Name: Alexander Wise **Lab 4 – Adding Data Memory to Datapath**

Instructor Name: Farzad Niknia **February 20, 2024**

Background

In the previous lab, we added a register file which is used to stage data for ALU operations. Building off of this, the purpose of this lab is to add data memory to the datapath. The data memory can store values from the register to be saved for later operations or load those values to the registers to be used in operations.

Pre-Lab

	Input								
Description	**regfile_WriteData**	**regfile_ReadData1**	**regfile_ReadData2**	**alu_input1**	**alu_input2**	**alu_take_branch**	**alu_ovf_flag**	**alu_output**	**DataMemOut**
write value 4 to regfile address 2	0004	0004	0000	0000	0004	0	0	0004	0000
write value at regfile address 2 to mem address 1	0004	0004	0000	0000	0001	0	0	0001	0004
Write value 6 to regfile address 3	0006	0006	0000	0000	0006	0	0	0006	0000
Write value at regfile address 3 to mem address 2	0006	0006	0000	0000	0002	0	0	0002	0006
Load value at mem address 2 to reg address 1	0006	0006	0006	0000	0002	0	0	0002	0006
load value at mem address 1 to reg address 1	0004	0006	0004	0000	0001	0	0	0001	0004
add values of reg address 0 and reg address 1	000A	0006	0004	0006	0004	0	0	000A	0000

Figure 1: Pre-Lab Table of VIO Inputs

	Output									
	RegWrite	**alu_input2_instr_src**	**ALUSrc1**	**ALUSrc2**	**ALUOp**	**MemWrite**	**MemToReg**	**regfile_ReadAddress1**	**regfile_ReadAddress2**	**regfile_WriteAddress**
write value 4 to regfile address 2	1	0004	1	1	0	0	0	2	0	2
write value at regfile address 2 to mem address 1	0	0001	1	1	0	1	1	2	2	2
Write value 6 to regfile address 3	1	0006	1	1	0	0	0	3	2	3
Write value at regfile address 3 to mem address 2	0	0002	1	1	0	1	1	2	3	3
Load value at mem address 2 to reg address 0	1	0002	1	1	0	1	1	0	3	0
load value at mem address 1 to reg address 1	1	0001	1	1	0	0	1	0	1	1
add values of reg address 0 and reg address 1	0	0001	0	0	0	0	0	0	1	1

Figure 2: Pre-Lab Table of VIO Outputs

Design Implementation

3.2.1 – Data Memory

The data memory was implemented using the Distributed Memory Generator found in the IP catalog on Xilinx Vivado IDE. Using this IP Core, the memory was set to a depth of 1024 bits and a width of 16 bits. After doing so, I verified the template matched the provided interface signals given in the lab manual and alu_datamem_top.sv file provided to us on Canvas.

3.2.2 – Adding the VIO

Next, using the VIO IP core, the inputs and outputs were set up to work with hardware and for testing purposes. Here, there are 9 input probes and 10 output probes. Each probe corresponds to an input and output probe as seen in the pre-lab tables (figures 1 and 2).

3.3 – Testing in Hardware

Once all IP Cores were configured and top-level modules were added and configured to work together, the system was ready to be tested on hardware. Images of the system working with hardware can be found in Appendix A, figures 3 through 8.

Conclusion

Building off previous labs, we continue to advance in the process of designing a RISC-V processor. Now with memory, the capabilities of our processor are increasing significantly. Similarly, we are continuing to advance our knowledge of System Verilog as well as the Xilinx Vivado IDE as we continue to incorporate new IP cores into our systems as well as learn more about top-level modules and their importance in building full circuits.

Appendix A – Images of System Working on Hardware

Note: All Multiplexer operations are flipped. A 1 in the pre-lab table for multiplexers corresponds to a 0 in the following images.

