Spring 2012 Special Topics - Digital Design using Verilog and FPGAs Project

Assigned date: March 20, 2012 Due date and time: 5p.m., April 30, 2012

Mode of operation: -

- You MUST form a team of 3 persons for this project. E-mail me the name of your team members before 5p.m. on March 21, 2012.
- You are encouraged to post your queries on the forum for this class on LMS.
- Inter team in-person discussion is allowed to a reasonable extent (e.g, verifying your understanding of something). But, you are required to post the summary of such inter-team discussion on the forum. This way common mode error will be avoided.
- Read project description at least 2 times and clarify your doubts before designing your microarchitecture.

Your deliverables and schedule: -

- You must submit following on the indicated due date: -
    - A report document in pdf format containing following information: -
        - Diagram(s) providing overview of your design's organization. All modules and their relationship should be shown in the diagram. It is not recommended to provide details of all wire connections between various modules, but you must provide some high-level description of the interface between relevant modules.
        - A brief description (less than 1 page) of why you organized your design in the specific manner chosen by you.
    - Verilog source code for all modules of your design.
    - Testbenches used for verifying top-level module of your design.
- Presentations for microarchitecture of your design will happen on March 30. Each team member should present for 5mts in the total 15mts scheduled for your presentation. Additionally, 5mts will be reserved for Q&A.
- LMS submission link for the project will be provided at a later time. You must submit all required files in a zip file named <roll_numbers_of_all_team_members>.zip before 5p.m. on April 30, 2012.
- Demonstrations of your design on SP605 evaluation board will happen on May 1, 2012. Specific details will be communicated at an appropriate time later.

---

1 Overview

In this project, you will implement a 3-stage in-order issue and in-order completion 32-bit toy RISC processor in Verilog HDL. You will first verify your design's functionality using Verilog simulation and then synthesize your design using Xilinx synthesis tool. Later, you will implement your design on the SP605 evaluation board.

Section 2 describes general architecture and some other relevant functionality that will help you in designing your processor. Specific instructions that should be executable on your processor are described along with their format in section 3. Your Verilog modules must follow the naming conventions indicated in section 4.

Performance objectives for your design are described in section 5 and section 6 provides information about when to start the process of implementing your design on SP605 evaluation board.

2 General architecture

The RISC processor for this project should consist of 3 stages. In the first stage, instructions are fetched from the instruction ROM. Instructions are 32-bit wide and instruction ROM should be able to hold a maximum of 512 instructions. Program counter is maintained in this stage for sequential

fetching of instructions. When program counter points to the largest address of the instruction ROM, it doesn't increment and a *done* signal is raised. Do not interpret this *done* signal as actual indication of processor's completion of your program because it will be raised before last few instructions are actually executed. Such functionality is being asked for reducing your coding effort for generating *done* signal otherwise. Generated *done* signal will be interpreted appropriately by instructor's test infrastructure.

In the second stage, instructions are decoded and most of the specific control signals needed by your design for implementing those instructions are generated. Required operands are also read from the register file in this stage. Data path is also 32-bit wide and there are 16 registers in the register file.

Care should be taken regarding propagating updated contents of the required register(s), if any, along with the decoded control signals to the 3rd stage. For example, in case of *Bz* instruction (see section 3.1.3), you should ensure that latest contents of register ra should be forwarded with corresponding decoded signals for *Bz* to the 3rd stage. Similarly if any other instruction (e.g. arithmetic, logical, memory etc.) in 2nd stage needs updated contents from 3rd stage, then it should be ensured that latest results of previously issued instructions are supplied to the current instruction.

All instructions (including branch and memory operations) complete execution in the 3rd stage and update corresponding register or memory locations as specified in their required functionality. There is a separate register that holds the carry of the addition and subtraction operations, which gets updated along with the updating of addition and subtraction results in the register file.

