ALU DIGITAL

Presented to : Dr. Yasser Elawady

Ahmed EL Sokkary, Amr Khaled, Ali Ehab, Wafaa Sherif

Cancel Start

Progress

Planning

ModeLsim simulation of ALU

Logisim circuit design

Hardware ALU

Planning

Table 1 : " Arithmetic Operations "

Operation	Output
Multiplication	C = A * B
Add	C = A + B
INC	C = A + 1
Subtraction	C = A - B
Bigger than	C = 1, if A>B, else, C = 0
Smaller than	C = 1, if A<B, else, C = 0

Table 2 : " Logic Operations "

Operation	Output
OR	C = A | B (bit by bit)
NAND	C = (A & B)' (bit by bit)
XOR	C = A ^ B (bit by bit)
Bigger than	C = 1, if A>B, else, C = 0
Smaller than	C = 1, if A<B, else, C = 0
XNOR	C = (A ^ B)' (bit by bit)
AND	C = A & B (bit by bit)

Table 3 : " Shift Operations "

Operation	Output
Arithmetic Shift Left	C = A <<<1
Arithmetic Shift Right	C = A >>>1 (extending sign bit)
Logical Shift Right	C = A >> 1 (Shift right 1 bit by adding 0)

Modelsim ALU

ALU

```
//Alu MUX8x1
module ALU16(input logic [3:0] A, B,input logic [3:0] sel,output logic [7:0] R);

logic [7:0] n1;
logic [7:0] n2;
logic [7:0] n3;
logic [7:0] n4;
logic [7:0] n5;
logic [7:0] n6;
logic [7:0] n7;
logic [7:0] n8;
logic [7:0] n9;
logic [7:0] n10;
logic [7:0] n11;
logic [7:0] n12;
logic [7:0] n13;
logic [7:0] n14;




NAND inst1(A, B, n1);
XOR_Gate inst2(A, B, n2);
XNOR_GATE inst3(A, B, n3);
OR inst4(A, B, n4);
AndOperation inst5(A, B, n5);
Multiplecation inst6(A,B,n6);
INC inst7(A,n7);
LOgAR_SH_RL inst8(A,n8);
Bigger_Smaller inst9(A,B,n9);
Bigger_Smaller2 inst10 (A,B,n10);
SSHHIFT_LEFT inst11 (A,n11);
ADDER inst12 (A,B,n12);
right inst13 (A,n13);
```

```
SSHHIFT_LEFT inst11 (A,n11);
ADDER inst12 (A,B,n12);
right inst13 (A,n13);
SUBTRACTOR inst14 (A,n14);



always_comb
begin

case(sel)
4'b0000: R=n1; //NAND
4'b0001: R=n2;  //XOR
4'b0010: R=n3;     //XNOR
4'b0011: R=n4;     //OR
4'b0100: R=n5;   //and
4'b0101: R=n6;   //Multiplecation
4'b0110: R=n7;   //INC
4'b0111: R=n8;   //logic_right
4'b1000: R=n9;    //Bigger_Smaller
4'b1001: R=n10;     //Bigger_Smaller
4'b1010: R=n11;    //AR_SH_left
4'b1011: R=n12;         //ADDER
4'b1100: R=n13;       //Arethmatic right
4'b1101: R=n14;      //SUBTRACTOR
4'b1110: R=8'bxxxxxxxx;   //DONT CARE
4'b1111: R=8'bxxxxxxxx;    //DONT CARE
```

Multiplier

```
module AND2_gate(input logic A,B , output logic z);

assign z=A&B;
endmodule


module half_adder2(input logic A,B , output logic SUM,C);

assign SUM=A^B;
assign C=A&B;
endmodule



module full_adder(input logic A,B,Cin, output logic SUM ,C);

assign SUM=A^B^Cin;
assign C=A&B + Cin&(A^B);
endmodule



module Multiplecation( input logic [3:0]A,B , output logic [7:0]P );

logic n0;
logic t;
logic q;
logic n1;
logic r;
logic e;
logic i;
logic O;
logic m;
```

```
logic m1;
logic m2;
logic n6;
logic n7;


AND2_gate inst1(A[0], B[0] , n0 );
assign P[0]=n0;
AND2_gate inst2(A[1], B[0] , t);
AND2_gate inst3(A[0], B[1] , q);
half_adder2 inst4(t, q , n1 ,r);
assign P[1]=n1;
AND2_gate inst5(A[2], B[0] , e);
AND2_gate inst6(A[1], B[1] , i);
full_adder inst7(e , i , r , O ,m);
AND2_gate inst8(A[0], B[2] , d);
half_adder2 inst9(O, d , n2 ,l);
assign P[2]=n2;
AND2_gate inst10(A[3], B[0] , q1);
AND2_gate inst11(A[2], B[1] , q2);
full_adder inst12(q1 , q2 , m , z1 ,z2);
AND2_gate inst13(A[1], B[2] , z3);
full_adder inst14(z1 , z3 , l , d1 ,d2);
AND2_gate inst15(A[0], B[3] , d3);
half_adder2 inst16(d1, d3 , n3 ,v1);
assign P[3]=n3;
AND2_gate inst17(A[3], B[1] , f1);
half_adder2 inst18(f1, z2 , x1 ,a1);
AND2_gate inst19(A[2], B[2] , x2);
full_adder inst20(x1 , x2 , d2 , k1 ,k2);
AND2_gate inst21(A[1], B[3] , k3);
full_adder inst22(k1 , k3 , v1 , n4 ,h2);
assign P[4]=n4;
```

```
assign P[6]=n6;
assign P[7]=n7;
```

