5:0] a_tb,b_tb;

                reg [3:0]  sel_tb;

                reg cin_tb;

                wire [5:0]  a_star_tb, b_star_tb, result_a_op_b_tb;

                wire bool_result_tb, cout_tb, overflow_tb;

        ALU uut (.a (a_tb), .b (b_tb), .sel (sel_tb), . cin (cin_tb), . a_star ( a_star_tb), . b_star
(b_star_tb),

        . result_a_op_b (result_a_op_b_tb), .boolean_result (bool_result_tb), . cout
(cout_tb), . overflow (overflow_tb));

        initial

        begin

                // test vector 1 should display result_op_b_tb = 000000

                a_tb = 6'b111111;

                b_tb = 6'b111111;

                sel_tb = 4'b0000;

                cin_tb = 1'b0;

                #200

                // test vector 2 should display a_star = 101010

                a_tb = 6'b101010;

                b_tb = 6'b000000;

                sel_tb = 4'b0001;

                cin_tb = 1'b0;

                #200

                // test vector 3 should display b_star = 101010

                a_tb = 6'b000000;

                b_tb = 6'b101010;

                sel_tb = 4'b0010;
```

```
cin_tb = 1'b0;

#200

// test vector 4 should display boolean_result = 1

a_tb = 6'b101011;

b_tb = 6'b101010;

sel_tb = 4'b0011;

cin_tb = 1'b0;

#200

// test vector 5 should display a_star = 0000001

a_tb = 6'b111111;

b_tb = 6'b000000;

sel_tb = 4'b0100;

cin_tb = 1'b0;

#200

// test vector 6 should display b_star = 0000001

a_tb = 6'b000000;

b_tb = 6'b111111;

sel_tb = 4'b0101;

cin_tb = 1'b0;

#200

// test vector 7 should display  a-star = 110001

a_tb = 6'b001110;

b_tb = 6'b000000;

sel_tb = 4'b0110;

cin_tb = 1'b0;

#200

// test vector 8 should display  b_star = 110001

a_tb = 6'b000000;

b_tb = 6'b001110;
```

```
sel_tb = 4'b0111;

cin_tb = 1'b0;

#200

// test vector 9 should display result_op_b_tb = 111111

a_tb = 6'b010101;

b_tb = 6'b101010;

sel_tb = 4'b1000;

cin_tb = 1'b0;

#200

// test vector 10 should display  a_star = 010101

a_tb = 6'b101010;

b_tb = 6'b000000;

sel_tb = 4'b1001;

cin_tb = 1'b0;

#200

// test vector 11 should display b_star = 010101

a_tb = 6'b000000;

b_tb = 6'b101010;

sel_tb = 4'b1010;

cin_tb = 1'b0;

#200

// test vector 12 should display result_op_b_tb = 010101, with cout_tb =1

a_tb = 6'b111110;

b_tb = 6'b101010;

sel_tb = 4'b1011;

cin_tb = 1'b1;

#200

// test vector 13 should display result_op_b_tb = 000000 and cout = 1

a_tb = 6'b010101;
```

```
                b_tb = 6'b101010;

                sel_tb = 4'b1100;

                cin_tb = 1'b1;

                #200

                // test vector 14 should display result_op_b_tb = 111111

                a_tb = 6'b000000;

                b_tb = 6'b000000;

                cin_tb = 1'b0;

                sel_tb = 4'b1111;

                #200


                $stop;


        end


        initial

                $monitor($stime,, a_tb,, b_tb,, sel_tb,, cin_tb,,, a_star_tb,,, b_star_tb,,,
result_a_op_b_tb,,, cout_tb ,,, overflow_tb);

