Name: Alexander Wise **Lab 3 – Adding Register File to ALU**

Instructor Name: Professor Consi **February 6, 2024**

Background

In the previous lab, we designed an ALU: the brains of the computer. In this lab, we are building off of the ALU and adding a register file. The register file is an array of registers that hold data. Each register can hold up to 16 bits of data and acts as a processor's memory to be accessed later by the ALU for further operations.

Pre-Lab

3.1.1 - Design a Register File:

	Inputs							
Description	**ALUSrc1**	**ALUSrc2**	**ALUOp**	**regfile_ReadAddress1**	**regfile_ReadAddress2**	**regfile_WriteAddress**	**regfile_WriteData**	**RegWrite**
Add zero register and immediate value of 12	1	1	0	x	x	x	x	0
the previous step to address 2	x	x	x	x	x	2	12	1
Add zero register and immediate value of 17	1	1	0	x	x	x	x	0
the previous step to address 3	x	x	x	x	x	3	17	1
Add values in address 2 and address 3	0	0	0	2	3	x	x	0

Figure 1: Pre-lab table of register file input values

	Outputs				
Description	**regfile_ReadData1**	**regfile_ReadData2**	**alu_input1**	**alu_input2**	**alu_output**
Add zero register and immediate value of 12	x	x	0	12	12
Wrtie the value from the previous step to address 2	x	x	x	x	x
Add zero register and immediate value of 17	x	x	0	17	17
Write the value from the previous step to address 3	x	x	x	x	x
Add values in address 2 and address 3	12	17	12	17	29

Figure 2: Pre-lab table of register file output values

3.1.2 - Create Test Vectors

	Input							
	ALUSrc1	**ALUSrc2**	**ALUOp**	**regfile_ReadAddress1**	**regfile_ReadAddress2**	**regfile_WriteAddress**	**regfile_WriteData**	**RegWrite**
Add zero register and immediate value of 5	1	1	0	x	x	x	x	0
Write the value from the previous step to address 6	x	x	x	x	x	6	5	1
Add address 6 and immediate value of 12	0	1	0	6	x	x	x	0
Write value from previous step to address 2	x	x	x	x	x	2	18	1
Invert bits of address 2	1	1	1	2	x	x	x	0
Store Previous step in address 3	x	x	x	x	x	3	13	1
Add values in address 6 and address 3	0	0	0	3	6	x	x	0
And zero register and address 2	1	0	2	x	2	x	x	0
Reset	x	x	x	x	x	x	x	x

Figure 3: Pre-lab table of register file input test vectors

	Output				
	regfile_ReadData1	**regfile_ReadData2**	**alu_input1**	**alu_input2**	**alu_output**
Add zero register and immediate value of 5	x	x	0	5	5
Write the value from the previous step to address 6	x	x	x	x	x
Add address 6 and immediate value of 12	5	x	6	12	18
Write value from previous step to address 2	x	x	x	x	x
Invert bits of address 2	18	x	18	x	13
Store Previous step in address 3	x	x	x	x	x
Add values in address 6 and address 3	13	5	13	5	18
And zero register and address 2	x	18	0	18	0
Reset	x	x	x	x	x

Figure 4: Pre-lab table of register file output test vectors

Design Implementation

3.2 – Entering Your Design

Each register has seven input ports, 3 of which are 1-bit inputs, 3 3-bit inputs, and 1 16-bit inputs. Similarly, each register has 2 16-bit output ports. To handle all 8 registers, an array of registers was created. Each of these registers was accessed by the wr_addr input. This input is an

address value that accesses the register at the array position *register[wr_addr]*. From here, data can be written to this register.

For accessing data, the two output ports are utilized. These values are assigned the stored values contained in the array at the address specified by the two input values rd0_addr and rd1_addr. The system verilog code for the register file can be found in Appendix A, labeled as figure 5.

