Fall 2017 EE 361L Lab 5 Report

Author: Lance Hwang
Lab Partners: Katherine Shun Gilmour

Date: 2017 December 12

Abstract: This report describes my Lab 5, which was a project where the main objective was to design a pipelined microprocessor using Verilog. The microprocessor used an architecture called LEGLite-Pipe which was a simplified version of the pipelined LEGv8 architecture. For background, LEGv8 is a simplified version of the ARMv8 architecture. The main objective was only partially complete with a microprocessor that inserted three NOP instructions after each instruction to avoid data and control hazards. The partially complete microprocessor was nevertheless programmed onto the Digilent Basys 3 Artix-7 FPGA Trainer Board using Xilinx Vivado WebPACK software. This report discusses the objectives of the lab in more detail and describes the design and approach towards accomplish the objectives of the lab.

1 Introduction and objective

This lab was a project where the main objective was to design and implement in Verilog a simplified version of the pipelined LEGv8 called LEGLite-Pipe [1]. LEGv8 is a simplified subset of the ARMv8 architecture [2]. The objective of this lab was divided into five subprojects.

Subproject 1 was to design in Verilog a microprocessor called LEGLite-Pipe0 (which is different from LEGLite-Pipe without the 0) so that the microprocessor can use a separate Verilog file called IM1.V as the instruction memory in the microprocessor and then demonstrate the microprocessor to the T.A. Avon Whitworth by running a testbench in a simulator. The file IM1.V was provided from a separate assignment, Homework 6 from EE 361 lecture [3]. LEGLite-Pipe0 is simpler than LEGLite-Pipe in that LEGLite-Pipe0 will always insert three NOP instructions (bubbles) into the pipeline after each instruction to ensure that the microprocessor never encounters data or control hazards. Thus, each LEGLite-Pipe0 instruction takes four clock cycles to execute.

Subproject 2 was to design and implement in Verilog LEGLite-Pipe0 so that it can use another file called IM2.V as the instruction memory. Just like IM1.V, IM2.V was provided in Homework 6 from EE 361 lecture.

Subproject 3 was to implement LEGLite-Pipe0 from Subproject 2 into an FPGA and demonstrate to the T.A. that the FPGA works. Subproject 4 was to design LEGLite-Pipe to run IM1.V so that it stalls to avoid both data and control hazards but otherwise pipelines its instructions whenever it does not cause a data or control hazard.

In Subproject 4, conditional branching is done by stall-on-branch, meaning that whenever a conditional branch instruction is put into the pipeline, the microprocessor will stall until the branch condition is determined.

Subproject 5 was to design LEGLite-Pipe to run IM2.V.

Our lab group only finished demos for Subprojects 1, 2, and 3. We did not complete Subprojects 4 and 5 for this lab report.

Katherine and I each did Subproject 1 separately, meaning that Katherine did Subproject 1 by herself and I also did Subproject 1 by myself. We did the subprojects separately in hopes of being able to better understand the project. I, at least, felt that at the beginning of Subproject 1 that I did not understand Verilog enough to simply write separate components of a design without understanding the entire structure as a whole, even after completely finishing the previously mentioned Homework 6 from EE 361 lecture. We both finished each of our subprojects at around the same time and we shared our code when we were done.

Transitioning from Subproject 1 to Subproject 2 was trivial because by the end of Subproject 1, most if not all of the components of the microprocessor was already in place to run IM2.V. Transitioning from Subproject 2 to Subproject 3 was also trivial because a Verilog file was provided with contained a module that only required slight modification to properly implement onto an FPGA. We attempted Subproject 4 momentarily, but due to time and motivation constraints, we decided to not proceed any further with Subproject 4.

In addition to the files provided with the assignment, we received various emails from our T.A. Avon Whitworth and another T.A. Jianqiu Cao that included hints and other information pertaining to the Lab 5 subprojects. We were also able to ask either T.A. questions about how we may improve our designs or gain a better understanding of the project.

