DIGITAL SYSTEM DESIGN LAB

(Course code - EC 204)

(PROJECT EVALUATION)

TITLE: SIMPLE ARITHMETIC CALCULATOR

STUDENT NAME:

1. RADHIKA (221EC143)
2. RAJKUMAR WALKE (221EC144)

CONTENT PAGE:

DESCRIPTION OF PROBLEM STATEMENT:

The project aims to develop a basic arithmetic calculator using Verilog hardware description language (HDL) and simulating it on Logisim. The calculator will be capable of performing fundamental arithmetic operations, including addition, subtraction, multiplication, and division on 2-bit positive binary numbers. The primary objective is to design a digital circuit that efficiently executes these operations.

KEY FEATURES:

1.Implement a straightforward user interface for input and output, allowing users to choose operands and select the desired operation.

2.Develop modules for addition, subtraction, multiplication and division. Ensure the accuracy and efficiency of each operation, considering both speed and resource utilization.

DESIGN AND WORKING:

The input to the calculator is 6 bit long.

The 1st two bits (a1, a0) and the last 2 bits (b1, b0) are input numbers of 2-bit positive binary numbers each.

The 3rd and 4th input bits (s1, s0) are for choosing the operation to be done on the input numbers. The code for different operations is as follows:

00 for Addition

01 for Subtraction

10 for Multiplication

11 for Division

The output is a 5-bit number (yrep, y3, y2, y1, y0).

- In case of addition and multiplication, yrep 0 for all cases, while y3 represents the MSB and y0 represents the LSB.
- In case of subtraction, yrep = 0 for all cases, y3 represents the sign of the output (1 for negative sign and 0 for positive sign), while y2 represents MSB and y0 represents LSB.
- In case of division, yrep = 1 if the decimals repeat and 0 if the decimal terminates. y3(MSB) and y2(LSB) are the bits before the decimal point,

TRUTH TABLE:

Input						Output				
S_0	S_1	A_0	A_1	B_0	B_1	Yrep	Y3	Y2	Y1	y0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	0	0	1	0

0	0	0	0	1	1	0	0	0	1	1
0	0	0	1	0	0	0	0	0	0	1
0	0	0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	0	0	0	1	1
0	0	0	1	1	1	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	1	0	0	0	1	0	0
0	0	1	0	1	1	0	0	1	0	1
0	0	1	1	0	0	0	0	0	1	1
0	0	1	1	0	1	0	0	1	0	0
0	0	1	1	1	0	0	0	1	0	1
0	0	1	1	1	1	0	0	1	1	0
0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	1	0	0	1
0	1	0	0	1	0	0	1	0	1	0
0	1	0	0	1	1	0	1	0	1	1

