Lab 8.2 Supplement: Full System Integration

(Presentation by Aaron Zeller)
Frank K. Gürkaynak
Seyyedmohammad Sadrosadati
ETH Zürich
Spring 2024
[21. May 2024]
Lab 8 Overview

- 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.
Lab 8 Sessions

- **Session I:** The Crawling Snake
- **Session II:** Speed Up the Snake
Lab 8 Session II: Speed Up the Snake

- In lab 8.1 you have made changes to the code template to produce a crawling snake on the 7-segment display.

- Lab 8.2 is similar and again requires careful reading of the lab manual.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top.v</td>
<td>Top level hierarchy that connects the MIPS processor to the I/O on the FPGA board.</td>
</tr>
<tr>
<td>top.xdc</td>
<td>Constraints file of the top level.</td>
</tr>
<tr>
<td>MIPS.v</td>
<td>The main processor.</td>
</tr>
<tr>
<td>DataMemory.v</td>
<td>The initial content of the data memory (composed of 64 32-bit words).</td>
</tr>
<tr>
<td>(datamem_h.txt)</td>
<td>The datamem_h.txt file contains the data part of the assembly program in a hexadecimal form. This module &quot;loads&quot; the data. You will only have to modify the .txt file if you do the challenges.</td>
</tr>
<tr>
<td>InstructionMemory.v</td>
<td>The ROM (composed of 64 32-bit words) contains the program.</td>
</tr>
<tr>
<td>(insmem_h.txt)</td>
<td>The insmem_h.txt file contains the assembly instructions we want to run on the MIPS processor in a hexadecimal form. This module &quot;loads&quot; the instructions. You will modify the .txt file for Part 2.</td>
</tr>
<tr>
<td>RegisterFile.v</td>
<td>Register file that creates two instances of reg_half.v as read ports and has one write port.</td>
</tr>
<tr>
<td>reg_half.v</td>
<td>Component describing a single port memory and binary description of how it is mapped in the FPGA.</td>
</tr>
<tr>
<td>reg_half.ngc</td>
<td>These are used to implement the register. You do not need to modify it.</td>
</tr>
<tr>
<td>ALU.v</td>
<td>ALU similar to the one from Lab 5.</td>
</tr>
<tr>
<td>ControlUnit.v</td>
<td>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).</td>
</tr>
<tr>
<td>snake_patterns.asm</td>
<td>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.</td>
</tr>
</tbody>
</table>
Let us first see some of the modifications made in lab 8.1.

IsMemWrite tells us if an instructions writes to memory or an I/O operation.

```plaintext
// 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 ? I0ReadData : ReadData; // Mux selects memory or I/O
// Select either the Data Memory (or I0) output or the ALU Result
assign Result = MemtoReg ? ReadMemIO : ALUResult; // Slightly modified to include above
```

IsMemWrite hence is 1 if an operation is a memory write operation and not an I/O operation.
Lab 8 Session I: The Crawling Snake

- **IsMemWrite** hence is 1 if an operation is a memory write operation and not an I/O operation.

- This is because there are I/O operations that also write to memory.

![Table showing the difference between MemWrite, DataMemWrite, and IOWriteEn](attachment:image.png)
Lab 8 Session I: The Crawling Snake

- **IsMemWrite** hence is 1 if an operation is a memory write operation and not an I / O operation.

- This is because there are I / O operations that also write to memory.

```verilog
// Memory Mapped I/O
assign IsIO = (ALUResult[31:4] == 28'h00007ff) ? 1 : 0; // 1: when datamem address falls into I/O address range

// **TODO** Part 1
assign IsMemWrite = 1; // Is 1 when there is a SW instruction on DataMem address
assign IOWriteData = 1; // This line is connected directly to WriteData
assign IOAddr = 1; // The LSB 4 bits of the Address is assigned to IOAddr
assign IOWriteEn = 1; // Is 1 when there is a SW instruction on IO address

assign ReadMemIO = IsIO ? IOReadData : ReadData; // Mux selects memory or I/O
assign Result = MemtoReg ? ReadMemIO : ALUResult; // Slightly modified to include above
```
Lab 8 Session II: Speed Up the Snake

- Extend the top-level hierarchy:
  - Modify the I/O controller to accept the inputs.

- Understand the provided assembly program and modify your assembly code to accept inputs.
  - The snake should crawl at different speeds for different inputs.
  - The inputs will be controlled by switches on the FPGA board.

- Optionally, you have two challenge tasks to complete.
  - Change the direction of the snake.
  - Change the pattern of the snake.
Lab 8 Session II: Speed Up the Snake

- You should proceed similarly to lab 8.1 here.

```cpp
// TODO for Part II of Lab 8
// The speed of the snake must be read as input and sent to the MIPS processor.
// Create the 32 bit IOReadData based on IOAddr value. Remember IOAddr is a 4-bit
// value.
assign IOReadData = ;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00007FF0</td>
<td>out</td>
<td>28 bits</td>
<td>Value to be sent to the 7-segment display.</td>
</tr>
<tr>
<td>0x00007FF4</td>
<td>in</td>
<td>2 bits</td>
<td>Speed step 0, 1, 2 or 3.</td>
</tr>
</tbody>
</table>

- The ALU is responsible for arithmetic, logic and memory address computation.
  - The **opcode** decides then what the result represents.
Lab 8 Session II: Speed Up the Snake

- You should proceed similarly to lab 8.1 here.

// TODO for Part II of Lab 8
// The speed of the snake must be read as input and sent to the MIPS processor. Remember IOAddr is a 4-bit value.

assign IOReadData = (IOAddr == 4 ? \_ : \_)

<table>
<thead>
<tr>
<th>Address</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00007FF0</td>
<td>out</td>
<td>28 bits</td>
<td>Value to be sent to the 7-segment display.</td>
</tr>
<tr>
<td>0x00007FF4</td>
<td>in</td>
<td>2 bits</td>
<td>Speed step 0, 1, 2 or 3.</td>
</tr>
</tbody>
</table>

Which 4 bits here are important? Remember that 4 bits in binary are one bit in hexadecimal.
Lab 8 Session II: Summary of the Flow

HW
- Edit Verilog files in Vivado
- Check Syntax
- Modify the XDC file if needed

SW
- Edit .asm files in MARS
- Assemble and simulate The code in MARS
- Dump the data & instruction memory as hex text files.

Generate the FPGA bit file in Vivado
Lab 8 Session II: Extending I/O

- **HW**
  - Edit Verilog files in Vivado
  - Check Syntax
  - Modify the XDC file if needed

- **SW**
  - Edit .asm files in MARS
  - Assemble and simulate The code in MARS
  - Dump the data & instruction memory as hex text files.

Generate the FPGA bit file in Vivado
Lab 8 Session II: Modifying the Assembly

**HW**
1. Edit Verilog files in Vivado
2. Check Syntax
3. Modify the XDC file if needed

**SW**
1. Edit .asm files in MARS
2. Assemble and simulate The code in MARS
3. Dump the data & instruction memory as hex text files.

---

Generate the FPGA bit file in Vivado
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

[07. June 2024 23:59]
Lab 8.2 Supplement: Full System Integration

Frank K. Gürkaynak
Seyyedmohammad Sadrosadati

ETH Zürich
Spring 2024
[21. May 2024]