endmodule
```

A_minus_b module waveform



A_minus_b module code:

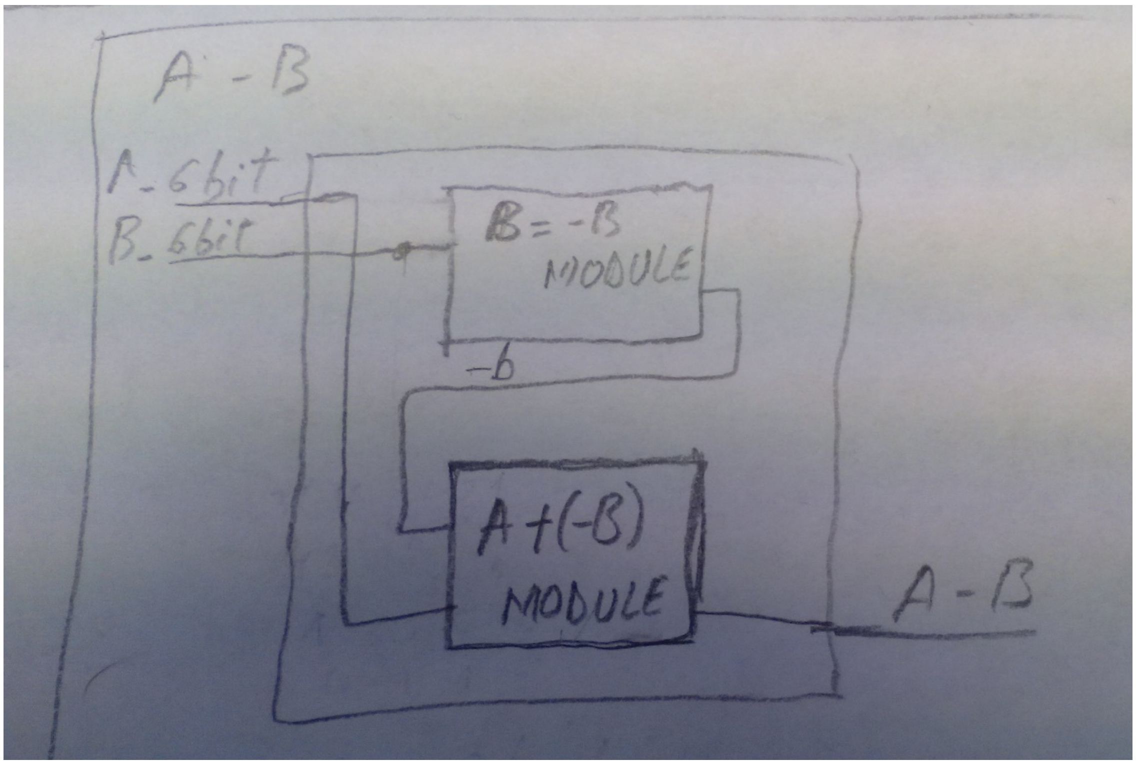
```
`timescale 1ns / 1ps
module minus_a_b
        (
                input wire [5:0] a_or_b,
                output wire [5:0] minus_a_or_b, a
        );
        // internal signal declaration
                reg one = 6'b000001, zero=1'b0;
                wire [5:0]x_inv,y;
                wire  cout, overflow,cin;
        assign y = one;
        assign cin =zero;
        // instantiating inverter modulle and adder module
        inverter a_minus_a (.ininvert(a_or_b),.outinvert(x_inv));
        bit6_ripple_adder a_plus_one
(.x(x_inv),.y(y),.cin_6(cin),.result(minus_a_or_b),.cout_6(cout),.overflow(overflow));

endmodule
```

Code for the a_minus_b testbench module:

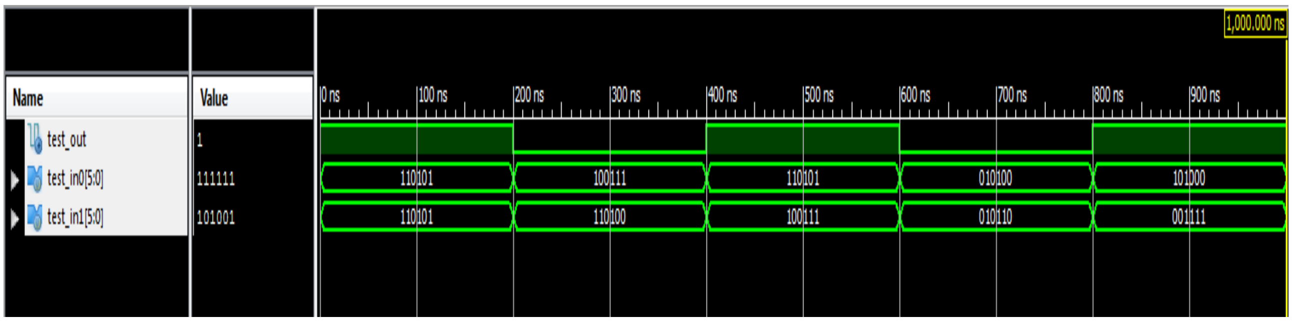
```
`timescale 1ns / 1ps

module a_minus_b_tb ();