To send data to the ALU, the system utilizes two 2-1 multiplexers. The first multiplexer takes in data stored at address rd0_addr as the first input and the zero_regiser as a secondary input. The zero_register is a register filled with zeroes. The second multiplexer takes in the data stored at address rd1_addr as the first input and can take in a second value from the user. The system verilog code for the multiplexer can be found in Appendix A, labeled as Figure 6.

3.3 - Simulating Your Design

To simulate the register file outputs designed in the previous section, a test bench was created to test custom values. This test bench utilized the test cases found in the tables pictured in figures 3 and 4. Because the register runs off of a clock, a clock signal had to be created to continually switch between high and low. Upon starting, the register file is reset using the rst input on the register file module. The system verilog code can be found in Appendix A, figures 7 and 8.

Once the test bench was designed, it was run, and a waveform was generated from the outputs. The generated waveform for this set of test vectors can be found in Appendix A, figure 9.

3.4 – Testing in Hardware

After verifying the results of the simulation against the table of test vectors, the VIO core was utilized to prepare the hardware for testing. There are a total of 16 inputs and outputs that have to be assigned through VIO. After doing this, the top module was added. Several modifications had to be made to account for differences in variable and file names. A screen shot of the code can be found in Appendix A, figures 10 and 11.

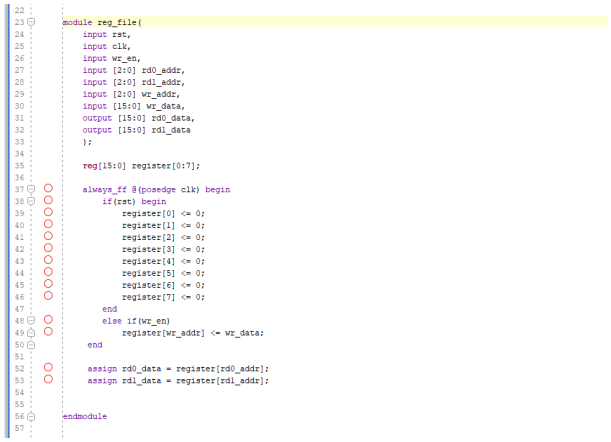
3.4.2 – Testing Your Design

After verifying the results of the testbench and setting up everything using the hardware manager, the whole circuit was uploaded to the PYNQ-Z2 board to be tested on the hardware. An image of the circuit working can be found in Appendix B, figure 12.

Conclusion

The ALU we designed in the previous lab is the powerhouse of processors but is limited by its inability to store the output values. The reg file has significantly increased the functionality of the ALU and allows for us to store the data that is generated by the ALU following its operations. Similarly, the lab introduced new System Verilog functionality and further increased our understanding of the HDL.

Appendix A – Xilinx Vivado Screen Captures



```
module reg_file(
    input rst,
    input clk,
    input wr_en,
    input [2:0] rd0_addr,
    input [2:0] rd1_addr,
    input [2:0] wr_addr,
    input [15:0] wr_data,
    output [15:0] rd0_data,
    output [15:0] rd1_data
    );

    reg[15:0] register[0:7];

    always_ff @(posedge clk) begin
        if(rst) begin
            register[0] <= 0;
            register[1] <= 0;
            register[2] <= 0;
            register[3] <= 0;
            register[4] <= 0;
            register[5] <= 0;
            register[6] <= 0;
            register[7] <= 0;
        end
        else if(wr_en)
            register[wr_addr] <= wr_data;
    end

    assign rd0_data = register[rd0_addr];
    assign rd1_data = register[rd1_addr];


endmodule
```

Figure 5: Screen Capture of the reg_file System Verilog code

```
module multiplexer(
    input [15:0] D0,
    input [15:0] D1,
    input s,
    output [15:0] y
    );

    assign y = s ? D1 : D0;

endmodule
```