Such execution requirement implies that branch is a two delay slot instruction. This means that two instructions immediately following branch instruction are always executed, irrespective of whether branch is taken or not. Branch instructions take effect (i.e. affect the program counter and instruction address supplied to ROM) in the 3nd stage because by then every instruction issued before branch is supposed to have written its result in the register file and carry flag register.

Data memory should be able to hold a maximum of 512 32-bit entries. Xilinx block RAMs must be used for implementing both instruction ROM and data memory. For the instruction ROM, we are not using actual ROMs available on the evaluation board because it will require additional effort to interface with them. Your design should ensure that processor can only read from the block RAM instantiated as instruction ROM. Hence using block RAMs should be sufficient for this project.

Reset for your design should be active low and clock should have 50% duty cycle.

You are required to prepare a microarchitecture of the design and present it in the class on the date indicated above. Microarchitecture should indicate various modules and interfaces between them that you will implement in Verilog afterwards.

3 Instruction set

Your processor should implement all instructions mentioned in section 3.1. Also, instructions are expected to be coded using the format specified in section 3.2.

3.1 Instruction details

Following naming conventions are used in this document for describing the instructions: -

- <rx> indicates contents of the register x.
- rx indicates index of the register x.
- Numbers are indicated by #Num.

3.1.1 Arithmetic instructions

3.1.1.1 Addition

Following instruction should perform unsigned 32-bit addition of <ra> and <rb>, and put the result in <rd> and carry flag register: -

Add rd, ra, rb

3.1.1.2 Subtraction

Following instruction should perform unsigned 32-bit subtraction of <ra> and <rb> (<ra>-<rb>), and put the result in <rd> and carry flag register: -

Sub rd, ra, rb

3.1.1.3 Multiplication

Following instruction should perform unsigned 16-bit multiplication of least significant 16-bits of <ra> and <rb>, and put the result in <rd>: -

Mul rd, ra, rb

3.1.2 Logical instructions

3.1.2.1 AND

Following instruction should perform 32-bit bitwise logical AND operation between <ra> and <rb>, and put the result in <rd>: -

And rd, ra, rb

3.1.2.2 OR

Following instruction should perform 32-bit bitwise logical OR operation between <ra> and <rb>, and put the result in <rd>: -

Or rd, ra, rb

3.1.2.3 NOT

Following instruction should perform 32-bit bitwise logical inversion of <ra>, and put the result in <rd>: -

Not rd, ra

3.1.2.4 Shift left logical

Following instruction should logically shift <ra> left by the amount specified by number ShiftAmnt, and put the result in <rd>: -

Sll rd, ra, #ShiftAmnt

ShiftAmnt is a 5-bit number supplied in the above instruction.

3.1.2.5 Shift right arithmetic

Following instruction should arithmetic shift <ra> right by the amount specified by number ShiftAmnt, and put the result in <rd>: -

Sra rd, ra, #ShiftAmnt

ShiftAmnt is a 5-bit number supplied in the above instruction. Recall that arithmetic shift replicates the MSB of <ra> while right shifting them.

3.1.3 Control flow related

3.1.3.1 Branch

Following instruction should result in fetching of subsequent instructions from the location specified by 9-bit number NextInstAddress: -

Br #NextInstAddress

3.1.3.2 Branch if zero

Following instruction should result in fetching of subsequent instructions from the location specified by 9-bit number NextInstAddress, only if <ra> is zero: -

Bz <ra>, #NextInstAddress

If <ra> is not zero, then instructions are processed in the normal sequential fashion.

3.1.3.3 Branch if carry

Following instruction should result in fetching of subsequent instructions from the location specified by 9-bit number NextInstAddress, only if carry flag register is set: -

Bc #NextInstAddress

If carry flag register is not set, then instructions are processed in the normal sequential fashion.

