104-2 Digital System Design Homework #4

Cache Unit Design

Announced at May 4, 2016
Deadline at 18:00, May 25, 2016

TA Information
郭泓圻，電二 232 實驗室
charleykuo@access.ee.ntu.edu.tw

1. Problem Statement

Many system designs involve the microprocessor, the memory, and peripherals which communicate with each other on a system bus or other protocols. Usually the clock speed of the processor is much higher than the memory and the bus system. In order to utilize the processor speed, caches are necessary to be added for buffering the data between the processor and the memory.

Figure 1 shows a pipelined MIPS with both data memory and instruction memory. This homework aims to design a cache unit to buffer data between the MIPS processor and one of the memories.

The I/O specification of the desired 32-word cache unit is illustrated in Figure 2. The cache should be implemented by using the direct-mapped architecture, which contains 8 blocks and 4 words in each block. Two types of the buffering mechanisms, direct write-through and write-back, are both available so that you may choose one by yourself.

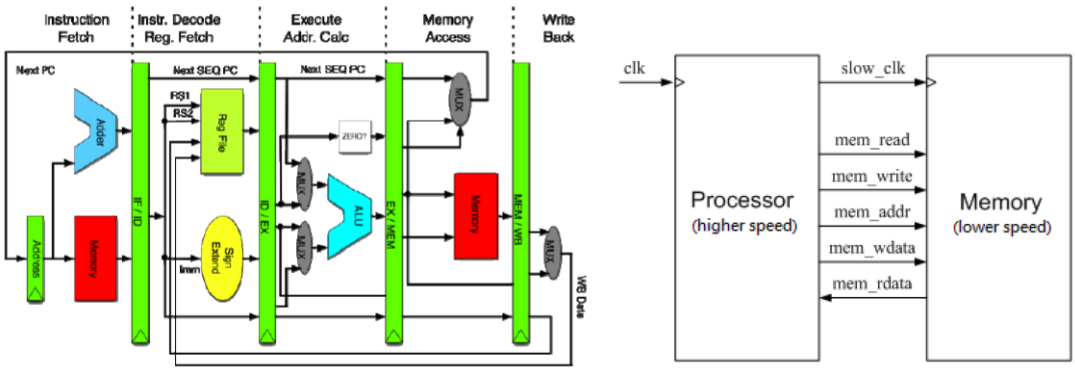


Figure 1. Pipelined MIPS architecture and the memory interface.

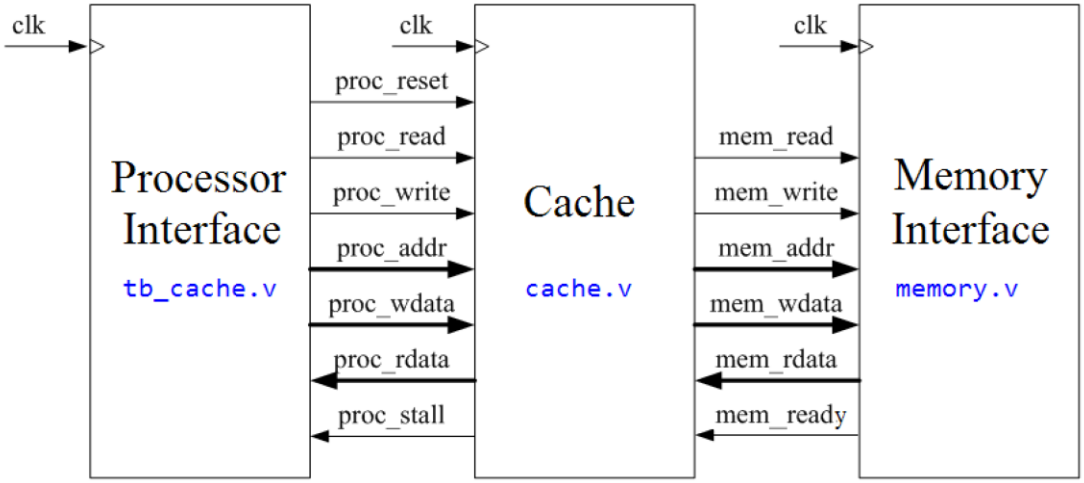


Figure 2. I/O illustration of the cache unit.