SUBTRACTOR

```
module AND (input logic a,b, output logic y);
assign y= a&b ;
endmodule

module XOR(input logic a,b, output logic y);
assign y= a^b ;
endmodule

module XOR3(input logic a,b,c, output logic y);
logic n1;
 XOR x(a,b,n1);
 XOR x2(n1,c,y);
endmodule

module OR (input logic a,b, output logic y);
assign y= a|b ;
endmodule



module full_adder_one_bit(input a,input b,input logic cin,output cout,output sum);
XOR3 x(a,b,cin,sum);
logic t1,t2,t3,t4,t5;
AND and1(a,b,t1);
XOR x2(a,b,t2);
AND and2(t2,cin,t4);
AND and3(a,b,t5);
OR fin(t4,t5,cout);
endmodule
```

```
module SUBTRACTOR(input [3:0] a, input [3:0] b, output [4:0] y);
logic [3:0] m;
NOT not0(b[0],m[0]);
NOT not1(b[1],m[1]);
NOT not2(b[2],m[2]);
NOT not3(b[3],m[3]);
logic n1;
logic n2;
logic n3;
logic n4;
full_adder_one_bit fa1(a[0],m[0],1,n1,y[0]);
full_adder_one_bit fa2(a[1],m[1],n1,n2,y[1]);
full_adder_one_bit fa3(a[2],m[2],n2,n3,y[2]);
full_adder_one_bit fa4(a[3],m[3],n3,n4,y[3]);
assign y[4]=n4;
endmodule
```

INC

```
module AND (input logic a,b, output logic y);
assign y= a&b ;
endmodule

module XOR(input logic a,b, output logic y);
assign y= a^b ;
endmodule

module XOR3(input logic a,b,c, output logic y);
logic n1;
 XOR x(a,b,n1);
 XOR x2(n1,c,y);
endmodule

module OR (input logic a,b, output logic y);
assign y= a|b ;
endmodule



module full_adder_one_bit(input a,input b,input logic cin,output cout,output sum);
XOR3 x(a,b,cin,sum);
logic t1,t2,t3,t4,t5;
AND and1(a,b,t1);
XOR x2(a,b,t2);
AND and2(t2,cin,t4);
AND and3(a,b,t5);
OR fin(t4,t5,cout);
endmodule
```

```
module INC(input logic[3:0] a,output logic[7:0] y);
logic n1;
logic n2;
logic n3;
logic n4;
full_adder_one_bit fa1(a[0],1,0,n1,y[0]);
full_adder_one_bit fa2(a[1],0,n1,n2,y[1]);
full_adder_one_bit fa3(a[2],0,n2,n3,y[2]);
full_adder_one_bit fa4(a[3],0,n3,n4,y[3]);
assign y[4]=n4;
assign y[5]=0;
assign y[6]=0;
assign y[7]=0;
endmodule
```

ADDER

```
module AND (input logic a,b, output logic y);
assign y= a&b ;
endmodule

module XOR(input logic a,b, output logic y);
assign y= a^b ;
endmodule

module XOR3(input logic a,b,c, output logic y);
logic n1;
 XOR x(a,b,n1);
 XOR x2(n1,c,y);
endmodule

module OR (input logic a,b, output logic y);
assign y= a|b ;
endmodule



module full_adder_one_bit(input a,input b,input logic cin,output cout,output sum);
XOR3 x(a,b,cin,sum);
logic t1,t2,t3,t4,t5;
AND and1(a,b,t1);
XOR x2(a,b,t2);
AND and2(t2,cin,t4);
AND and3(a,b,t5);
OR fin(t4,t5,cout);
endmodule

module ADDER(input [3:0] a,input [3:0]b,output [4:0] y);
logic n1=0;
logic n2;
logic n3;
logic n4;
full_adder_one_bit fa1(a[0],b[0],0,n1,y[0]);
full_adder_one_bit fa2(a[1],b[1],n1,n2,y[1]);
full_adder_one_bit fa3(a[2],b[2],n2,n3, y[2]);
full_adder_one_bit fa4(a[3],b[3],n3,n4,y[3]);
assign y[4]=n4;
endmodule
```

NAND

```
module AND (input logic [3:0]x,y, output logic [3:0]z);
assign z = x & y;
endmodule

module NOT (input logic [3:0]x, output logic [3:0]y);
assign y = ~x;
endmodule

module NAND (input logic [3:0]x,y, output logic [7:0]z);
logic[3:0] n2;
AND and2(x,y,n1);
NOT not1(n1,z);
assign z = n2;
endmodule
```

Bigger/ smaller than

```
always_comb
begin

if (A > B) Z=8'b11111111;
else if (A < B) Z=8'b00000000;
else Z=8'bxxxxxxxx;


end
endmodule


module tb_Bigger_Smaller2();

logic [3:0]A,B;
logic [7:0]Z;

Bigger_Smaller dut (A,B,Z);
initial begin
```