This report is organized into sections as described as follows. Section 2 lists the equipment used in the lab. Section 3 describes the LEGLite-Pipe0 architecture to be implemented in Subprojects 1, 2, and 3. Section 4 and its subsections describe the design description and approach of Subprojects 1 and 2. Section 5 describes the process implementing LEGLite-Pipe0 onto the Basys 3 FPGA. Section 5 is the conclusion to this report.

2 Setup

For this lab, the following equipment was used.

- Computer (running Windows with Xilinx Vivado WebPACK installed)
- Digilent Basys 3 Artix-7 FPGA Trainer Board with USB connector

3 Description of LEGLite-Pipe0

LEGLite-Pipe0 is a simplification of the pipelined LEGv8 architecture. LEGv8 is a simplified subset of the ARMv8 architecture [2]. The registers of LEGLite-Pipe0 each have 16-bits unlike the registers of LEGv8 which each have 32-bits. LEGLite-Pipe0 has seven general purpose registers and one register that is always valued as zero as described in Table 1. LEGLite-Pipe0 is also considered to have four instruction formats, R, I, D, and CB as described in Table 2. LEGLite-Pipe0 has seven instructions as described in Table 3. A block diagram of LEGLite-Pipe0 is shown in Figure 1. As shown in Figure 1, LEGLite-Pipe0 has five pipeline stages: instruction fetch (IF), instruction decode (ID), execution (EX), memory (MEM), and write-back (WB). Between each of the five stages in Figure 1 is a register block for a total of four register blocks (IFID, IDEX, EXMEM, and MEMWB). Each register block updates its outputs on each positive edge of the clock depending on the values being input into each register

block when the positive clock edge occurs. This allows an instruction to pass through each stage of the pipeline as the clock transitions over time.

Table 1. LEGLite-Pipe0 16-bit registers [3].

register name	register number	usage
X0-X6	0-6	general purpose registers
XZR	7	zero register (always zero valued)

Table 2. LEGLite-Pipe0 instruction formats. Although formats I, D, and CB could technically be considered the same format, they are considered to be different formats to be consistent with LEGv8 [3].

format	[15:13]	[12:10]	[9:6]	[5:3]	[2:0]	example
R	opcode	Xm	N/A	Xn	Xd	ADD Xd, Xn, Xm
I	opcode	constant		Xn	Xd	ADDI Xd, Xn, #constant
D	opcode	constant		Xn	Xd	STUR Xd, [Xn, #constant]
CB	opcode	(PC-relative offset)/2		N/A	Xd	CBZ Xd, Target

Table 3. LEGLite-Pipe0 instructions. The numbers in the bit fields from [12:0] are provided as examples which are shown on the rightmost column of the table [3].

name	format	[15:13]	[12:10]	[9:6]	[5:3]	[2:0]	example
ADD	R	0	3	0	2	1	ADD X1, X2, X3
SUB	R	1	3	0	2	1	SUB X1, X2, X3
LD	D	3	50		2	1	LD X1, [X2, #50]
ST	D	4	50		2	1	ST X1, [X2, #50]
CBZ	CB	5	(PC-relative offset)/2		0	1	CBZ X1, Target
ADDI	I	6	50		2	1	ADDI X1, X2, #50
ANDI	I	7	50		2	1	AND X1, X2, #50

