Jyresa Mae M. Amboang

Frances Nicole S. Gigataras

Crishelle A. Agopitac

Erica Dianne Q. Canillo

Laboratory Submitted: June 13, 2023

Laboratory Hours: 336 hours

# Laboratory Activity No. 5

**32-Bit ALU and Testbench 110 Points**

**Laboratory Activity Overview**

In this laboratory activity, you will design the 32-bit Arithmetic Logic Unit (ALU) that is described in Section 5.2.4 of the text. Your ALU will become an important part of the MIPS microprocessor that you will build in later laboratory activity. In this laboratory activity you will design an ALU in SystemVerilog. You will also write a SystemVerilog testbench and test vector file to test the ALU.

**SystemVerilog code**

You will only be doing ModelSim simulation in this lab, so you can use your favorite text editor (such as WordPad) instead of Quartus if you like.

Create a 32-bit ALU in SystemVerilog. Name the file alu.sv. It should have the following module declaration:

module alu(input logic [31:0] a, b,

input logic [2:0] f, output logic [31:0] y, output logic zero);

The output zero should be TRUE if y is equal to zero. An adder is a relatively expensive piece of hardware. Be sure that your design uses no more than one adder.

# 

# Simulation and Testing

Now you can test the 32-bit ALU in ModelSim. It is prudent to think through a set of input vectors. Develop an appropriate set of test vectors to convince a reasonable person that your design is probably correct. Complete Table 1 to verify that all 5 ALU operations work as they are supposed to. Note that the values are expressed in **hexadecimal** to reduce the amount of writing.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **F[2:0]** | **A** | **B** | **Y** | **Zero** |
| ADD 0+0 | 2 | 00000000 | 00000000 | 00000000 | 1 |
| ADD 0+(-1) | 2 | 00000000 | FFFFFFFF | FFFFFFFF | 0 |
| ADD 1+(-1) | 2 | 00000001 | FFFFFFFF | 00000000 | 1 |
| ADD FF+1 | 2 | 000000FF | 00000001 | 00000100 | 0 |
| SUB 0-0 | 6 | 00000000 | 00000000 | 00000000 | 1 |
| SUB 0-(-1) | 6 | 00000000 | FFFFFFFF | 00000001 | 0 |
| SUB 1-1 | 6 | 00000001 | 00000001 | 00000000 | 1 |
| SUB 100-1 | 6 | 00000100 | 00000001 | 000000FF | 0 |
| SLT 0,0 | 7 | 00000000 | 00000000 | 00000000 | 1 |
| SLT 0,1 | 7 | 00000000 | 00000001 | 00000001 | 0 |
| SLT 0,-1 | 7 | 00000000 | FFFFFFFF | 00000000 | 1 |
| SLT 1,0 | 7 | 00000001 | 00000000 | 00000000 | 1 |
| SLT -1,0 | 7 | FFFFFFFF | 00000000 | 00000001 | 0 |
| AND FFFFFFFF, FFFFFFFF | 0 | FFFFFFFF | FFFFFFFF | FFFFFFFF | 0 |
| AND FFFFFFFF, 12345678 | 0 | FFFFFFFF | 12345678 | 12345678 | 0 |
| AND 12345678, 87654321 | 0 | 12345678 | 87654321 | 2244220 | 0 |
| AND 00000000, FFFFFFFF | 0 | 00000000 | FFFFFFFF | 00000000 | 0 |
| OR FFFFFFFF, FFFFFFFF | 1 | FFFFFFFF | FFFFFFFF | FFFFFFFF | 0 |
| OR 12345678, 87654321 | 1 | 12345678 | 87654321 | 97755779 | 0 |
| OR 00000000, FFFFFFFF | 1 | 00000000 | FFFFFFFF | FFFFFFFF | 0 |
| OR 00000000, 00000000 | 1 | 00000000 | 00000000 | 00000000 | 1 |

**Table 1. ALU operations**

Build a self-checking testbench to test your 32-bit ALU and this is inside the alu.tv with all your vectors in Table 1.

**ALU.tv file**

2\_00000000\_00000000\_00000000\_1

2\_00000000\_FFFFFFFF\_FFFFFFFF\_0

2\_00000001\_FFFFFFFF\_00000000\_1

2\_000000FF\_00000001\_00000100\_0

6\_00000000\_00000000\_00000000\_1

6\_00000000\_FFFFFFFF\_00000001\_0

6\_00000001\_00000001\_00000000\_1

6\_00000100\_00000001\_000000FF\_0

7\_00000000\_00000000\_00000000\_1

7\_00000000\_00000001\_00000001\_0

7\_00000000\_FFFFFFFF\_00000000\_1

7\_00000001\_00000000\_00000000\_1

7\_FFFFFFFF\_00000000\_00000001\_0

0\_FFFFFFFF\_FFFFFFFF\_FFFFFFFF\_0

0\_FFFFFFFF\_12345678\_12345678\_0

0\_12345678\_87654321\_02244220\_0

0\_00000000\_FFFFFFFF\_00000000\_1

1\_FFFFFFFF\_FFFFFFFF\_FFFFFFFF\_0

1\_12345678\_87654321\_97755779\_0

1\_00000000\_FFFFFFFF\_FFFFFFFF\_0

1\_00000000\_00000000\_00000000\_1

**ALU.sv file.**

module alu(input logic [2:0] f,input logic [31:0] a,b, output logic [31:0] y, output logic zero );