XOR

```
module XOR_Gate(input logic [3:0]A,B ,output logic [7:0]C );

assign C=A^B;
endmodule


module tb_XOR_Gate();

logic [3:0]A,B;
logic [7:0]C;

XOR_Gate dut (A,B,C);
initial begin
```

XNOR

```
module NOT_Gate(input logic [3:0]X , output logic [3:0]Y );

assign Y=~X;
endmodule

module XNOR_GATE(input logic [3:0]A,B ,output logic [7:0]C );

logic [7:0]q;
logic [7:0]n;


XOR_GATE inst1(A,B,q);
NOT_Gate inst2(q,n);
assign C=n;
endmodule
```

Arithmetic shift left

```
module MUltiplexerX41_SH(input logic A,L, input logic SEL ,output logic H);

always_comb
begin
case(SEL)
0: H=A;
1: H=L;


default: H=1'bx;
endcase
end
endmodule



module SSHHIFT_LEFT(input logic [0:3]A,  output logic [0:3]H);

MUltiplexerX41_SH inst1 (0,A[1],1,H[0]);
MUltiplexerX41_SH inst2 (A[0],A[2],1,H[1]);

MUltiplexerX41_SH inst3 (A[1],A[3],1,H[2]);
MUltiplexerX41_SH inst4 (A[2],0,1,H[3]);


endmodule
```

Arithmetic shift right

```
module two_and(input x, input y, output z);
assign z=x&y;
endmodule

module two_or(input x, input y, output z);
assign z=x|y;
endmodule

module two_mux(input x, input y, input select, output z);
logic f;
logic s;
two_and first(select, x,f);
two_and second(~select, y,s);
two_or fin(f,s,z);
endmodule

module right(input [3:0]x, output [7:0]z);
two_mux first(1,0,x[0], z[3]);
two_mux second(1,0,x[3], z[2]);
two_mux third(1,0,x[2], z[1]);
two_mux fourth(1,0,x[1], z[0]);
endmodule
```

Logic shift right

```
module LOgAR_SH_RL(input logic [0:3]A, output logic [0:3]H);

MUX41_SH inst1 (0,A[1],0,H[0]);
MUX41_SH inst2 (A[0],A[2],0,H[1]);

MUX41_SH inst3 (A[1],A[3],0,H[2]);
MUX41_SH inst4 (A[2],0,0,H[3]);


endmodule



module tb_LOgAR_SH_RL();

logic [3:0]A;
logic [3:0]H;

LOgAR_SH_RL dut(A,H);


initial begin
for (int i =0; i<16; i = i+1) begin

A = i;
#10;
end
end
endmodule
```

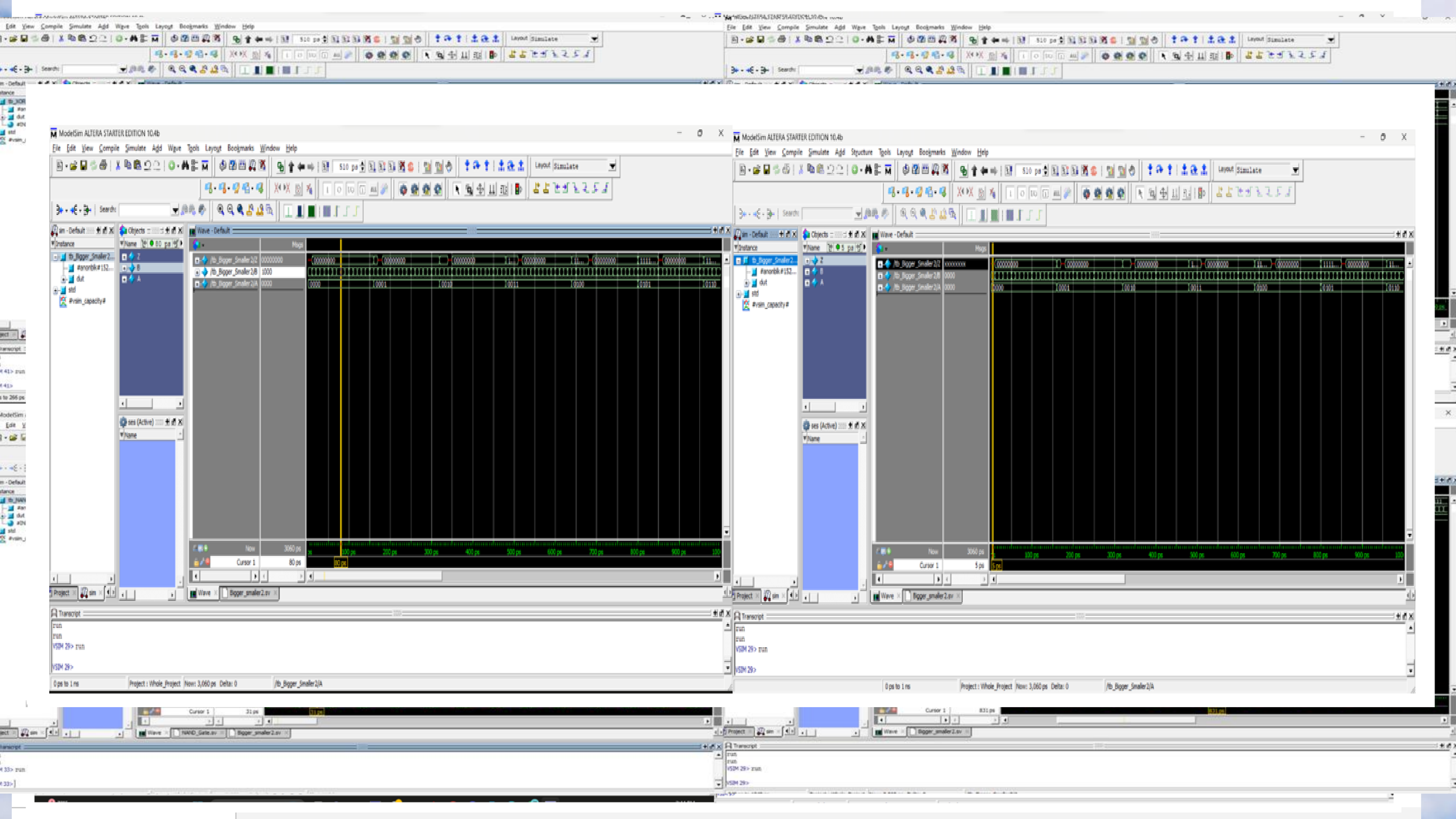
Test bench(simple)

```
module tb_LOgAR_SH_RL();

logic [3:0]A;
logic [3:0]H;

LOgAR_SH_RL dut(A,H);


initial begin
for (int i =0; i<16; i = i+1) begin

A = i;
#10;
end
end
endmodule
```

