UCSD CSE140L Fall 2013

LAB#4("Small computer design")

(Due: See course web page)

Instructor: Dr. Choon Kim

Objective

- Based on the experience from all previous LABs including **combinational**, **sequential** and **FSM**, learn how to design, implement, simulate, program and test **a small computer system** using Altera Quartus II CAD SW and DE1 FPGA board.

Instructions

---------- Preparation ----------

Step 1:

Download the following zip file which contains **LAB4 base project** from our Ted Discussion Board. It will be used as our base design.

LAB4_base.zip

Unzip and **load** the project("lab4_de1.qbf") to Quartus II Web Edition SW v.9.0 sp2 which we have been using for previous LABs. DO NOT use other version of Quartus II. **Compile** and **program** it to Altera board without any modification. You should not have any problem during the process.

Step 2: [Testing downloaded LAB4 base project to make sure you got a correct one]

The downloaded LAB4 base project is O.K. and ready to be used if it passes the following tests.

Note1: The " + P/R " means " followed by turning power on or reset(=pressing KEY[3])".

Note2: In some case, you may get the output without performing the " + P/R " step. However, it is a rule in LAB4 that the " + P/R " step **must be performed when specified**. If you miss the " + P/R " step, you make a mistake!

Note3: Ignore an initial HEX[3:0]="0000" display, if exists, occurring at the moment of + P/R

If SW[2:0] = 000 + P/R, "0353"
If SW[2:0] = 001 + P/R, a pattern("AAAA bbbb") repeats continuously.
If SW[2:0] = 010 + P/R, a pattern("0 1 2 3 4 5") repeats continuously.

Step 3:

Study the following document with **LAB4 base project** thoroughly. You need to make yourself familiar with the Computer & CPU operations by playing with the built-in example input memory file, "add.mif"(included in the document), on Altera DE1 board.

LAB4_tinycpu.pdf

---------- Main ----------

1. You design, implement and test a **small computer system** on DE1 FPGA board. You were given a base design(LAB4_base.zip). Your job is to update the base design following the specifications given in PART sections below. Carefully analyze & study the base project codes(Don't depend on comments too much. They are helpful in general in most cases. However they were written not by you but by someone else. Note that it is possible they can be confusing or even inaccurate in the worst case.)

2. [**REMINDER:** Golden solutions and rule to use]
   Golden solutions are provided to you. You need to play with them as a reference during design whenever you have a question. Remember the following rules when using the golden solution.

> Each Part of the LAB specifies most **input conditions** for your design. However, it is possible that not all the possible input conditions are specified.
>
> 1) For specified **input conditions**, your design must behavior exactly same as the golden solution.
> 2) For other **input conditions** which were NOT specified, your design does NOT have to behavior same as the solution provided. Your design is allowed to behavior anyway you want.
>
> Example:
> Suppose a Part specifies "If SW0 is up, turn LED0 on." and no further specifications. In this case, your design must behavior as follows.
>
> ```
> IF SW0=up    /* i.e., specified input condition */
>     LED0=on.     /* LED0 must be same as golden solution  ==> will be tested during Demo*/
>
> IF SW0=down  /* i.e.,  NOT specified condition(s)  ==> will not be tested during Demo */
>     LED0= can be either on or off   /* regardless of golden solution */
> ```

The golden solutions of LAB4 are:

random.pof , random.sof ,
addind.pof , addind.sof ,
addpcr.pof , addpcr.sof

3. During demonstration, different .mif files may be given to you from tutor/instructor for testing your design. If requested, you should compile & program your design with them during demo.

4. Additional or updated instructions may be given from Dr. Kim during class or "News&Updates" section of course web page.

---------- LAB#4 Project Operation Flow ----------

Warning: Following operations are ***__**prerequisite**__*** conditions. You will get **zero(0) point** for LAB#4 if you fail these operations.

1) Initial state: You should turn off power first. Then turn on power(with initial setting).
Before you turn on power, you should have following initial setting.
   - all sw are in DOWN position
   - no key is PRESSED

After you turn on power, your output must be as follows, and you are now in the initial state
   - all leds(i.e., ledg and ledr) are OFF
   - **HEX[3:0] displays your CID**.

2) Return to initial state from any other state
Turn off power first. Then turn on power(with initial setting above).

3) ** PART#1-4 testing must start always from the initial state.*****

For example, a sequence of testing should be,

initial state --> testing PART1 --> initial state --> testing PART2 --> initial state --> testing PART3 --> initial state --> testing PART4 --> initial state(see below for PART5) --> testing PART5

4) ** PART#5 testing must start from the initial state with one exception in the initial setting, sw[8]=1(instead of sw[8]=0).**

PART 1:

(Displaying your CID number on HEX[3:0])

****

Reminder: ** PART(#1-4) testing must start always from the initial state.*****

On initial state, display your CID on HEX . For example, HEX[3:0]=**0027** if your CID is 27.
(Note that this is not a new task but a part of above ***__**prerequisite**__*** conditions)

PART 2:

(Adding a new instruction: **RANDOM**)

****

Step 1(Updating the base project):

1. Add a new instruction RANDOM to our LAB4 base design. New instruction is defined as follows.

Instruction	Mnemonic	Operation Preformed	Opcode Value(in hex)
RANDOM	Data	AC <= 4-bit random number Starting from initial value, **9**	0x35

2. A 4-bit random number is generated in sequence by using the diagram in Lecture#5 slide#8 titled "Pseudo Random Sequencer". Note that Q0=**LSB**(Least Significant Bit), Q3=**MSB**(Most Significant Bit). For example, if Q3=1, Q2=1, Q1=0, and Q0=0, then the random number is C in hex format.

