COMP3211/COMP9211 Computer Architecture

Lab 2 Single Cycle Processor

Goals

1. Study how to model a single cycle processor core using HDL
2. Build simple single cycle processor

The lab is based on an existing single cycle core model (in VHDL) created by a previous research student Ms. Lih Wen Koh (a similar Verilog model written by Dr. Sajid Hussain is also available on the course website). The block diagram of the core is given in single_cycle_core.pdf. All the related HDL files can be found in single_cycle_core.zip (available on the Labs page). The instructions to create a VHDL project and run simulation with Xilinx Vivado for the processor are given below. The same procedure can also be applied to the Verilog model.

1. Build the single cycle core project

- Download and unzip single_cycle_core.zip to a suitable location on your computer and uzip it to a folder.
- Start Xilinx Vivado and create a new project.
- Add single_cycle_core.vhdl and other vhdl files in the folder to the project, as shown in Figure 1.

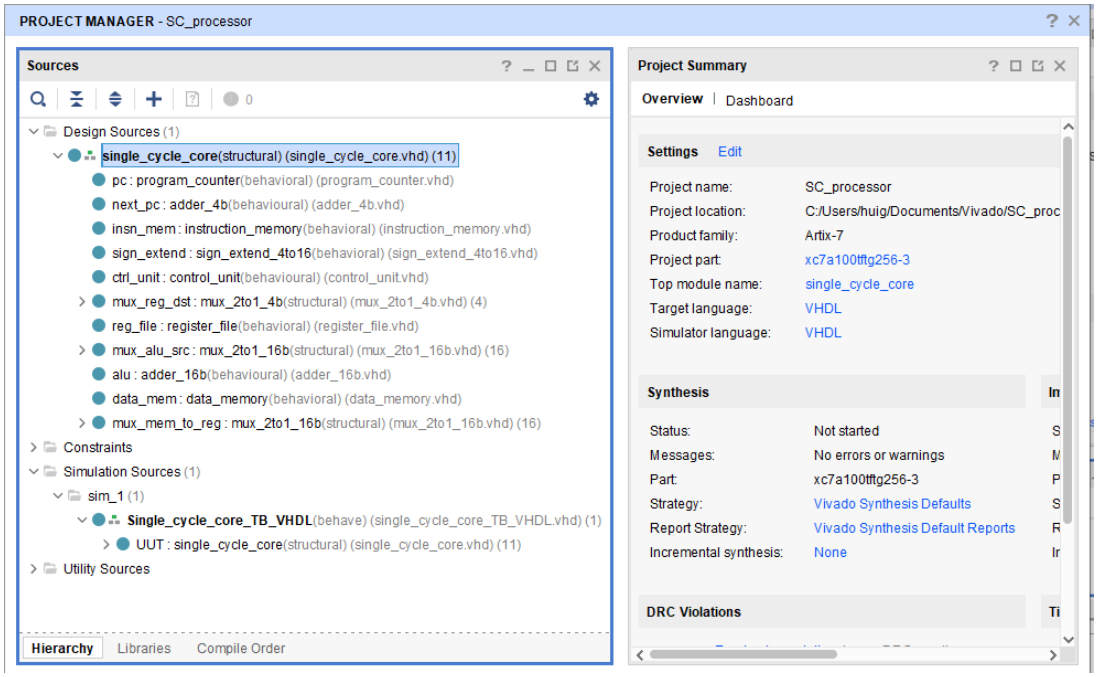

Figure 1

2. Simulation

- Click Simulate Behavioral Model.
- In the simulation window, add signals you are interested in to the waveform window, as detailed below.
    - Expand the model hierarchy in the Scope box. There will be a list of corresponding signals displayed in the Objects box.
    - Drag signals you want to investigate from the box to the waveform viewer, as shown in Figure 2.
    - Rerun the simulation to generate the waveforms for all signals selected, as shown in Figure 3.
- Save the simulation to a .wcfg file. You can use this file as the configuration for future simulation.

