CECS 341 – LAB 2

ALU and Register File

17 February 2020

Rifa Safeer Shah

017138353

I certify that this submission is my original work

[illegible]

Lab Report: Lab Assignment 2 – ALU and Register File

1. Goal: The goal of the ALU and Register File Lab Assignment is to use Structural Verilog to model part of a CPU datapath by combining functional blocks that is capable of executing ALU operations on data contained in the register file.

2. Steps:
   a. Go to edaplayground.com and log in to your account.
   b. Rewrite the outline provided with the lab including flopr.v and regfile.v.
   c. Add the ALU structural model from lab 1
   d. Identify what is to be done to produce specified output.
   e. Fill in the statements for the test cases in the testbench.sv
   f. Check for errors
   g. Run the code for test cases.

3. Results: In this lab assignment we tested for different ALU operations using Structural Verilog. The output displayed 8 test cases (0-7) which was the same amount that was expected. It also created up to 32 registers. Since the index starts from 0 the number of registers in the output is 0-31. The output displays each test case with an operation performed. There are instructions and operations stored in the registers. The output indicates if the test case is passed after a specific operation is carried out.

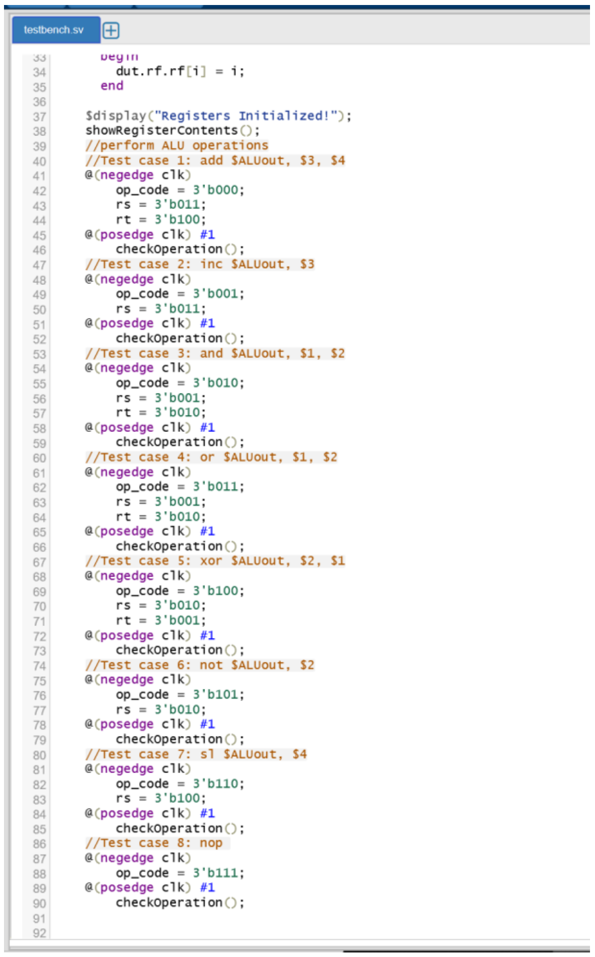
4. Conclusion: In conclusion of this lab, I learned to generate ALU output using the first lab and the components of the second lab. I also learned a way to convert operations and operand(s) into binary. I learned about the op_code, rs, and rt. Some challenges I faced during this lab included rewriting the skeleton code without any errors and learning to write the correct syntax. Another challenge was to fill in the ports for the registers in the testbench.

5. Notes: The ALU operation code 000 refers to Add operation. 001 refers to Inc operation. 010 refers to And operation. 011 refers to Or operation. The 100 refers to Xor operation. 101 refers to Not operation. The 110 ALU operation code refers to Shl operation. 111 refers to Nop operation.

6. Output cases:

Test Case	Operation	op_code	rs	rt
1	add $ALUout, $3, $4	000	011	100
2	inc $ALUout, $3	001	011	
3	and $ALUout, $1, $2	010	001	010
4	or $ALUout, $1, $2	011	001	010
5	xor $ALUout, $1, $2	100	010	001
6	not $ALUout, $2	101	010	
7	sl $ALUout, $4	110	100	
8	nop	111		