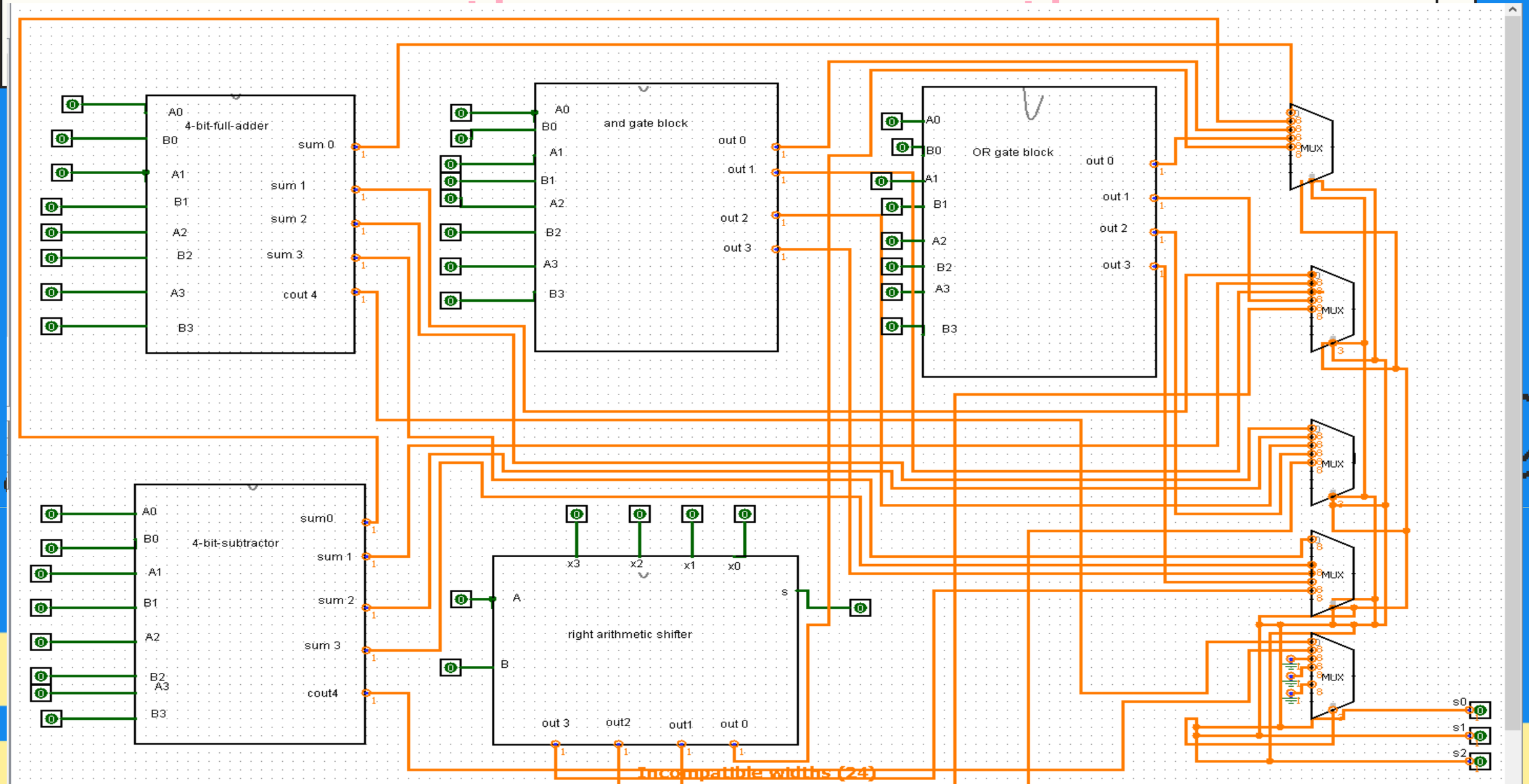
Test bench

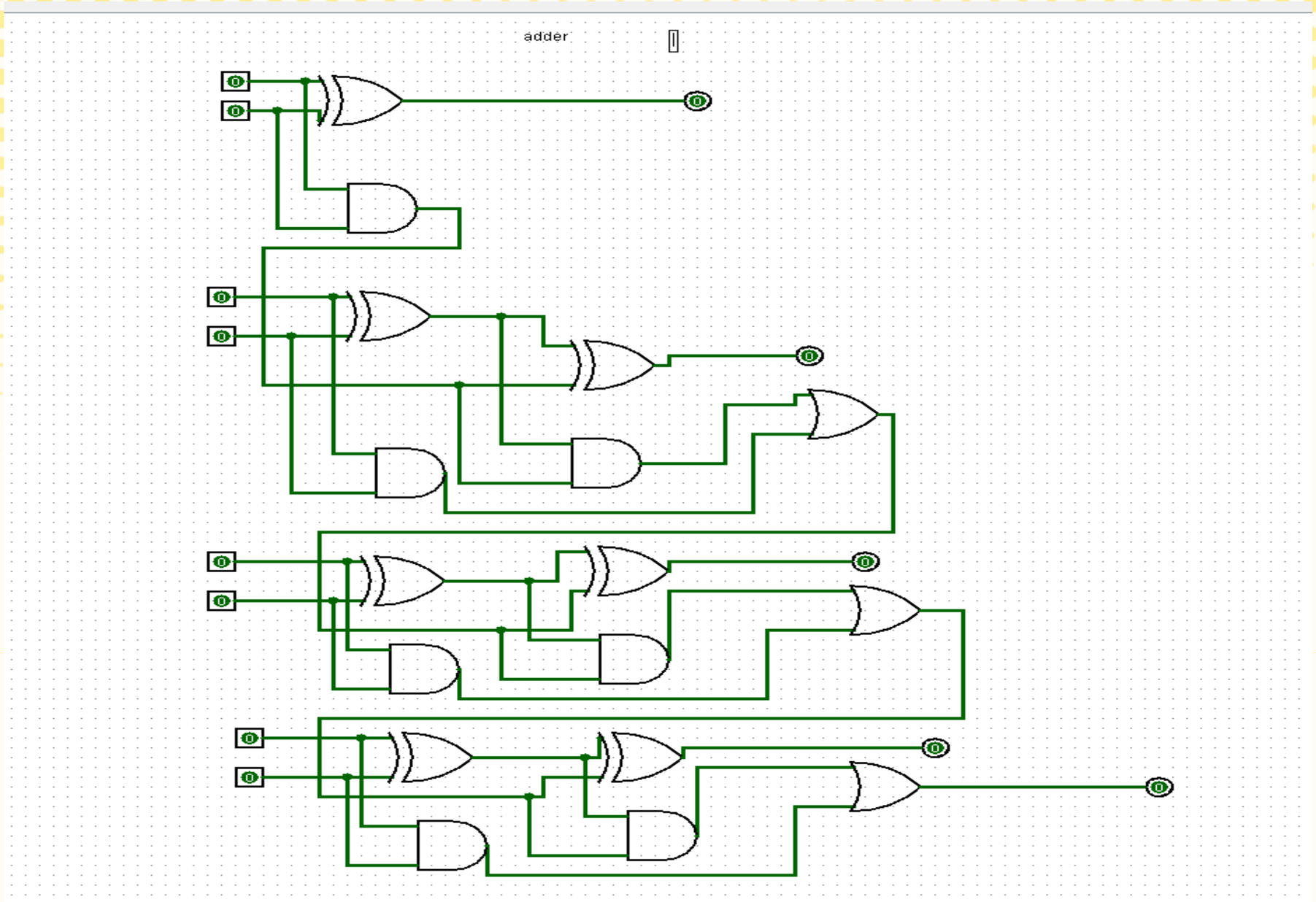
```
module tb_testvectors();

logic [3:0] A,B,sel;
logic [7:0]R;
logic [7:0] R_expected;


logic clk , reset;
logic [3:0] i,errors;
logic [6:0] testvecctors[39:0];



ALU16 dut (A,B,sel,R);


initial begin

$readmemb("Testbench_whole_project ",testvecctors);
i=0; errors=0; reset=1; #6; reset=0;
end

always begin
clk=1; #10; clk=0; #10;
end


always@(posedge clk) begin
{A,B,sel,R_expected} = testvecctors[i];
end


always@(negedge clk) begin
```