2. I/O Specification of the Processor Interface

Name	I/O	Width	Description
`clk`	I	1	positive edge trigger clock
`proc_reset`	I	1	asynchronous active-high reset signal
`proc_read`	I	1	synchronous active-high read enable signal
`proc_write`	I	1	synchronous active-high write enable signal
`proc_addr`	I	30	address bus (*word address*)
`proc_wdata`	I	32	data bus for writing to cache
`proc_rdata`	O	32	data bus for reading from cache
`proc_stall`	O	1	active-high control signal for processor that indicates

3. I/O Specification of the Memory Interface

Name	I/O	Width	Description
`mem_read`	O	1	synchronous active-high read enable signal
`mem_write`	O	1	synchronous active-high write enable signal
`mem_addr`	O	28	address bus (*4-word address*)
`mem_wdata`	O	128	data bus for writing to slow memory
`mem_rdata`	I	128	data bus for reading from slow memory
`mem_ready`	I	1	asynchronous active-high one-cycle signal that indicates data arrives from memory

4. Address Mapping

There are 8 blocks inside the cache unit and each contains 4 words. The processor accesses one word at a time, while the cache unit can only access 4 words at a time from the memory. That is, the last (rightmost) 2 bits of `proc_addr` should be translated as the *offset* inside one block, indicating which word of the 4 words should be accessed. The leftmost 28 bits are then directly translated to be the `mem_addr` while accessing. Figure 3 demonstrates the address mapping between the processor, the memory, and the cache unit.

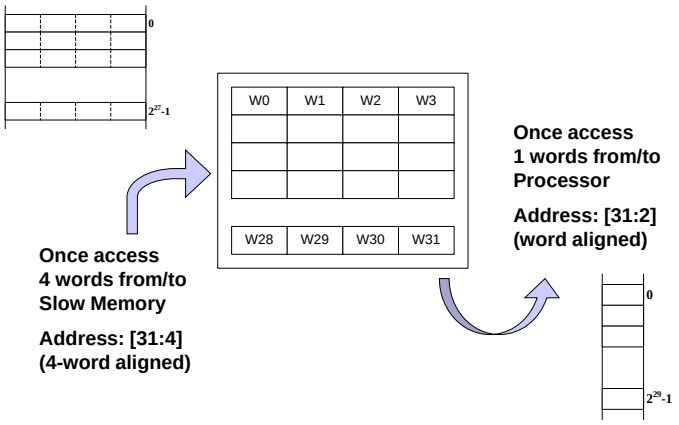


Figure 3. Address mapping of the system.

5. Functional and Timing Specification of the Processor Interface

The processor interface is actually a pseudo processor that only performs the memory accessing tasks in order to verify the functionality of the cache unit. Therefore, all signals from the processor interface change at the rising clock edge with a very slight input delay.

Whenever the cache needs to access data from memory and wait for the memory, `proc_stall` signal should be set high to stall the processor. As for timing specification, the `proc_stall` signal should be set high before the positive clock edge so that the processor can be stalled correctly. Also, the `proc_rdata` should be also prepared before the positive clock edge if the data should be sent to the processor at that cycle.

There are two possible cases where one stall is necessary:

(a) *Read miss* in both write-through and write-back caches;

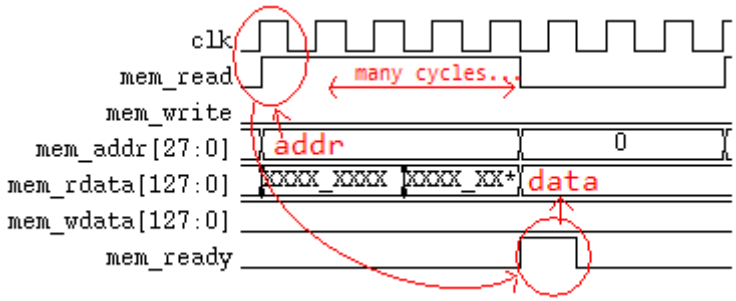
(b) *Write hit* in *only write-through* caches.