// signal declaration

                    reg [5:0] a_tb, b_tb;

                    wire[5:0] a_minus_b_tb;

// instantiation of the unit under tess

  a_minus_b uut (. a (a_tb), . b (b_tb), . aminusb ( a_minus_b_tb));

//  test vector generator

initial

begin

          //  test vector 1

          a_tb = 6'b000001;

          b_tb = 6'b000000;

   #200;

          //  test vector 2

          a_tb = 6'b000001;

          b_tb = 6'b000001;

   #200;

          //  test vector 3

          a_tb = 6'b100010;

          b_tb = 6'b100001;

   #200;

          //  test vector 4

          a_tb = 6'b111110;

          b_tb = 6'b111111;

   #200;

          //  test vector 5

          a_tb = 6'b100000;
```

```
            b_tb = 6'b011111;
   #200;
// stop simulation
  $stop;
end initial
  $monitor($stime,,  a_tb,,  b_tb ,,, a_minus_b_tb);
Endmodule
```

Block Diagram for the Six_bit_gt_eq module:



Six_bit_gt_eq module waveform



Code for the Six_bit_gt_eq module:

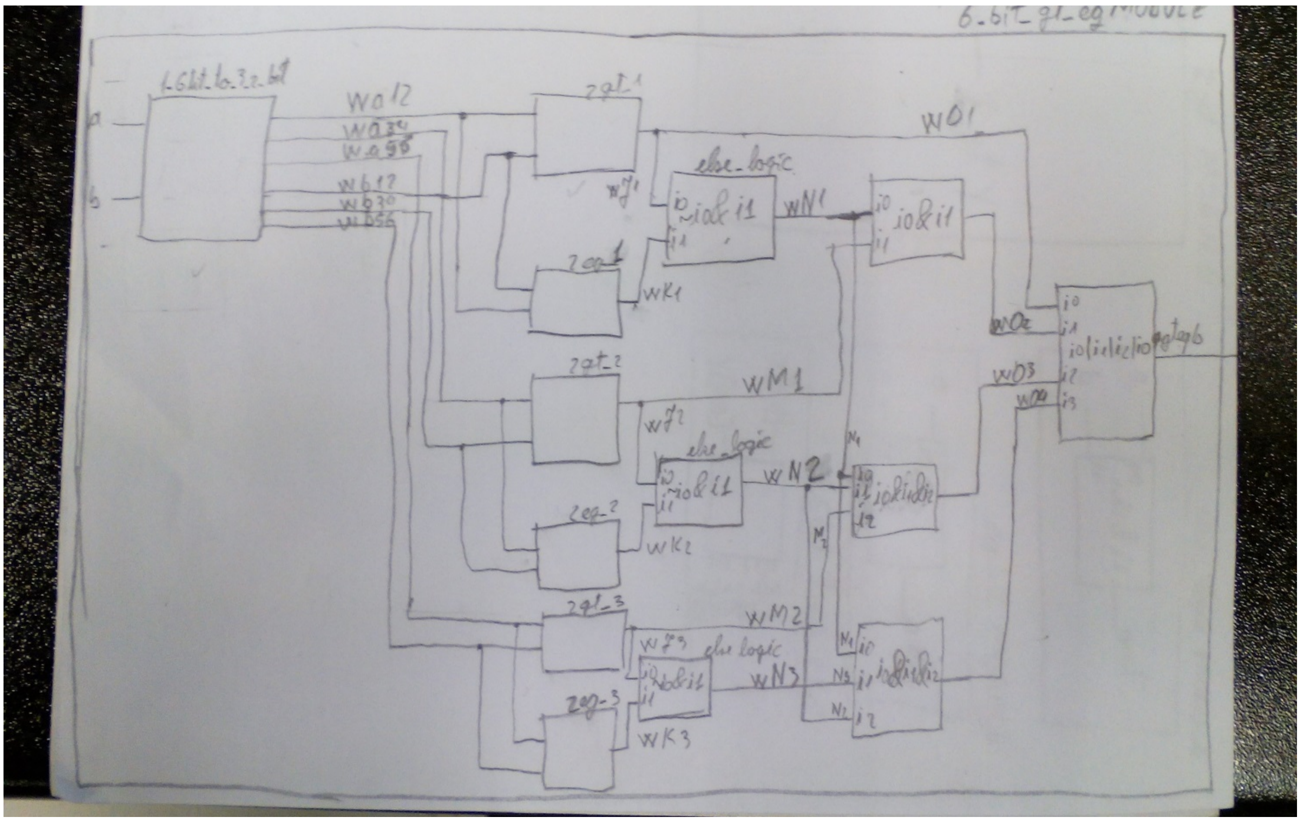
```
module six_bit_gt_eq