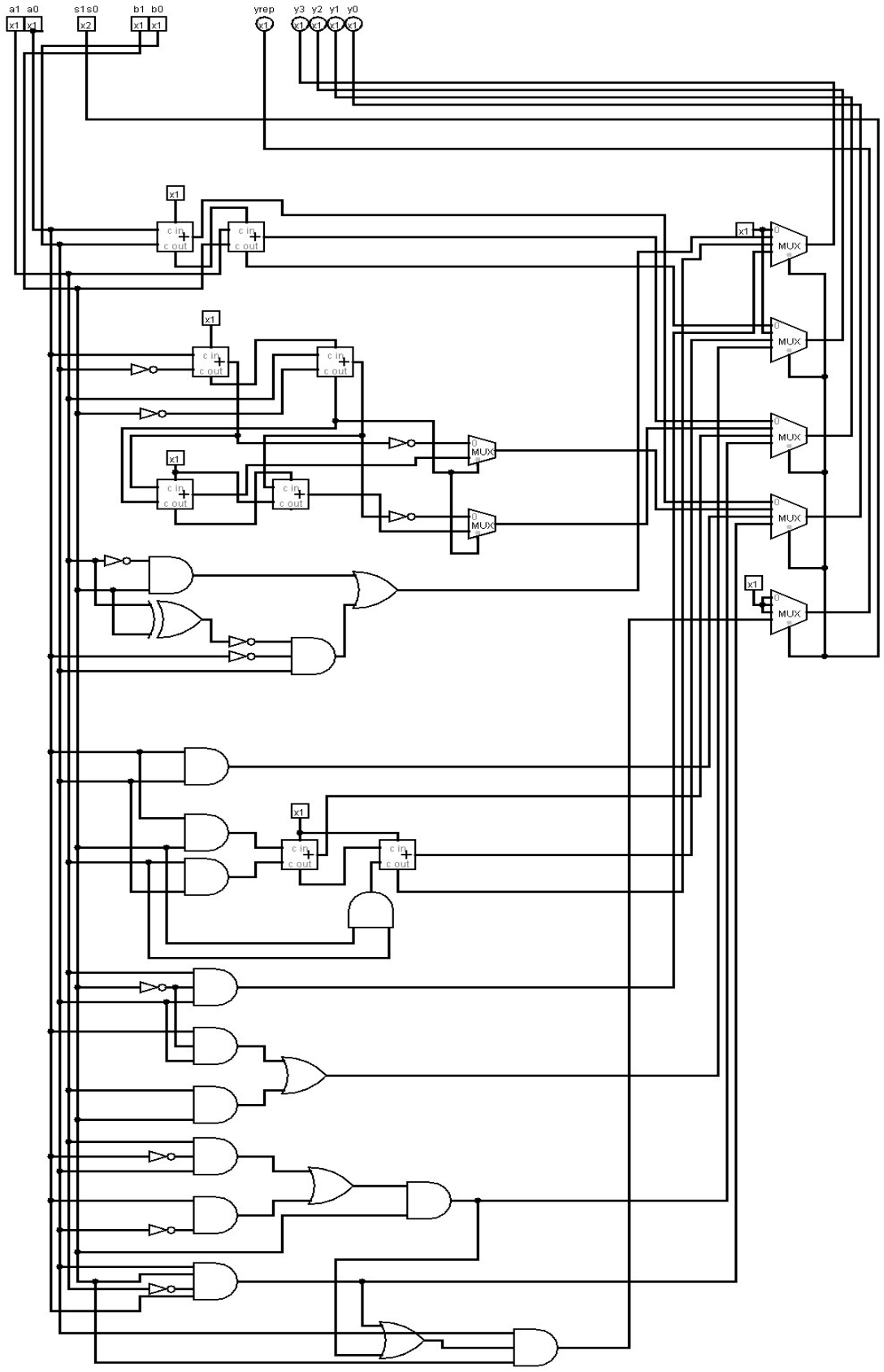
0	1	0	1	0	0	0	0	0	0	1
0	1	0	1	0	1	0	0	0	0	0
0	1	0	1	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	1	0
0	1	1	0	0	1	0	0	0	0	1
0	1	1	0	1	0	0	0	0	0	0
0	1	1	0	1	1	0	1	0	0	1
0	1	1	1	0	0	0	0	0	1	1
0	1	1	1	0	1	0	0	0	1	0
0	1	1	1	1	0	0	0	0	0	1
0	1	1	1	1	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0

1	0	0	0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	1
1	0	0	1	1	0	0	0	0	1	0
1	0	0	1	1	1	0	0	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	1	0
1	0	1	0	1	0	0	0	1	0	0
1	0	1	0	1	1	0	0	1	1	0
1	0	1	1	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	1	1
1	0	1	1	1	0	0	0	1	1	0
1	0	1	1	1	1	0	1	0	0	1
1	1	0	0	0	0	X	X	X	X	X

1	1	0	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0
1	1	0	1	0	0	X	X	X	X	X
1	1	0	1	0	1	0	0	1	0	0
1	1	0	1	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0	0	0	1
1	1	1	0	0	0	X	X	X	X	X
1	1	1	0	0	1	0	1	0	0	0
1	1	1	0	1	0	0	0	1	0	0
1	1	1	0	1	1	1	0	1	1	0
1	1	1	1	0	0	X	X	X	X	X
1	1	1	1	0	1	0	1	1	0	0
1	1	1	1	1	0	0	0	1	1	0

1	1	1	1	1	1	0	0	1	0	0

Logisim:



Verilog Code:

```
//Module that defines a 2:1 mux using gate level implementation
module mux21(s, i0, i1, y);
    input s, i0, i1;
    wire w0, w1a, w1b;
    output y;
    //y = S.I1 + ~S.I0 where S- select line and Ii are inputs and y - output
    not(w0, s);
    and(w1a, s, i1); and(w1b, w0, i0);
    or(y, w1a, w1b);
endmodule

//Module that defines a 4:1 mux using gate level implementation
module mux41(s0, s1, i0, i1, i2, i3, y);
    input s0, s1; //Select lines
    input i0, i1, i2, i3; //Input
    output y;
    wire wa, wb, w0a, w0b, w0c, w0d, w1a, w1b, c1, c2; //Connecting wires
    //y = ~s1.(~s0.i0 + s0.i1) + s1.(~s0.i2 + s0.i3)
    not(wa, s0); not(wb, s1);
    and(w0a, wa, i0); and(w0b, s0, i1); and(w0c, wa, i2); and(w0d, s0, i3);
    or(w1a, w0a, w0b); or(w1b, w0c, w0d);
```

```
    and(c1, wb, w1a); and(c2, s1, w1b);
    or(y, c1, c2);
endmodule

//Module defining 1-bit full bit adder using gate level implementation
module full_adder(a,b,cin,sum,cout);
    input a,b,cin; //a,b - 1-bit inputs;  cin - input carry
    output sum, cout;
    wire w0, w1, w2;//connecting wires
    //sum = a^b^cin;
    //cout = ab + cin(a^b)
    xor(w0, a,b);
    xor(sum, w0, cin);
    and(w1, a,b); and(w2, w0, cin);
    or(cout, w1, w2);
endmodule

//Main module for defining simple calculator using gate level implementaion
module simple_calci(a1, a0, b1, b0, y3, y2, y1, y0, s0, s1, yrep);
    input a1, a0, b1, b0; //2 bit binary positive input a=a1a0 and b=b1b0
    output y3, y2, y1, y0; //4-bit output y3-for sign (in case of subtraction), y1, y0-bits after decimal point (in case of division),
```

```
                    //else y3-MSB and y0-LSB
    output yrep; //1 bit to represent nonterminating decimal points in case of division to differentiate the output of 2/3 and 3/2
    input s0, s1; //si - select lines, {00 - add, 01 - subtract, 10 - multiply, 11-divide}

    //Addition
    wire sum0, sum1, carry1, carry0;
    full_adder fadd1(a0, b0,1'b0, sum0, carry0);
    full_adder fadd2(a1, b1, carry0, sum1, carry1);

    //Subtraction
    wire diff0, diff1;
    wire b0bar, b1bar, a00, a10;
    wire s00, s10, s0bar, s1bar, c1, c0, c00, c01;
    not(b0bar, b0); not (b1bar, b1);
    full_adder fsub1(a0, b0bar,1'b0, a00, c0);
    full_adder fsub2(a1, b1bar, c0, a10, c1);
    full_adder fsub3(a00, c1,1'b0, s00, c00);
    full_adder fsub4(a10,1'b0, c00, s10, c01);
    not(s0bar, a00); not(s1bar, a10);
    mux21 msub1(c1, s0bar, s00, diff0);
    mux21 msub2(c1, s1bar, s10, diff1);
    //Subatraction(check whether the output is negative or not)
```

```
    wire wsgn0, wsgn1, wsgn2, a0bar, a1bar;
    wire diff3;
    not(a0bar, a0); not(a1bar, a1);
    xnor(wsgn1, a1, b1);
    and(wsgn0, a1bar, b1); and(wsgn2, a0bar, b0, wsgn1);
    or(diff3, wsgn0, wsgn2);

    //Multiplication
    wire wpro0, wpro1, wpro2, wpro3;
    wire pro0, pro1, pro2, pro3;
    and(pro0, a0, b0); and(wpro0, a0, b1); and(wpro1, a1, b0);
and(wpro3, a1, b1);
    full_adder fmul1(wpro0, wpro1,1'b0, pro1, wpro2);
    full_adder fmul2(wpro3, wpro2,1'b0, pro2, pro3);

    //Division
    wire quo0, quo1, quo2, quo3;
    wire wdiv0;
    wire wdiv11, wdiv12, wdiv13;
    wire wdiv21, wdiv22;
    wire wrep;
    xor(wrep, a1, a0); and(yrep, b1, b0, wrep, s0, s1);
    and(quo3, a1, b1bar, b0);
    and(wdiv21, b1bar, b0, a0); and(wdiv22, a1, b1);
    or(quo2, wdiv21, wdiv22);
```