Since *write miss* can be regarded as "*read miss then write hit*," a write miss in write-through caches involve two successive stalls, while only one stall is needed in write-back caches.

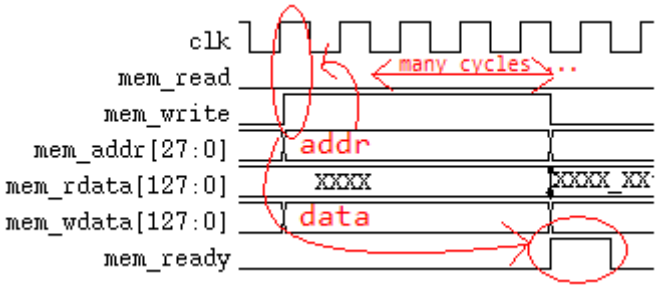
The pseudo processor in the given testbench performs only three things. It initially reads the whole data from the memory, then rewrites the whole memory with new data, and finally read the rewritten data from the memory. All possible cases are considered, so the finite state machine in the cache unit should be considered very carefully and neatly.

6. Functional and Timing Specification of the Memory Interface

The memory interface is given with the functional model of an ordinary memory or a system bus. Figure 4 demonstrates the timing diagram of the memory interface. After the memory is set to read or write mode, it finishes the operation in many cycles with a `mem_ready` signal, which indicates the required data is ready in read mode or the data is written correctly in write mode. If the `mem_read` and `mem_write` signals are both set high and low, the memory would do nothing.



(a) Read operation.



(b) Write operation.

Figure 4. Timing diagram of the memory interface.

7. Homework Requirements

Verilog Code. A *synthesizable* Verilog code of the cache unit named "**cache.v**" is required. The submitted design should pass the given testbench. You are not required to synthesize the code, but there should be *no latches* and *no timing expressions* in your design which are not allowed in a synthesizable design. (Please check this issue by the command "`read_verilog`" in *Synopsys Design Vision*.)

Report. The report should be named as "**report.pdf**" in **PDF** format, and should have at least four parts, including:

(a) General specification of the cache unit (such as numbers of words, placement policy, and so on);
(b) Read/write policy (write-through or write-back);
(c) Design architecture or the finite state machine of the cache unit;
(d) Performance evaluation of your cache design, including the miss rates of read/write operations, the execution cycles, the stalled cycles, and so on.

Besides, more appreciation will be given if you give some discussions of the cache architectures, or describe your design guideline in details and the experiences you gain from this homework.

Grading policy. Note that *the report is as important as the Verilog code*, so it is suggested that the code should be finished as soon as possible to reserve time for a qualified report. Here are the rules.

(a) A qualified code and report will be each graded as full score.
(b) If the code passes the RTL simulation but is not synthesizable, you would only get at most 60% of the coding score.
(c) If the report is just copy-paste of the simulation results, you would also get at most 60% of report score.
(d) The scores of the late submission would be 10% off per day.

Bonus. If your code is implemented with other skills that enhance your cache unit significantly, there will be an extra bonus up to 10 points to your grade. ***Prerequisite:*** You should describe your methods or architectures in detail in the report!

8. Submission Requirement

1. All the files need to be compressed as a single ZIP or RAR file. Send this file to TA via FTP:
   Address : **140.112.20.128** Port : **1232**
   Account : **DSD_STUDENT** Password : **dsd2016**

   Example of filename
   DSD_HW4_b00943135.zip
   DSD_HW4_b00943135_v2.zip
   .
   .
   .

   Your submitted file should include the following files:
   DSD_HW4_b00943135/
   cache.v
   Report_ HW4_b00943135.pdf

2. The homework will be graded ONLY IF the filename of your submission are correct!