**Lab Assignment 5**

Due: 30 Oct 2020

By: Alex Oswald

EECE 2323

**Introduction**

During this lab experiment, a new type of storage, *Data Memory*, is to be added to the datapath. Data memory is typically slower but larger than registers. It is accessed with load and store instructions. The goal of this experiment is to properly design and connect data memory into the partially assembled central processing unit.

**Verilog Code for pdatapath\_top\_lab5.v:**

1. `timescale 1ns / 1ps
3. module pdatapath\_top(
4. input wire clk,
5. input wire rst\_general
6. );
8. wire [7:0] alu\_1st\_input, alu\_2nd\_input, alu\_2nd\_input\_vio;
9. wire [7:0] alu\_output;
10. wire [2:0] ALUOp;
11. wire       alu\_ovf\_flag;
12. wire       alu\_take\_branch\_output;
14. wire RegWrite;//Write enable
15. wire RegRead;//Read enable
16. wire [1:0] regfile\_read\_address1;//source register1 address
17. wire [1:0] regfile\_read\_address2;//source register2 address
18. wire [1:0] regfile\_write\_address;//destination register address
19. wire [8:0] regfile\_write\_data;//result data
20. wire [8:0] read\_data1;//source register1 data
21. wire [8:0] read\_data2;//source register2 data
23. wire ALUSrc1, ALUSrc2;
24. wire [8:0] alu\_result;
25. wire [8:0] zero\_register;
27. wire MemtoReg;
28. wire MemWrite;
30. wire [8:0] data\_mem\_out;
32. assign zero\_register = 8'b0;//ZERO constant
33. /\* Instantiate the reg-file, MUXes, ALU that you have created here \*/
34. assign alu\_result = {alu\_ovf\_flag, alu\_output}; // 9 bits (concatenated)
35. assign regfile\_write\_data = MemtoReg ? data\_mem\_out : alu\_result; // 9 bits
37. alu\_regfile ALU\_RegFile (
38. .ReadData1(read\_data1),
39. .ReadData2(read\_data2),
40. .ALUsrc1(ALUSrc1),
41. .ALUsrc2(ALUSrc2),
42. .ALUop(ALUOp),
43. .Instr\_i(alu\_2nd\_input\_vio),
44. .zero\_register(zero\_register),
46. .rst(rst\_general),
47. .clk(clk),
48. .wr\_en(RegWrite),
49. .rd0\_addr(regfile\_read\_address1),
50. .rd1\_addr(regfile\_read\_address2),
51. .wr\_addr(regfile\_write\_address),
52. .wr\_data(regfile\_write\_data),
54. .input\_1(alu\_1st\_input),
55. .input\_2(alu\_2nd\_input),
56. .result(alu\_result),
57. .ovf(alu\_ovf\_flag),
58. .take\_branch(alu\_take\_branch\_output)
59. );
61. /\* Instantiate the VIO that you have created here,
62. make sure the number of probes and their width are correctly configured \*/
63. vio\_0 vio (
64. .clk(clk),              //  input wire clk
66. .probe\_in0(regfile\_write\_data),     //  [8:0] WriteData
67. .probe\_in1(read\_data1),             //  [7:0] ReadData1
68. .probe\_in2(read\_data2),             //  [7:0] ReadData2
69. .probe\_in3(alu\_1st\_input),          //  [7:0] input1
70. .probe\_in4(alu\_2nd\_input),          //  [7:0] input2
71. .probe\_in5(alu\_take\_branch\_output), //  [0:0] take\_branch
72. .probe\_in6(alu\_ovf\_flag),           //  [0:0] alu\_ovf
73. .probe\_in7(alu\_output),             //  [7:0] alu\_out
74. .probe\_in8(data\_mem\_out),           //  [8:0] DataMemOut
76. .probe\_out0(RegWrite),              //  [0:0] RegWrite
77. .probe\_out1(alu\_2nd\_input\_vio),     //  [7:0] Instr\_i
78. .probe\_out2(ALUSrc1),               //  [0:0] ALUSrc1
79. .probe\_out3(ALUSrc2),               //  [0:0] ALUSrc2
80. .probe\_out4(ALUOp),                 //  [2:0] ALUOp
81. .probe\_out5(MemWrite),              //  [0:0] MemWrite
82. .probe\_out6(MemtoReg),              //  [0:0] MemToReg
83. .probe\_out7(regfile\_read\_address1), //  [1:0] ReadAddr1
84. .probe\_out8(regfile\_read\_address2), //  [1:0] ReadAddr2
85. .probe\_out9(regfile\_write\_address)  //  [1:0] WriteAddr
86. );
88. /\* Instantiate the data memory that you have created here\*/
89. // NOT USING RAM IP--MY OWN FILE
90. data\_memory datamem (
91. .rst(rst\_general),
92. .clk(clk),
93. .write\_enable(MemWrite),
94. .addr(alu\_output),  // ALU\_RESULT or ALU\_OUTPUT ??
95. .write\_data(read\_data2),
96. .read\_data(data\_mem\_out)
97. );
99. endmodule

**Verilog Code for data\_memory.v:**

1. `timescale 1ns / 1ps
3. module data\_memory(
4. //inputs
5. input rst, // rst=reset OR clr=clear
6. input clk,
7. input write\_enable,
8. input [7:0] addr,
9. input [8:0] write\_data,
10. output reg [8:0] read\_data);
12. reg [8:0] MEM[7:0];
13. integer i;
15. always@(addr)
16. begin
17. **if**(rst)
18. **for** (i=0; i<128; i=i+1)
19. MEM [i] = 0;
20. **else** **if**(write\_enable)
21. MEM[addr] = write\_data;
22. **else**
23. read\_data <= MEM[addr];
24. end
26. endmodule

**VIO Output Visualization:**

Evidence is located in attached **lab5-test.mp4** file. The following sequence was followed.

1. Write 8’d18 to Reg 1 and then save to memory at address 8’d0.
2. Compute 18 >>> 1 and save to Reg 2.
3. Save 8’d5 to Reg 3.
4. Compute Reg 2 + Reg 3 = Reg 0 (saving to Reg 0).
5. Store Reg 0 to memory at address 8’h0E.
6. Read address 8’d0 from memory (18).

**Conclusion**

During this lab assignment, the concept of data memory was successfully explored by way of designing a memory bank and successfully implementing it into our data path. Memory banks act by design similarly to how registers do, but they have significantly more depth. The data memory was designed in Verilog rather than using the IP Source single-port RAM option provided by Xilinx in order to allow for the capabilities of a reset button. To extend academic inquiry, one might examine how to add instruction decoding to the datapath.