COE758 – Digital Systems Engineering

Project #1 – Memory Hierarchy: Cache Controller

Objectives

- To learn the functionality of a cache controller in "hit-miss" modes and its interaction with block-memory (SRAM based) and SDRAM-controllers.
- To get practical experience in the design and implementation of custom logic controllers and interfacing to SRAM memory units and other logic devices.
- To learn VHDL-coding technique in the environment of the Xilinx ISE CAD system and a hardware evaluation platform based on the Xilinx Spartan-3E FPGA.

Background

Most computers are designed to operate with different types of memory categorized in a structure known as a memory hierarchy. At first glance, it can be noticed that as the distance from the processor increases, so does the volume of the memory. This structure, with the appropriate operating mechanisms, allows memory to be accessed in shorter periods of time the closer it is to the processor. The closest to the CPU is the Cache Memory. Cache memory is a type of memory for temporary storage of data that is likely to be used again soon. Each entry in the cache is composed of a number of identifiers; they include the **index**, **tag**, **valid** and **dirty** bits. All the above identifiers are associated with the block of data. A block of data contains a number of data-words stored in a certain order. Thus, the **block offset** is also included into the list of data-identifiers. These identifiers give the cache information about whether the requested data item is in the cache or not. The information stored in the identifiers is processed by a Cache Controller. This part of the cache determines hit or miss conditions for a data item and delivers the item to the CPU or writes the data sent by the CPU to a given location in a block. Two other parts of the cache are the Block RAM and Main Memory Controller (Figure 1). The Block RAM is an SRAM based memory module which contains actual data-blocks. The Main Memory Controller provides access to the DRAM based main memory and thus performs the procedure for the block replacement in case of miss.

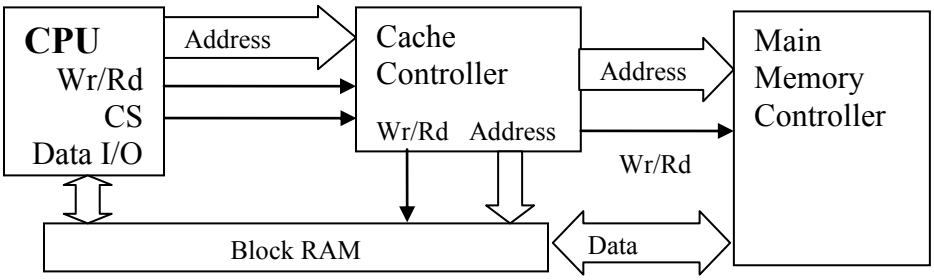

Figure 1: Cache block diagram

Application: Hit/Miss determination of a Cache system

1. Description

A 256 byte block-organized cache has a 32 byte block containing 32 data words. Figure 2 shows the organization of the cache. Each data word is 1 byte in size. Following the boot-up or a reset of the computer, the contents of the cache are empty or invalid. When trying to access the cache after these occurrences, it is inevitable that a cache misses will occur. A block replacement procedure must then read an entire block from main memory and replace the corresponding block in the cache.
In the example cache-based system, the CPU issues addresses to access the cache. Depending on the cache's contents, it will either provide the CPU with immediate data from SRAM or transfer data from the main memory controller into its contents and then provide the CPU with its requested data. Figure 3 illustrates this concept.

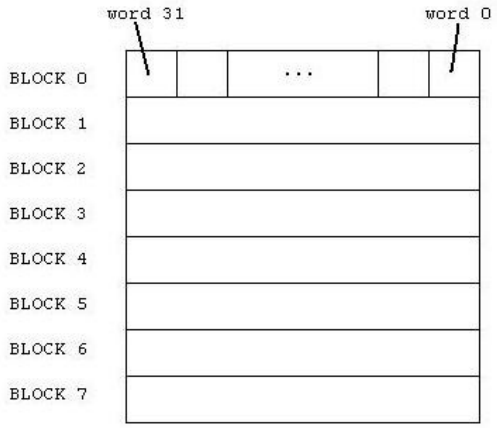

Figure 2. Organization of the Cache (on the base of Block RAM)

