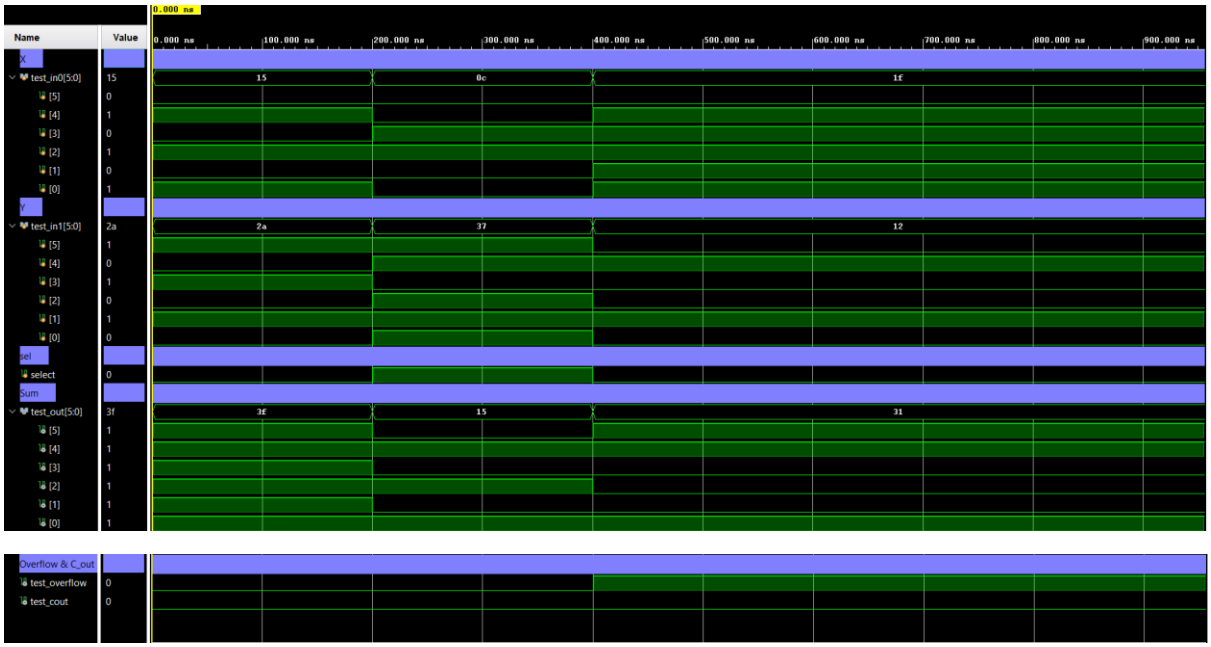
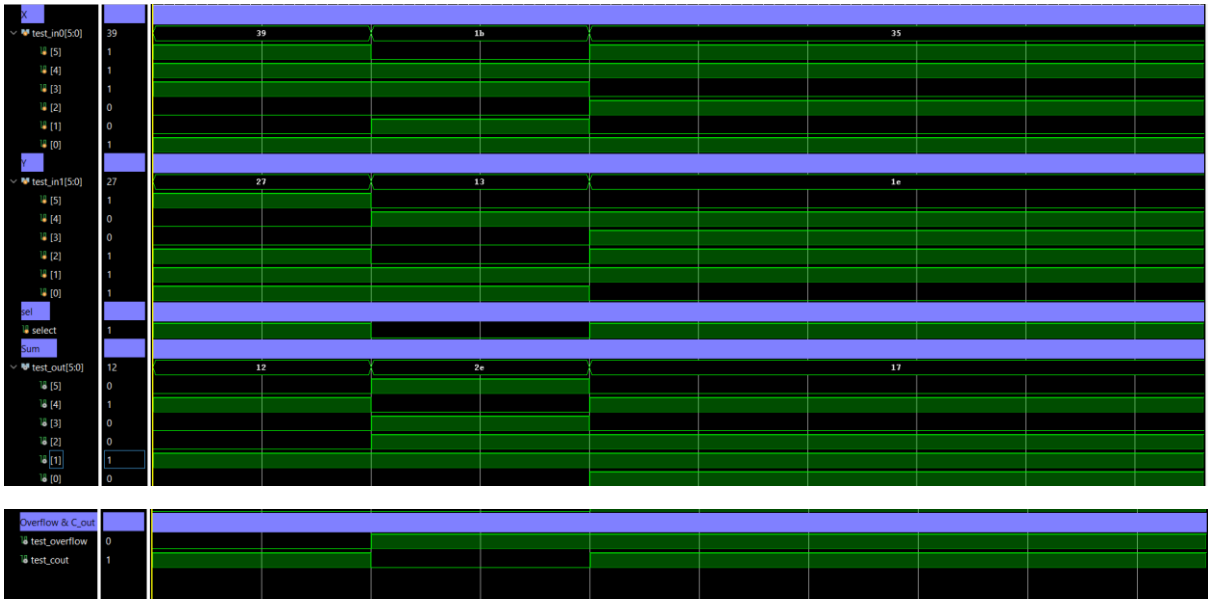
Test ID	x	y	sel	Cout (exp)	Overflow (exp)	Sum (exp)	Cout (obs)	Overflow (obs)	Sum (obs)	Pass/Fail
1	6'b010101	6'b101010	0	0	0	6'b111111	0	0	6'b111111	Pass
2	6'b001100	6'b110111	1	0	0	6'b010101	0	0	6'b010101	Pass
3*	-6'b100001	6'b010010	0	0	1	6'b110001	0	1	6'b110001	Pass
4	6'b111001	6'b100111	1	1	0	6'b010010	1	0	6'b010010	Pass
5*	6'b011011	-6'b101101	0	0	1	6’b101110	0	1	6’b101110	Pass
6	6'b110101	6’b011110	1	1	1	6’b010111	1	1	6’b010111	Pass
7	6’b111111	6’b111111	0	1	0	6’b111110	1	0	6’b111110	Pass
8	6’b000000	6’b000000	1	1	0	6’b000000	1	0	6’b000000	Pass
9	-6’b011010	-6’b101011	0	1	0	6’b111011	1	0	6’b111011	Pass