Figure 6: Screen capture of the multiplexer's System Verilog code

```
23  module alu_reg_file_tb();
24  // Regfile:
25  logic rst, clk, RegWrite;
26  logic [2:0] ReadAddress1, ReadAddress2, WriteAddress;
27  logic [15:0] WriteData, ReadData1, ReadData2;
28  // MUX
29  logic ALUSrc1, ALUSrc2;
30  logic [15:0] alu_input2_instr_src, zero_register, alu_input1, alu_input2;
31  // ALU
32  logic ovf, take_branch;
33  logic [2:0] ALUOp;
34  logic [15:0] result;
35
36  reg_file register(.rst(rst), .clk(clk), .wr_en(RegWrite), .rd0_addr(ReadAddress1), .rd1_addr(ReadAddress2), .wr_addr(WriteAddress), .wr_data(WriteData), .rd0_data(ReadData1), .rd1_data(ReadData2));
37
38  multiplexer mux0(.D0(ReadData1), .D1(zero_register), .s(ALUSrc1), .y(alu_input1));
39  multiplexer mux1(.D0(ReadData2), .D1(alu_input2_instr_src), .s(ALUSrc2), .y(alu_input2));
40
41  alu UTT(.a(alu_input1), .b(alu_input2), .s(ALUOp), .f(result), .ovf(ovf), .take_branch(take_branch));
42
43  initial begin
44      clk = 0;
45
46      #10 rst = 1;
47      #10 rst = 0;
48
49      #10
50      RegWrite = 0;
51      ALUSrc1 = 1;
52      ALUSrc2 = 1;
53      ALUOp = 0;
54      alu_input2_instr_src = 5;
55
56      #10
57      RegWrite = 1;
58      WriteAddress = 6;
59      WriteData = result;
60
61      #10
62      RegWrite = 0;
63      ALUSrc1 = 0;
64      ALUSrc2 = 1;
65      ALUOp = 0;
66      ReadAddress1 = 6;
67      ReadData1 = 5;
68      alu_input2_instr_src = 12;
69
70      #10
71      RegWrite = 1;
72      WriteAddress = 2;
73      WriteData = result;
74
```

Figure 7: Screen capture of reg file test bench, from start to line 74

```
75      #10
76      RegWrite = 0;
77      ALUSrc1 = 0;
78      ALUSrc2 = 1;
79      ALUOp = 1;
80      ReadAddress1 = 2;
81      ReadData1 = 18;
82
83      #10
84      RegWrite = 1;
85      WriteAddress = 3;
86      WriteData = result;
87
88      #10
89      RegWrite = 0;
90      ALUSrc1 = 0;
91      ALUSrc2 = 0;
92      ALUOp = 0;
93      ReadAddress1 = 6;
94      ReadAddress2 = 3;
95      ReadData1 = 5;
96      ReadData2 = 13;
97
98      #10
99      ALUSrc1 = 1;
100     ALUSrc2 = 0;
101     ReadAddress2 = 6;
102     ReadData2 = 18;
103     ALUOp = 2;
104
105     #10
106     rst = 1;
107     #10
108     rst = 0;
109
110 end
111
112 always #10 clk = ~clk;
113
114
115 endmodule
116
```

