A Basic Calculator

CSN-221 : Computer Architecture and Microprocessors
Coding Project - 1

OCTOBER 8, 2018

SUBMITTED TO
Dr. Sudip Roy

SUBMITTED BY
Gopi Kishan (17114035) : Group ID - 61
Natansh Mathur (17114051) : Group ID - 64
Harsh Kumar Bhartiya (17114036) : Group ID - 61
Chirag Dalmia (17114022) : Group ID - 64

A Brief Description

This Calculator project comprises of a Python (Tkinter) based Graphical User Interface (GUI) and an implementation of the Arithmetic and Logic Unit (ALU) in Verilog. First of all, the user gives input in the GUI based calculator which is then stored in an input text file in the format of Operation Code - First Operand - Second Operand. Now, we simulate the behavioral model of the test bench in ISE Project Navigator. This leads to the creation of the waveform in ISim GUI. Moreover, it stores the output in another text file. The GUI based Calculator now reads the output and displays it to the user.

The ALU module takes 3 input arguments and gives 1 output. The input consists of one 2-bit Op-Code (ALU_Op) and two 4-bit Operands (A and B). Now, according to the Op-Code, the operation is performed on the 2 Operands - 00 for Addition, 01 for Subtraction, 10 for OR, 11 for 2s Complement for which, we have used the switch case. The output is in ALU-Out register which is also a 4-bit register.

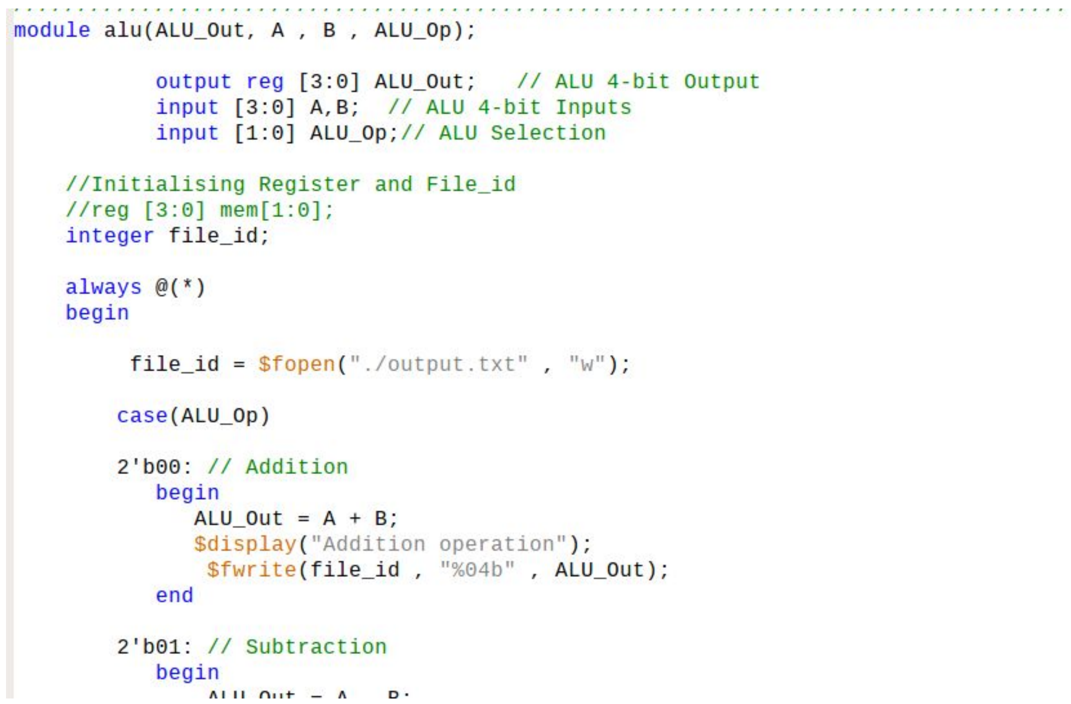
Steps for Running

The following steps are described for Linux (Ubuntu 18.04):

