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`:

```verilog
`timescale 1ns / 1ps

module pdatapath_top(
        input wire clk,
        input wire rst_general
    );

    wire [7:0] alu_1st_input, alu_2nd_input, alu_2nd_input_vio;
    wire [7:0] alu_output;
    wire [2:0] ALUOp;
    wire       alu_ovf_flag;
    wire       alu_take_branch_output;

    wire RegWrite;//Write enable
    wire RegRead;//Read enable
    wire [1:0] regfile_read_address1;//source register1 address
    wire [1:0] regfile_read_address2;//source register2 address
    wire [1:0] regfile_write_address;//destination register address
    wire [8:0] regfile_write_data;//result data
    wire [8:0] read_data1;//source register1 data
    wire [8:0] read_data2;//source register2 data

    wire ALUSrc1, ALUSrc2;
    wire [8:0] alu_result;
    wire [8:0] zero_register;

    wire MemtoReg;
    wire MemWrite;

    wire [8:0] data_mem_out;

    assign zero_register = 8'b0;//ZERO constant
    /* Instantiate the reg-file, MUXes, ALU that you have created here */
    assign alu_result = {alu_ovf_flag, alu_output}; // 9 bits (concatenated)
    assign regfile_write_data = MemtoReg ? data_mem_out : alu_result; // 9 bits

    alu_regfile ALU_RegFile (
        .ReadData1(read_data1),
        .ReadData2(read_data2),
        .ALUsrc1(ALUSrc1),
        .ALUsrc2(ALUSrc2),
        .ALUop(ALUOp),
        .Instr_i(alu_2nd_input_vio),
        .zero_register(zero_register),

        .rst(rst_general),
        .clk(clk),
        .wr_en(RegWrite),
        .rd0_addr(regfile_read_address1),
        .rd1_addr(regfile_read_address2),
        .wr_addr(regfile_write_address),
        .wr_data(regfile_write_data),

        .input_1(alu_1st_input),
        .input_2(alu_2nd_input),
        .result(alu_result),
        .ovf(alu_ovf_flag),
        .take_branch(alu_take_branch_output)
        );

    /* Instantiate the VIO that you have created here,
    make sure the number of probes and their width are correctly configured */
    vio_0 vio (
        .clk(clk),              //  input wire clk

        .probe_in0(regfile_write_data),      //  [8:0] WriteData
        .probe_in1(read_data1),              //  [7:0] ReadData1
        .probe_in2(read_data2),              //  [7:0] ReadData2
        .probe_in3(alu_1st_input),           //  [7:0] input1
        .probe_in4(alu_2nd_input),           //  [7:0] input2
        .probe_in5(alu_take_branch_output),  //  [0:0] take_branch
        .probe_in6(alu_ovf_flag),            //  [0:0] alu_ovf
        .probe_in7(alu_output),              //  [7:0] alu_out
        .probe_in8(data_mem_out),            //  [8:0] DataMemOut

        .probe_out0(RegWrite),               //  [0:0] RegWrite
        .probe_out1(alu_2nd_input_vio),      //  [7:0] Instr_i
        .probe_out2(ALUSrc1),                //  [0:0] ALUSrc1
        .probe_out3(ALUSrc2),                //  [0:0] ALUSrc2
        .probe_out4(ALUOp),                  //  [2:0] ALUOp
        .probe_out5(MemWrite),               //  [0:0] MemWrite
        .probe_out6(MemtoReg),               //  [0:0] MemToReg
        .probe_out7(regfile_read_address1),  //  [1:0] ReadAddr1
        .probe_out8(regfile_read_address2),  //  [1:0] ReadAddr2
        .probe_out9(regfile_write_address)   //  [1:0] WriteAddr
        );

    /* Instantiate the data memory that you have created here*/
    // NOT USING RAM IP--MY OWN FILE
    data_memory datamem (
        .rst(rst_general),
        .clk(clk),
        .write_enable(MemWrite),
        .addr(alu_output),  // ALU_RESULT or ALU_OUTPUT ??
        .write_data(read_data2),
        .read_data(data_mem_out)
        );

endmodule
```

Verilog Code for `data_memory.v`:

```
`timescale 1ns / 1ps

module data_memory(
    //inputs
    input rst, // rst=reset OR clr=clear
    input clk,
    input write_enable,
    input [7:0] addr,
    input [8:0] write_data,
    output reg [8:0] read_data);

    reg [8:0] MEM[7:0];
    integer i;

    always@(addr)
    begin
        if(rst)
            for (i=0; i<128; i=i+1)
                MEM [i] = 0;
        else if(write_enable)
            MEM[addr] = write_data;
        else
            read_data <= MEM[addr];
    end

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.