Figure 8: Screen capture of reg file test bench, from line 74 to the end

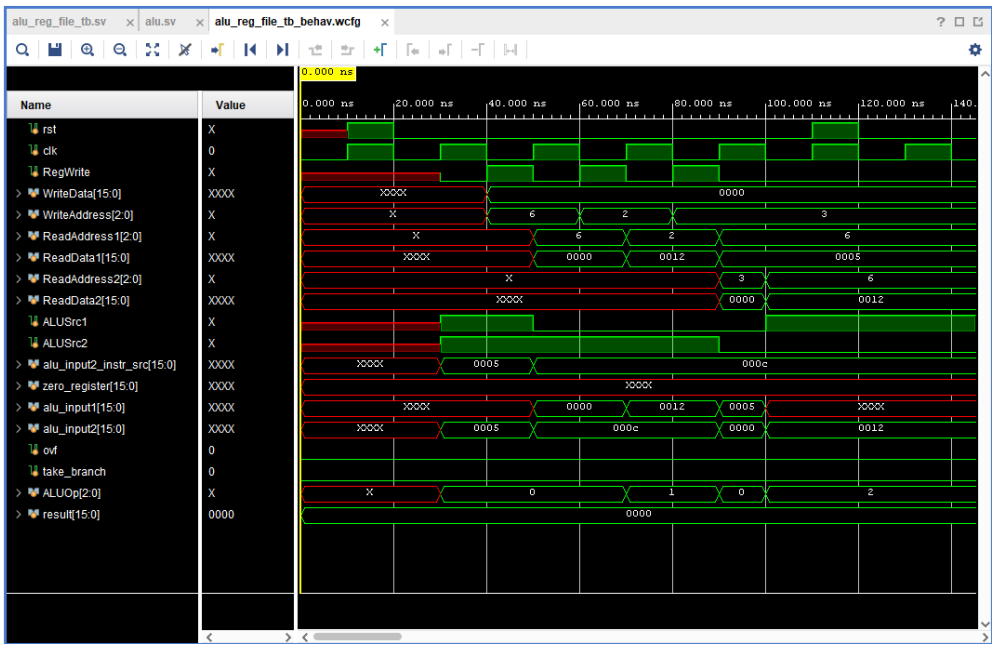


Figure 9: Screen capture of Waveform generated from simulating the test vectors

```
45  module alu_regfile_top(
46          input logic clk,    // clock for vio and regfile
47          input logic reset,
48          output logic ovf_ctrl,
49          output logic take_branch,
50          output logic [3:0] disp_en,     // 7-Segment display enable
51          output logic [6:0] seg7_output    // LD3
52      );
53
54      logic [15:0] alu_input1, alu_input2,alu_input2_instr_src;
55      logic [15:0] alu_output;
56      logic [3:0] ALUOp;
57      logic RegWrite;
58      logic alu_ovf_flag, alu_take_branch;
59
60      logic [2:0] regfile_ReadAddress1;   //source register1 address
61      logic [2:0] regfile_ReadAddress2;   //source register2 address
62      logic [2:0] regfile_WriteAddress;   //destination register address
63      logic [15:0] regfile_WriteData;     //result data
64      logic [15:0] regfile_ReadData1;     //source register1 data
65      logic [15:0] regfile_ReadData2;     //source register2 data
66      logic ALUSrc1, ALUSrc2;
67      logic [15:0] zero_register = 0;
68
69      //assign led = alu_output;
70      assign ovf_ctrl = alu_ovf_flag;
71      assign take_branch = alu_take_branch;
72      // Instantiate RegFile module here
73      reg_file reg1(.rst(reset),.clk(clk),.wr_en(RegWrite),.rd0_addr(regfile_ReadAddress1),
74      .rd1_addr(regfile_ReadAddress2),.wr_addr(regfile_WriteAddress),
75      .wr_data(regfile_WriteData),.rd0_data(regfile_ReadData1),.rd1_data(regfile_ReadData2)
76      );
77      //Instantiate muxes here
78      multiplexer mux1(.D0(regfile_ReadData1),.D1(zero_register),.s(ALUSrc1),.y(alu_input1));
79      multiplexer mux2(.D0(regfile_ReadData2),.D1(alu_input2_instr_src),.s(ALUSrc2),.y(alu_input2));
80    //Instantiate the sixteenbit_alu module here
81      alu sixteenbit_alu(
82          .s(ALUOp),
83          .a(alu_input1),
84          .b(alu_input2),
85          .f(alu_output),
86          .ovf(alu_ovf_flag),
87          .take_branch(alu_take_branch)
88          );
89      Adaptor_display display(
90          .clk(clk),                  // system clock
91          .input_value(alu_output),   // 16-bit input [15:0] value to display
92          .disp_en(disp_en),          // output [3:0] 7 segment display enable
93          .seg7_output(seg7_output)   // output [6:0] 7 segment signals
94      );
95
```

