**EECE 2323 Digital Logic Design Lab Report**

Lab 4 Register File

| Student Name:  Ousmane Toure & Kaitlyn O'Flaherty | Section #: 1 |
| --- | --- |
| Instructor Name: Dr. Aboelela  Lab TA Name: Keshav Bharadwaj Vaidyanathan, Srinidi Somasundaram Subbulakshmi, Yuchao Su | Date: 07/29/22 |

1. **Background & Purpose**:

In this experiment, we implemented a processor datapath utilizing a Register file with a zero register and an Arithmetic Logic Unit. Our goal was to implement a sequential logic (logic that holds a state and is timed with a clock signal) into the processor datapath using an ALU, zero register and register file. From our previous labs, our ALU performs all necessary operations to generate new values and needs a register file to remember and store these values in memory. resThe results were then displayed utilizing the VIO Dashboard on Vivado and the LEDs on the PYNQ. Register files are essential because they give central processing units the ability to remember values while it performs arithmetic logic. This is important because it allows the CPU to do functions like addition, division, subtraction etc. The goal of this lab was to utilize verilog code in vivado to create a register file with ALU functions and virtually test it before physically implementing it. Furthermore enhancing confidence and proficiency in vivado and computational architecture.

This lab is important to the scientific community because each processor needs to be able to store values in its memory. This allows the processor to quickly access information, and without memory a computer would not be able to function properly.

1. **Pre-Lab Response:**

**Table 1:**

| **rst** | **rd1\_addr** | **rd2\_addr** | **wr\_addr** | **wr\_data** | **wr\_en** | **Instr\_i** | **ALUSrc2** | **ALUSrc1** | **ALUOp**  **(3’d\_)** | **result** | **input1** | **input2** | **ovf** | **take\_branch** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | x | -x | 0 | 0 | 0 | x | x | x | x | x | x | x | x | x |
| **0** | x | x | 00 | -8’d86  10101010 | 1 | x | x | x | x | x | x | x | x | x |
| **0** | 00 | x | x | x | 0 | x | x | 0 | 001  (inv) | x | -8’d86  10101010 | x | 0 | 0 |
| **0** | x | x | 01 | 8d’85  01010101 | 1 | x | x | x | x | x | x | x | x | x |
| **0** | 00 | 01 | x | x | 0 | x | 0 | 0 | 010  (and) | 8’d0 | -8’d86 | 8d’85 | 0 | 0 |
| **0** | 00 | 01 | x | x | 0 | x | 0 | 0 | 000  (+) | -8’d1 | -8’d86 | 8d’85 | 0 | 0 |
| **0** | x | x | 01 | -8’d86  10101010 | 0 | x | x | x | x | x | x | x | x | x |
| **0** | 00 | 01 | x | - | 0 | x | 0 | 0 | 011  (or) | -8’d1  11111111 | -8’d86 | 8d’85 | 0 | 0 |
| **0** | x | x | 10 | -8’d86  10101010 | 1 | x | x | x | x | x | x | x | x | x |
| **0** | x | x | 11 | -8d’100  10011100 | 1 | x | x | x | x | x | x | x | x | x |
| **0** | 10 | 11 | x | x | 0 | x | 0 | 0 | 000  (+) | 8’d70  01000110 | -8’d86  10101010 | -8d’100  10011100 | 1 | 0 |
| **0** | 10 | 11 | x | x | 0 | -8d’2 11111110 | 1 | 0 | 100  (sra) | -8d’43 11010101 | -8’d86  10101010 | -8d’2 11111110 | 0 | 0 |
| **0** | 10 | 11 | x | x | 0 | x | 0 | 1 | 110  (beq) | 8’d0 | 8’d0 | -8d’100  10011100 | x | 0 |
| **0** | 10 | 11 | x | x | 0 | -8d’2 11111110 | 1 | 1 | 110  (beq) | 8’d0 | -8d’2 11111110 | 0 | 0 | 0 |
| **0** | 10 | 11 | x | x | 0 | -8d’2 11111110 | 1 | 1 | 111  (branch not equal) | 8’d0 | -8d’2 11111110 | 0 | 0 | 1 |