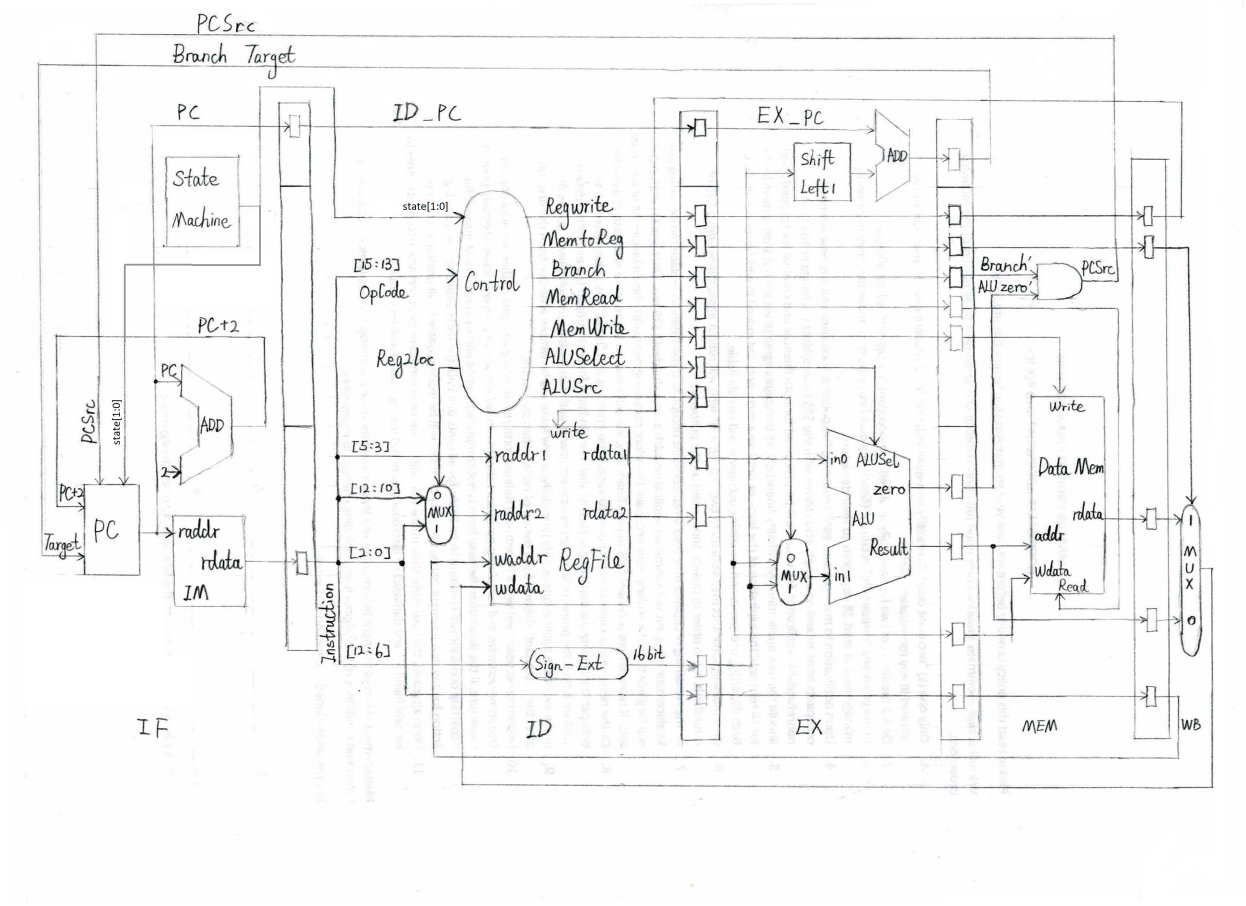

Figure 1. LEGLite-Pipe0 block diagram. This diagram was modified from [4].

4 Design Description of Subprojects 1 and 2

From Figure 1, the components that required nontrivial design included the state machine, the program counter, the controller, and the register blocks. The remaining components such as the instruction memory, register file, ALU, data memory, adders, multiplexers, etc. were provided in Verilog files. Aside from designing the state machine, program counter, controller, and register blocks, the main design challenge was merely to instantiate the appropriate modules and select the appropriate inputs and outputs for each instantiation.

4.1 Description of the State Machine, PC Logic, and Controller

The state machine, PC logic, and controller were all integrated in their design to ensure that the microprocessor would stall for exactly three clock cycles between each instruction sent down the pipeline as described in Table 4 and Figure 2. It should be noted that the state machine and the PC Logic were both separate sequential circuits that each updated its parameters on each positive clock edge. The controller on the other hand was designed to be a combinational circuit where the output depends on the direct input into the circuit as opposed to only updating its parameters on a clock edge.