The circuit has passed all the tests as the observed output, overflow and sum values to what was expected. If I were to conduct this experiment again, I would increase the amount of test vectors with select as 1 in order to further verify the accuracy of the circuit when preforming 2’s complement subtraction.

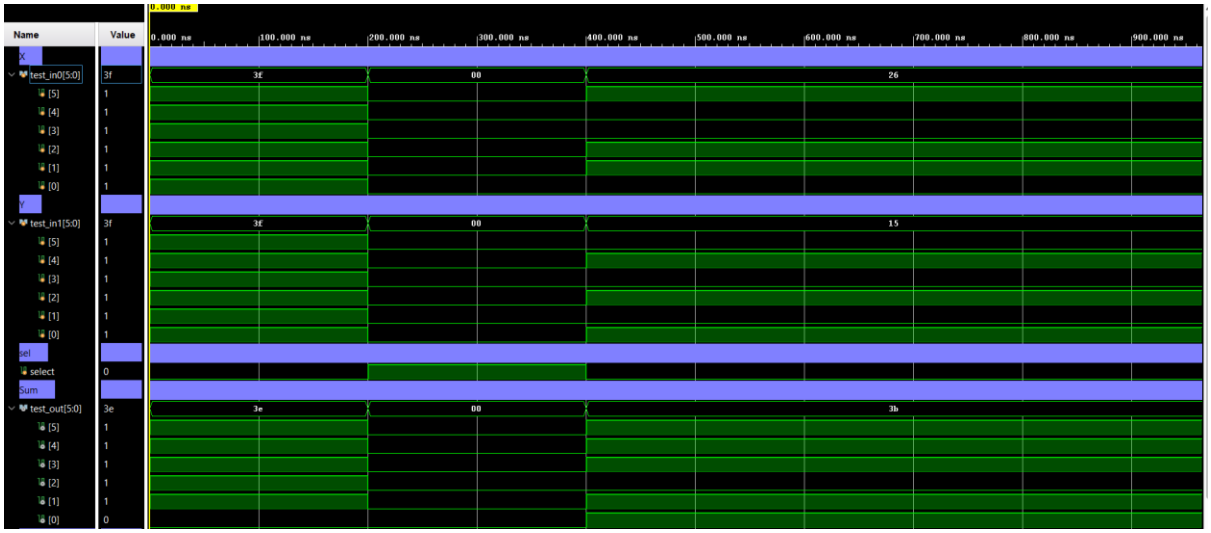
Testbench Waveform[1-3]

