ECSE425 Project2 - Direct-Mapped Cache

Winter 2021

due date: Feb 18, 2021

The goal of this project is to implement to a basic, direct-mapped cache circuit. After this project, you should have a fundamental understanding for cache memory and will later use the concepts from this assignment in the final project of MIPS processor implementation.

1 Introduction

Cache structure:

- direct-mapped
- write-back policy
- 32-bit words
- 128-bit blocks (4 words)
- 4096-bit of data storage   4096/128 = 32 blocks
- flag bits 'valid' and 'dirty'

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.

Main memory:

- 32-bit address
- 32,768 bytes (32,768/4 = 8192 words)

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. Note that ignoring the last two bits does not mean eliminating the last two bits!

Your cache VHDL entity will have the inputs and outputs shown in the figure below, **deviation from this interface by the addition or exclusion of ports will result in lost marks**:

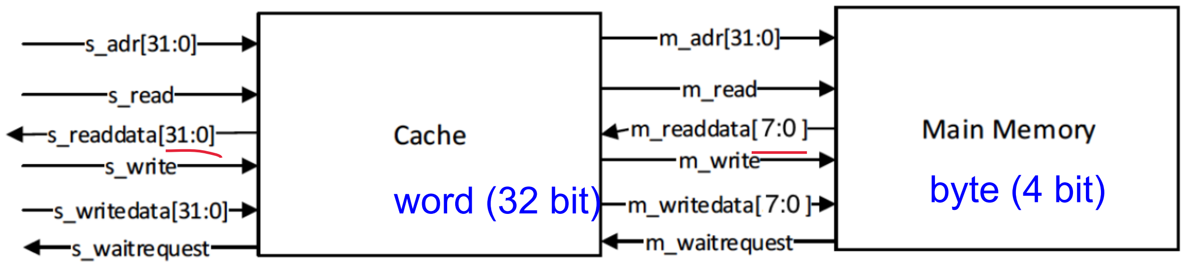

Figure 1: Cache Circuit I/O Ports

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):

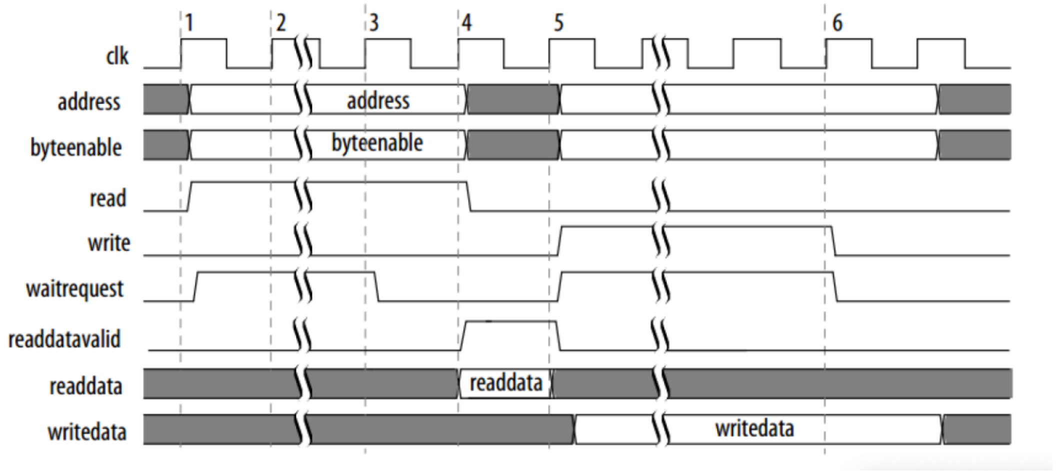

Figure 2: Avalon Interface

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

Your cache should use a Finite-State-Machine to implement the Avalon interfaces. For this assignment, since we're only using the *waitrequest* signal and no *readdatavalid* signal, we just set *waitrequest* high by default. When the transaction is complete, the slave sets *waitrequest* low for a clock cycle (indicating the read data is valid or write operation complete), then sets it high again. From the Avalon interface manual: "An Avalon-MM slave may assert *waitrequest* during idle cycles. An Avalon-MM master may initiate a transaction when *waitrequest* is asserted and wait for that signal to be deasserted. To avoid system lockup, a slave device should assert *waitrequest* when in reset."

2 Where to start

2.1 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.

2.2 Files provided

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.

3 Grading

Your deliverable will be evaluated based on the (a) correctness of your design, (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, and (c) report quality (similar format and standard as Project1 - FSM)

You are expected to write a short report (IEEE format, max 4 pages) summarizing your implementation. Your report should at least include the following content:

- Calculation of the number of bits required for tag, block index, and block offset. Block offset has two components, word offset and byte offset, clearly indicate the number of bits required for these two components as well.
- Explain your implementation and provide the state diagram of your implementation (i.e. your VHDL implementation should follow the state diagram in your report).
- State the cases that are impossible and explain why they are impossible.
- For each test case in your cache_tb.vhd, show the test results and explain how the test memory access relates to the test case. You might want to show the signal values provided by ModelSim for a better explanation.
- If you have additional files other than the ones provided to you, provide explanation for those additional files.

4 Submission

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

- cache.vhd, cache_tb.vhd, memory.vhd, memory_tb.vhd
- cache.tcl file that compiles the files and runs the simulation
- 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).
- Report **(failure to submit the report will result in a grade of 0)**

Name your submission folder as "GroupXX_Cache"