ECE 443

MIPS-16, Part 1: Designing a 16-bit RISC style microprocessor

Lab #4: Basic components and Register File

Assigned **2/21/2023**
Report due **2/28/2023**

Introduction

The goal of this laboratory is to introduce you to the design of a simple microprocessor system. The microprocessor is a 16-bit machine that is based on the 32-bit Reduced Instruction Set Computer (RISC) machine that is presented by Patterson and Hennessey in Chapter 4 of their book "Computer organization and design: the hardware/software interface". The register file we are creating in this lab should look similar to the one that was presented in class (look it up in the slides or in the textbook).

Equipment:

Intel Quartus Prime (Lite) 20.1+ and **DE2-115** board with **Cyclone IV EP4CE115F29C7**

Procedure:

1. Locate the files produced for Lab 0.
2. Gather the material for digital logic components and VHDL from EE 243.
3. Carry out attached worksheet and exercises.

MIPS-16:

The processor is a 16-bit machine with the following instruction formats;

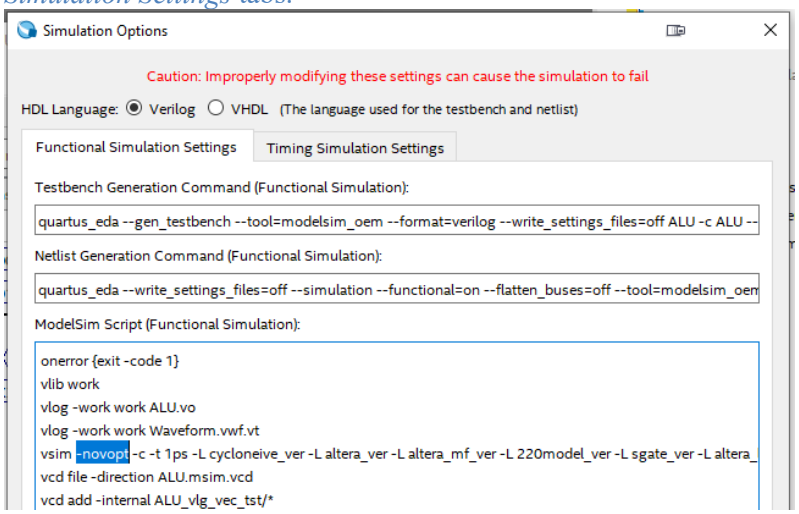
```
R-Type   opcode = I[15:12], Rs = I[11:9], Rt = I[8:6], Rd = I[5:3], funct = I[2:0]
I-Type   opcode = I[15:12], Rs = I[11:9], Rt = I[8:6], imm = I[5:0]
J-Type   opcode = I[15:12], adr = I[11:0]
```

The instruction set is given in Table 1.

Worksheet:

1. Register File Design and Implementation (NOTE: you should have done some of these steps in Lab0)
   (a) Create an 8-bit register component, **REG8**, in VHDL. The component has an 8-bit input D[7:0], a 1-bit enable, EN, a rising-edge triggered clock, CLK, and an 8-bit output Q[7:0].
   (b) Create a 4-input, 4-bit multiplexer, **MUX4x4**, in VHDL (use any appropriate VHDL statement, not gates) and use it to create appropriately sized multiplexers that will be used to select registers read by instructions (i.e., think about the size of multiplexers that will be needed to read TWO registers from the register file – see the step (d) for more details). Hint: those will be 16-bit wide and we shall call them **MUX8X16**.
   (c) Create a 3-to-8, **DCD3x8**, in VHDL (this component will select **where to write**).
   (d) Use these components in a structural architecture (i.e., use PORT MAPs) to design an 8×16-bit register file, **REG8x16** in VHDL. This component should have the following ports:
      i. Three 3-bit address inputs (ADD_R1[2:0], ADD_R2[2:0], and ADD_W[2:0]) for specifying **two read and one write register address** (i.e., your component should be capable of outputting TWO registers at the same time).
      ii. Two 1-bit inputs – WE (write enable) and CLK (clock).
      iii. One 16-bit input, DIN[15:0].
      iv. Two 16-bit outputs: DOUT1[15:0] (the output from a register specified by ADD_R1) and DOUT2[15:0].
   (e) Compile for Cyclone IV EP4CE115F29C7. Record the chip resources used (can be found in the compilation report created after each successful compilation; you can access it by clicking on *Processing⇨Compilation Report*.)
   (f) Analyze timing delays given in tables that are part of the compilation report.
2. Perform the functional and timing analysis using University Program VWF