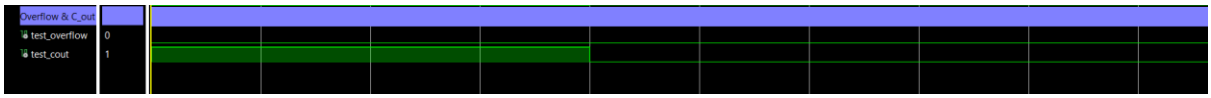


Testbench Waveform[3-6]

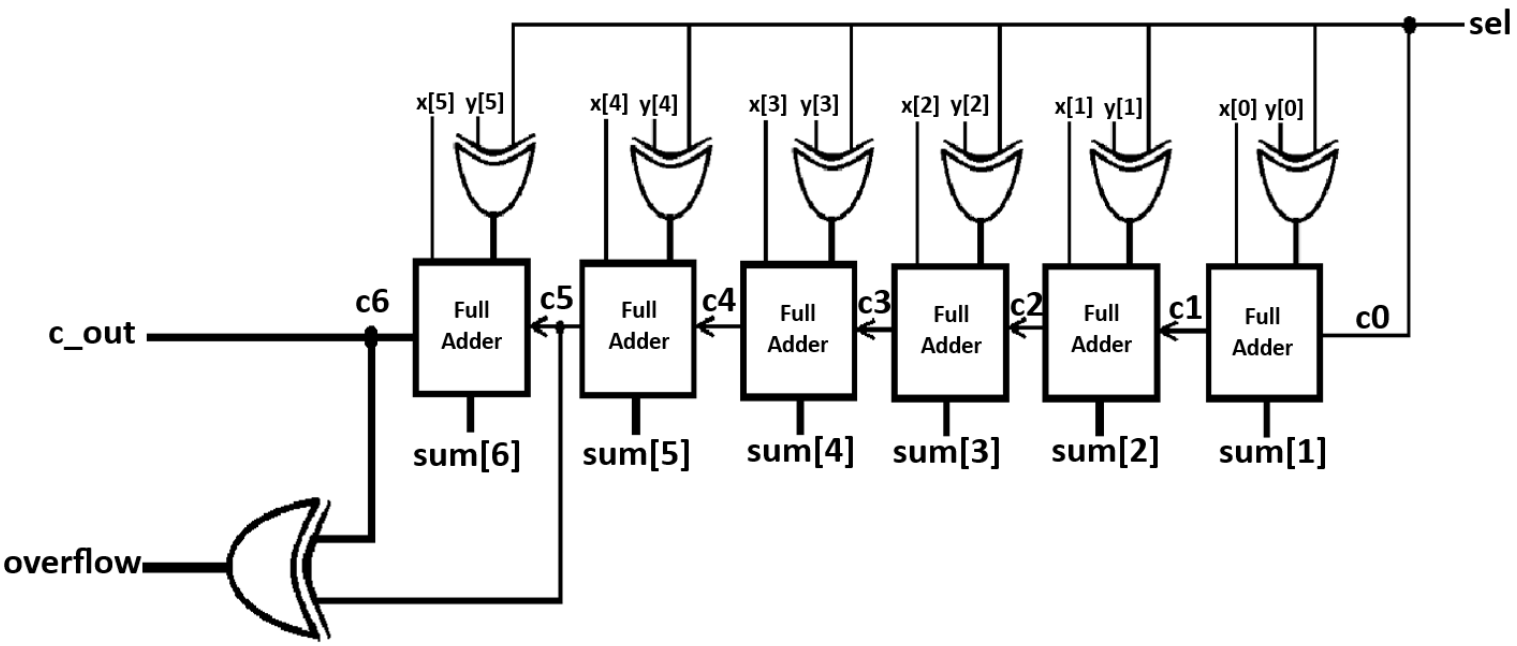


Testbench Waveform[6-9]





Block Diagram



full_adder.v (from Blackboard)

```
module FullAdder(a, b, cin, s, cout);
 // 3C7 LabD 2010
 // a and b are the bits to add
 // cin is carry in
 input wire a, b, cin;

 // s is the sum of a and b. cout is any carry out bit
 // wires since just using assign here
 output wire s, cout;

 // logic for sum and carry
 assign s = cin ^ a ^ b;
 assign cout = (b & cin) | (a & cin) | (a & b);

endmodule
```

6bit_ripple_adder.v

```
module sixbit_ripple_adder(
   input wire [5:0] x, y,              // 6-bit two's complement numbers to add
   input sel,                          // sel -> [0 = add] [1 = sub]
   output wire overflow,               // Output flagging overflow in the sum output
            [5:0] sum,                 // 2s complement sum of x and y
             c_out,                    // MSB Carry out from the sum
         wire c1, c2, c3, c4, c5, c6   // carry output for each full adder
   );
   assign overflow = c6 ^ c5 ;         //overflow = c6 xor c5
   assign c0 = sel;                    //initialise 1st output as the current value of sel
   FullAdder adder_1(.a(x[0]),.b(y[0] ^ sel), .cin(sel), .s(sum[0]), .cout(c1));   // 1st bit
   FullAdder adder_2(.a(x[1]),.b(y[1] ^ sel), .cin(c1), .s(sum[1]), .cout(c2));    // 2nd bit
   FullAdder adder_3(.a(x[2]),.b(y[2] ^ sel), .cin(c2), .s(sum[2]), .cout(c3));    // 3rd bit
   FullAdder adder_4(.a(x[3]),.b(y[3] ^ sel), .cin(c3), .s(sum[3]), .cout(c4));    // 4th bit
   FullAdder adder_5(.a(x[4]),.b(y[4] ^ sel), .cin(c4), .s(sum[4]), .cout(c5));    // 5th bit
   FullAdder adder_6(.a(x[5]),.b(y[5] ^ sel), .cin(c5), .s(sum[5]), .cout(c6));    // 6th bit
   assign c_out = c6;                  // declare final carry output as output of ripplr adder
endmodule
```

sbra_testbench.v (LabB - eq2_tb.v used as reference)

```
// Listing 1.7
// The `timescale directive specifies that
// the simulation time unit is 1 ns  and
// the simulation timestep is 10 ps
`timescale 1 ns/10 ps


module sbra_testbench;
  // signal declaration
  reg  [5:0] test_in0, test_in1;
  reg select;
  wire [5:0] test_out;
  wire test_overflow;
  wire test_cout;
  // instantiate the circuit under test
  sixbit_ripple_adder uut (.x(test_in0),.y(test_in1),.sum(test_out),.sel(select),.overflow(test_overflow),.c_out(test_cout));
```

```
//  test vector generator
initial
begin
  //test vector 1
  test_in0 = 6'b010101;
  test_in1 = 6'b101010;
  select = 0;
  #200;
  // test vector 2
  test_in0 = 6'b001100;
  test_in1 = 6'b110111;
  select = 1;
  #200;
  // test vector 3
  test_in0 = -6'b100001;
  test_in1 = 6'b010010;
  select = 0;
  #200;
  //test vector 4
  test_in0 = 6'b111001;
  test_in1 = 6'b100111;
  select = 1;
  #200;
  // test vector 5
  test_in0 = 6'b011011;
  test_in1 = -6'b101101;
  select = 0;
  #200;
  // test vector 6
  test_in0 = 6'b110101;
  test_in1 = 6'b011110;
  select = 1;
  #200;
  // test vector 7
  test_in0 = 6'b111111;
```

```
    test_in1 = 6'b111111;
    select = 0;
    #200;
    // test vector 8
    test_in0 = 6'b000000;
    test_in1 = 6'b000000;
    select = 1;
    #200;
    // test vector 9
    test_in0 = -6'b011010;
    test_in1 = -6'b101011;
    select = 0;
    #200;
    // stop simulation
    //$stop;
  end
endmodule
```