Digital Design & Computer Arch.

Lab 8.1 Supplement: Full System Integration

(Presentation by Aaron Zeller)
Frank K. Gürkaynak
Seyyedmohammad Sadrosadati
ETH Zürich
Spring 2024
[14 May 2024]

Lab 8 Overview

- You will build a whole single-cycle processor and write assembly code that runs on the FPGA board.
- Don't worry! You have 2 sessions for the lab, and it will give you up to 14 points (+6 points from the reports).
- You will learn how a processor is built.
- Learn how the processor communicates with the outside world.
- Implement the MIPS processor and demonstrate a simple "snake" program on the FPGA starter kit.

Lab 8 Sessions

- **Session I:** The Crawling Snake

- **Session II:** Speed Up the Snake

Lab 8 Session I: The MIPS Processor

- Download the Vivado project from the course website
- A lot of parts are already implemented for you!
- What you will have to implement:
  - Compute the Instruction Memory address and read the instruction.
  - Connect the ALU.
  - Compute the Data Memory address and add the necessary wires.
  - Instantiate the Control Unit.

Lab 8 Session I: Memory-Mapped I/O

- Your goal is to control the 7-segment display with your assembly program.

- You will need to complete the I/O controller so the output of the processor will be correctly mapped to the display.

I/O in Assembly

- How do we communicate with the display?

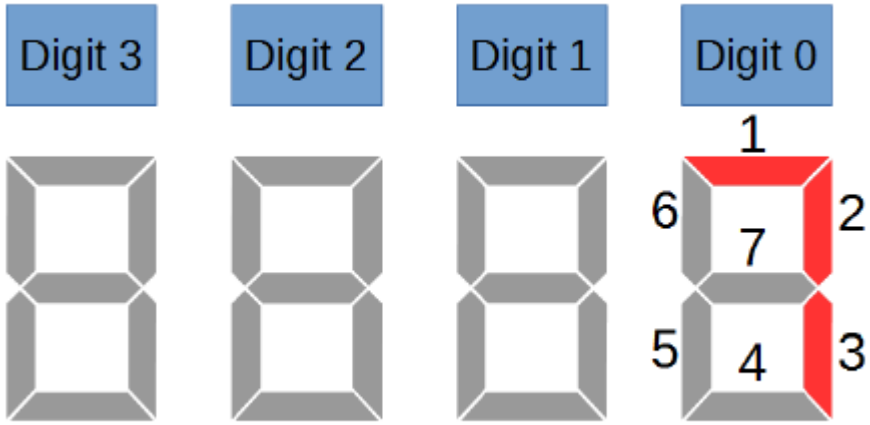
- Memory Mapped I/O
  - We designate specific addresses for the I/O
  - We can read and write to those addresses.
    - Example

```
# write contents of $t0 into memory at address 0x7FF0
# so that the I/O controller can send it to the display

                    sw $t0, 0x7FF0($0)
```

Lab 8 Session I : The Crawling Snake

- You learned how to write assembly code in Lab 7
- We provide you with the code for a crawling snake on the 7-segment display



- We will run this program on the processor you build in this lab!

Lab 8 Session I : The Given Code

- Read the lab manual carefully! It contains important information that will save you a lot of time.

- The code contains `//TODO` markers to show you where you are supposed to add things.

Lab 8 Session I : The Given Code

- The lab manual tells you what files contain and which files you need to modify.

top.v	Top level hierarchy that connects the MIPS processor to the I/O on the FPGA board. *You will modify this file for Part 2.*
top.xdc	Constraints file of the top level. *You will modify this file for Part 2.*
MIPS.v	The main processor. *For Part 1, you have to add code inside of this file only.*
DataMemory.v (datamem_h.txt)	The initial content of the data memory (composed of 64 32-bit words). *The* datamem_h.txt *file contains the data part of the assembly program in a hexadecimal form. This module "loads" the data. You will only have to modify the .txt file if you do the challenges.*
InstructionMemory.v (insmem_h.txt)	The ROM (composed of 64 32-bit words) contains the program. *The* insmem_h.txt *file contains the assembly instructions we want to run on the MIPS processor in a hexadecimal form. This module "loads" the instructions. You will modify the .txt file for Part 2.*
RegisterFile.v	Register file that creates two instances of *reg_half.v* as read ports and has one write port. *This is the implementation of a register. You do not need to modify it.*
reg_half.v reg_half.ngc	Component describing a single port memory and binary description of how it is mapped in the FPGA. *These are used to implement the register. You do not need to modify it.*
ALU.v	ALU similar to the one from Lab 5. *You should not change anything in this file, but if you want, you can use your own implementation (just make sure that it works).*
ControlUnit.v	The unit that does the instruction decoding and generates nearly all the control signals. Table 7.5 on page 379 lists most of them and their truth tables (only the `AluOp` signal is generated differently in the exercise). *This is just a combinational circuit, and it's already given; you don't need to change anything here.*
snake_patterns.asm	Assembly program corresponding to the *datamem_h.txt* and *insmem_h.txt* dump files that displays a crawling snake on the 7-segment display when all the parts are connected properly. *You have to modify this file for Part 2, where you will also learn how to generate the dump files.*

Lab 8 Session I : The Given Code

- For each `//TODO` in the code there is a part in the lab manual that explains what you are supposed to do.

```
// Memory Mapped I/O
assign IsIO = (ALUResult[31:4] == 28'h00007ff) ? 1 : 0; // 1: when datamemory address
                                                        // falls into I/O  address range
// TODO Part 1
assign IsMemWrite  =                // Is 1 when there is a SW instruction on DataMem
address
assign IOWriteData =                // This line is connected directly to WriteData
assign IOAddr      =                // The LSB 4 bits of the Address is assigned to IOAddr
assign IOWriteEn   =                // Is 1 when there is a SW instruction on IO address


assign ReadMemIO   = IsIO ? IOReadData : ReadData;   // Mux selects memory or I/O
// Select either the Data Memory (or IO) output or the ALU Result
assign Result = MemtoReg ? ReadMemIO : ALUResult;    // Slightly modified to include above
```

What's the difference between `MemWrite`, `DataMemWrite`, and `IOWriteEn`?

			Signal		
			`MemWrite`	`DataMemWrite`	`IOWriteEn`
Instruction	SW instruction	on DataMem address	1	1	0
		on IO address		0	1
	Non-SW instruction	D.C. (don't care)	0	0	0

Lab 8 Session I : Verilog Endianness

- Verilog does not assume endianness of buses.
- The endianness is determined depending on how you declare the bus.
- `wire[10:0] A` is **little endian**.
- `wire[0:10] B` is **big endian**.

Lab 8 Session I : Verilog Endianness

- `wire[10:0] A` is **little endian**.
  - `A[10]` is the most significant bit
  - A ends in the least significant bit.
  - `[10], [9], ... , [0]`

- `wire[0:10] B` is **big endian**.
  - `B[10]` is the least significant bit
  - B ends in the most significant bit.
  - `[0], [1], ... , [10]`

- `A[1:0]` selects the lower 2 bits in both cases.

Last Words

- You will build a whole single-cycle processor and write assembly code that runs on the FPGA board.
- You will learn how a processor is built.
- Learn how the processor communicates with the outside world.
- Implement the MIPS processor and demonstrate a simple "snake" program on the FPGA starter kit.
- You will have some questions to answer in the report.

Report Deadline

[31. Mai 2024 23:59]

Digital Design & Computer Arch.

Lab 8.1 Supplement: Full System Integration

Frank K. Gürkaynak
Seyyedmohammad Sadrosadati

ETH Zürich
Spring 2024
[14 May 2024]