    (

    input wire [5:0] a, b,

    output wire agteqb

    );

    // designation of internal wires used to connect the input and outputs of the module

    wire k1, k2, k3 ,n1 ,n2, n3, m1 ,m2 , o1, o2 ,o3 ,o4;

            wire [1:0] a12, a34, a56 ,b12 ,b34, b56;

    // create an instance of the six_to _2 _bit module

    six_to_2_bit instance_1 (.a(a), .b(b), .a12(a12), .a34(a34), .a56(a56), .b12(b12), .b34(b34),
.b56(b56));

    // module code that checks if a 6 bit number is greater or eaqual to another 6 bit number

           //instantiate 3 2-bit greater-than comparators


           gt2 gt_1 (.a(a56), .b(b56), .agtb(o1)); // check the 2 most significant bits

           gt2 gt_2 (.a(a34), .b(b34), .agtb(m1)); // check the middle bits

           gt2 gt_3 (.a(a12), .b(b12), .agtb(m2)); // check the least significant bits
```

```
	//instantiate 3 2-bit equal-to comparators
	eq2 eq2_1 (.a(a56), .b(b56), .aeqb(k1)); // check the 2 most significant bits
	eq2 eq2_2 (.a(a34), .b(b34), .aeqb(k2)); // check the middle bits
	eq2 eq2_3 (.a(a12), .b(b12), .aeqb(k3)); // check the least significant bits

	//defining wires conforme the block diagram anexed

	assign n1 = ~o1 & k1; // coresponds to the else_logic block in case the most
	//significant 2 bits are not greater than but could still be equal
	assign n2 = ~m1 & k2; // coresponds to the else_logic block in case the middle
	//2 bits are not greater than but could still be equal
	assign n3 = ~m2 & k3; // coresponds to the else_logic block in case the most
	//least significant 2 bits are not greater than but could still be equal

	assign o2 = n1 & m1; //  considers the case
	//when the most significant 2 bits are equal but not grater than
	assign o3 = n1 & n2 & m2;// considers the case
	//when the most significant and the middle 2 bits are equal but not grater than
  assign o4 = n1 & n2 & n3;//  considers the case
	//when the most significant and the middle 2 bits are equal but not grater than and
	//checks to see if the last two bits are either greater than or equal

	assign agteqb = o1 | o2 | o3 | o4;

endmodule
```

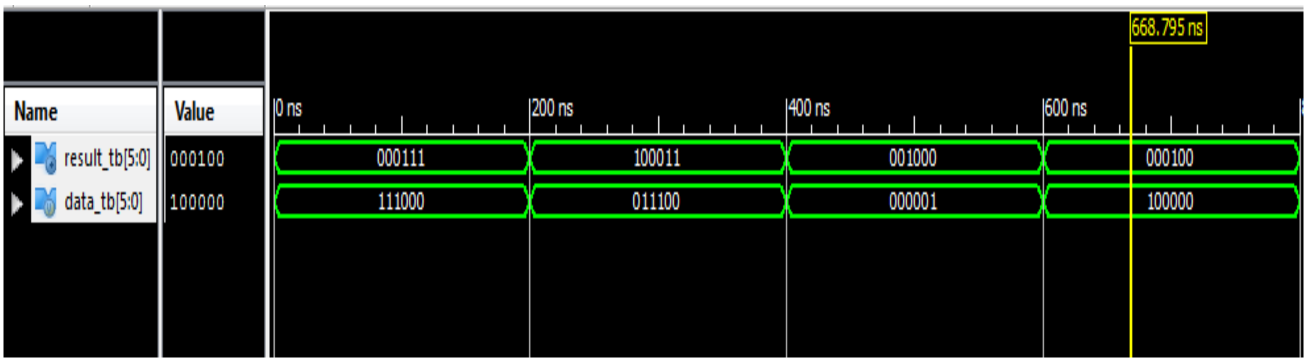
Code for the Six_bit_gt_eq_testbench module:

```
`timescale 1 ns/10 ps
module six_bit_gt_eq_testbench;
  // signal declaration
  reg  [5:0] test_in0, test_in1;
  wire  test_out;
  // instantiate the circuit under test
  six_bit_gt_eq uut  (.a(test_in0), .b(test_in1), .agteqb(test_out));
  //  test vector generator
  initial
  begin
    // test vector 1
    test_in0 = 6'b110101;
    test_in1 = 6'b110101;
    # 200;
    // test vector 2
    test_in0 = 6'b100111;
    test_in1 = 6'b110100;
    # 200;
    // test vector 3
    test_in0 = 6'b110101;
    test_in1 = 6'b100111;
    # 200;
    // test vector 4
    test_in0 = 6'b010100;
    test_in1 = 6'b010110;
    # 200;
    // test vector 5
    test_in0 = 6'b101000;

    test_in1 = 6'b001111;

    # 200;

    // test vector 6

    test_in0 = 6'b111111;

    test_in1 = 6'b101001;

    # 200;

    // stop simulation

    $stop;

  end

  initial

         $monitor($stime,, test_in0,, test_in1,,, test_out);

Endmodule
```

Block Diagram for the Six_bit_gt_eq module:



Barrel_shifter_3bits module waveform



```
result_tb[5:0]  000100   000111   100011   001000   000100
data_tb[5:0]    100000   111000   011100   000001   100000
```

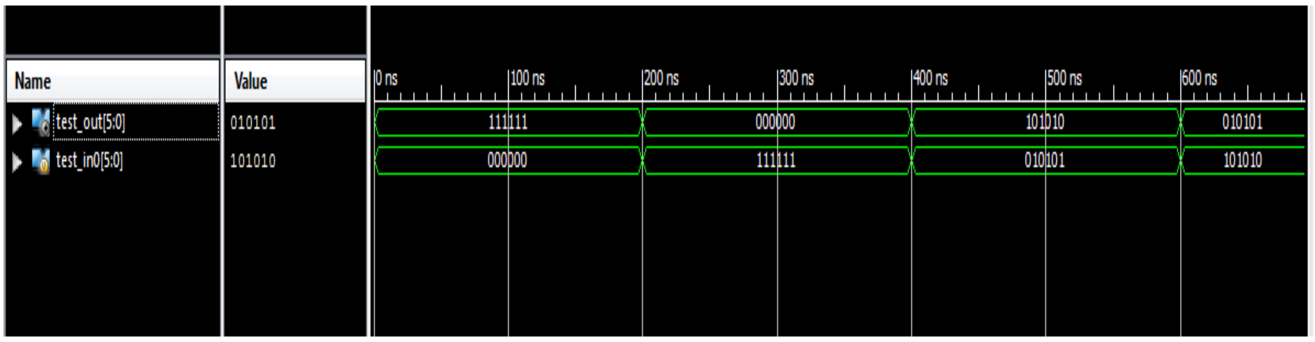
Code for the Barrel_shifter_3bits module:

```
`timescale 1ns / 1ps
module barrel_shifter_3bits
        (
        input  wire [5:0] data, //6-bit input
        output wire [5:0]  result // result after the shifting
        );
wire [11:0] data_in_double; // this will store two copies of the input
//wire [5:0] select; // amount to shift by; in this case it will be hardcode to 3

//Concatenate the input signal
assign data_in_double = {data,data};
//The same as signal[select + 63 : select] this is where the shifting happens altghough
// it is a pseudeo shifting since the result is obtained by reading 6 bits,
//over to the right by 3 from the concatenated inputs
assign result = data_in_double[3+5-:6];
endmodule
```

<u>Code for the Barrel_shifter_3bits_tb module:</u>