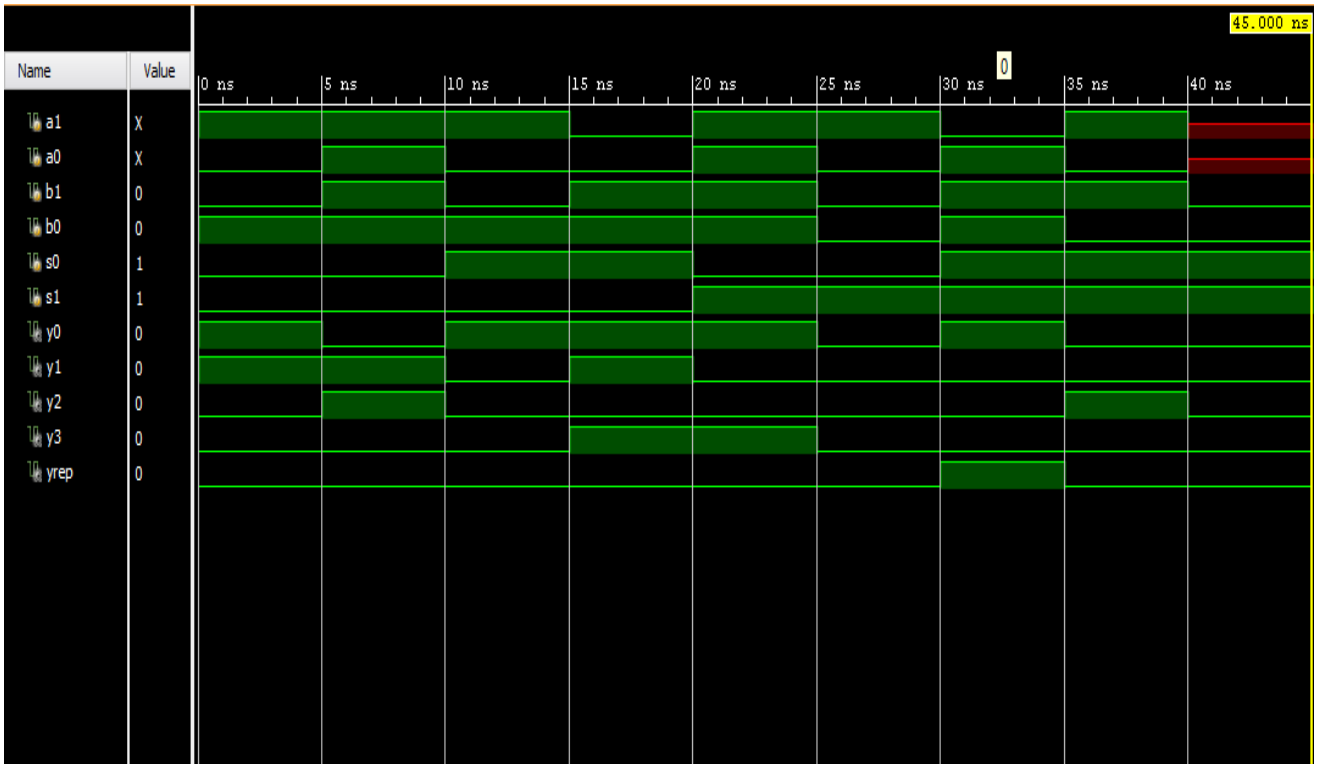
```
    and(wdiv11, a0, b0bar); and(wdiv12, a0bar, b0, a1); or(wdiv13,
wdiv11, wdiv12);
    and(quo1, b1, wdiv13);
    and(quo0, b1, b0, a1bar, a0);

    //Connecting the outputs to mux for selected output
according to the user input
    mux41 m1(s0, s1, sum0, diff0, pro0, quo0, y0);
    mux41 m2(s0, s1, sum1, diff1, pro1, quo1, y1);
    mux41 m3(s0, s1, carry1,1'b0, pro2, quo2, y2);
    mux41 m4(s0, s1,1'b0, diff3, pro3, quo3, y3);

endmodule
```

TEST BENCH:

```
module simple_calci_tb();
    reg a1, a0, b1, b0;
    reg s0, s1;
    wire y0, y1, y2, y3;
    wire yrep;

    simple_calci sc(a1, a0, b1, b0, y3, y2, y1, y0, s0, s1, yrep);

    initial
        begin
```

```
        s0=1'b0; s1=1'b0; a1=1; a0=0; b1=0; b0=1; #5;
        a1=1; a0=1; b1=1; b0=1; #5;
        s0=1'b1; s1=1'b0; a1=1; a0=0; b1=0; b0=1; #5;
        a1=0; a0=0; b1=1; b0=1; #5;
        s0=1'b0; s1=1'b1; a1=1; a0=1; b1=1; b0=1; #5;
        a1=1; a0=0; b1=0; b0=0; #5;
        s0=1'b1; s1=1'b1; a1=0; a0=1; b1=1; b0=1; #5;
        a1=1; a0=0; b1=1; b0=0; #5;
        a1=1'bx; a0=1'bx; b1=0; b0=0; #5;
        $finish;
        end

endmodule
```

WAVEFORM:



CONCLUSION:

To summarise, this project is about building a simple calculator that performs basic functions like addition, subtraction, multiplication and division on 2-bit positive binary numbers. The calculator is constructed using basic combinational circuit elements and is simulated using the simulation tool, Logisim. It is later modelled using gate-level modelling, using the HDL Verilog in Vivado software.