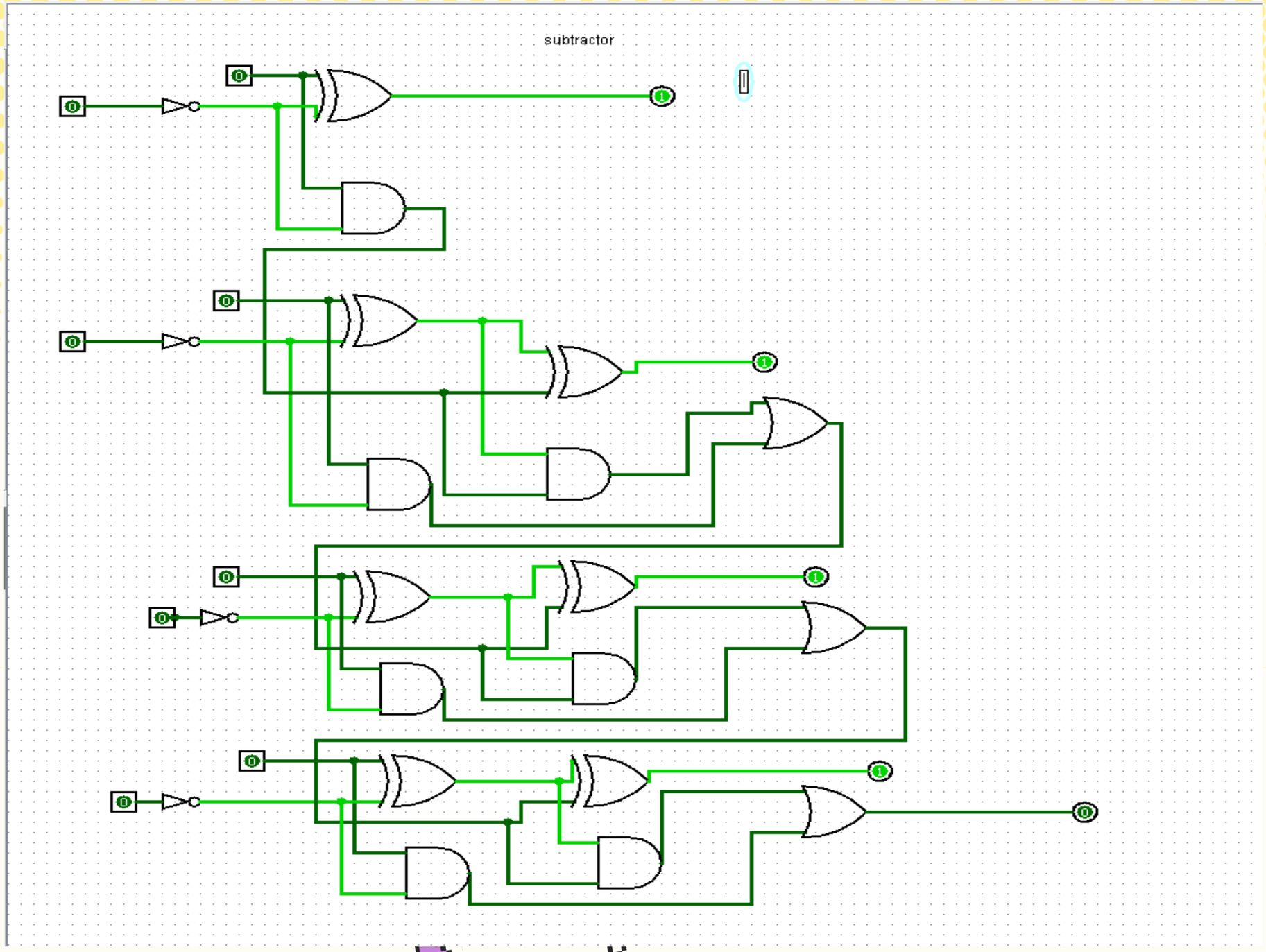
```
always@(negedge clk) begin

if (R!==R_expected) begin
$display ("Error testcase %b, %b ,%b  failed",A,B,sel);
$display ("output: %b, expected output: %b ",R,R_expected);
errors = errors +1;
end
i=i+1;
if (testvecctors[i]=== 4'bx) begin
$display ("%d number of test cases completed with %d errors" ,i,errors);
$stop;

end

end

endmodule
```