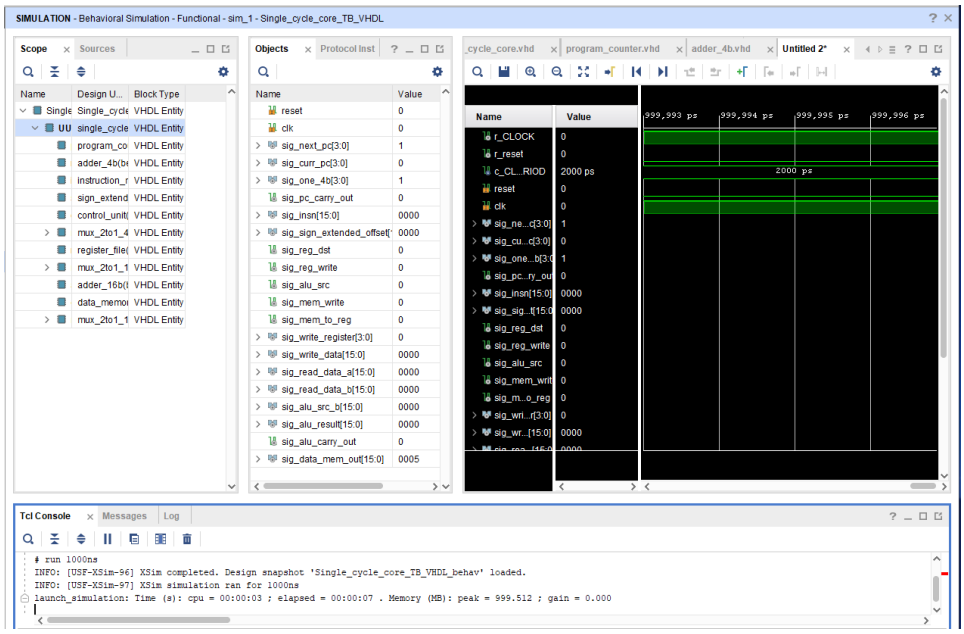

Figure 2

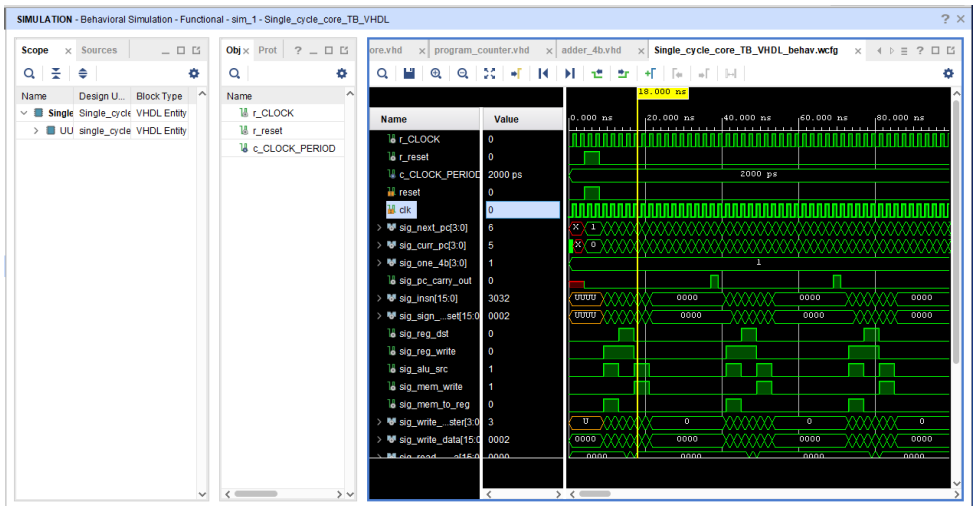

Figure 3

3. Tasks

Task 1: (20 marks)

- Study the HDL code in the single_cycle_core project.
- Modify the design to include the following two instructions
    - left-shift instruction, *SLL Rd, Rs, Rt*, that left-shifts contents in Rs the number of bits given by Rt and saves the result to Rd.
    - Conditional branch instruction, *BNE Rs, Rt, imme*: When the contents of the two registers Rs and Rt are not the same, the execution flow changes to the branch target. You need to decide how the PC is updated for this instruction.
- Verify your design.