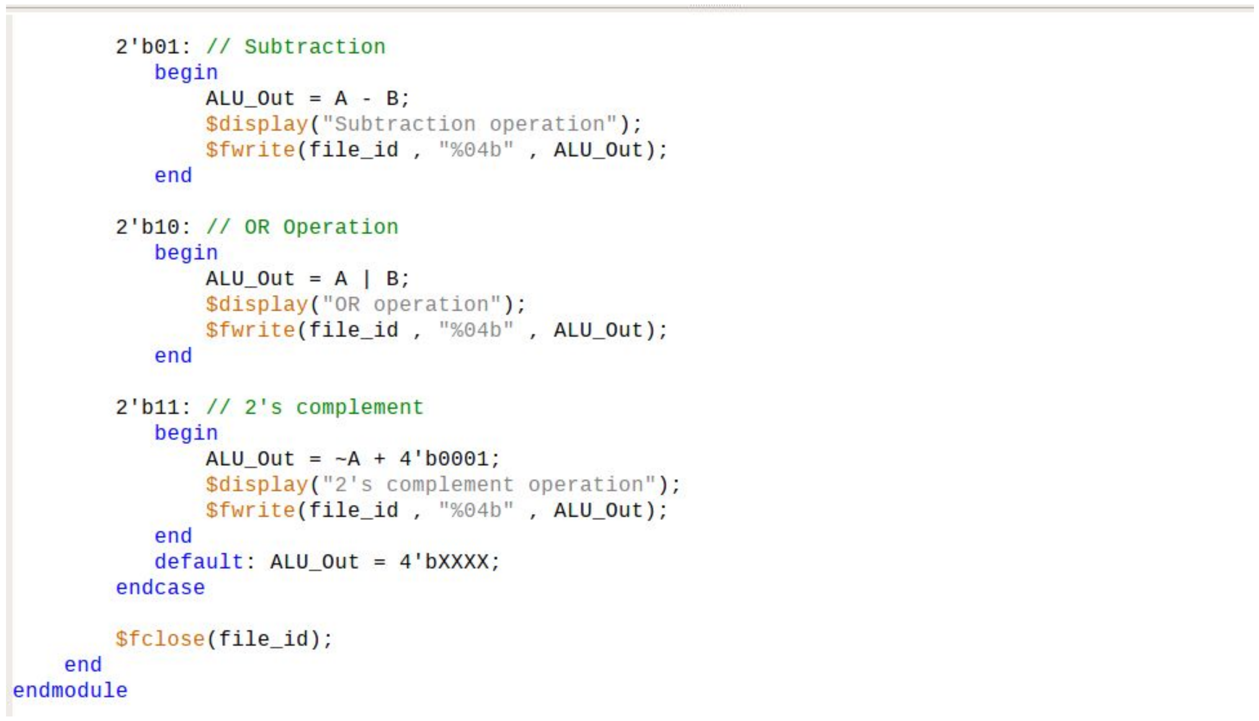
1. Run the Python based Calculator GUI in the terminal by running : ‘python Calc.py’.
2. Click on Operand 1 digit button. Click on Operation button. Click on Operand 2 digit button. (Do not click on ‘Equals to’ button yet.)
3. Open the ISE Design Suite Calculator project, select testbench(tb_alu.v) and then click on Simulate Behaviour Model option to generate output text file.
4. Now, click on the ‘Equals to (=)’ button in GUI to get the answer displayed.

Verilog Code

ALU MODULE:

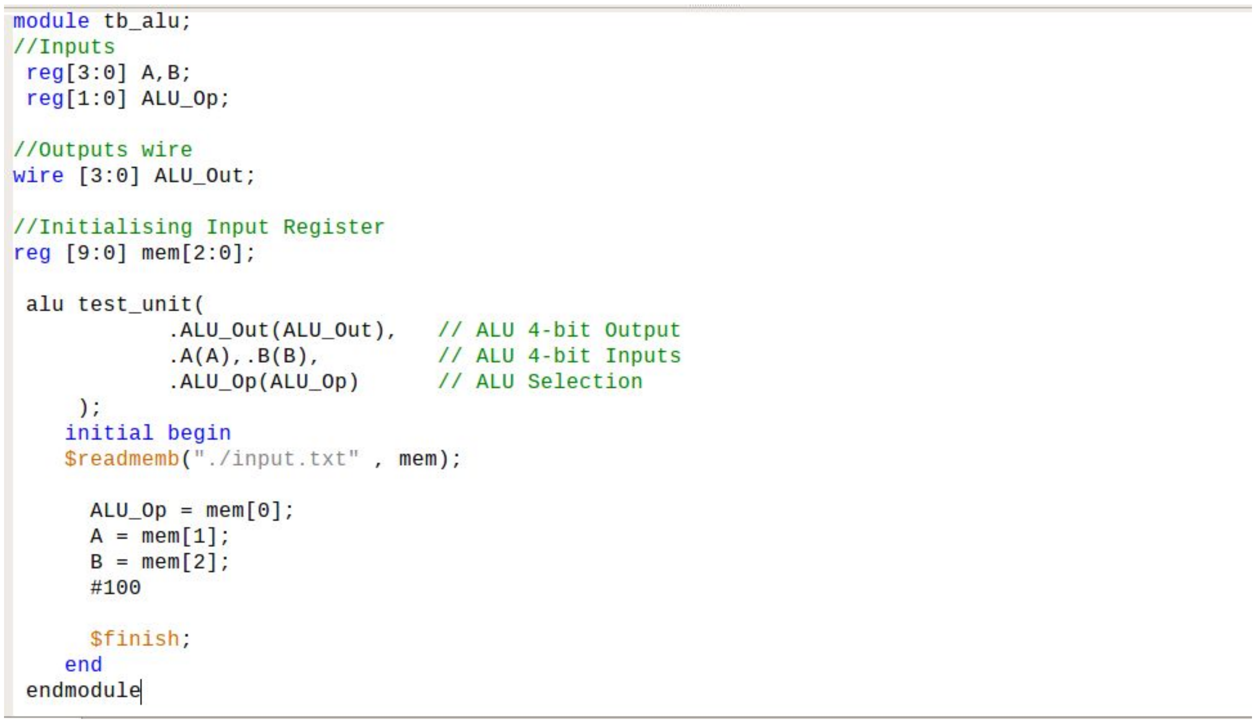


```
module alu(ALU_Out, A , B , ALU_Op);

           output reg [3:0] ALU_Out;   // ALU 4-bit Output
           input [3:0] A,B;  // ALU 4-bit Inputs
           input [1:0] ALU_Op;// ALU Selection

    //Initialising Register and File_id
    //reg [3:0] mem[1:0];
    integer file_id;

    always @(*)
    begin

         file_id = $fopen("./output.txt" , "w");

        case(ALU_Op)

        2'b00: // Addition
           begin
              ALU_Out = A + B;
              $display("Addition operation");
               $fwrite(file_id , "%04b" , ALU_Out);
           end

        2'b01: // Subtraction
           begin
```



```
        2'b01: // Subtraction
           begin
               ALU_Out = A - B;
               $display("Subtraction operation");
               $fwrite(file_id , "%04b" , ALU_Out);
           end

        2'b10: // OR Operation
           begin
               ALU_Out = A | B;
               $display("OR operation");
               $fwrite(file_id , "%04b" , ALU_Out);
           end

        2'b11: // 2's complement
           begin
               ALU_Out = ~A + 4'b0001;
               $display("2's complement operation");
               $fwrite(file_id , "%04b" , ALU_Out);
           end
           default: ALU_Out = 4'bXXXX;
        endcase

        $fclose(file_id);
    end
endmodule
```

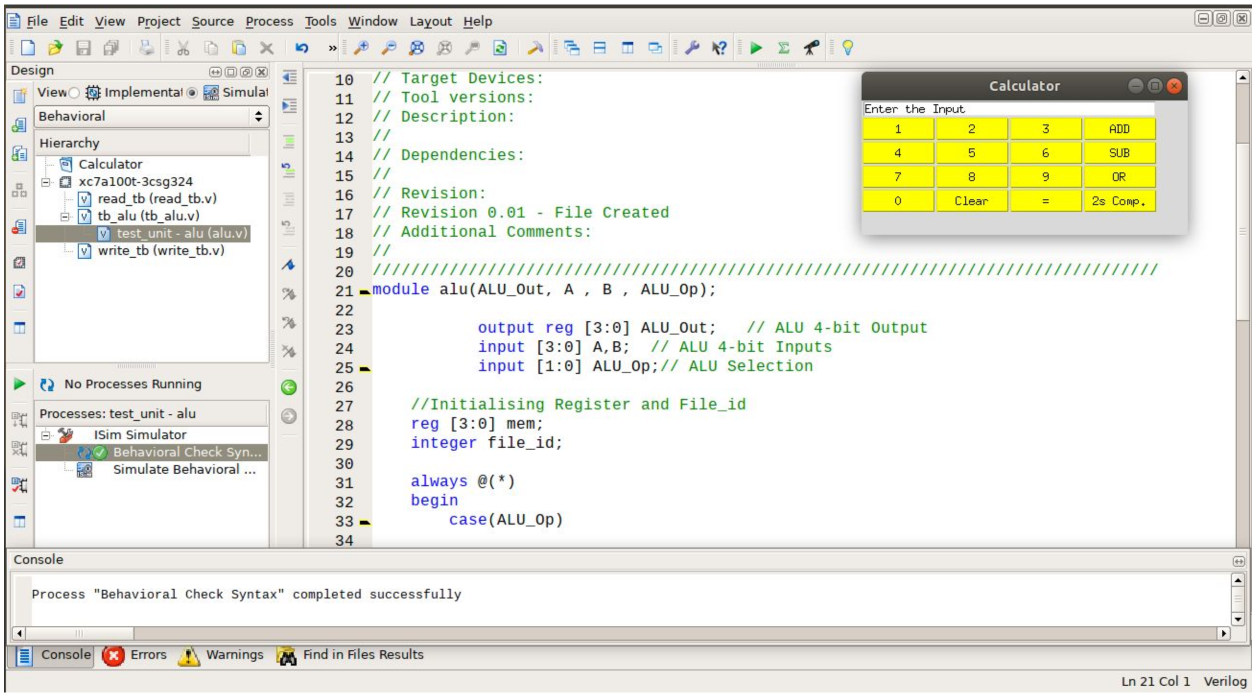
TEST BENCH :



```
module tb_alu;
//Inputs
 reg[3:0] A,B;
 reg[1:0] ALU_Op;

//Outputs wire
wire [3:0] ALU_Out;

//Initialising Input Register
reg [9:0] mem[2:0];

 alu test_unit(
            .ALU_Out(ALU_Out),   // ALU 4-bit Output
            .A(A),.B(B),         // ALU 4-bit Inputs
            .ALU_Op(ALU_Op)      // ALU Selection
     );
    initial begin
    $readmemb("./input.txt" , mem);

      ALU_Op = mem[0];
      A = mem[1];
      B = mem[2];
      #100

      $finish;
    end
 endmodule
```

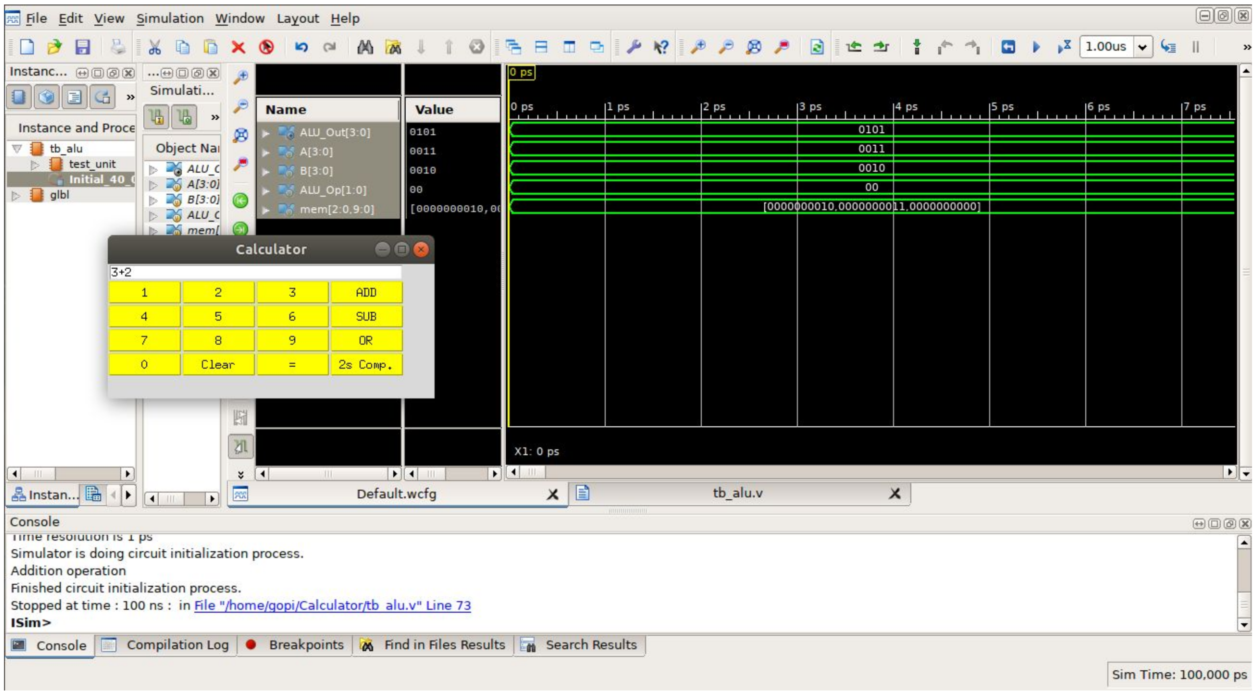
Testing and Simulation

Test Module : tb_alu.v



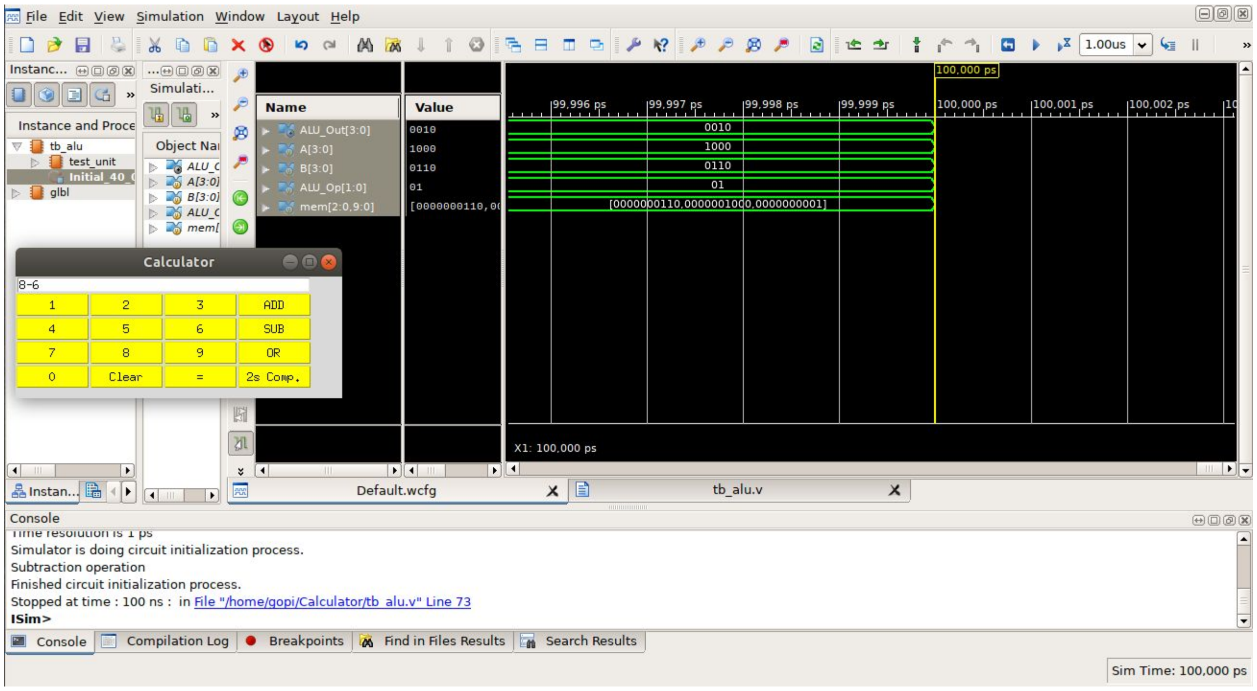
```
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module alu(ALU_Out, A , B , ALU_Op);

           output reg [3:0] ALU_Out;   // ALU 4-bit Output
           input [3:0] A,B;  // ALU 4-bit Inputs
           input [1:0] ALU_Op;// ALU Selection

    //Initialising Register and File_id
    reg [3:0] mem;
    integer file_id;

    always @(*)
    begin
        case(ALU_Op)
```

```
Process "Behavioral Check Syntax" completed successfully
```



Name	Value
ALU_Out[3:0]	0101
A[3:0]	0011
B[3:0]	0010
ALU_Op[1:0]	00
mem[2:0,9:0]	[0000000010,00

```
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Addition operation
Finished circuit initialization process.
Stopped at time : 100 ns :  in File "/home/gopi/Calculator/tb_alu.v" Line 73
ISim>
```



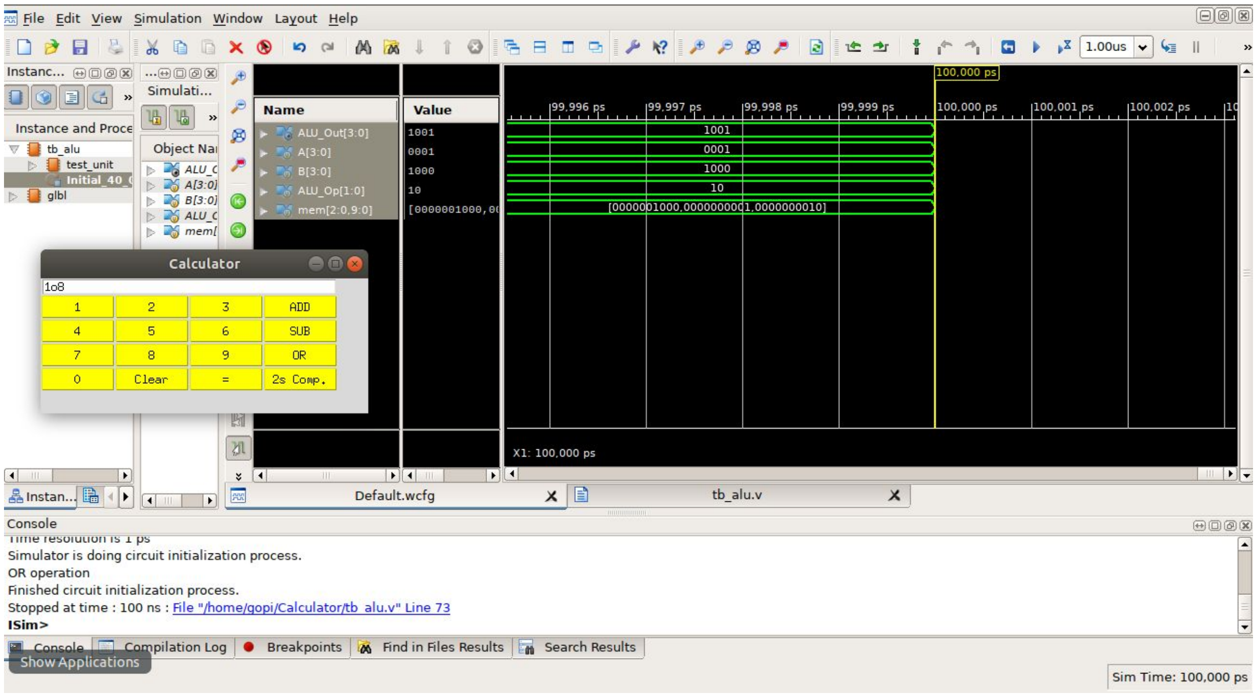
Name	Value
ALU_Out[3:0]	0010
A[3:0]	1000
B[3:0]	0110
ALU_Op[1:0]	01
mem[2:0,9:0]	[0000000110,0000001000,0000000001]

Calculator

```
8-6
1	2	3	ADD
4	5	6	SUB
7	8	9	OR
0	Clear	=	2s Comp.
```

```
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Subtraction operation
Finished circuit initialization process.
Stopped at time : 100 ns :  in File "/home/gopi/Calculator/tb_alu.v" Line 73
ISim>
```

Sim Time: 100,000 ps



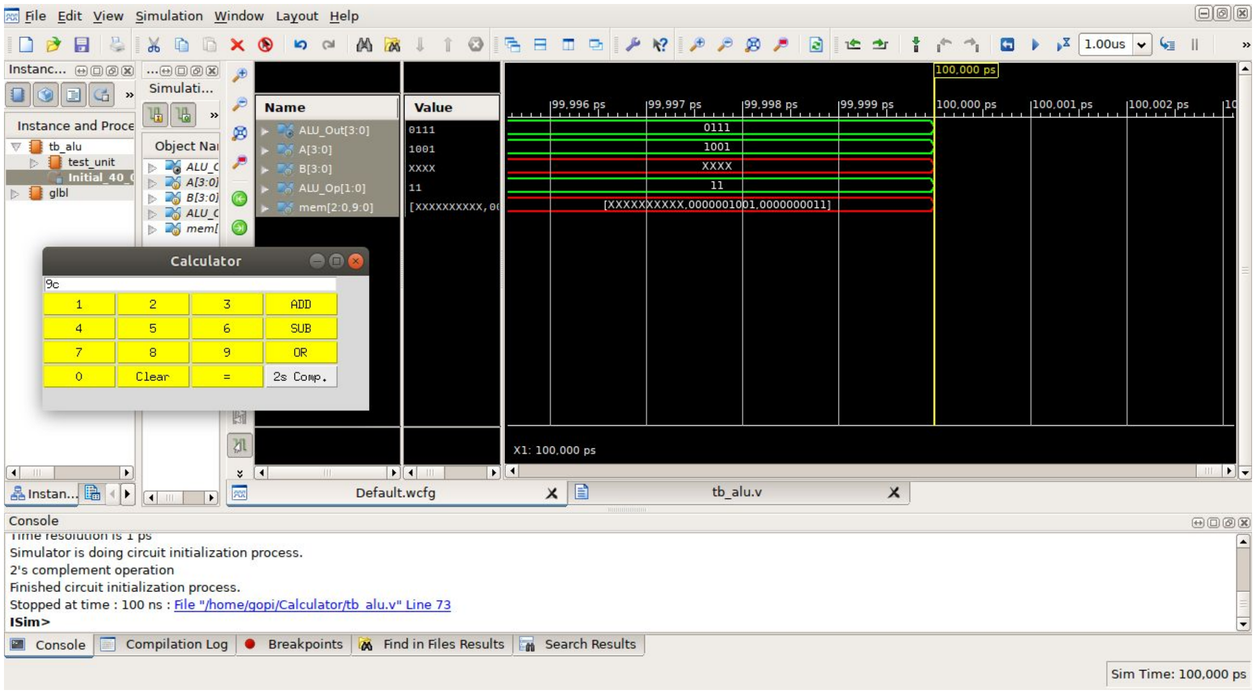
Name	Value
ALU_Out[3:0]	1001
A[3:0]	0001
B[3:0]	1000
ALU_Op[1:0]	10
mem[2:0,9:0]	[0000001000,0000000001,0000000010]

Calculator

```
1o8
1	2	3	ADD
4	5	6	SUB
7	8	9	OR
0	Clear	=	2s Comp.
```

```
Time resolution is 1 ps
Simulator is doing circuit initialization process.
OR operation
Finished circuit initialization process.
Stopped at time : 100 ns : File "/home/gopi/Calculator/tb_alu.v" Line 73
ISim>
```

Sim Time: 100,000 ps



```
9c
```

Name	Value
ALU_Out[3:0]	0111
A[3:0]	1001
B[3:0]	XXXX
ALU_Op[1:0]	11
mem[2:0,9:0]	[XXXXXXXXXX,0000001001,0000000011]

```
Time resolution is 1 ps
Simulator is doing circuit initialization process.
2's complement operation
Finished circuit initialization process.
Stopped at time : 100 ns : File "/home/gopi/Calculator/tb_alu.v" Line 73
ISim>
```

Sim Time: 100,000 ps

What We learnt from the Project

From this project, we were introduced to programming of ALUs in actual code. We got to know how data is communicated from the GUI to the ALU, processed there and returned back to the GUI. This very concept of how such a simple design can be used to handle a practical problem was quite intriguing.
Moreover, we got a First-Hand experience of using the Xilinx ISE Project Navigator and implementing a module using Verilog Code. We got to experience how we could collaborate as a team and learn things from each other to pull out a Project together.

Thank You!