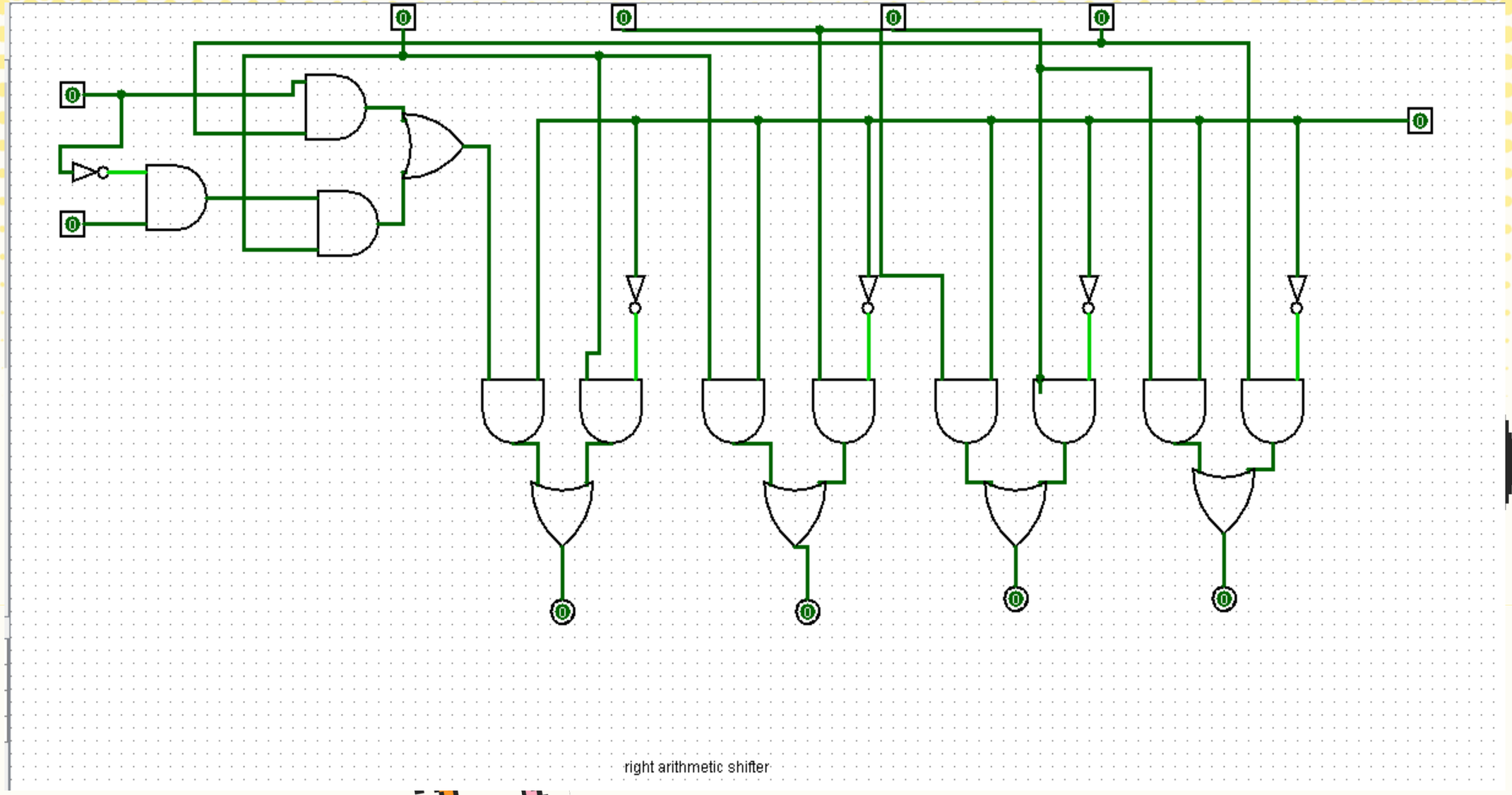

ModelSim ALTERA STARTER EDITION 10.4b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Transcript
run
VSIM 29> run
VSIM 29>
Cursor 1
80 ps
5 ps
Project : Whole_Project
Now: 3,060 ps Delta: 0
/tb_Bigger_Smaller2/A

