CECS 323 - Lab 5

"ALU and Register File"

Due date: 03/10/21

Student Name: Larry Delgado

Student ID: 016376137

I certify that this submission is my original work

Your signature or electronic signature

Lab Report: Lab Assignment 5 - "ALU and Register File"

1. **Goal:** Model part of a CPU data path using structural Verilog.
2. **Steps:**
    a. We include the previous lab file alu.v
    b. We include flopr.v
    c. We include design.v and fill in the ports
    d. We create random values in test bench.
3. **Results:** The results is what we expect from MIPS instructions based off Table 1.
4. **Conclusion:** I learned how to implement register file and ALU using verilog. The challenges were figuring out the port configuration for register file.

Table 1:

Table 1:

Test Case	Operation	op_code	rs	rt
1	add $ALUout, $3, $4	0	3	4
2	inc $ALUout, $3	1	3	1
3	and $ALUout, $1, $2	2	1	2
4	or $ALUout, $1, $2	3	1	2
5	xor $ALUout, $2, $1	4	2	1
6	not $ALUout, $2	5	2	null
7	sl $ALUout, $4	6	4 << 1	null
8	nop	7	null	null

Design.sv

SV/Verilog Design

```sv
// Larry Delgado
// Lab 5
// CECS 341 Section 11
`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 - 8
  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
```

TestBench.sv

SV/Verilog Testbench

```sv
    d_in = 0;

    //load registers with values
    dut.rf.rf[0] = 0;
    dut.rf.rf[1] = 32'hFFFF0000;
    dut.rf.rf[2] = 32'h0A0A0A0A;
    dut.rf.rf[3] = 32'd500;
    dut.rf.rf[4] = 32'd1000;
    dut.rf.rf[5] = 32'd1001;
    dut.rf.rf[6] = 32'd1002;
    dut.rf.rf[7] = 32'd1003;
    dut.rf.rf[8] = 32'd1004;
    dut.rf.rf[9] = 32'd1005;
    dut.rf.rf[10] = 32'd1006;
    dut.rf.rf[11] = 32'd1007;
    dut.rf.rf[12] = 32'd1008;
    dut.rf.rf[13] = 32'd1009;
    dut.rf.rf[14] = 32'd1010;
    dut.rf.rf[15] = 32'd1011;
    dut.rf.rf[16] = 32'd1012;
    dut.rf.rf[17] = 32'd1013;
    dut.rf.rf[18] = 32'd1014;
    dut.rf.rf[19] = 32'd1015;
    dut.rf.rf[20] = 32'd1016;
    dut.rf.rf[21] = 32'd1017;
    dut.rf.rf[22] = 32'd1018;
    dut.rf.rf[23] = 32'd1019;
    dut.rf.rf[24] = 32'd1020;
    dut.rf.rf[25] = 32'd1021;
    dut.rf.rf[26] = 32'd1022;
    dut.rf.rf[27] = 32'd1023;
    dut.rf.rf[28] = 32'd1024;
    dut.rf.rf[29] = 32'd1025;
    dut.rf.rf[30] = 32'd1126;
    dut.rf.rf[31] = 32'd1027;
    //load remaining registers with random values
```

Output:

[2021-03-01 20:15:45 EST] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out

design.sv:10: warning: timescale for simple_datapath inherited from another file.

./alu.v:6: ...: The inherited timescale is here.

design.sv:24: warning: Port 3 (load) of flopr expects 1 bits, got 2.

design.sv:24:   : Pruning (signed) 1 high bits of the expression.

Registers Initialized!

	Register	Content
0	0x00000000	
1	0xffff0000	
2	0x0a0a0a0a	
3	0x000001f4	
4	0x000003e8	
5	0x00000005	
6	0x00000006	
7	0x00000007	
8	0x00000008	
9	0x00000009	
10	0x0000000a	
11	0x0000000b	
12	0x0000000c	
13	0x0000000d	
14	0x0000000e	
15	0x0000000f	
16	0x00000010	
17	0x00000011	
18	0x00000012	
19	0x00000013	
20	0x00000014	
21	0x00000015	
22	0x00000016	
23	0x00000017	
24	0x00000018	
25	0x00000019	
26	0x0000001a	
27	0x0000001b	

28 0x0000001c

29 0x0000001d

30 0x0000001e

31 0x0000001f

Checking test case

srca = 500, srcb = 1000

add $ALUout, $ 3, $ 4 performed

sum = 1500

test passed

Checking test case

srca = 500, srcb = 1000

increment $ALUout, $ 3 performed

d_out = 501

test passed

Checking test case

srca = ffff0000, srcb = 0a0a0a0a

and $ALUout, $ 1, $ 2 performed

d_out = 0a0a0000

test passed

Checking test case

srca = ffff0000, srcb = 0a0a0a0a

or $ALUout, $ 1, $ 2 performed

d_out = ffff0a0a

test passed

Checking test case

srca = 0a0a0a0a, srcb = ffff0000

xor $ALUout, $ 2, $ 1 performed

d_out = f5f50a0a

test passed

Checking test case

srca = 0a0a0a0a, srcb = ffff0000

not $ALUout, $ 2 performed

d_out = 00000000

Test Failed!!!

Checking test case

srca = 1000, srcb = 4294901760

sll $ALUout, $ 4, performed

d_out = 2000

test passed

Checking test case

srca = 000003e8, srcb = ffff0000

nop operation performed

d_out = 00000000

test passed

Done