3. When SW[2:0] = 001 + P/R, your design must start the random number sequence on HEX[0] from the value **9**. If you fail starting from correct initial value(i.e., **9**), you will get **zero(0)** point. HEX[3:1]=000.

HEX[0] = " **9** 2 5 A 4 8 0 1 3 7 E d b 6 C "

REMINDER: Ignore an initial HEX[3:0]="0000" display, if exists, occurring at the moment of + P/R

Step 2(Testing your work):

Reminder: ** PART(#1-4) testing must start always from the initial state.*******

1. Download the following mif file into your LAB4 project directory: random.mif

2. Replace the following line in tc140l.v
   RAM.init_file = "add.mif" with
   RAM.init_file = "random.mif"
and compile & program the updated project to Altera board.

3. You should observe the followings on HEX(Ignore initial "0000" value at the moment of power turned on or reset). Note that leading 0s were not listed below.
If you fail starting from correct initial value(**9**), you will get **zero(0)** point.

If SW[2:0] = 000 + P/R, Your CID number.
If SW[2:0] = 001 + P/R, a pattern(" **9** 2 5 A 4 8 0 1 3 7 E d b 6 C ") repeats continuously on HEX[0].
If SW[2:0] = 010 + P/R, a pattern("0 1 2") repeats continuously.
If SW[2:0] = 011 + P/R, a pattern("3500 300") repeats continuously -- here, use turn power on to see the result better than using reset.

PART 3:

(Adding a new instruction: ADDIND)

****

Step 1(Updating the base project):

1. Add a new instruction ADDIND to our base design. It is defined as follows.

Instruction	Mnemonic	Operation Preformed	Opcode Value(in hex)
ADDIND	Address(**indirect**)	AC <= AC + contents of **indirect** memory address	0x36

2. In **indirect** addressing mode, a memory location pointed by **Address** field of instruction register contains a value which is a pointer to another memory location where data is located.

Step 2(Testing your work):

Reminder: ** PART(#1-4) testing must start always from the initial state.*****

1. Download the following mif file into your LAB4 project directory: addind.mif

2. Replace the line in tc140l.v

RAM.init_file = "add.mif"

with

RAM.init_file = "addind.mif"

and compile & program the updated project to Altera board.

3. You should observe the followings on HEX[3:0]. Ignore an initial HEX[3:0]="0000" display occurring at the moment of + P/R.

If SW[2:0] = 000 + P/R, Your CID number.
If SW[2:0] = 001 + P/R, a pattern(" 3 C F 18 1b 24 ") repeats continuously.
If SW[2:0] = 010 + P/R, a pattern("0 1 2 3 4 5 6 7") repeats continuously.
If SW[2:0] = 011 + P/R, instruction codes in mif file repeats continuously -- here, use turn power on to see the result better than using reset.

PART 4:

(Adding a new instruction: ADDPCR)
(note: **PCR** = Program Counter-relative addressing)
****

Step 1(Updating the base project):

1. Add a new instruction ADDPCR to our base design. It is defined as follows.

Instruction	Mnemonic	Operation Preformed	Opcode Value(in hex)
ADDPCR	Address(**PCR**)	AC <= AC + contents of **PCR** memory address	0x37

2. In **PCR** addressing mode, the effective address is calculated by adding PC value and the offset data in the address field of the instruction_register, i.e., effective address = PC + IR[7:0].

Step 2(Testing your work):

Reminder: ** PART(#1-4) testing must start always from the initial state.*****

1. Download the following mif file into your LAB4 project directory: addpcr.mif

2. Replace the line in tc140l.v
        RAM.init_file = "add.mif"
with
        RAM.init_file = "addpcr.mif"

and compile & program the updated project to Altera board.

3. You should observe the followings on HEX. Ignore an initial HEX[3:0]="0000" display occurring at the moment of + P/R.

If SW[2:0] = 000 + P/R, Your CID number.
If SW[2:0] = 001 + P/R, a pattern("A F 16 21") repeats continuously.
If SW[2:0] = 010 + P/R, a pattern("0 1 2 3 4 5") repeats continuously.
If SW[2:0] = 011 + P/R, instruction codes in mif file repeats continuously -- here, use turn power on to see the result better than using reset.

PART 5:

(Speed up Computer operation 5 times faster)

****

****** PART#5 testing must start from the initial state with one exception in the initial setting, sw[8]=1(instead of sw[8]=0)******

Warning: *DO NOT make sw[8]=1 in the middle of normal operation. It may confuse the system clock and you may see the system failing (or showing unexpected behavior).*

Up to PART4, we have turned on power with sw[8]=0 following initial state condition specification.

Now, if you set up the initial setting with one exception, sw[8]=1(instead of sw[8]=0) and turn on the power, your design's system clock speed should run **five times faster** than the speed of LAB4_base project given to you. As a result, all the operations(random, addind, addpcr) will run faster.

PART5 task is, you should demonstrate at least one of those operations(random, addind, addpcr) running **five times faster** than the speed of LAB4_base project given to you.

---------- The END of LAB#4 ----------