Table 4. Description of PC updates and controller output to IDEX register block on each stage of the state machine [1].

state	PC updates	controller output to IDEX
0	PC = PC + 2	bubble
1	PC = PC (hold)	control signals that are dependent of the opcode in IFID
2	PC = PC (hold)	bubble
3	PC = target branch address if the branch condition is true. Otherwise, PC = PC (hold)	bubble

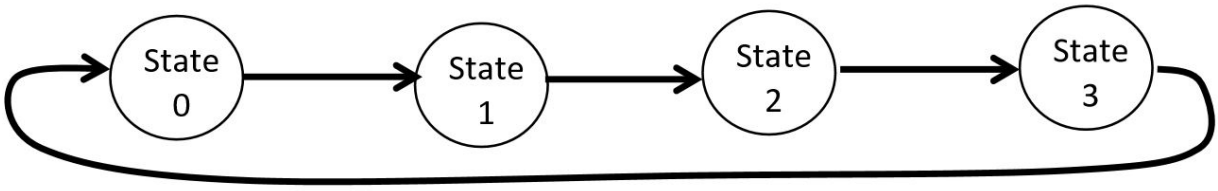

Figure 2. State diagram of the state machine. State transition occurs on each positive clock edge [1].

4.2 Design Approach of Subprojects 1 and 2

Once all of the individual components were designed, each component depicted in Figure 1 was either instantiated or implemented with lines of Verilog code. In the LEGLiteP0.V file, each of the wire and reg parameters that were not provided as inputs or outputs to the LEGLiteP0 module were declared towards the top of the file. Then the provided inputs and outputs in the LEGLiteP0 module together with the declared wire and reg parameters were either typed into the appropriate instantiation fields for each instantiated module or used to implement the register blocks by setting the registers in the register blocks to update its output to whatever its input is on each positive clock edge. The sign extension was incorporated using the concatenation syntax in Verilog. Shift left and add could be incorporated with simple operations in Verilog (<< and +). Combinational AND gates could be incorporated with the bitwise AND operation in Verilog (&). Once Subproject 1 was complete, transitioning to Subproject 2 was trivial because the code required to run LD and ST instructions was already implemented in Subproject 1 using the same design approach described previously. The testbenches for Subprojects 1 and 2 were demonstrated to the T.A. Avon Whitworth upon completion of the subprojects.

5 Description of Subproject 3

A Verilog file titled SubProject3FPGA.V was provided and it only required slight modifications in order to use it to implement LEGLite-Pipe0 into the Basys 3 FPGA. The Basys 3 would be programmed using Xilinx Vivado WebPACK running on Windows while being connected to the computer via USB. When the Basys 3 was programmed with LEGLite-Pipe0, the four 7-segment displays would each display 0 when hardware switch sw[0] was set to 0 and similarly the four 7-segment displays would each display 1 when the hardware switch sw[0] was

set to 1. This programmed Basys 3 was shown to the T.A. Avon Whitworth to complete the demo for Subproject 3.

6 Conclusions

Subprojects 1, 2, and 3 were completed and demoed to the T.A. Avon Whitworth while Subprojects 4 and 5 were not completed. Once Subprojects 1 to 3 were completed, no major problems were encountered neither in the testbench simulations nor the implementation onto the Basys 3 FPGA.

References

[1] G. Sasaki. (2017, Oct 24). *EE 361L Fall 2017 Pipelined LEGLite* [Online]. Available: https://laulima.hawaii.edu/access/content/attachment/MAN.70882.201810/Assignments/2f4d1033-3685-40f5-8871-0bc1d2b9721c/EE361L-Fall2017-Lab5.pdf

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface ARM® Edition*, 1st ed. Amsterdam, Netherlands: Elsevier, 2016.

[3] G. Sasaki. (2017, Oct 19). *EE 361 16-Bit Single-Cycle LEGLite Project* [Online]. Available: https://laulima.hawaii.edu/access/content/attachment/MAN.70882.201810/Assignments/422ba15f-9771-4b30-9463-df1edfd5d557/Hw6LEGLiteSingle.zip

[4] J. Cao. (2017, Nov 25). *EE-361-001 [MAN.70882.FA17]: Corrections on Lab 5 block diagram* [Online]. Available e-mail: jianqiuc@hawaii.edu Message: