CS -268 | MIDSEM REPORT FILE

Submitted By -

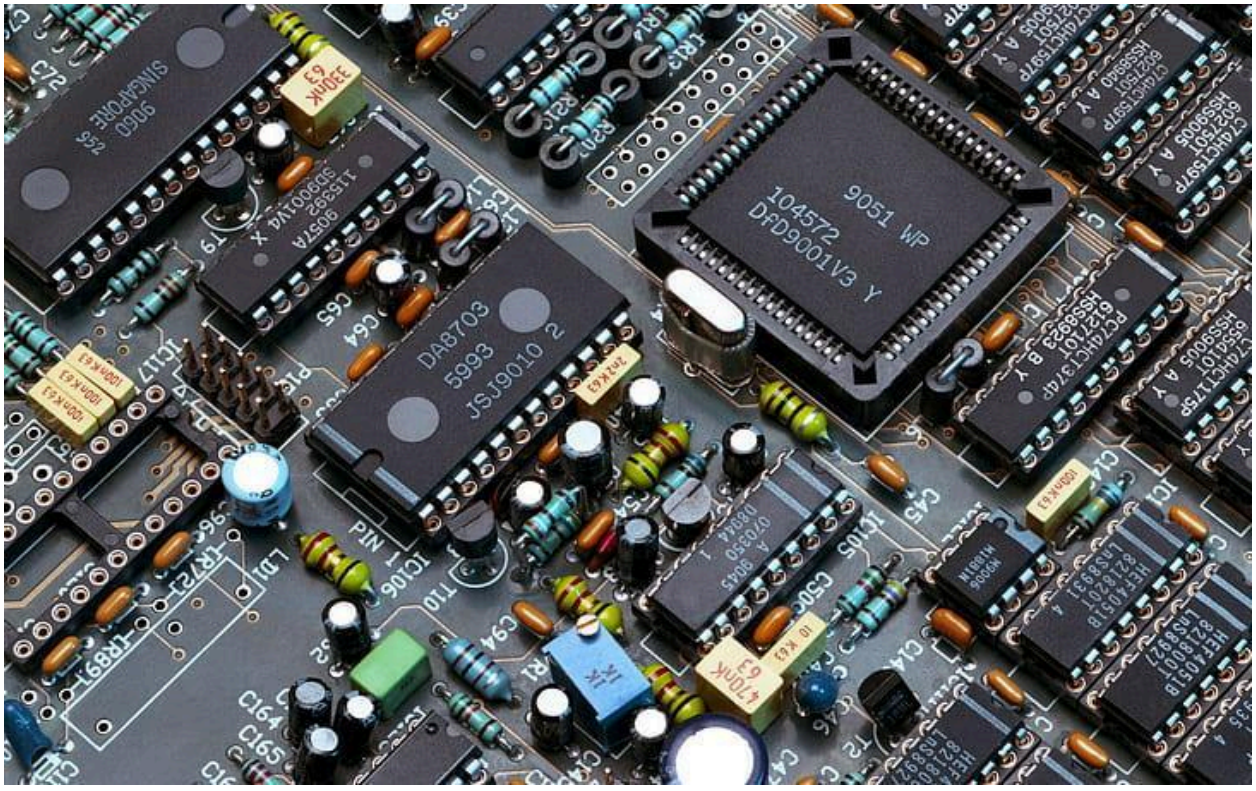
Pradumya Gaurav | ID – 202251096

Sourabh Ramakrishna Patil | ID – 202251095

Malaika Varshney | ID - 202251069

Submitted To - Dr. Kamal Kishore Jha

CS - 268 | Lab Report



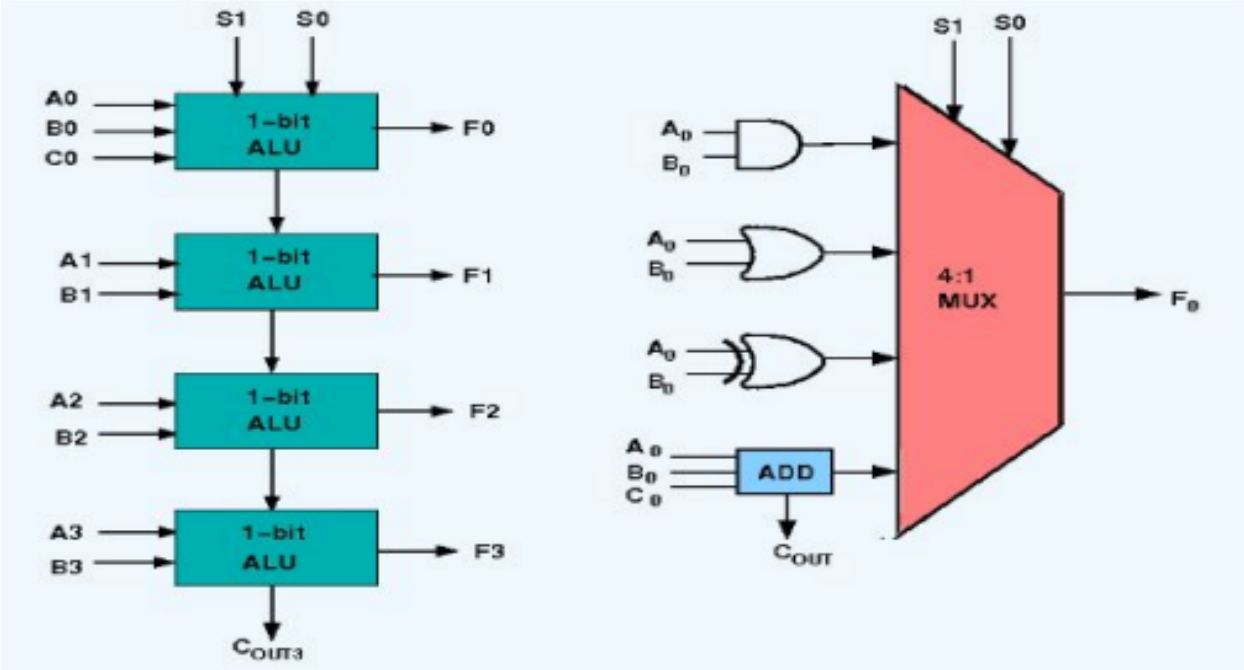
Report File | Week - 01

Question:

Designing of 1-bit and 4-bit ALU using Logisim.

Theory:

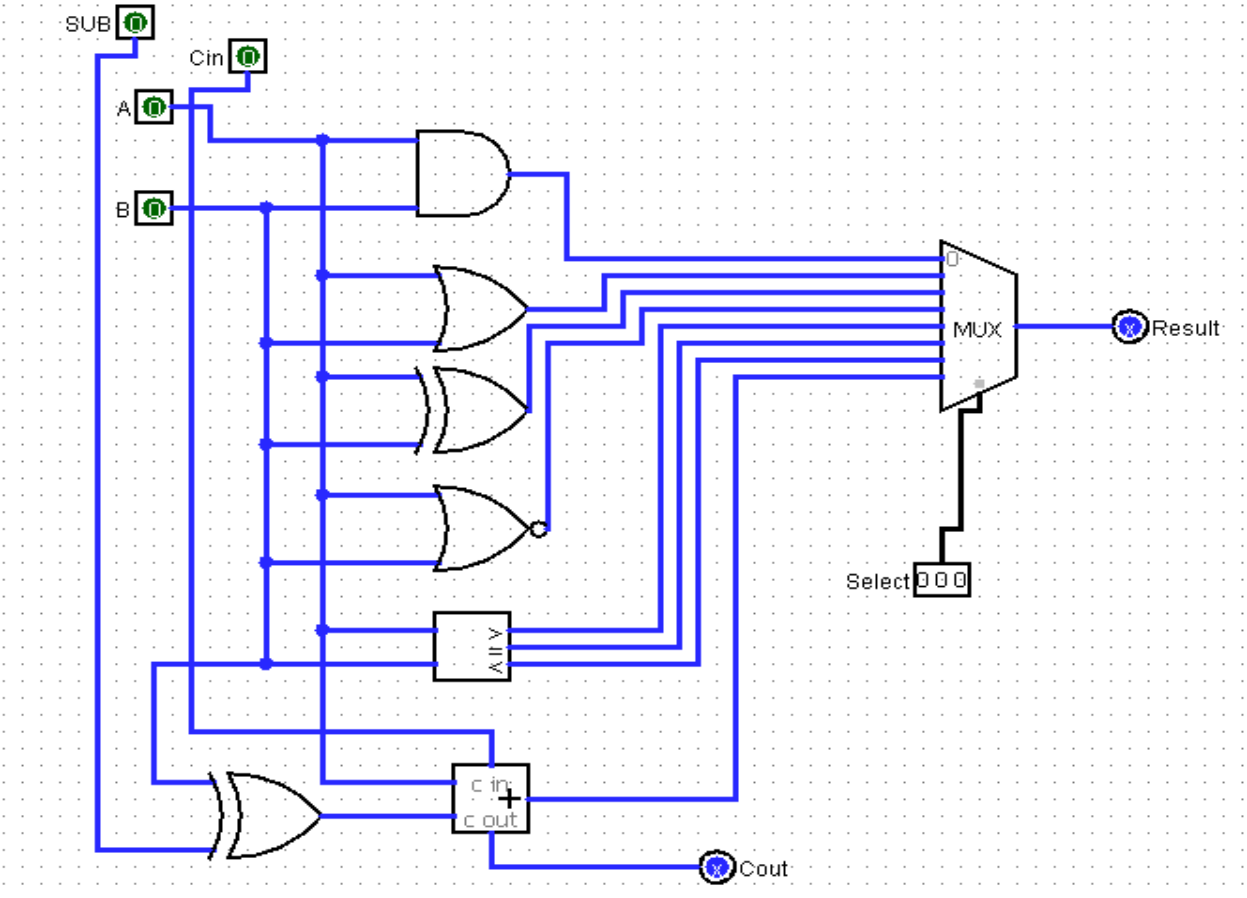
ALU or Arithmetic Logical Unit is a digital circuit to do arithmetic operations like addition, subtraction,division, multiplication and logical oparations like and, or, xor, nand, nor etc. A simple block diagram of a 4 bit ALU for operations and,or,xor and Add is shown here .



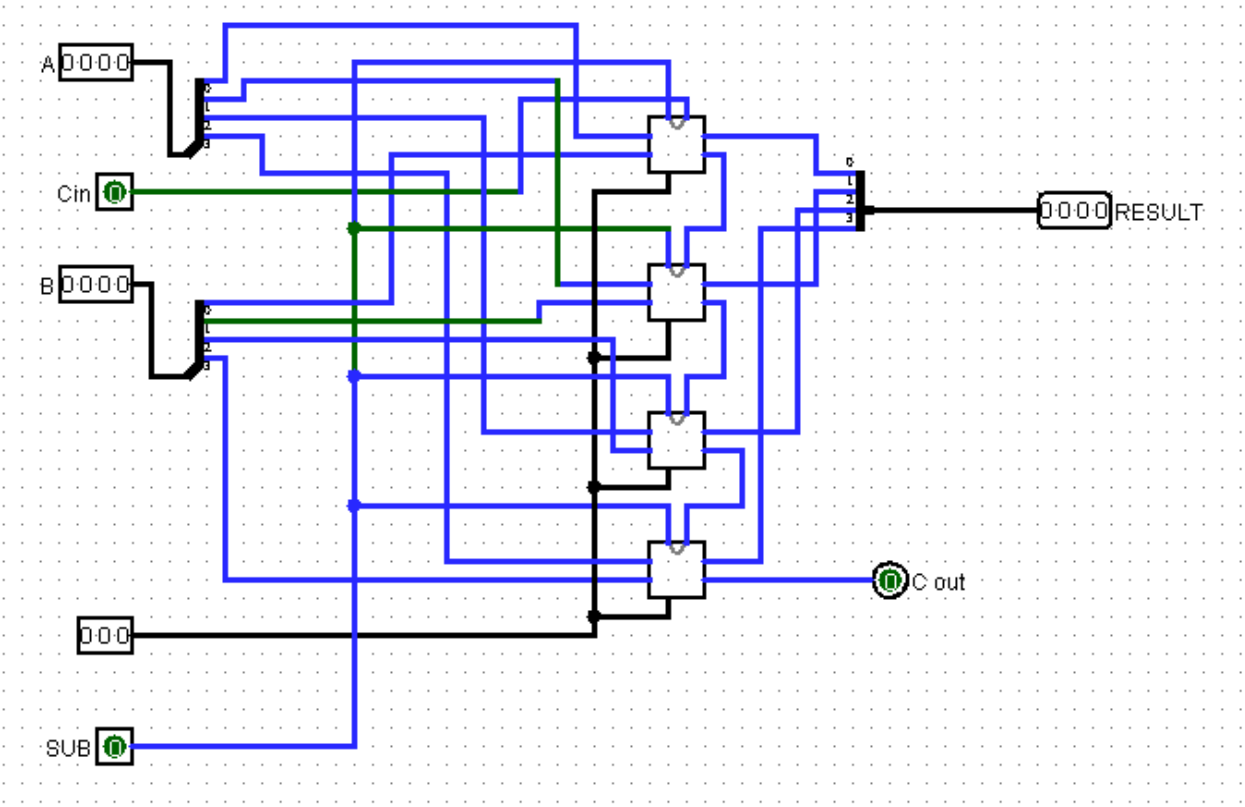
Design & Procedure:

The figure shown below depicts 1-bit ALU. It performs 8 operations :

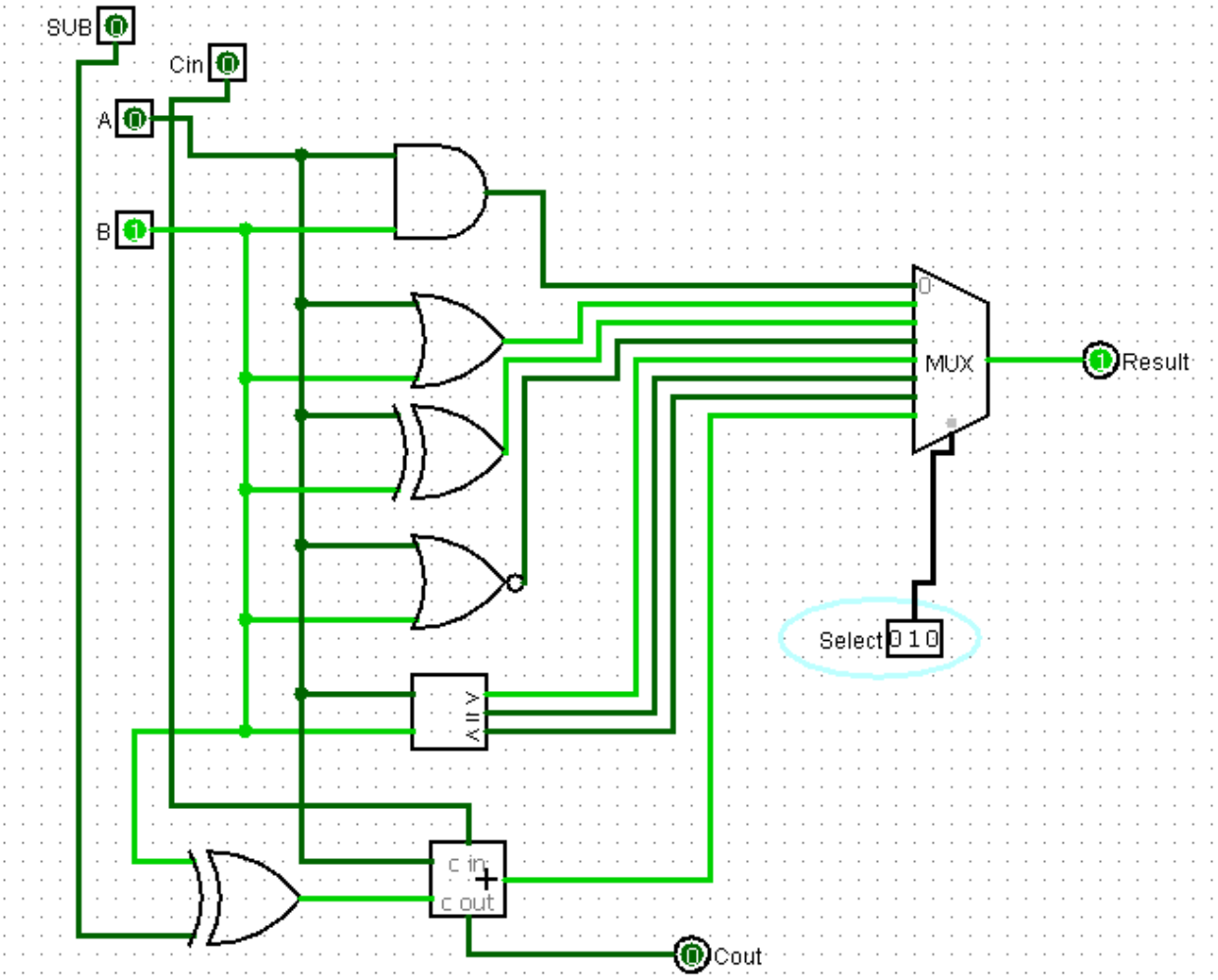
1. AND Operation
2. OR Operation
3. XOR Operation
4. NOR Operation
5. GREATER THAN Comparator
6. EQUAL TO Comparator
7. LESSER THAN Comparator
8. ADDER/SUBTRACTOR Operation



With the help of 1-bit ALU, we constructed a 4-bit ALU where carry/borrow propogates from 1st 1-bit ALU to others. Below is the figure of 4-bit ALU.



Testing of 1-bit ALU:



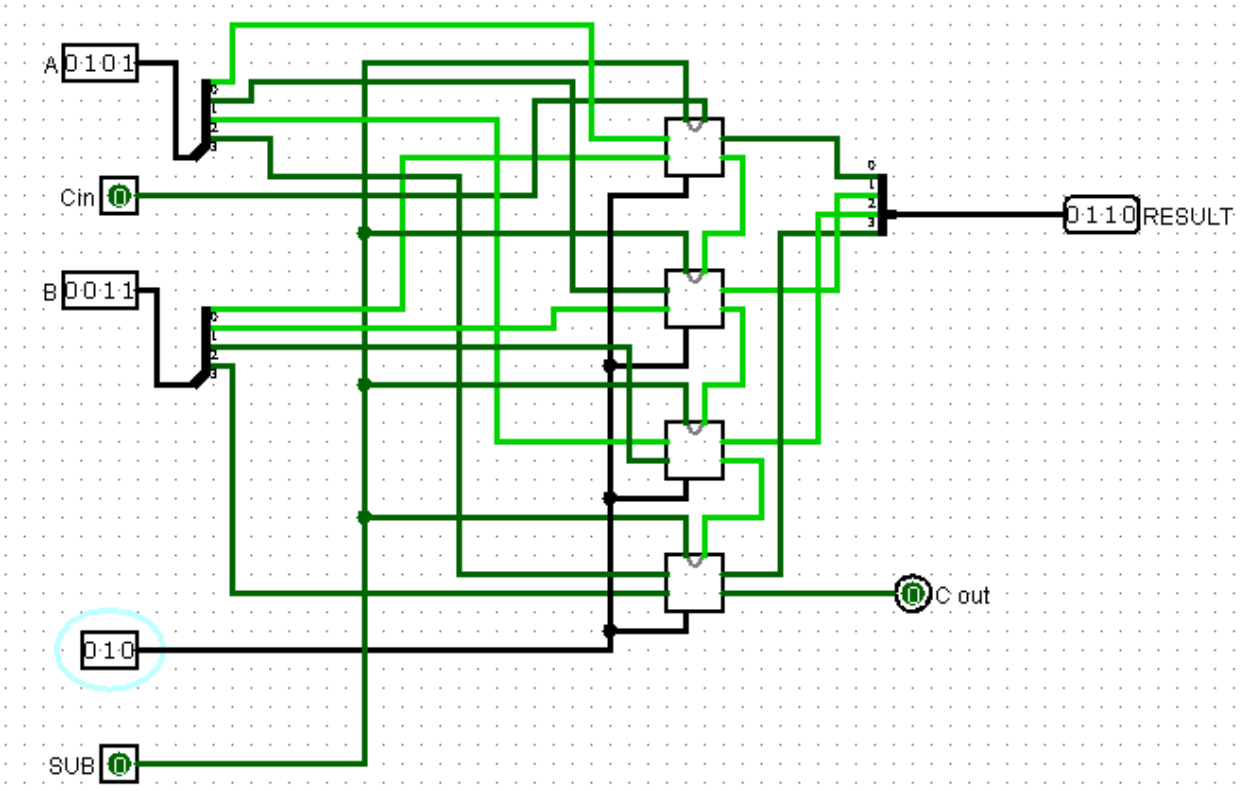
- First, we will choose the select line in 8:1 MUX. Then, we will provide inputs in A and B and finally the result gets displayed based on the select line chosen by the user.