Task 2: (optional, 25 bonus marks)

In some application areas, a processor design that leads to a small code size is desired. One possible way to reduce code size is using stack-based operations, where most instructions are performed on stack. Since the locations of the instruction operands are known and do not need to be explicitly specified in the instruction, the instruction size can be small. Some examples of register-based instruction and stack-based instruction are given in the table below for comparison.

Table 1 Register-based Instructions vs Stack-based Instructions

Operation	Register-based		Stack-based	
	example	meaning	example	meaning
read memory	load Rd, A	Rd <= mem(A)	push A	stack(SP)<= mem(A), SP <=SP-1
write memory	store A, Rs	mem(A) <= Rs	pop A	mem(A) <= stack(SP), SP<=SP+1
addition	add Rt, Rs	Rt <= Rt+Rs	add	stack(SP+1) <= stack(SP)+stack(SP+1), SP<=SP+1

* In the table, SP is the stack pointer, stack(SP) the top of the stack and A the memory address; The stack grows to the smaller address.

For this task,

- Convert the single_cycle_core model into a new design so that all operations carried out by the existing instructions are now performed on stack (rather than registers). Specifically,

    - The new core has a stack which can be implemented by a set of registers, as demonstrated in Figure 4 (a), where the stack bottom is Reg15 and the stack grows towards smaller register number. The stack pointer always points to the top of the stack. The shaded registers hold valid values. When the POP instruction is executed, the value on the top is popped out to the memory and the SP is changed to point to the new top (Reg13), as shown in Figure 4(b). Similarly, after PUSH operation, a memory data is saved on to the stack and the SP is changed to new top (Reg11), as shown in Figure 4(c).
    - Create a new instruction set and related instruction format(s).

- Verify your design

Note: Task 2 is optional. But you can earn up to 25 marks if you attempt it.

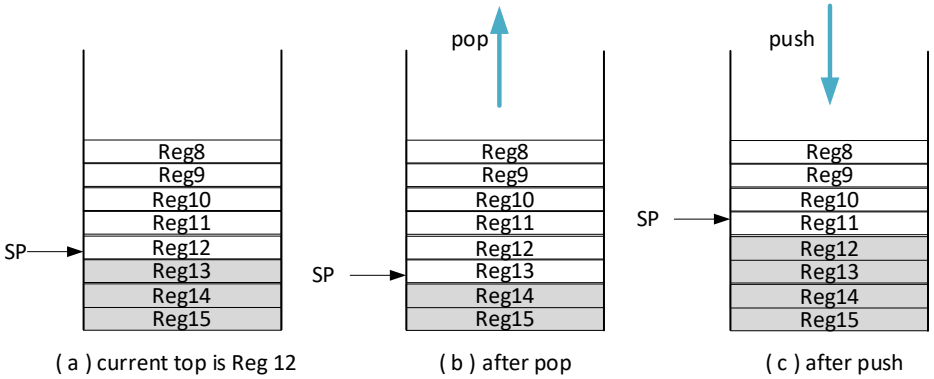

Figure 4

Due Time:

- Task 1, your TLB class in Week 4
  - Lab presentation for **peer assessment.** The scheme is given below.
- Task 2, *give* submission, 18:00, Friday, Week 5
  - Zip your Vivado project for the task and submit it using *give*.
    - Your code should be well commented and easy to read.

Peer Assessment Scheme:

Your work for **Task 1** will be assessed by your peer students and tutor. As for Lab 1, for this lab

- Your TLB class is randomly divided into two assessement groups, each capped at 11 students. This is to allow sufficient time for the assessment to be completed in the 2-hour lab class.
- Each student is given 6 mins for presentation and 2 mins for Q&A.
  - The presentation covers three areas: design idea, HDL model, simulation results.
- The work is assessed based on three categories:
  - Presentation (0-4)
    - clarity
    - logic
    - timing management
  - Completion (0-4)
  - Quality (0-4)
- The link of the assessment form will is available in your MS team.
- Your tutor will organize and steer the lab.

- By the end of the lab class, all students are required to submit their assessment forms. Your participation to the assessment will be taken into account to the overall lab participation marks.