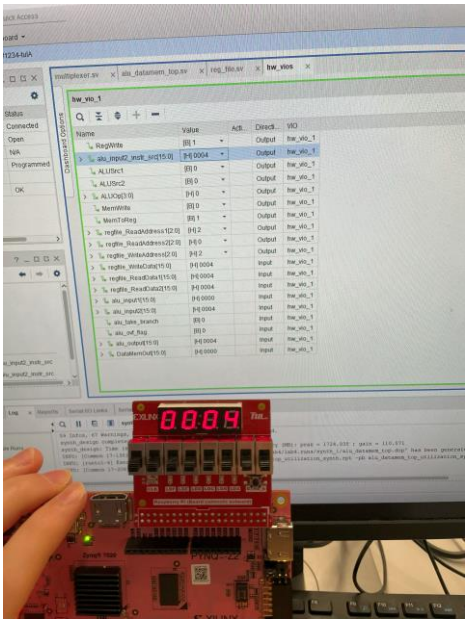

Figure 3: Hardware Working for Test Case 1

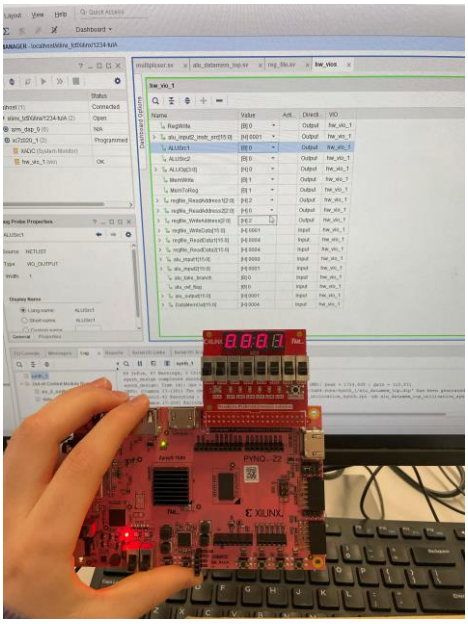

Figure 4: Hardware Working for Test Case 2

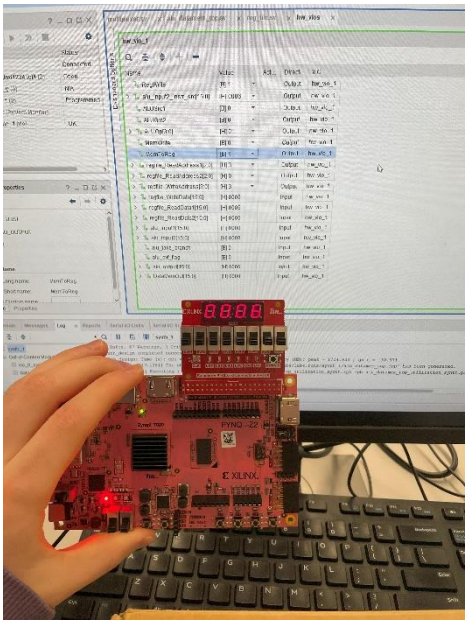

Figure 5: Hardware Working for Test Case 3

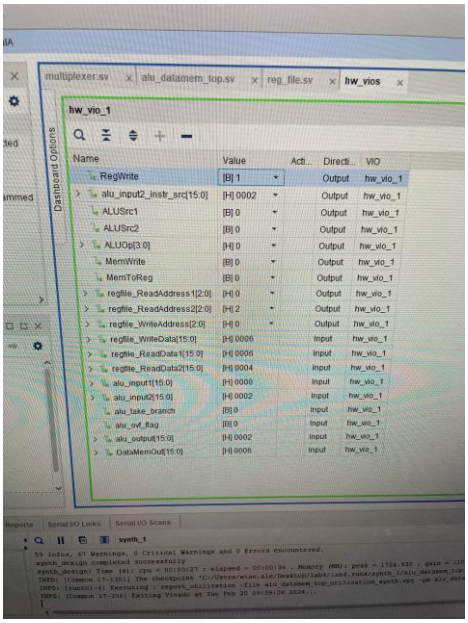

Figure 6: Hardware Working for Test Case 5

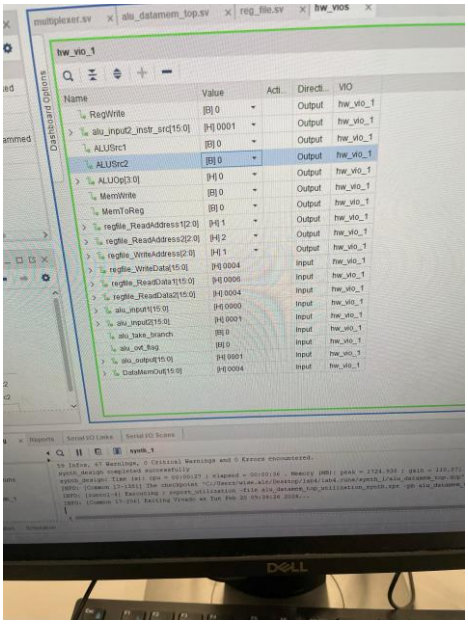

Figure 7: Hardware Working for Test Case 6

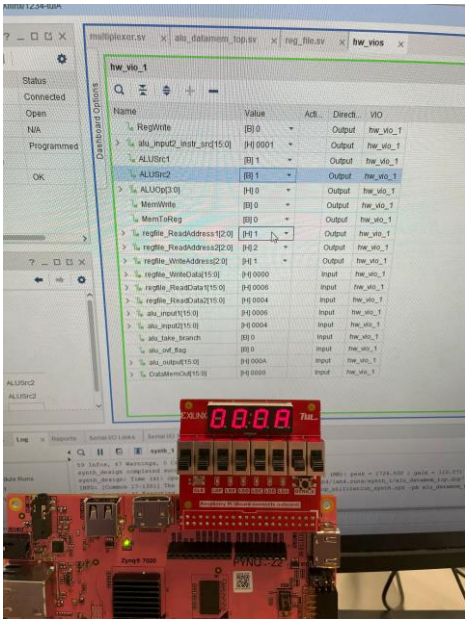

Figure 8: Hardware Working for Test Case 7