- In the above example,select line chosen is 010 i.e, XOR Operator.

- By providing the input in A and B as 0 and 1 respectively we get the result as 1.
- Similarly, we can perform other operations too.

Testing of 4-bit ALU:

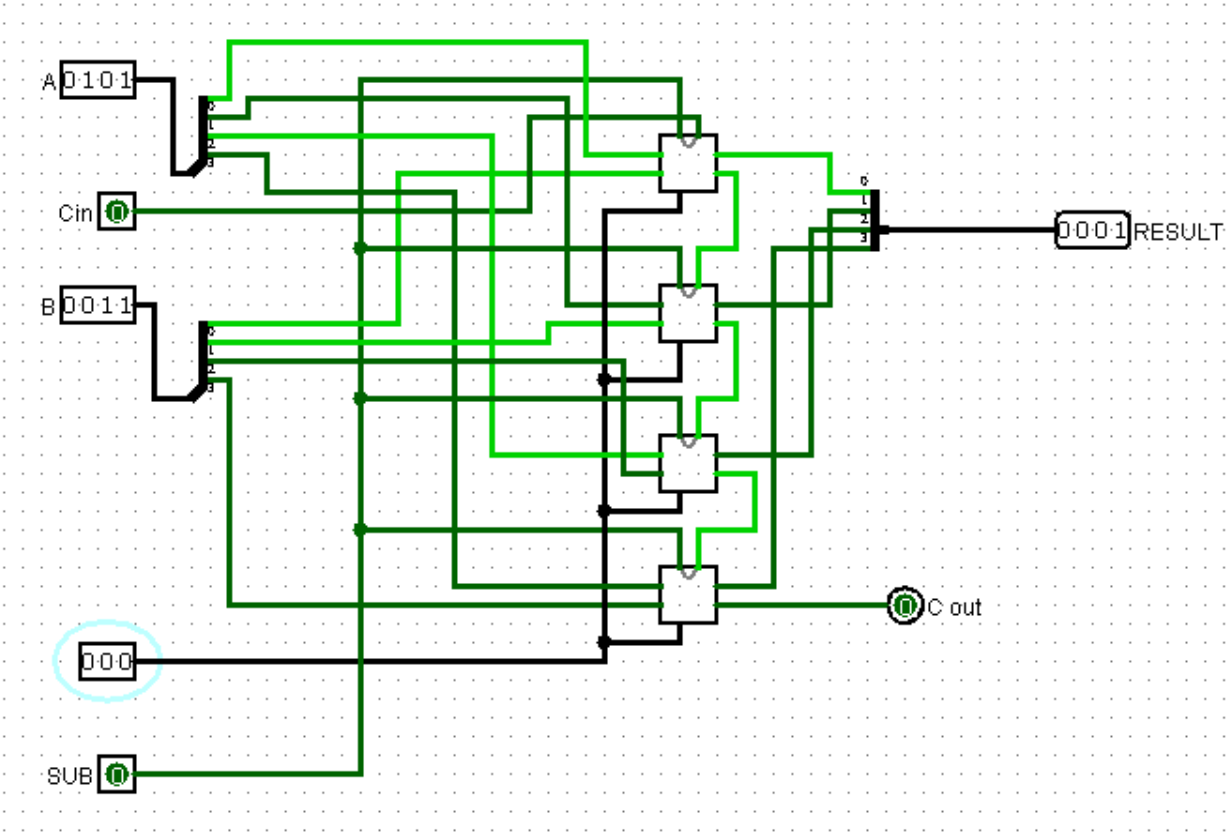
XOR Operation:



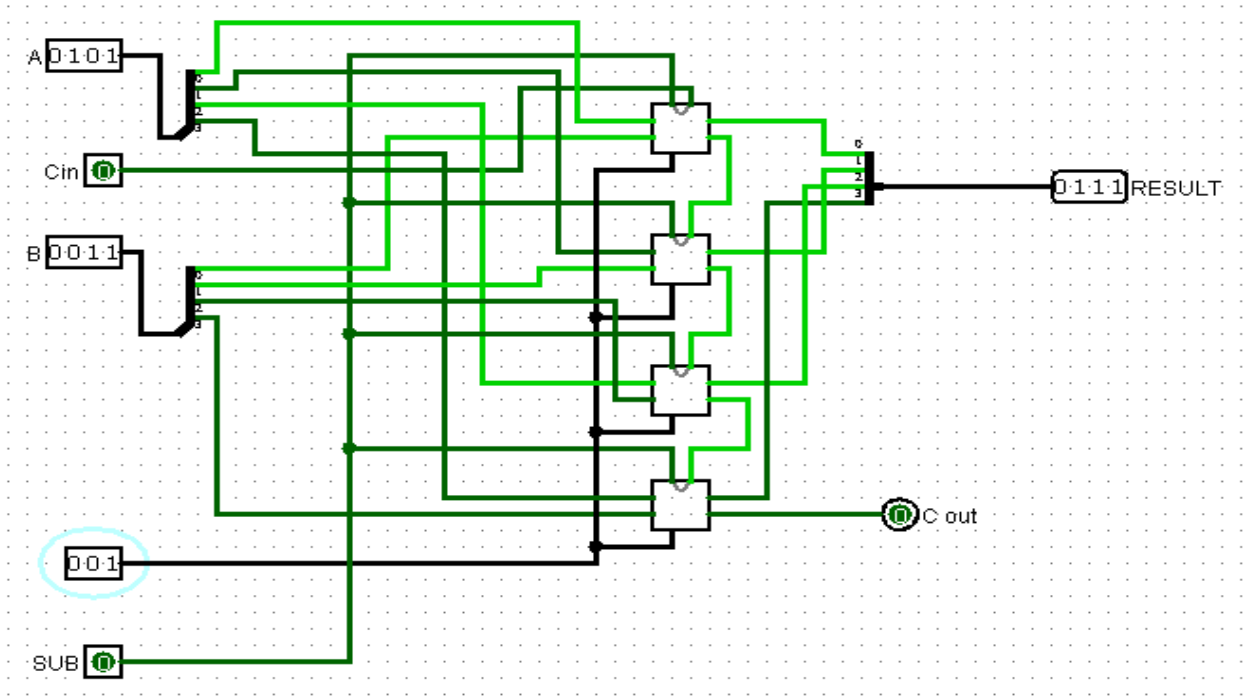
- Initially, we set the input of A and B as ‘0101’ and ‘0011’ respectively. The 4-bit ALU performs the same operations as 1-bit ALU and displays the result in 4-bit binary number.

- On choosing the same select line for XOR Operator as we did in 1-bit ALU i.e., ‘010’, we get the result as ‘0110’.
- Similarly, we can do this for other operations.

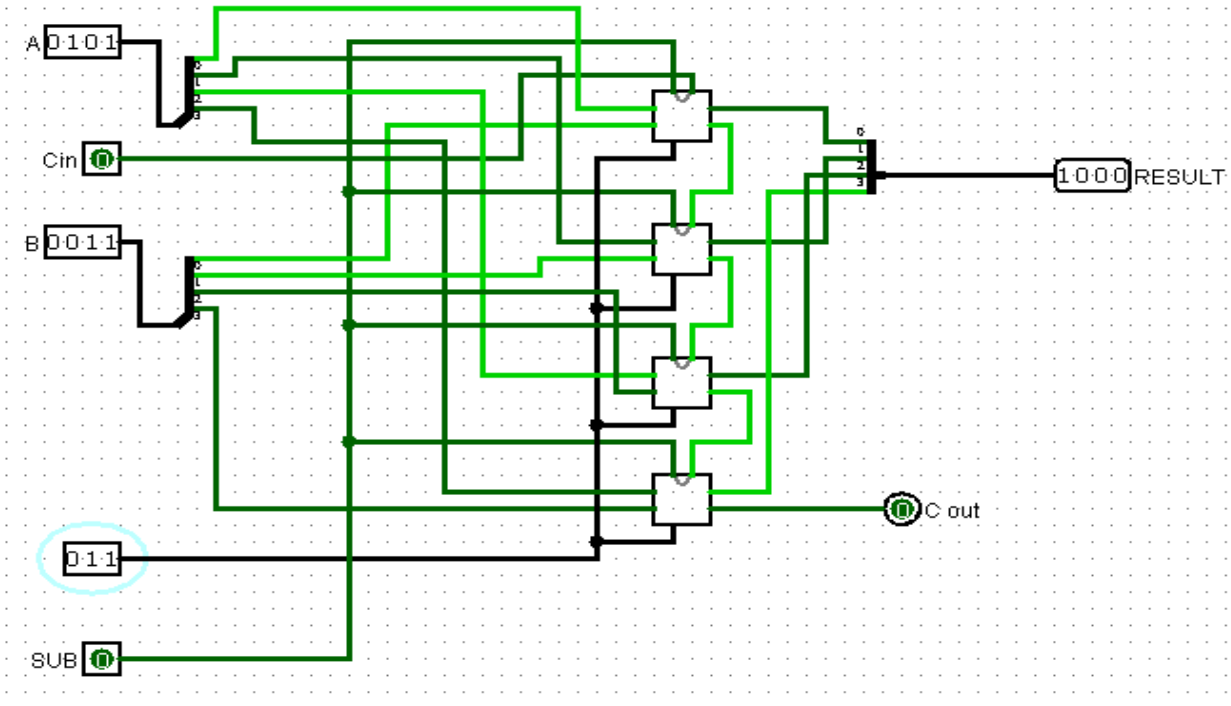
AND Operation:



OR Operation:

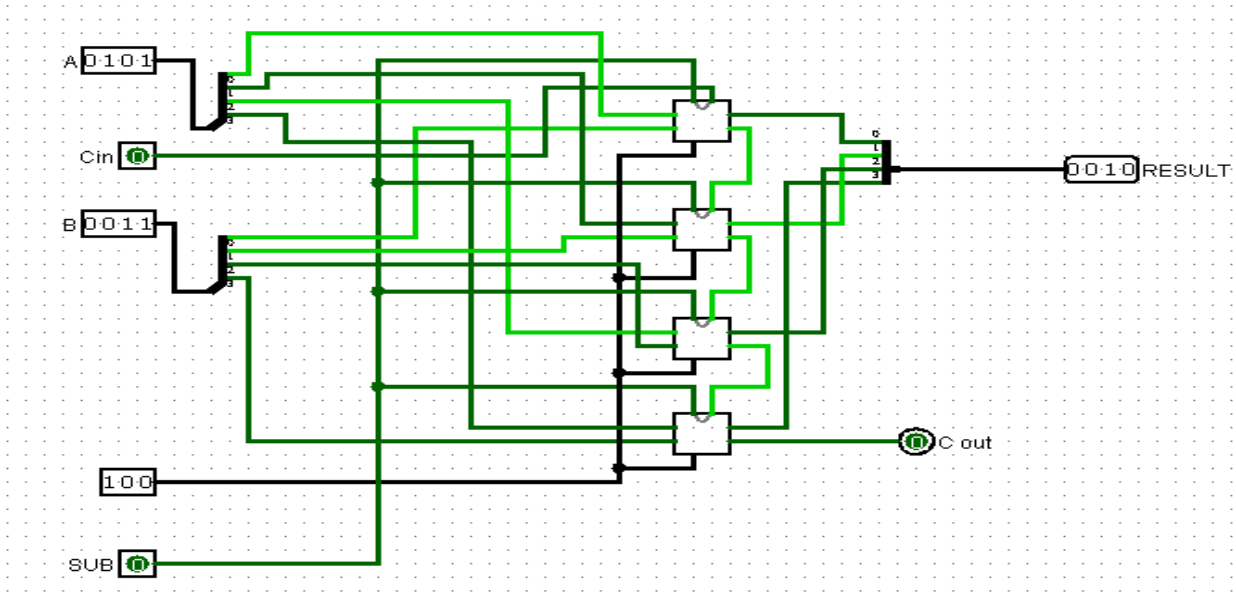


NOR Operation:

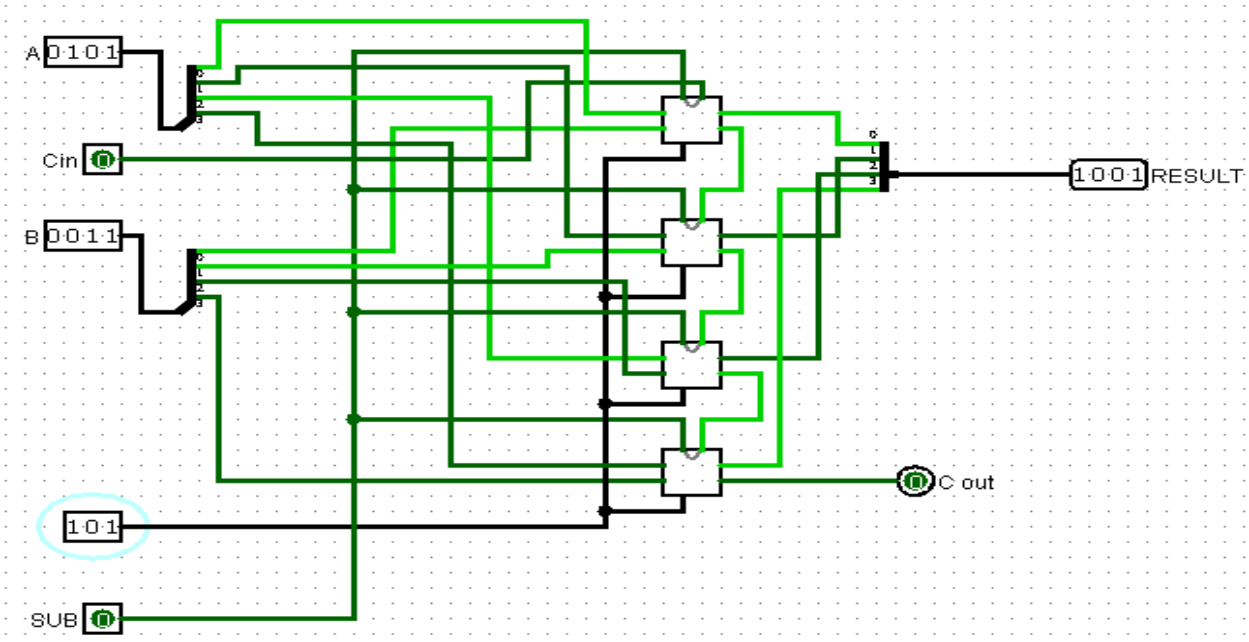


COMPARATOR Operation:

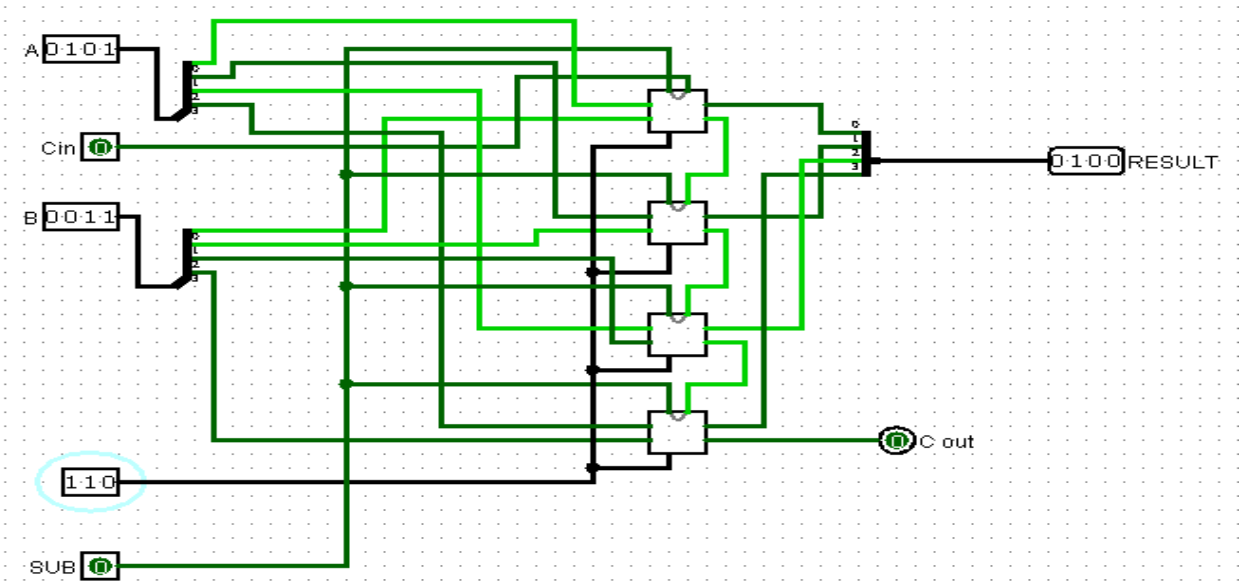
- ***Less Than Operation:***



- ***Equal To Operation:***

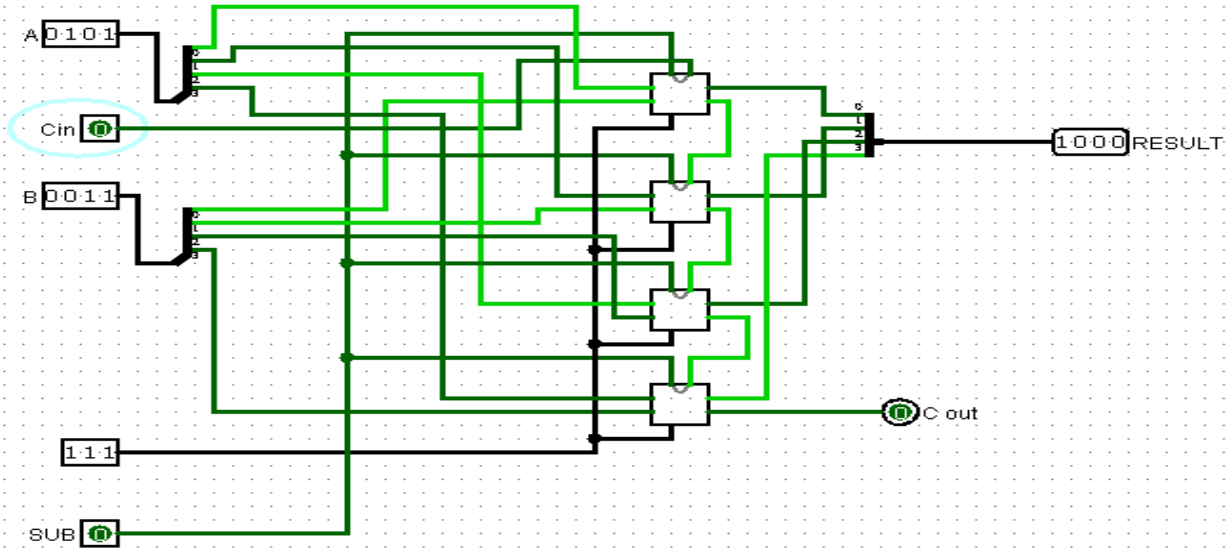


- ***Greater Than Operation:***

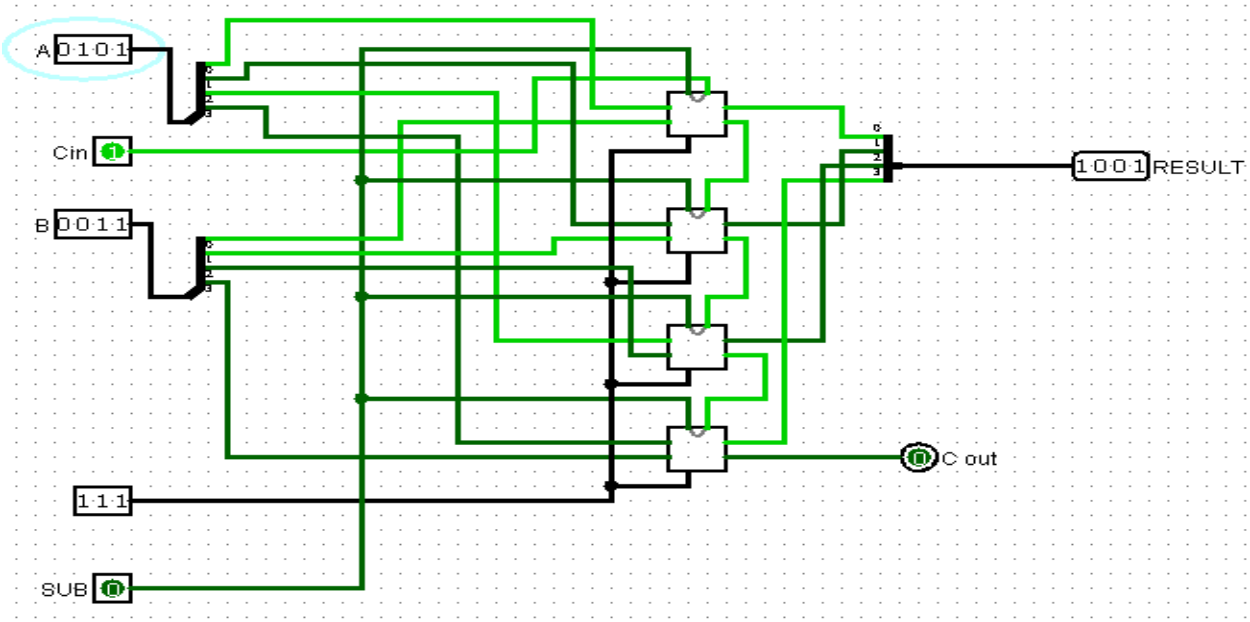


ADDER Operation:

- ***Carry Input As 0:***

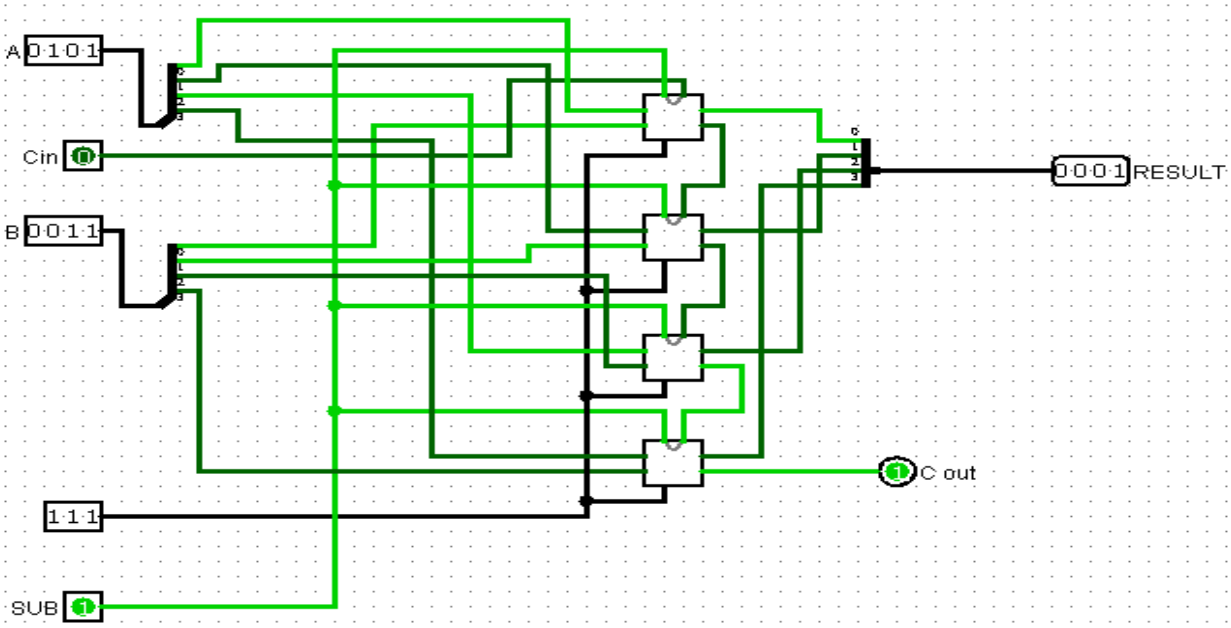


- ***Carry Input As 1:***

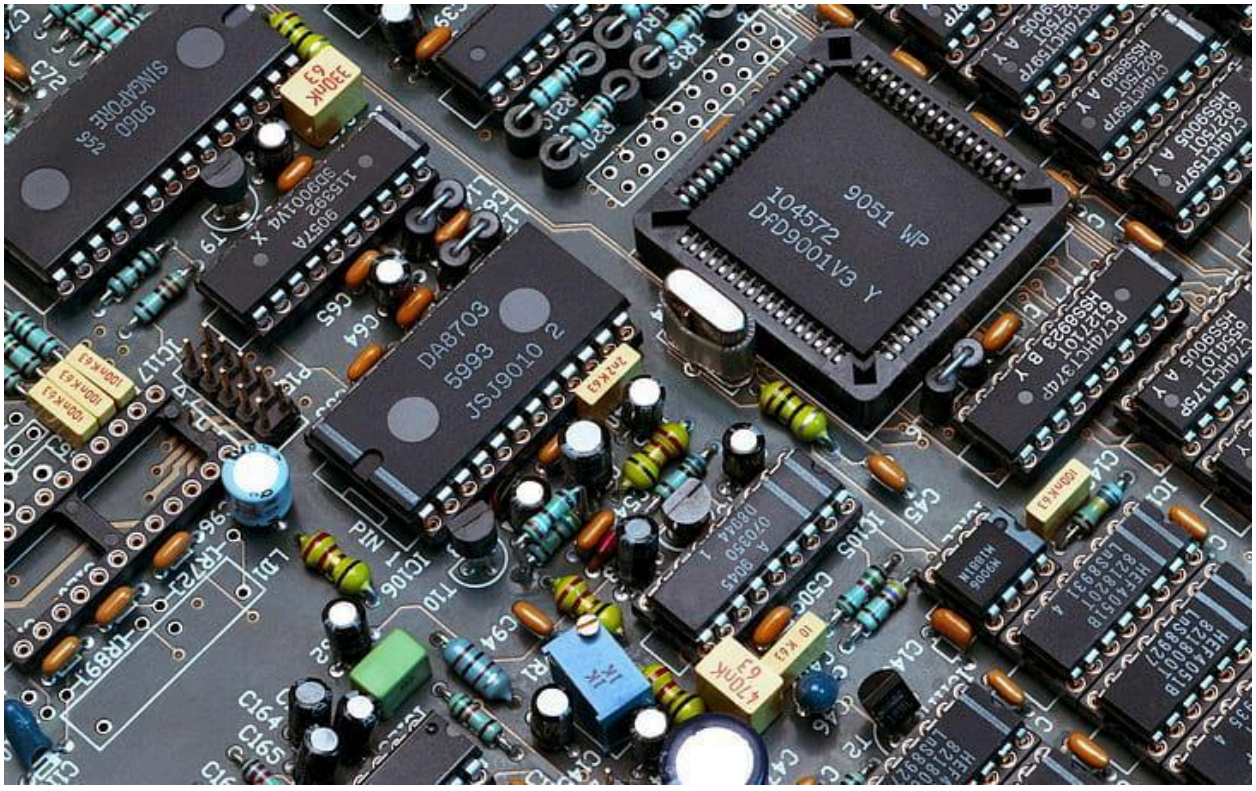


SUBTRACTOR Operation:

- ***Sub Input As 1:***



CS - 268 | Lab Report



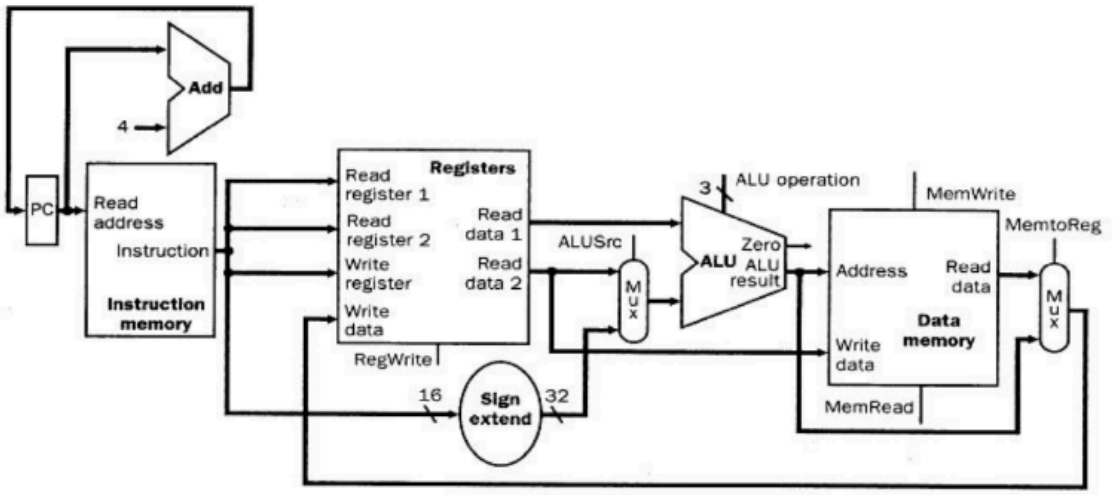
Report File | Week - 02

Question:

To implement Data Path Architecture structure using 4-bit ALU

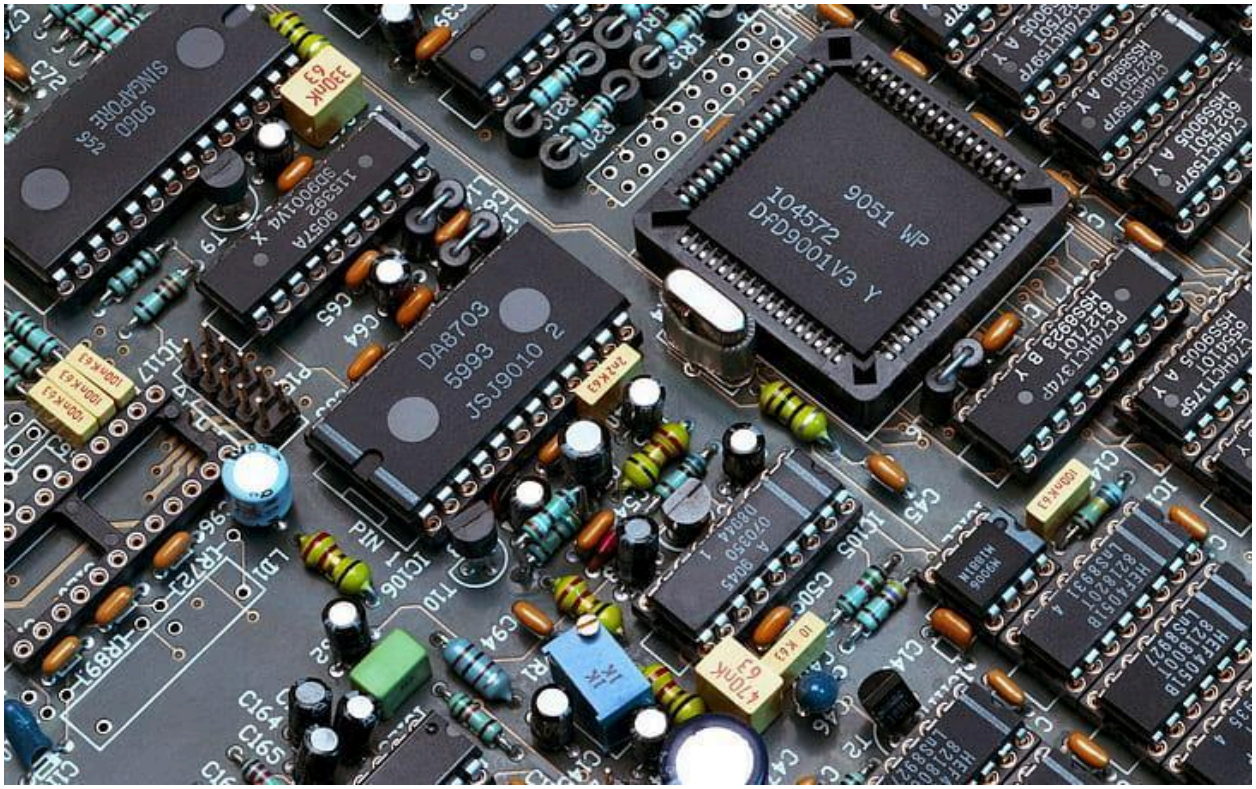
Theory & Design:

- A data path (also written as datapath) is a set of functional units that carry out data processing operations.
- Datapaths, along with a control unit, make up the CPU (central processing unit) of a computer system.
- A larger data path can also be created by joining more than one together using multiplexers.
- Currently, data paths can only be configured once. Researchers are trying to find ways to imprint data paths on fabrics and make them reconfigurable.
- This action would allow them to be configured at runtime, providing for improved efficiency and power savings.



Test & Output:

CS - 268 | Lab Report



Report File | Week - 03

Aim:

Design and implementation of Booth Multiplier in Verilog

Objective:

- To find the error in BoothMultiplier.v file
- To ensure the proper working of BoothMultiplier
- To ensure that proper state are activated with respect to q0 an qm1

In this experiment we (202251069)Malaika Varshney, (202251096)Pradumya Gorav and (202251095)Sourabh Patil have analyzed the code part and found 4 errors.

```
module boothmultiplier

(LdA,LdQ,LdM,clrA,clrq,clrff,sftA,sftQ,addsub,ldcnt,data_in,decr,clk,q0,qm1,eqz);//
objective M*Q  Datapath

      input LdA,LdQ,LdM,clrA,clrq,clrff,sftA,sftQ,addsub,ldcnt,decr,clk;

      input [15:0] data_in;

      output q0,qm1,eqz;

      wire temp;

      wire[15:0] A,Q,M,Z;

      wire[5:0] count;



      assign eqz=~|count;



      shiftreg AR(LdA,clrA,Z,sftA,A,clk,A[15]);     //Submodule for accumulator

      shiftreg QR(LdQ,clrQ,data_in,sftQ,Q,clk,A[0]);//Submodule for multiplier
```

```
	dff QM1(Q[0],q0,clk,clrff);//Submodule for buffer

	dff Q0(q0,qm1,clk,clrff);

	addsum AS(Z,A,M,addsub);//Submodule for adder

	PIPO MR(LdM,data_in,M,clk);//Submodule for Multiplicand

	CNTR CN(ldcnt,clk,decr,count);//Submodule for counter



endmodule



// To make the ASR shift

module shiftreg (ld,clr,din,sft,dout,clk,s_in);

	input ld,clr,sft,clk,s_in;

	output reg [15:0] dout;

	input [15:0] din;

	always@(posedge clk) //clock  Shift register A & Q

		begin

			if(clr)

				dout<=0;

			else if (ld)

				dout<=din;

			else if (sft)

				dout<={s_in,dout[15:1]};

		end

endmodule



// To load multiplier

module PIPO(ld,din,dout,clk);
```

```
	input[15:0] din;
	input ld,clk;
	output reg [15:0] dout;
	always@(posedge clk)   //clock for register M
		if (ld)
			dout<=din;
endmodule

// To load q0,qm1
module dff(d,q,clk,clr);
	input clr,clk,d;
	output reg q;
	always@(posedge clk) // clock for buffer register
		if (clr)
			#1 q<=0;
		else
			#1 q<=d;
endmodule

// To make AC=AC+M OR AC=AC-M
module addsum(out,in1,in2,addsub);
	input[15:0] in1,in2;
	input addsub;
	output reg[15:0] out;
	always@(*)
		begin
```

```
                    if (addsub==0)
                            out=in1-in2;
                    else
                            out=in1+in2;
            end
endmodule


// To make a counter for clock
module CNTR (ldcnt,clk,decr,count);
      input ldcnt,decr,clk;
      output reg[5:0] count;
      always@(posedge clk) // clock for counter
            begin
                    if (ldcnt)
                            count=5'b10000;
                    else if(decr)
                            count<=count-1;
            end
endmodule


// To check state
module controller
(LdA,LdQ,LdM,clrA,clrQ,clrff,sftA,sftQ,addsub,ldcnt,decr,done,clk,Q0,qm1,eqz,start);//
control path code
      input clk,Q0,qm1,eqz,start;
      output reg LdA,LdQ,LdM,clrA,clrQ,clrff,sftA,sftQ,addsub,ldcnt,decr,done;
      reg[2:0] state;
```

```
	parameter s0=3'b000,s1=3'b001,s2=3'b010,s3=3'b011,s4=3'b100,s5=3'b101,s6=3'b110;

	always @(posedge clk)

		begin

			case(state)

				s0: if(start) state<=s1;

				s1: #2 state<=s2;

				s2:  if({Q0,qm1}==2'b01) state<=s3;

					else if({Q0,qm1}==2'b10) state<=s4;

					else state<=s5;

				s3: state<=s5;

				s4:  state<=s5;

				s5: #2 if (({Q0,qm1}==2'b01) && !eqz) state<=s3;

					else if (({Q0,qm1}==2'b10) && !eqz) state<=s4;

					else if(eqz) state<=s6;

				s6: state<=s6;

				default: state <=s0;

			endcase

		end

	always@(state)

		begin

			case(state)

				s0: begin
LdA=0;clrA=0;sftA=0;LdQ=0;clrQ=0;clrff=1;LdQ=0;sftQ=0;LdM=0;done=0;end

				s1: begin clrA=1;clrff=0;ldcnt=1;LdQ=1;end

				s2: begin clrA=0;clrff=0;ldcnt=0;LdM=1;LdQ=0;end

				s3: begin LdA=1;addsub=1;LdQ=0;sftA=0;sftQ=0;decr=0;end

				s4: begin LdA=1;LdQ=0;sftA=0;sftQ=0;decr=0;addsub=0;end
```

```
                            s5: begin LdA=0;sftA=1;LdQ=0;sftQ=1;decr=1;end

                            s6: done=1;

                            default: begin LdA=0;clrA=0;LdQ=0;clrQ=0;clrff=0;LdM=0;end

                        endcase

                end

endmodule
```

Test Bench:

```
module BoothMultiplier_tb();

    reg [15:0] data_in;

    reg clk,start;

    wire done;

    boothmultiplier BM
(LdA,LdQ,LdM,clrA,clrq,clrff,sftA,sftQ,addsub,ldcnt,data_in,decr,clk,q0,qm1,eqz);

    controller
CL(LdA,LdQ,LdM,clrA,clrq,clrff,sftA,sftQ,addsub,ldcnt,decr,done,clk,q0,qm1,eqz,start);


    initial

        begin

            $monitor("Time Q A state");

            clk=0;

            start=1;

        end


    always #5 clk=~clk;


    initial

        begin

            # 10 data_in=3;
```

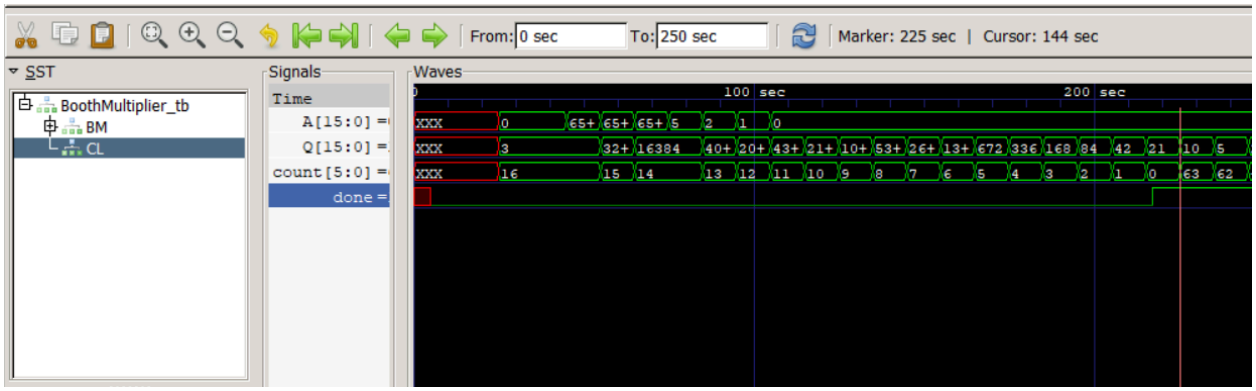
```
            # 20 data_in=7;

        end

    initial


        begin

            $monitor($time, "%b %b %d %d",BM.Q,BM.A,CL.state,CL.done,);

            $dumpfile("test_booth.vcd");

            $dumpvars(0,BoothMultiplier_tb);

            #250 $finish;

        end

endmodule
```

Terminal:

```
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_03> iverilog -o test .\BoothMultiplier.v
.\BoothMultiplier_tb.v
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_03> vvp test
VCD info: dumpfile test_booth.vcd opened for output.
                  0xxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx x x
                  5xxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx 0 0
                 15xxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx 1 0
                 250000000000000011 0000000000000000 1 0
                 270000000000000011 0000000000000000 2 0
                 350000000000000011 0000000000000000 4 0
                 450000000000000011 1111111111111001 5 0
                 551000000000000001 1111111111111100 5 0
                 650100000000000000 1111111111111110 5 0
                 670100000000000000 1111111111111110 3 0
                 750100000000000000 0000000000000101 5 0
                 851010000000000000 0000000000000010 5 0
                 950101000000000000 0000000000000001 5 0
                1051010100000000000 0000000000000000 5 0
                1150101010000000000 0000000000000000 5 0
                1250101010000000000 0000000000000000 5 0
                1350001010100000000 0000000000000000 5 0
                1450000101010000000 0000000000000000 5 0
                1550000010101000000 0000000000000000 5 0
                1650000001010100000 0000000000000000 5 0
                1750000000101010000 0000000000000000 5 0
                1850000000010101000 0000000000000000 5 0
                1950000000001010100 0000000000000000 5 0
                2050000000000101010 0000000000000000 5 0
                2150000000000010101 0000000000000000 5 0
                2170000000000010101 0000000000000000 6 1
```

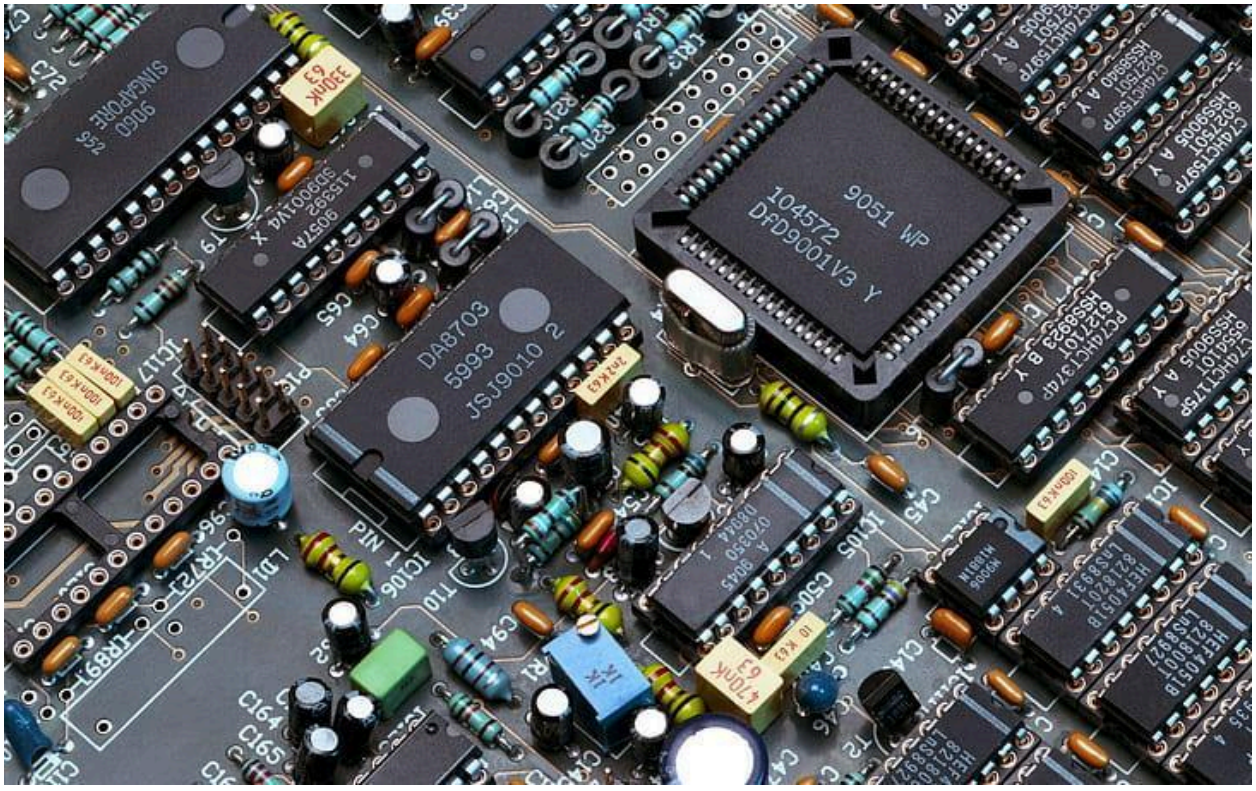
```
                2250000000000001010 0000000000000000 6 1
                2350000000000000101 0000000000000000 6 1
                2450000000000000010 0000000000000000 6 1
.\BoothMultiplier_tb.v:28: $finish called at 250 (1s)
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_03> gtkwave test_booth.vcd

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

[0] start time.
[250] end time.
```

Gtkwave:



Conclusion:

CS - 268 | Lab Report



Report File | Week - 04

Aim:

Design and implementation of memory in Verilog

Objective:

● Design & simulate a memory which has a 5 bit address line and 32 bit data line, the given memory has two read ports & one write port. It is mandatory to use the positive edge of the clock pulse to write data in the memory but the reading of data from memory has no such condition.

● It is recommended to use continuous assignment for reading the data from memory and procedural assignment for writing the data in memory.

In this experiment we (202251069)Malaika Varshney, (202251096)Pradumya Gorav and (202251095)Sourabh Patil have analyzed the code part and created a testbench in this experiment.

```
module regbank_v4 (
    rdData1,rdData2,wrData,sr1,sr2,dr,write,reset,clk);
    input clk,write,reset;
    input [4:0] sr1,sr2,dr;
    input [31:0] wrData;
    output [31:0] rdData1,rdData2;
    integer k;
    output reg [31:0] regfile[0:31];
    always @(posedge clk ) begin
        begin
            if (reset) begin
                for (k =0 ;k<32 ;k=k+1 ) begin
                    regfile[k] <= 0;
                end
            end
            else if (write) begin
                regfile[dr] <= wrData;
            end
        end
    end
    assign rdData1=regfile[sr1];
    assign rdData2=regfile[sr2];
endmodule
```

Test Bench:

```
module regbank_v4_tb();

    reg clk,write,reset;

    reg [4:0] sr1,sr2,dr;

    reg [31:0] wrData;

    wire [31:0] rdData1,rdData2;

    wire [31:0] regfile[31:0];

    integer k;


    regbank_v4 uut(rdData1,rdData2,wrData,sr1,sr2,dr,write,reset,clk);


    always  #5  clk=~clk;


    initial begin

        clk=0;

        $dumpfile("test.vcd");

        $dumpvars(0,regbank_v4_tb);

        #7 reset=1;

        #10 reset=0;

        for (k = 0;k<32 ;k=k+1 ) begin

            #5

            write=1;

            dr=k;

            wrData=10*k+1;

            #5 write=0;

        end
```

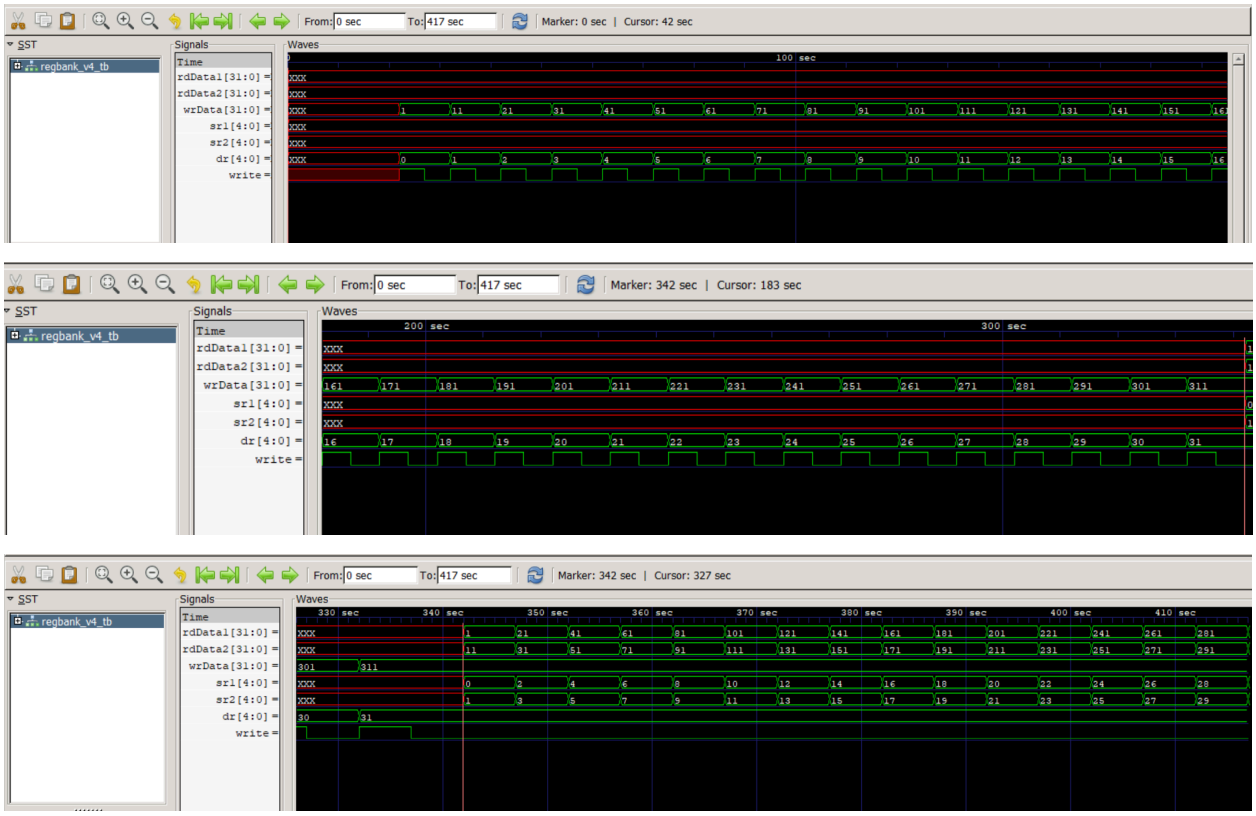
```
        for (k =0 ;k<32 ;k=k+2) begin
            #5
            sr1=k;
            sr2=k+1;
            $monitor("%d %d",rdData1,rdData2);
            $dumpvars(regfile[k]);
        end
         $finish();
    end
endmodule
```