```
`timescale 1ns / 1ps
module barrel_shifter_3bits_tb( );
//signal declaration
        reg [5:0] data_tb;
        wire [5:0] result_tb;
//instantiate unit under test
        barrel_shifter_3bits uut ( .data(data_tb),.result(result_tb));
// test vectors for the simulation
initial
begin
         //  test vector 1 result should be 000111
         data_tb = 6'b111000;
  #200;
         //  test vector 2 result should be 100011
         data_tb = 6'b011100;
  #200;
         //  test vector 3 result should be 001000
         data_tb = 6'b000001;
  #200;
         //  test vector 4 result should be 000100
         data_tb = 6'b100000;
  #200;
// stop simulation
 $stop;
end initial
$monitor($stime,,data_tb ,,, result_tb);
Endmodule
```

Inverter testbench waveform



Code for the Inverter module:

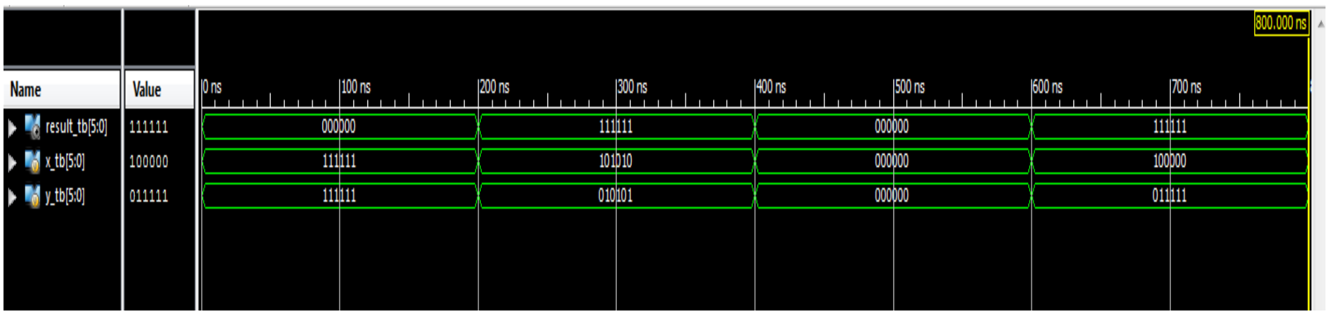
```
module inverter(ininvert,outinvert);
  // 3C7Assignment 1
  // in _invert is the number to be inverted
  input wire [5:0] ininvert;
  // out_invert is the inverted number inverted
  output [5:0] wire outinvert;
  // assignment statement
  assign outinvert = ~ininvert;
endmodule
```

Code for the Inverter module:

```
module inverter_testbench;
   // signal declaration
   reg  [5:0] test_in0 ;
   wire  [5:0] test_out;
   // instantiate the circuit under test
   inverter uut
      (.ininvert(test_in0), .outinvert(test_out));
   //  test vector generator
   initial
```

```
  begin
    // test vector 1
    test_in0 = 6'b000000;
    # 200;
    // test vector 2
    test_in0 = 6'b111111;
    # 200;
    // test vector 3
    test_in0 = 6'b010101;
    # 200;
    // test vector 4
    test_in0 = 6'b101010;
    # 200;
    $stop;
  end
endmodule
```

Bitwise6_xor module waveform:



Code for the Bitwise6_xor module:

```
`timescale 1ns / 1ps
module bitwise6_xor
        (
                input wire [5:0] x,y,
                output wire [5:0] result
```

```
            );
// internal signal declaration
wire b0,b1,b2,b3,b4,b5;

// instantiate
bitwise_xor bit0 (.bit_a0(x[0]),.bit_b0(y[0]),.bit_xor0(b0));
bitwise_xor bit1 (.bit_a0(x[1]),.bit_b0(y[1]),.bit_xor0(b1));
bitwise_xor bit2 (.bit_a0(x[2]),.bit_b0(y[2]),.bit_xor0(b2));
bitwise_xor bit3 (.bit_a0(x[3]),.bit_b0(y[3]),.bit_xor0(b3));
bitwise_xor bit4 (.bit_a0(x[4]),.bit_b0(y[4]),.bit_xor0(b4));
bitwise_xor bit5 (.bit_a0(x[5]),.bit_b0(y[5]),.bit_xor0(b5));