1. **Summary of Design Implementation**
   1. **Results and Analysis:**

When conducting this experiment, the ALU from Lab 3 was used with the addition of two muxes to perform operations on the values that were being written to our register file. The simulation that was run using the Test Bench Code resulted in the correct storage of values in the register file which were then used to complete the following operations of the ALU: addition, bitwise NOT, bitwise AND, and bitwise OR, arithmetic shift right, logical shift left, and branch if equal/ not equal. The output also clearly indicated when overflow occurred in the addition. Each of the data lines read from the register file were put through a respective mux. The first mux used the select line “ALUSrc1” to choose between rd0\_data and a zero register. The second mux used the select line “ALUSrc2” to choose between rd1\_data and instr\_i, which is an arbitrary value. The alu\_register\_file implements the basic mux file, alu file created in lab 3, and the new register file just created. All files can be found in Appendix A. The test bench utilized this alu\_register\_file to test each of the wires within the circuit. The result of the simulation being run is the Test Bench WaveForm Simulation which clearly shows the reset, rd1\_addr(read address 1), rd2\_addr(read address 2), wr\_addr(write address), wr\_data(write data), wr\_en(enable), Instr\_i, ALUSrc2, ALUSrc1, ALUOp, result, input1(result of mux 1), input2(result of mux 2), the output, whether there is overflow or not, and the branch if equal/not equal. The results of the test bench verified that our code in fact does work and gives us the green light to program straight into the PYNQ-Z2 board. After programming our board and connecting the add on board, we opened the virtual input output (VIO) dashboard which allowed us to test our values as our PYNQ board did not have enough physical inputs. Storing values in the register and all of the necessary ALU outputs were tested using the VIO dashboard and the resulting screenshots were placed in Appendix B. All tests resulted in values that were consistent with our pre lab test bench truth table highlighting that our circuit was in fact correct. You could face many errors when conducting this lab. For example, when creating your virtual input output, if you didn’t correctly instantiate your values in your top file, Vivado will leave you with constants with no values.

* 1. **Conclusion & Recommendations:**

Based on our results, we can conclude that Lab 4 consists of creating a Register file that can do basic arithmetic operations utilizing our Arithmetic Logical Unit from Lab 3. Testing its implementation virtually with a test bench confirmed that our verilog code was correct which gave us the green light to program the PYNQ Board using Virtual Input Output (VIO) ports. . The lab resulted in successfully being able to perform different arithmetic and logical functions like addition and bit shifting, BNE, as well as checking if values were equal. These tools are crucial for a Central Processing Unit, and gives it the ability to perform varying tasks of different degrees. Completing this lab showed that the clock is an essential part of a sequential logic circuit. This is because it provides the register file with a sense of time to determine when the data input will be looked at, this is necessary so that the register file isn’t reading data it is not supposed to at incorrect times. Furthermore, the zero register was very resourceful as it made setting all register values to zero, making the testability of the register file much more efficient.

Recommendations going forward would be to give a set of test vectors that is standard across the class, this allows more collaboration between groups and students. Furthermore, not using all the ALU functions as long as all the wires are tested in the circuit would speed up the lab while still ensuring accuracy.

**Appendix A: Design Program Files (Verilog modules, testbenches, etc)**

**Verilog Code:**

**ALU REGFILE:**

module alu\_regfile(input1,

zero\_reg,

alu\_src1,

input2,

wr\_data,

instr\_i,

alu\_src2,alu\_op,alu\_result,ovf,take\_branch,rd0\_addr, rd1\_addr, wr\_addr,wr\_en, clk, rst);

wire [7:0] rd0\_data, rd1\_data;

output [7:0] input1,input2,alu\_result;

input wire[8:0] wr\_data;

input wire[7:0] zero\_reg,instr\_i;

input wire [1:0] alu\_src1,alu\_src2, rd0\_addr, rd1\_addr, wr\_addr;

input wire [2:0] alu\_op;

output ovf,take\_branch;

input wire wr\_en, clk, rst;

regfile rf(.rd0\_data(rd0\_data),.rd1\_data(rd1\_data),.wr\_data(wr\_data),.rd0\_addr(rd0\_addr),.rd1\_addr(rd1\_addr),.wr\_addr(wr\_addr),.wr\_en(wr\_en),.clk(clk),.rst(rst));

Mux m1(.in1(rd0\_data),.sel(alu\_src1),.in2(zero\_reg),.out(input1)); //instantiate template