3.1.4 Memory operations and data movement

3.1.4.1 Load

Following instruction should fetch the 32-bit data in the register rd from data memory at the location specified by 9-bit number MemAddress: -

Ld <rd>, #MemAddress

3.1.4.2 Store

Following instruction should store the <ra> at the location specified by 9-bit number MemAddress: -

St <ra>, #MemAddress

3.1.4.3 Move from carry flag register

Following instruction should save the contents of carry flag register in the least significant bit of rd register: -

Movc <rd>

Most significant 31-bits of the <rd> should be cleared to 0.

3.2 Instruction format

Table 1. Instruction format

Instruction	**Opcode (most significant 4 bits, instr[31:28])**	**Operand 1 (next 4 bits, instr[27:24])**	**Operand 2 (next 4 bits, instr[23:20])**	**Operand 3 (next 4 bits, instr[19:16])**	**Least significant 16 bits, instr[15:0]**
Add rd, ra, rb	0000	rd	ra	rb	Don’t care
Sub rd, ra, rb	0001	rd	ra	rb	Don’t care
Mul rd, ra, rb	0010	rd	ra	rb	Don’t care
And rd, ra, rb	0011	rd	ra	rb	Don’t care

Or rd, ra, rb	0100	rd	ra	rb	Don't care
Not rd, ra	0101	rd	ra	Don't care	Don't care
Sll rd, ra, #ShiftAmnt	0110	rd	ra	Don't care	Least significant 5-bits for #ShiftAmnt. Remaining bits don't care.
Sra rd, ra, #ShiftAmnt	0111	rd	ra	Don't care	Least significant 5-bits for #ShiftAmnt. Remaining bits don't care.
Br #NextInstAddress	1000	Don't care	Don't care	Don't care	Least significant 9-bits for #NextInstAddress Remaining bits don't care.
Bz <ra>, #NextInstAddress	1001	Don't care	ra	Don't care	Least significant 9-bits for #NextInstAddress Remaining bits don't care.
Bc #NextInstAddress	1010	Don't care	Don't care	Don't care	Least significant 9-bits for #NextInstAddress Remaining bits don't care.
Ld <rd>, #MemAddress	1011	rd	Don't care	Don't care	Least significant 9-bits for #MemAddress Remaining bits don't care.
St <ra>, #MemAddress	1100	Don't care	ra	Don't care	Least significant 9-bits for #MemAddress Remaining bits don't care.
Movc <rd>	1101	rd	Don't care	Don't care	Don't care

4 Naming convention for Verilog modules

While you are free to specify and name internal modules and ports of your design in any manner, but you must follow specified naming conventions below for indicated higher-level modules and their ports. Notice that Verilog is case sensitive. Italics formatting for the names is used only for making viewing easier with respect to the surrounding text.

Few global conventions that should be followed: -

- Clock should be named *clk*.
- Reset should be named *rst*.

4.1 Instruction ROM

You must name your instruction ROM module as *instruction_rom.* Address port should be named *irom_addr* and port supplying instruction should be named *irom_data*.

4.2 Data Memory

You must name your data memory module as *data_mem.* Address port should be named *dmem_addr*. Data write port should be named as *dmem_wr_data* and port supplying read data should be named *dmem_rd_data*.

4.3 Core of the processor

Core module of the processor should be named as *iiitb_toy_proc*. Done signal mentioned in section 2 should be named as *done*.

5 Performance objectives

Your design should operate at the maximum possible clock frequency for the microarchitecture that you will be developing. Maximum clock frequency in this case may be limited by following factors among others: -

- Capabilities of the synthesis tool to obtain a fast implementation for your microarchitecture design.
- Your microarchitecture itself.
- Capacity of the Spartan 6 FPGA on the SP605 evaluation board.

Since your design is not dealing with the outside world or any other module not being coded by you, so input and output delays used during synthesis and physical design are not required for this project.

6 Final demonstration

You must demonstrate your design's functionality on SP605 evaluation board available in the CEEMS lab. Please contact me after you have verified the functionality of your design in simulation and synthesized it such that it is ready to be implemented on the FPGA board. I will allocate a SP605 board to you at that time.

Demonstration date is mentioned in the schedule above.