//concatenate the individual results to for the overall result
assign result={b0, b1, b2, b3, b4, b5};
endmodule
```

<u>Code for the Bitwise_xor module:</u>

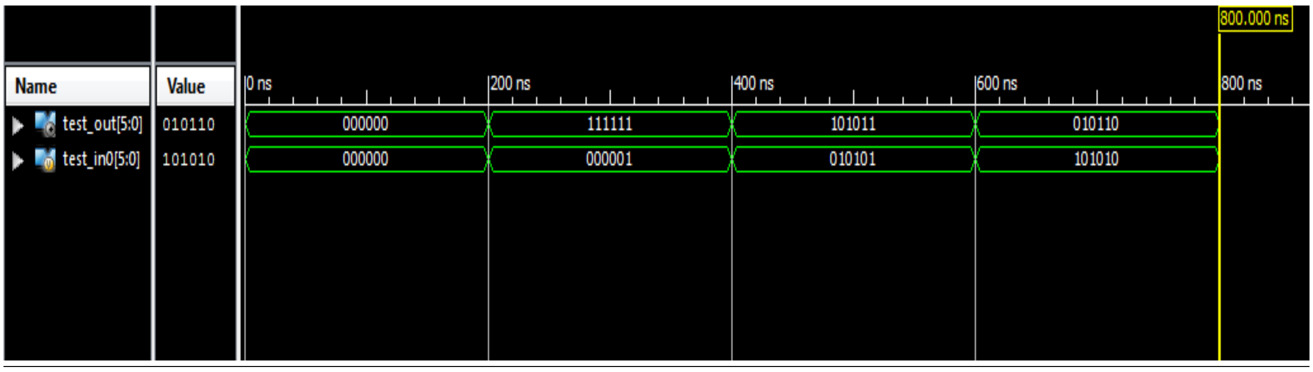
```
`timescale 1ns / 1ps
module bitwise_xor
           (
                    input wire bit_a0,bit_b0,
                    output wire bit_xor0
            );
  assign bit_xor0 = bit_a0 ^ bit_b0;
endmodule
```

<u>Code for the Bitwise6_xor_tb module:</u>

```
`timescale 1ns / 1ps
module bitwise6_xor_tb();
//signal declaration
        reg [5:0] x_tb,y_tb;
        wire [5:0] result_tb;
//instantiate unit under test
        bitwise6_xor uut ( .x(x_tb), .y(y_tb),.result(result_tb));
// test vectors for the simulation
initial
begin
        //  test vector 1 result should be zero
        x_tb = 6'b111111;
        y_tb = 6'b111111;
  #200;
        //  test vector 2 result should be one
        x_tb = 6'b101010;
        y_tb = 6'b010101;
  #200;
        //  test vector 3 all zeros no chage
        x_tb = 6'b000000;
        y_tb = 6'b000000;
  #200;
        //  test vector 4
        x_tb = 6'b100000;
        y_tb = 6'b011111;
  #200;
// stop simulation
 $stop;
```

```
end initial

$monitor($stime,,x_tb ,, y_tb,,, result_tb);

endmodule
```

Minus _a_b waveform:



Code for the Minus_a_b module:

```
`timescale 1ns / 1ps

module minus_a_b

        (

                input wire [5:0] a_or_b,

                output wire [5:0] minus_a_or_b, a

        );

        // internal signal declaration

                reg one = 6'b000001, zero=6'b000000;

                wire [5:0]x_inv,y,cin;

                wire  cout, overflow;

        assign y = one;

        assign cin =zero;

        // instantiating inverter modulle and adder module

        inverter a_minus_a (.ininvert(a_or_b),.outinvert(x_inv));

        bit6_ripple_adder a_plus_one
(.x(x_inv),.y(y),.cin_6(cin),.result(minus_a_or_b),.cout_6(cout),.overflow(overflow));

endmodule
```

Code for testbench for Minus_a_b:

```
`timescale 1ns / 1ps
module minus_a_b_tb;
          reg       [5:0] test_in0 ;
  wire  [5:0] test_out;
  // instantiate the circuit under test
  minus_a_b uut
    (.a_or_b(test_in0), .minus_a_or_b(test_out));
  //  test vector generator
  initial
  begin
    // test vector 1 (all zeros)
    test_in0 = 6'b000000;
    # 200;
    // test vector 2 (1)
    test_in0 = 6'b000001;
    # 200;
    // test vector 3 (21)
    test_in0 = 6'b010101;
# 200;
    // test vector 4 (-21)
    test_in0 = 6'b101010;
    # 200;
    $stop;
  end
endmodule
```