Mux m2(.in1(rd1\_data),.sel(alu\_src2),.in2(instr\_i),.out(input2)); //instantiate template

eightbit\_alu a1(.a(input1),.b(input2),.sel(alu\_op),.f(alu\_result),.ovf(ovf),.take\_branch(take\_branch));

endmodule

**REGFILE**

module regfile #(parameter WordSize = 8, AddrSize = 2)(

output [WordSize-1: 0] rd0\_data, rd1\_data,

input [WordSize: 0] wr\_data,

input [AddrSize-1: 0] rd0\_addr, rd1\_addr, wr\_addr,

input wr\_en, clk, rst);

//32bit x32words memory

reg [WordSize-1:0] RegFile[2\*\*AddrSize-1:0];

integer i = 0; //for the loop

assign rd0\_data = RegFile[rd0\_addr];

assign rd1\_data = RegFile[rd1\_addr];

always @ (posedge clk, posedge rst)

if (rst) //clear the contents of all registers

for (i=0; i < 2\*\*AddrSize; i=i+1) RegFile[i] <= 0;

else if (wr\_en == 1'b1) RegFile[wr\_addr] <= wr\_data;

endmodule

**MUX CODE:**

**EIGHTBIT ALU:**

module eightbit\_alu(

input signed [7:0] a,

input signed[7:0] b,

input [2:0] sel,

output reg [7:0] f,

output reg ovf,

output reg take\_branch

);

always@(a or b or sel)

begin

case(sel)

3'b000: //add

begin

f = a + b;

ovf =((a[7] & b[7]) &!f[7]) | ((!a[7] & !b[7])&f[7]);

take\_branch = 0;

end

3'b001: //b inversion

begin

f= !b;

ovf = 0;

take\_branch = 0;

end

3'b010: //AND

begin

f = a \* b;

ovf = 0;

take\_branch = 0;

end

3'b011: //OR

begin

f = a | b;

ovf = 0;

take\_branch = 0;

end

3'b100: //Shift right

begin

f = a >>> 1;

ovf = 0;

take\_branch = 0;

end

3'b101: //shift left

begin

f = a << b;

ovf = 0;

take\_branch = 0;

end

3'b110: //branch if equal

begin

f = 0;

ovf = 0;

take\_branch = (a == b);

end

3'b111: //Branch if not equal

begin

f = 0;

ovf = 0;

take\_branch = (a != b);

end

default:

begin

f = 0;

take\_branch=0;

ovf =0;

end

endcase

end

endmodule

**TOP FILE:**

`timescale 1ns / 1ps

// Create Date: 10/06/2020 04:43:08 PM

// Module Name: alu\_regfile\_vio\_top

// By: Ousmane Toure & Kaitlyn O’Flaherty

//////////////////////////////////////////////////////////////////////////////////

module alu\_regfile\_vio\_top(

input clk, // clock for vio and RegFile

input reset, // BTN0 for for reset

output [7:0] led, // add-on board led[5:0], + LD0, LD1

output wire ovf\_ctrl // LD3

);

wire [7:0] alu\_input1, alu\_input2, alu\_input2\_instr\_src;

wire [7:0] alu\_output;

wire [2:0] ALUOp;

wire alu\_ovf;

wire take\_branch;

wire RegWrite; //Write enable

wire [1:0] regfile\_ReadAddress1; //source register1 address

wire [1:0] regfile\_ReadAddress2; //source register2 address

wire [1:0] regfile\_WriteAddress; //destination register address

wire [8:0] regfile\_WriteData; //result data

wire [8:0] regfile\_ReadData1; //source register1 data

wire [8:0] regfile\_ReadData2; //source register2 data

wire ALUSrc1, ALUSrc2;

reg [7:0] zero\_register = 0;

// Assign LEDs

assign led = alu\_output;

assign ovf\_ctrl = alu\_ovf;

// Instantiate RegFile module here

regfile rf(

.rd0\_data(regfile\_ReadData1),

.rd1\_data(regfile\_ReadData2),

.wr\_data(regfile\_WriteData),

.rd0\_addr(regfile\_ReadAddress1),

.rd1\_addr(regfile\_ReadAddress2),

.wr\_addr(regfile\_WriteAddress),

.wr\_en(RegWrite),

.clk(clk),

.rst(reset)

);

// Instantiate Muxes here

Mux m1(

.in1(regfile\_ReadData1),

.sel(ALUSrc1),

.in2(zero\_register),

.out(alu\_input1)

);

Mux m2(

.in1(regfile\_ReadData2),

.sel(ALUSrc2),

.in2(alu\_input2\_instr\_src),

.out(alu\_input2)

);

// Instantiate ALU module here

eightbit\_alu al(

.sel(ALUOp),

.a(alu\_input1),

.b(alu\_input2),

.f(alu\_output),

.ovf(alu\_ovf),

.take\_branch(take\_branch)

);

// Instantiate VIO module here

vio\_0 vio(

.clk(clk),

.probe\_in0(alu\_output),

.probe\_in1(alu\_ovf),

.probe\_in2(take\_branch),

.probe\_in3(regfile\_ReadData1),

.probe\_in4(regfile\_ReadData2),

.probe\_in5(alu\_input1),

.probe\_in6(alu\_input2),

.probe\_in7(reset),

.probe\_out0(alu\_input2\_instr\_src),

.probe\_out1(ALUOp),

.probe\_out2(ALUSrc1),

.probe\_out3(ALUSrc2),

.probe\_out4(RegWrite),

.probe\_out5(regfile\_ReadAddress1),

.probe\_out6(regfile\_ReadAddress2),

.probe\_out7(regfile\_WriteAddress),

.probe\_out8(regfile\_WriteData)

);

endmodule

**TestBench Code:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 07/29/2022 11:56:48 AM

// Design Name:

// Module Name: alu\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module alu\_tb;

reg [1:0] ALUSrc1, ALUSrc2;

reg wr\_en, clk, rst = 1;

reg [1:0] rd0\_addr, rd1\_addr, wr\_addr;

reg [2:0] ALUOp;

reg [7:0] Instr\_i , zero\_reg =8'b0;

reg [8:0] wr\_data;

wire [7:0] result, input1, input2;

wire ovf, take\_branch;

alu\_regfile UUT(

.alu\_src1(ALUSrc1),

.alu\_src2(ALUSrc2),

.wr\_en(wr\_en),

.clk(clk),

.rst(rst),

.rd0\_addr(rd0\_addr),

.rd1\_addr(rd1\_addr),

.wr\_addr(wr\_addr),

.alu\_op(ALUOp),

.instr\_i(Instr\_i),

.wr\_data(wr\_data),

.alu\_result(result),

.input1(input1),

.input2(input2),

.ovf(ovf),

.zero\_reg(zero\_reg),

.take\_branch(take\_branch));

initial

begin

$monitor("result = ", result);

$monitor("input1 = ", input1);

$monitor("input2 = ", input2);

$monitor("ovf = ", ovf);

$monitor("take\_branch = ", take\_branch);

end

initial

begin

clk = 0;

forever #20 clk = ~clk;

end

initial

begin

// reset

#102 rst = 1;

// turn off reset

#100 rst = 0;

// writing to spot 0 the value -86

#100

wr\_en = 1'b1;

wr\_addr = 2'b00;

wr\_data = 8'b010101010;

// inverting -86 to 85 (input1 = -86)

#100

wr\_en = 1'b0;

rd0\_addr = 2'b00;

ALUSrc1 = 1'b0;

ALUOp = 3'b001;

// writing to spot 1 the value 85

#100

wr\_en = 1'b1;

wr\_addr = 2'b01;

wr\_data = 8'b01010101;

// -86 and 85 = 0 (input1 = -86, input2 = 85)

#100

wr\_en = 1'b0;

rd0\_addr = 2'b00;

rd1\_addr = 2'b01;

ALUSrc2 = 1'b0;

ALUOp = 3'b010;

// -86 + 85 = -1 (input1 = -86, input2 = 85)

#100

ALUOp = 3'b000;

// writing to spot 1 the value 85

#100

wr\_addr = 2'b01;

wr\_data = 8'b010101010;

// writing to spot 1 the value 85 when wr\_en is off and -86 or 85 = -1

#100

ALUOp = 3'b011;

// writing to spot 2 the value -86

#100

wr\_en = 1'b1;

wr\_addr = 2'b10;

wr\_data = 8'b010101010;

// writing to spot 3 the value -100

#100

wr\_addr = 2'b11;

wr\_data = 8'b010011100;

// -86 + -100 = 70 ovf 1 (input1 = -86, input2 = -100)

#100

wr\_en = 1'b0;

rd0\_addr = 2'b10;

rd1\_addr = 2'b11;

ALUOp = 3'b000;

// -86 (Instr\_i in mux) SRA = -43 (input1 = -86, input2 = -2)

#100

Instr\_i = 8'b11111110;

ALUSrc2 = 1'b1;

ALUOp = 3'b100;

// -86 beq 0 = take\_branch 0 (input1 = 0, input2 = -100)

#100

ALUSrc1 = 1'b1;

ALUSrc2 = 1'b0;

ALUOp = 3'b110;

// -2 beq 0 = take\_branch 0 (input1 = 0, input2 = -2)

#100

ALUSrc2 = 1'b1;

// -2 bne 0 = take\_branch 1 (input1 = 0, input2 = -2)

#100

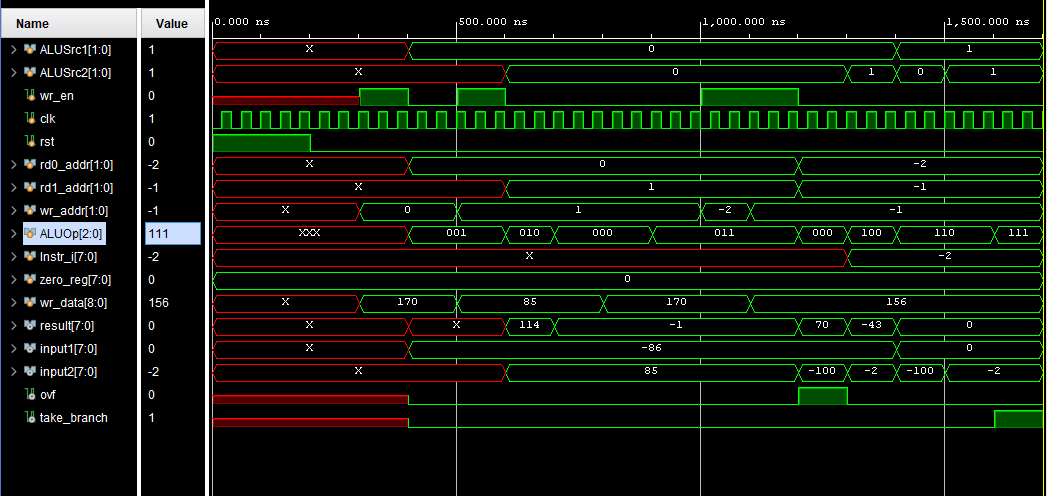
ALUOp = 3'b111;

#100 $finish;

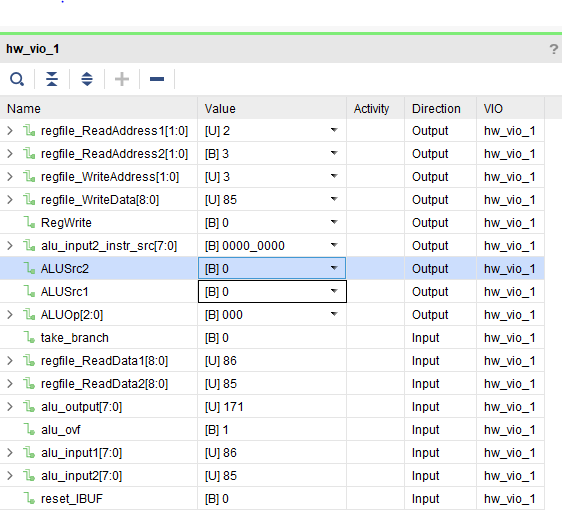
end

endmodule

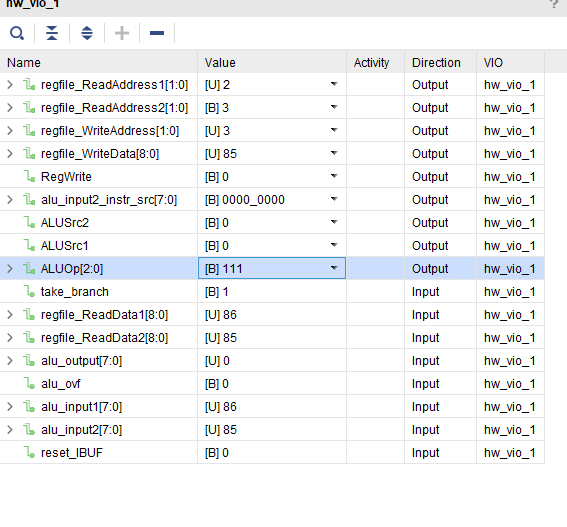
**Appendix B: Captures of the output screens and simulation waveforms.**



**Figure 1:** Test Bench WaveForm Simulation



**Figure 2:** VIO Dashboard addition



**Figure 2:** VIO Dashboard BNE