Reminder - once in the *Simulation Waveform Editor*, click on *Simulation→ Simulation Settings* and remove the option "-novopt" from vsim command in both *Functional Simulation Settings* and *Timing Simulation Settings* tabs.



(a) Write the data into two different registers of your choice. Read the contents of these two registers simultaneously. What is the delay between the rising edge of the clock and the moment when the data becomes stable? Is there any difference between those times for different registers? Can you explain why this might happen?

(b) Conduct simulations of the register file using the fastest clock possible, showing its operation (write data, read data) for several input combinations. In order to do this, you will have to perform Timing Analysis (using the Timing Analyzer) to determine the longest propagation delay through your circuit. For details how to do this, please refer to EE243 Lab2 (posted).

Requirements:

1. The report should include printouts of all design files and all simulation files. Annotate the output waveforms to explain what is happening during simulation.
2. Follow the report template given on Blackboard.
3. **Upload the report to Blackboard. Include listings of your VHDL files in the Appendix.**

The best way to include code listings is to open the file in Notepad++, click on *Plugins→NppExport→Copy RTF to Clipboard* and paste it into your Word file. Play with font sizes and tab positions until it looks pretty.

TABLE 1. MIPS-16 INSTRUCTION SET

Instruction	*HDL*	*Type*
add Rd, Rs, Rt	Rd := Rs + Rt	R-type
sub Rd, Rs, Rt	Rd := Rs - Rt	R-type
and Rd, Rs, Rt	Rd := Rt and Rd	R-type
or Rd, Rs, Rt	Rd := Rt or Rd	R-type
slt Rd, Rs, Rt	if(Rs < Rt)then Rd = 0001_{16} (hexadecimal) else Rd := 0000_{16}	R-type
lw Rd, imm(Rs)	Rd := M(Rs + imm)	I-type
sw Rd, imm(Rs)	M(Rs + imm) := Rd	I-type
beq Rs, Rt, imm	if(Rs = Rd) then PC := PC + (imm×2)	I-type
j adr	PC := PC[15:13]:adr:0	J-type

TABLE 2. MIPS-16 ALU OPERATIONS

Operation	*SEL2*	*SEL[1:0]*
AND	0	00
OR	0	01
ADD	0	10
SUB	1	10
SLT	1	11

TABLE 3. MIPS-16 Instruction Set and Formats

Format	**Fields**	**HDL**	**Example**
R-Format	opcode:Rs:Rt:Rd:funct	Rd ← f(Rs, Rt)	add $1, $2, $3
I-format	opcode:Rs:Rt:imm	Rt ← M(Rs + imm)	lw $1,addr($2)
		if(Rt == Rs) then PC ← PC + (imm×2)	beq $1,$2, addr
J-format	opcode:addr	PC ← PC(15..p):addr:'0'	j addr

Note: The current value of the PC is the address of the next sequential instruction to be executed.

TABLE 4. MIPS-16 OPCODES	
Instruction	***Hexadecimal Opcode***
add	0_{16}
sub	0_{16}
and	0_{16}
or	0_{16}
slt	0_{16}
lw	B_{16}
sw	F_{16}
beq	4_{16}
j	2_{16}

TABLE 5. MIPS-16 Instruction Opcodes and Function Fields

Instruction	***Format***	***Hexadecimal Opcode***	***Function Field***
add	R	0_{16}	010
sub	R	0_{16}	110
and	R	0_{16}	000
or	R	0_{16}	001
slt	R	0_{16}	111
lw	I	B_{16}	n/a
sw	I	F_{16}	n/a
beq	I	4_{16}	n/a
j	J	2_{16}	n/a