Logisim circuit design

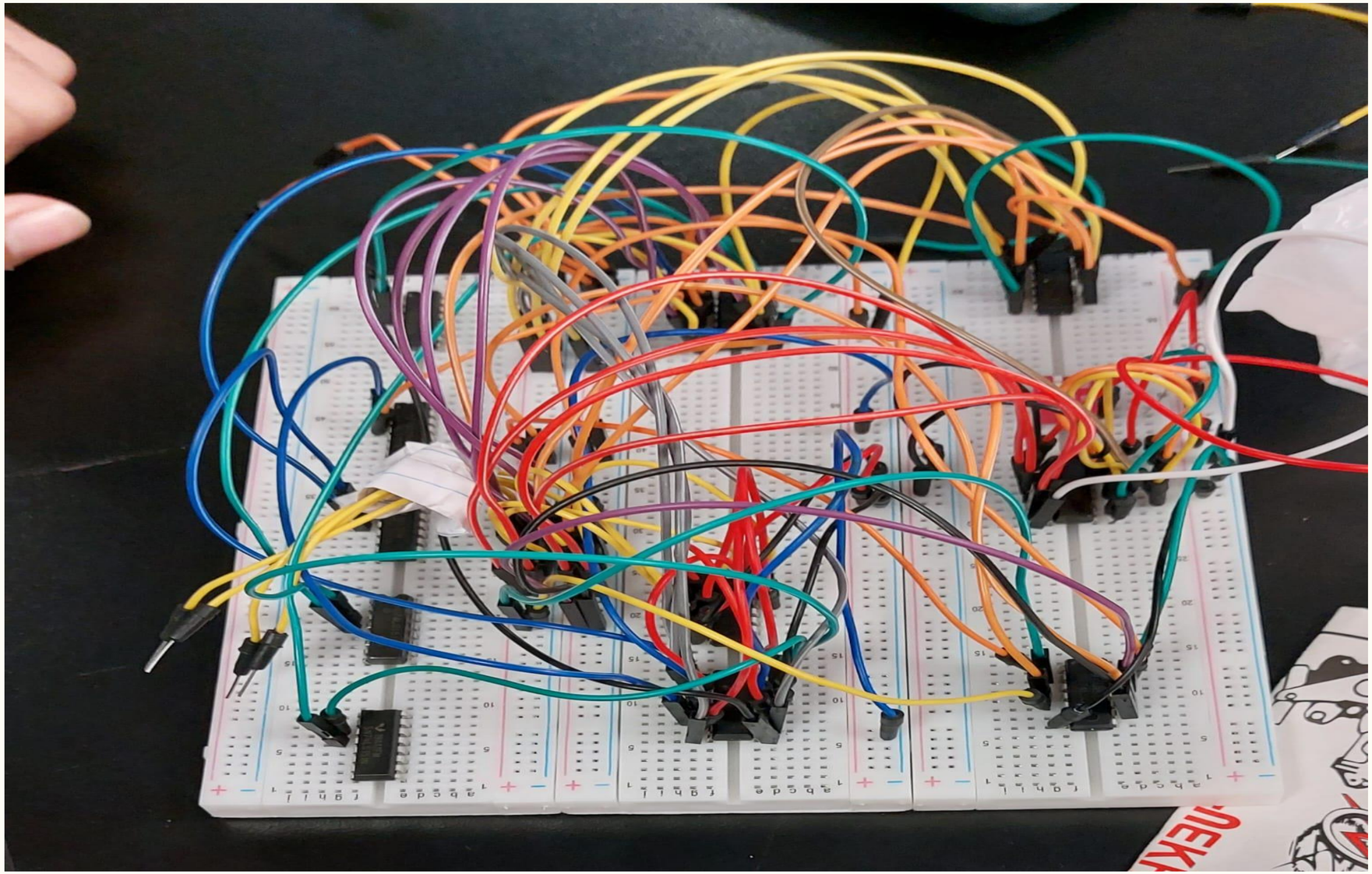



adder

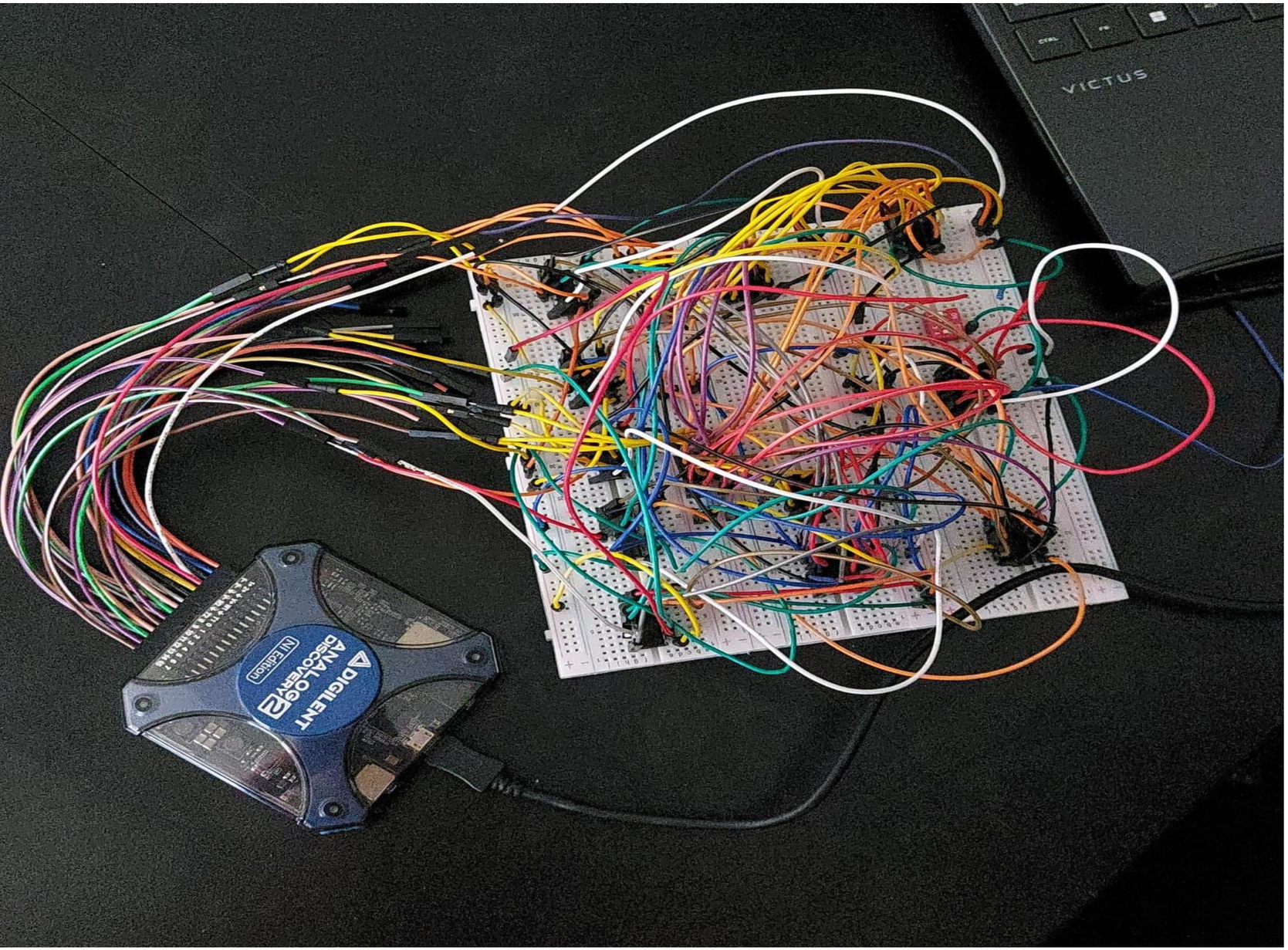

subtractor


right arithmetic shifter

Hardware design



Hardware design



THANK YOU!