Terminal:

```
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_04> iverilog -o test .\lab_04_c.v
.\lab_04_c_tb.v
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_04> vvp test
VCD info: dumpfile test.vcd opened for output.
rdData1=          1 rdData2=         11
rdData1=         21 rdData2=         31
rdData1=         41 rdData2=         51
rdData1=         61 rdData2=         71
rdData1=         81 rdData2=         91
rdData1=        101 rdData2=        111
rdData1=        121 rdData2=        131
rdData1=        141 rdData2=        151
rdData1=        161 rdData2=        171
rdData1=        181 rdData2=        191
rdData1=        201 rdData2=        211
rdData1=        221 rdData2=        231
rdData1=        241 rdData2=        251
rdData1=        261 rdData2=        271
rdData1=        281 rdData2=        291
.\lab_04_c_tb.v:33: $finish called at 417 (1s)
rdData1=        301 rdData2=        311
```

```
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_04> gtkwave .\test.vcd

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

[0] start time.
[417] end time.
```

Gtkwave:

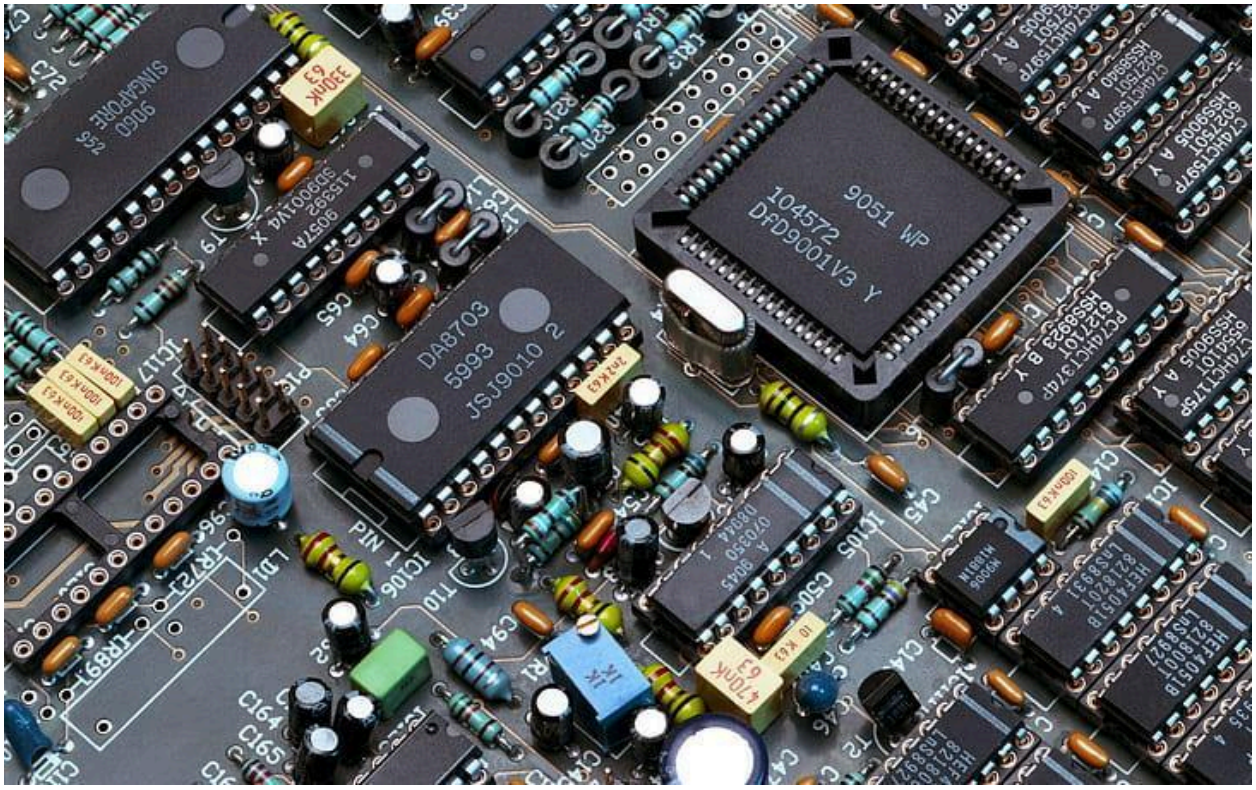


Conclusion:

- The clock signal toggles every 5 time units.
- The reset signal transitions from 2'b00 to 2'b01 to 2'b10 at time intervals of 5 time units.
- The state machine transitions through s0, s1, and s2 in response to the clock signal.

Within GTKWave, it is possible to view waveforms depicting signals like clock, reset, and light. The patterns displayed should demonstrate the state machine cyclically transitioning through its states, potentially with an initial reset. The light signal's waveform is expected to mirror the color corresponding to each state.

CS - 268 | Lab Report



Report File | Week - 05

Aim:

Design and implementation of pipeline in Verilog

Objective:

● Design & simulate a memory which has a 5 bit address line and 32 bit data line, the given memory has two read ports & one write port. It is mandatory to use the positive edge of the clock pulse to write data in the memory but the reading of data from memory has no such condition.

● It is recommended to use continuous assignment for reading the data from memory and procedural assignment for writing the data in memory.

In this experiment we (202251069)Malaika Varshney, (202251096)Pradumya Gorav and (202251095)Sourabh Patil have analyzed the code part and created a testbench in this experiment.

```
module pipe_ex (

    F,A,B,C,D,clk

);

    parameter N = 10;

    input [N-1:0] A,B,C,D;

    input clk;

    output [N-1:0] F;



    reg [N-1:0] L12_x1,L12_x2,L12_D,L23_x3,L23_D,L34_F;



    assign F=L34_F;

    always @(posedge clk )

        begin

            L12_x1 <= #4A+B;

            L12_x2 <= #4C-D;

            L12_D  <=D;



            L23_x3 <= #4L12_x1+L12_x2;

            L23_D  <=L12_D;



            L34_F  <= #6L23_x3*L23_D;

    end

endmodule
```

Test Bench:

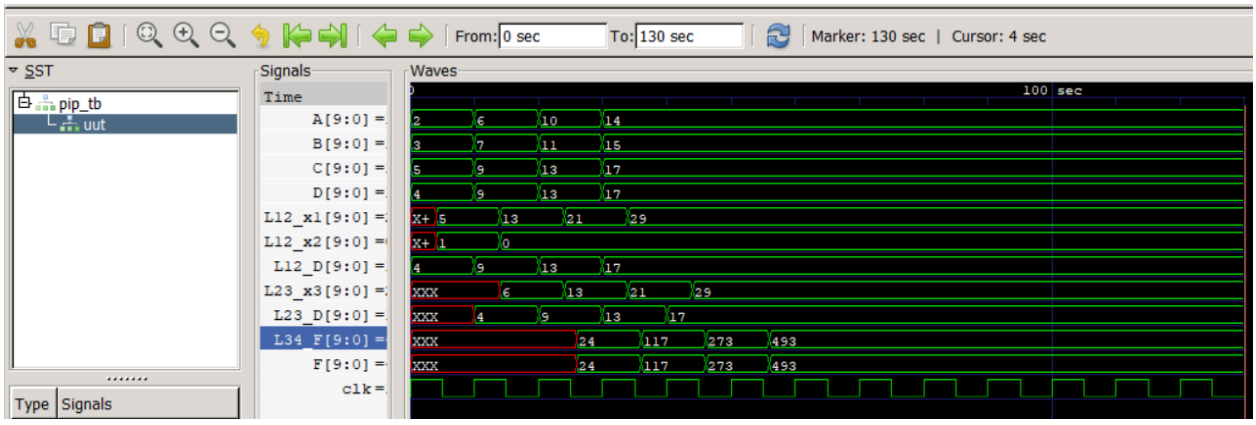
```
module pip_tb();

    parameter N = 10;

    reg [N-1:0] A,B,C,D;

    reg clk;

    wire [N-1:0] F;


    pipe_ex uut (F,A,B,C,D,clk);


    always #10 clk = ~clk;


    initial begin

        $dumpfile("pip_tb.vcd");

        $dumpvars(0,pip_tb);

        clk=1;

        A=2;B=3;C=5;D=4;

        $monitor("A=%d B=%d C=%d D=%d F=%d",A,B,C,D,F);

        #10;

        A=6;B=7;C=9;D=9;

        $monitor("A=%d B=%d C=%d D=%d F=%d",A,B,C,D,F);

        #10;

        A=10;B=11;C=13;D=13;

        $monitor("A=%d B=%d C=%d D=%d F=%d",A,B,C,D,F);

        #10;

        A=14;B=15;C=17;D=17;

        $monitor("A=%d B=%d C=%d D=%d F=%d",A,B,C,D,F);
```

```
        #100;

        $finish();

    end

endmodule
```

Terminal:

```
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_05> iverilog -o test .\pip.v pip_tb.v
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_05> vvp test
VCD info: dumpfile pip_tb.vcd opened for output.
A=   2 B=   3 C=   5 D=   4 F=   x
A=   6 B=   7 C=   9 D=   9 F=   x
A=  10 B=  11 C=  13 D=  13 F=   x
A=  10 B=  11 C=  13 D=  13 F=  24
A=  14 B=  15 C=  17 D=  17 F=  24
A=  14 B=  15 C=  17 D=  17 F= 117
A=  14 B=  15 C=  17 D=  17 F= 273
A=  14 B=  15 C=  17 D=  17 F= 493
pip_tb.v:27: $finish called at 130 (1s)
PS D:\IMPORTANT\SEM-4\LAB\CS268\Lab_05> gtkwave .\pip_tb.vcd

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

[0] start time.
[130] end time.
```

Gtkwave:



Conclusion:

- The clock signal toggles every 5 time units.
- The reset signal transitions from 2'b00 to 2'b01 to 2'b10 at time intervals of 5 time units.
- The state machine transitions through s0, s1, and s2 in response to the clock signal.

Within GTKWave, it is possible to view waveforms depicting signals like clock, reset, and light. The patterns displayed should demonstrate the state machine cyclically transitioning through its states, potentially with an initial reset. The light signal's waveform is expected to mirror the color corresponding to each state.

CS-268

MID-SEM REPORT FILE COMPLETED

THANK YOU