logic [31:0] bb, b2, add\_res, and\_res, or\_res, slt\_res;

assign bb = ~b;

assign b2 = f[2] ? bb : b;

assign add\_res = a + b2 + f[2]; // handle 2&#39;s complement

assign and\_res = a &amp; b2;

assign or\_res = a | b2;

assign slt\_res = add\_res[31] ? 32&#39;b1 : 32&#39;b0;

always\_comb

case (f[1:0])

2’b00: y = and\_res;

2’b01: y = or\_res;

2’b10: y = add\_res;

2’b11: y = slt\_res;

endcase

assign zero = (y == 32’b0);

endmodule

**ALU Testbench.sv file**.

module testbench();

logic clk;

logic [31:0] a,b,y, yexpected;

logic [2:0] f;

logic zero, zeroexpected;

logic [31:0] vectornum, errors;

logic [104:0] testvectors [10000:0];

alu dut(f,a,b,y,zero);

always

begin

clk=1; #50; clk=0; #50;

end

initial

begin

$readmemh("alu.tv",testvectors);

vectornum = 0; errors=0;

end

always @(posedge clk)

begin

#1;

f = testvectors[vectornum] [102:100];

a = testvectors[vectornum] [99:68];

b = testvectors[vectornum] [67:36];

yexpected = testvectors[vectornum] [35:4];

zeroexpected = testvectors[vectornum] [0];

end

always @(negedge clk)

begin

if (y !== y\_expected || zero !== zero\_expected) begin

$display(“Error in vector %d”;, vectornum);

$display(“ Inputs : a = %h, b = %h, f = %b”, a, b, f);

$display(&quot; Outputs: y = %h (%h expected), zero = %h (%h expected)&quot;,

y, y\_expected, zero, zero\_expected);

errors = errors+1;

end

vectornum = vectornum + 1;

if (testvectors[vectornum][0] === 1’bx) begin

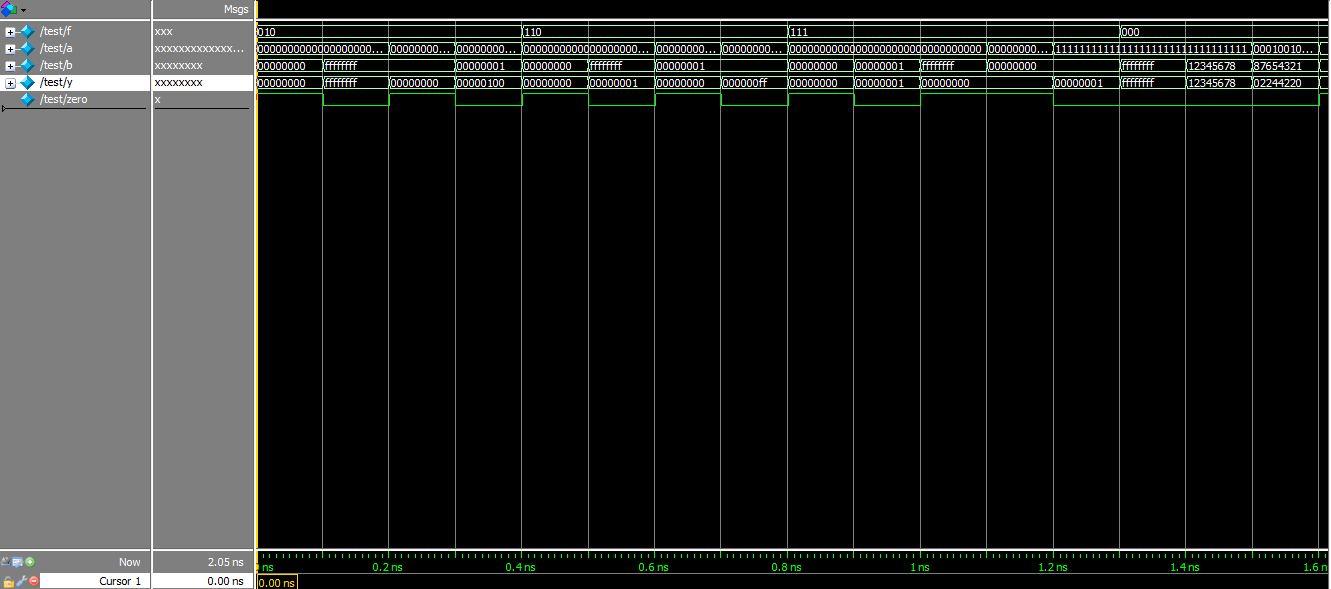
$display(“%d tests completed with %d errors”, vectornum, errors);

