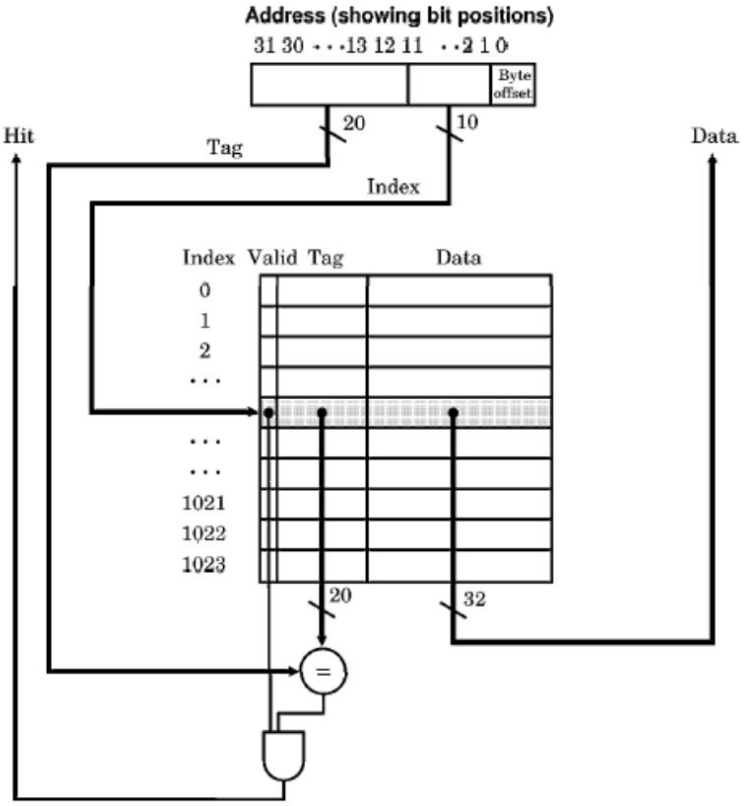
ECSE 425: Computer Organization and Architecture
VHDL Project, Part 1: Cache

Due February 10, 2017, 11:59 PM

Introduction

The goal of this assignment is to make a cache circuit. You will implement a design similar to the one in the figure below:



Later, you will connect your cache to a MIPS processor.

Your cache should have the following parameters:

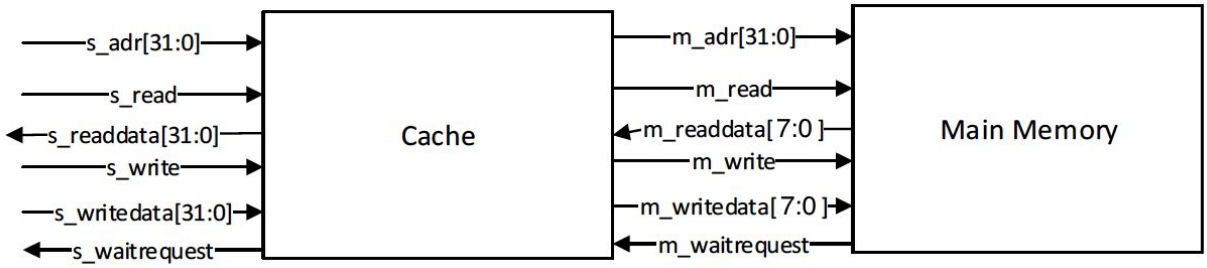
- Write-back policy
- Direct-mapped
- 32-bit words
- 128-bit blocks (4 words per block)
- 4096 bits of data storage (32 blocks)
- 32-bit addresses

You should implement the storage for data, tags, and flags (= dirty bit and valid bit) as a VHDL array. The `memory.vhd` file provided to you shows an example of using arrays.

While the MIPS processor uses 32-bit addresses and can theoretically address 2^32 different bytes, the main memory here has only 2^15 bytes (32768 bytes). Hence, your cache should simply use the lower 15 bits of the address and ignore the rest.

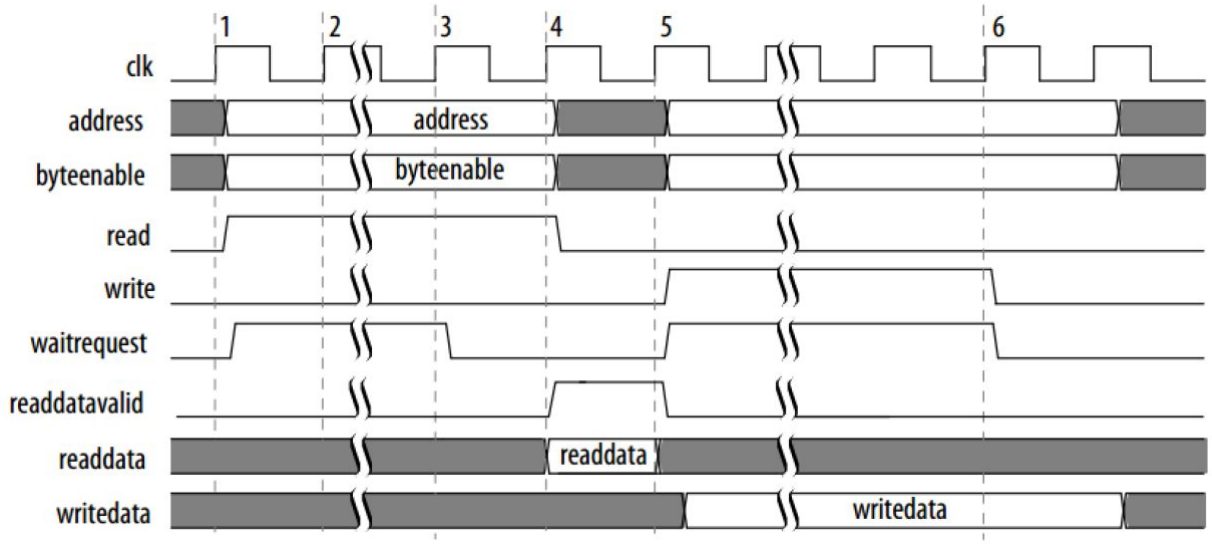
Additionally, although MIPS uses byte addressing, your cache should read and write complete words rather than individual bytes. Thus, you can assume for this assignment that the processor will only ask to read and write words and will provide addresses which are word-aligned (multiples of 4). That means that your cache should also ignore the last two bits of the input address.

Your cache VHDL entity will have the inputs and outputs shown in this figure:



Deviation from this interface by the addition or exclusion of ports will result in lost marks.

The Altera Avalon interface defines the timing with which data should be transferred between a master device and slave device. Your cache should interact with the main memory (and any circuit in which you instantiate the cache) using the Avalon interface. An example Avalon timing diagram is shown here (ignore the `byteenable` and `readdatavalid` signals, which you will not need to use):



More information about the Avalon interface can be found at https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf. Note that while the memory testbench we provide uses "`wait until ...`" to communicate with the memory, this is not synthesizable behaviour (i.e., you can't use statements with "wait" to generate hardware; it's only appropriate for simulation). Instead, your cache should use an FSM to implement the Avalon interfaces.

Test cases

Since a cache block can be {valid/invalid, dirty/not dirty}, and an access to a block can be {read/write, tag equal/tag not equal}, there are theoretically 2^4 = 16 different cases your FSM must handle. (Of course, some of these are impossible; for example, a block won't be marked dirty if the data are invalid.)

Your testbench must cover all possible combinations of {valid/invalid, dirty/not dirty, read/write, tag equal/tag not equal} for a cache access. You are required to come up with a set of memory accesses which will trigger all the relevant cases.

Where to start

Four files are provided to you for this assignment:

- `cache.vhd`: Put your code here. *Do not change the port structure.*
- `cache_tb.vhd`: Put your testbench here.
- `memory.vhd`: This is the lower-level memory with which your cache will interact. *Do not change this file.*
- `memory_tb.vhd`: This testbench shows an example of accesses to the lower-level memory.

Grading

Your deliverable will be evaluated based on the (a) correctness of your design as evaluated with our testbench, and (b) completeness of your testbench, with respect to test coverage. Indicate in your code comments how your test memory accesses relate to the different test cases.

Hand In Procedure

Hand in, via MyCourses, in a single ZIP file:

- `cache.vhd`
- `cache_tb.vhd`
- Any other VHDL source files you need to implement the cache (e.g., you may wish to encapsulate the FSM and cache storage array within separate files).