Figure 10: Screen capture of top file from start to line 94

```
96      //Instantiate the VIO core here
97      //Find the instantiate template from Sources Pane, IP sources -> Instantiation Template -> vio_0.veo (double click to open the file)
98      vio_0 vio (
99            .clk(clk),                // input wire clk
100           .probe_in0(alu_output),    // input wire [15 : 0] probe_in0
101           .probe_in1(alu_ovf_flag),    // input wire [0 : 0] probe_in1
102           .probe_in2(alu_take_branch),    // input wire [0 : 0] probe_in2
103           .probe_in3(regfile_ReadData1),
104           .probe_in4(regfile_ReadData2),
105           .probe_in5(alu_input1),
106           .probe_in6(alu_input2),
107           .probe_out0(alu_input2_instr_src),  // output wire [15 : 0] probe_out0
108           .probe_out1(ALUOp),  // output wire [3 : 0] probe_out1
109           .probe_out2(ALUSrc1),
110           .probe_out3(ALUSrc2),
111           .probe_out4(RegWrite),
112           .probe_out5(regfile_ReadAddress1),
113           .probe_out6(regfile_ReadAddress2),
114           .probe_out7(regfile_WriteAddress),
115           .probe_out8(regfile_WriteData)
116       );
117   endmodule
```

Figure 11: Screen capture of top file from line 96 to the end

Appendix B – Images of Program Working on Hardware

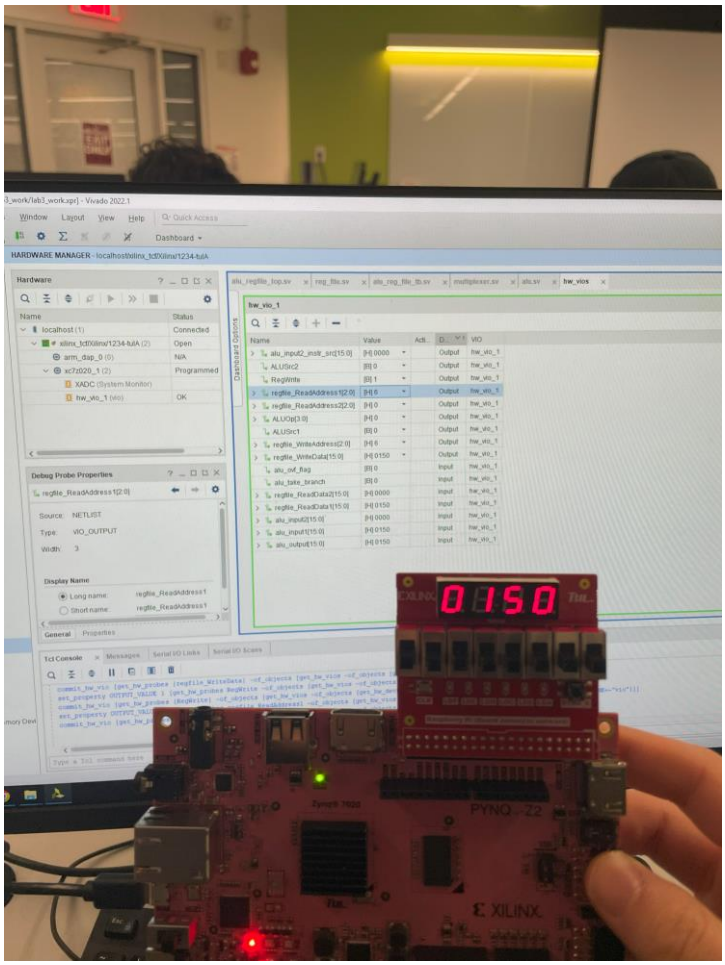


Figure 12: Reg file working on hardware