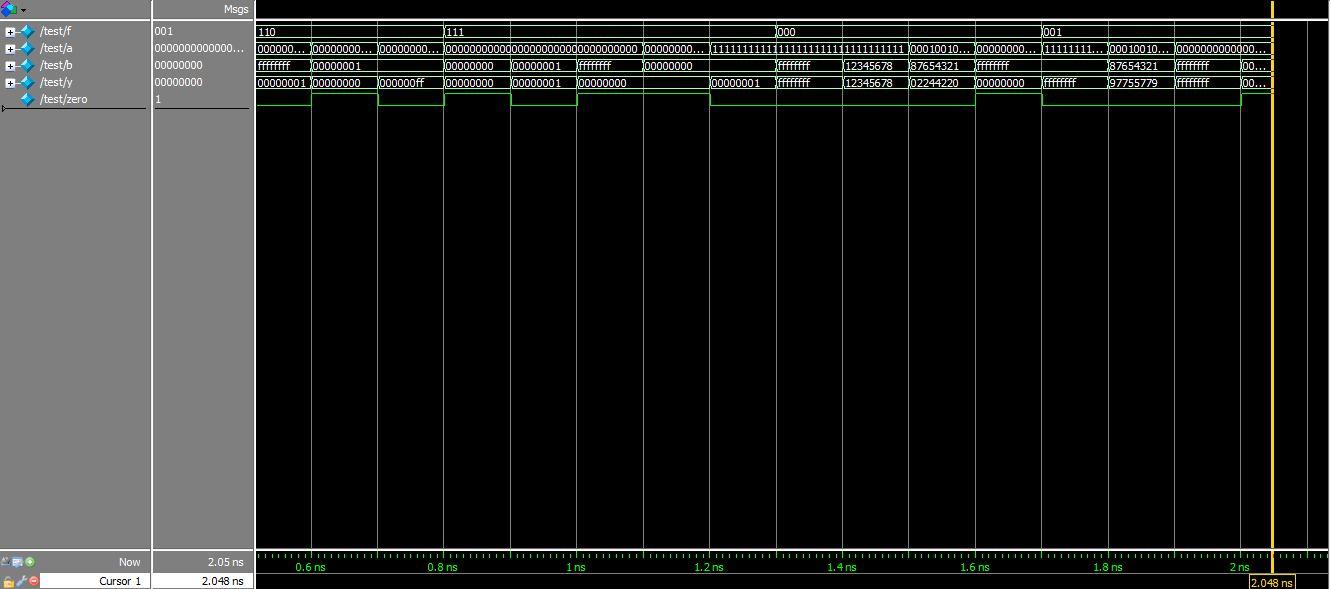
$stop;

end

end

endmodule

**Modelsim Simualtion:**

****

**Conclusion**

What did we learn or take away from performing this laboratory activity?

Creating a 32-bit Arithmetic Logic Unit (ALU) using SystemVerilog helps digital hardware designers create 32-bit Arithmetic Logic Units (ALUs). It enables us to grasp its inner workings by studying the ALU's operation and attributes, teaches hardware arithmetic and logic operations, laying the groundwork for more complicated digital systems and processors. SystemVerilog ALU implementation improves coding and hardware description language knowledge which hardware designs are captured using SystemVerilog that, simplifies code and module definitions. It also aids data type learning and combinational logic, and the encoding feature generates a 32-bit ALU, reducing the requirement for several 1-bit ALUs. Encoding reduces code lines and optimizes hardware use in digital systems. The lab activity taught us how to build an ALU test bench, and the testbench verified ALU inputs by applying test vectors and analyzing outputs. This strategy focuses on thorough testing and guarantees that the Arithmetic Logic Unit (ALU) works reliably across various circumstances. Test benches improve test vector generation and simulation analysis using ModelSim. This popular simulation program helps designers learn industry standards and approaches to solve real-world hardware design problems by simulating digital circuits and evaluating their performance using test vectors. The laboratory activity teaches us digital hardware design, SystemVerilog programming, optimum hardware design, testbench construction, and simulation. These abilities allow us to create microprocessors and other digital devices and improve our digital hardware skills and confidence in the future.