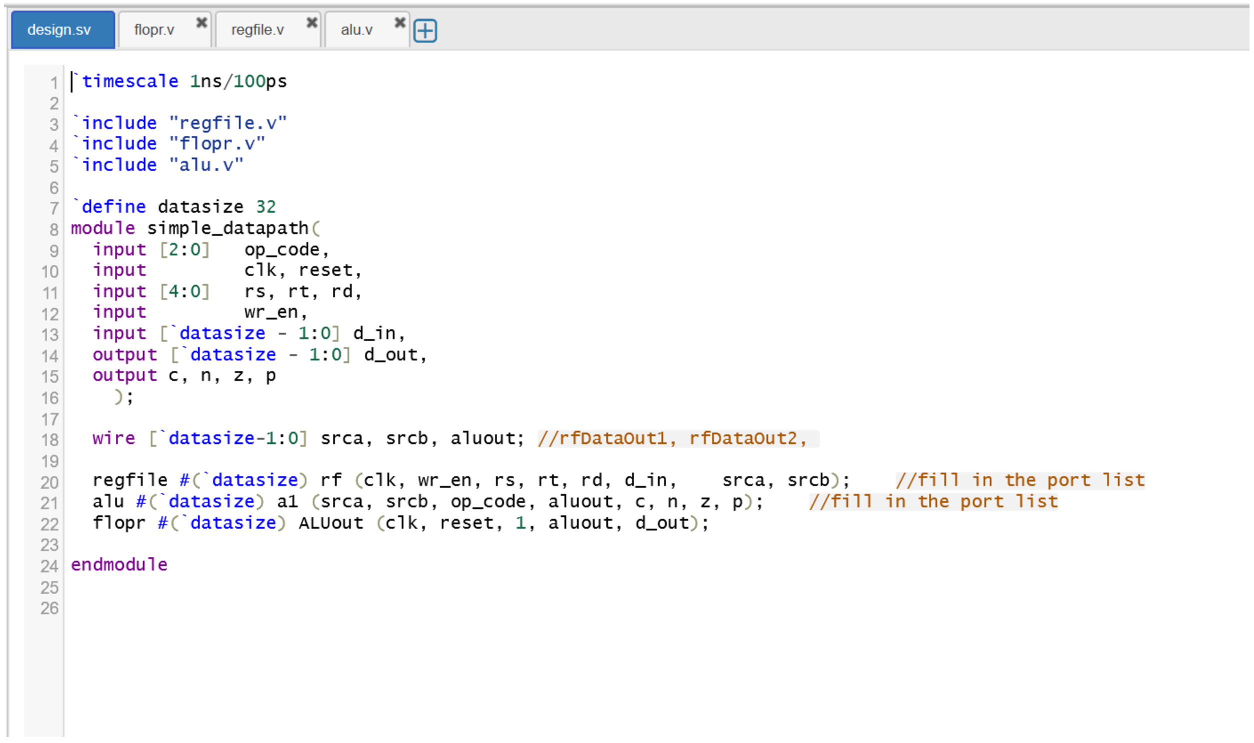
7. Lab Screenshots:



```
      begin
        dut.rf.rf[i] = i;
      end

    $display("Registers Initialized!");
    showRegisterContents();
    //perform ALU operations
    //Test case 1: add $ALUout, $3, $4
    @(negedge clk)
        op_code = 3'b000;
        rs = 3'b011;
        rt = 3'b100;
    @(posedge clk) #1
        checkOperation();
    //Test case 2: inc $ALUout, $3
    @(negedge clk)
        op_code = 3'b001;
        rs = 3'b011;
    @(posedge clk) #1
        checkOperation();
    //Test case 3: and $ALUout, $1, $2
    @(negedge clk)
        op_code = 3'b010;
        rs = 3'b001;
        rt = 3'b010;
    @(posedge clk) #1
        checkOperation();
    //Test case 4: or $ALUout, $1, $2
    @(negedge clk)
        op_code = 3'b011;
        rs = 3'b001;
        rt = 3'b010;
    @(posedge clk) #1
        checkOperation();
    //Test case 5: xor $ALUout, $2, $1
    @(negedge clk)
        op_code = 3'b100;
        rs = 3'b010;
        rt = 3'b001;
    @(posedge clk) #1
        checkOperation();
    //Test case 6: not $ALUout, $2
    @(negedge clk)
        op_code = 3'b101;
        rs = 3'b010;
    @(posedge clk) #1
        checkOperation();
    //Test case 7: sl $ALUout, $4
    @(negedge clk)
        op_code = 3'b110;
        rs = 3'b100;
    @(posedge clk) #1
        checkOperation();
    //Test case 8: nop
    @(negedge clk)
        op_code = 3'b111;
    @(posedge clk) #1
        checkOperation();


```



```
`timescale 1ns/100ps

`include "regfile.v"
`include "flopr.v"
`include "alu.v"

`define datasize 32
module simple_datapath(
  input [2:0]   op_code,
  input         clk, reset,
  input [4:0]   rs, rt, rd,
  input         wr_en,
  input [`datasize - 1:0] d_in,
  output [`datasize - 1:0] d_out,
  output c, n, z, p
    );

  wire [`datasize-1:0] srca, srcb, aluout; //rfDataOut1, rfDataOut2,

  regfile #(`datasize) rf (clk, wr_en, rs, rt, rd, d_in,    srca, srcb);    //fill in the port list
  alu #(`datasize) a1 (srca, srcb, op_code, aluout, c, n, z, p);    //fill in the port list
  flopr #(`datasize) ALUout (clk, reset, 1, aluout, d_out);

endmodule


```