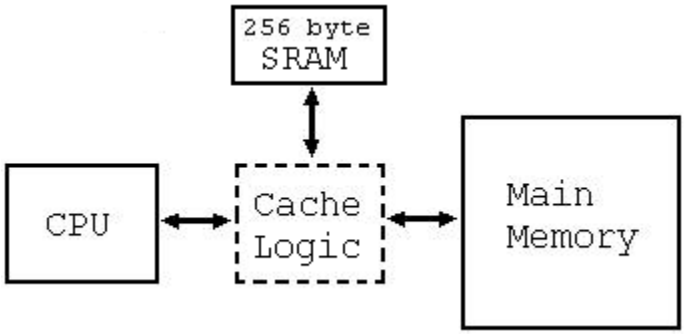

Figure 3. Cache based system

2. Specification

CPU Address Word Organization

The 16 bit address word is issued by the CPU with *RD/ WR*-signal and CS-strobe. The address word is received by the Cache Controller and stored in the Address Word Register (AWR). AWR consist of three fields according to cache organization (see Figure 4). A 256 byte cache memory consists of: 8 entries (blocks) with 32 words (bytes) / block. Therefore: the I**ndex** field consists of 3 bits and the **Offset** field consists of 5 bits. Finally, the upper 8 bits of the address word represent the **Tag**. Figure 4 illustrates the AWR field division.

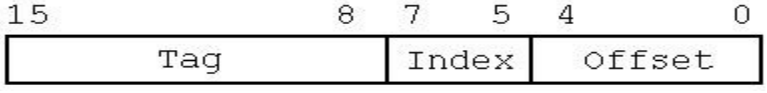

Figure 4. Address Word Register Fields

3. Design

You are to design a logic circuit that determines whether the data requested by the CPU is found in the cache memory (i.e., a cache hit or cache miss). This requires you to compare the **tag** value from the CPU address word with its corresponding value found in the tag registers. The following cases have to be considered and implemented in the design of the Cache controller:

Design Cases

1. Write a word to cache [hit]
   The cache is provided an address by the CPU, which also contains information that tells us the command is to be a write. In this situation, the request made by the CPU is found in the cache, thus being a cache hit. The **index** and **offset** portion of the address supplied by the CPU is to be sent to the Block RAM as the write address for the new data (the Cache memory module should be implemented using Block RAM according to Tutorial 3). Also, the cache's **dirty** bit (in the Cache Controller) and the **Write Enable** (WE) signal to the BRAM must both be asserted.

2. Read a word from cache [hit]
   The cache is provided an address by the CPU, which also contains information that tells us the command is to be a read. In this situation, the request made by the CPU is found in the cache, thus being a cache hit. The **index** and **offset** portion of the address supplied by the CPU is sent to the SRAM which contains the data needed by the CPU (implementation of the data access is based on Block RAM according to Tutorial 3).

3. Read/Write from/to cache [miss] and dirty bit = 0
   In this situation, the request made by the CPU is not found in cache, resulting in a cache miss. This requires performing a block replacement in the cache. First the **dirty** bit for the corresponding CPU address must be analyzed. If the **dirty** bit is low (0) then the entire CPU address with the **offset** portion set to "00000" is passed to the main memory controller which will replace the block in the cache. Finally, the **tag** value from the CPU address needs to replace the value in the corresponding tag register and the **dirty** bit must be set low (0).

4. Read/Write from/to cache [miss] and dirty bit = 1
   In this situation, the request made by the CPU is not found in cache, resulting in a cache miss. This requires performing a block replacement in the cache. Again the **dirty** bit for the corresponding CPU address must be analyzed. If the **dirty** bit is high (1) then the recently used (and soon to be replaced) block must be written back into main memory. The data block in BRAM must be sent to the main memory controller with the following address: [**Tag** & **Index** & 00000]. The **Tag**

is the 8-bit value stored in the Tag- register for the corresponding block. While "00000" is the offset value for starting block address (at word # 0).

Next, a new block must be copied into the cache. This requires reading data from the main memory controller by issuing the following address: [**Tag** & **Index** & 00000]. The **Tag** now is the 8-bit value of the Tag-field of AWR requested by the CPU recently. Since the block of this address comes from the main memory controller, the **Write Enable** (WE) signal must be asserted to write into the BRAM. Finally, the **Tag** value from the CPU address needs to replace the value in the corresponding tag register and the **dirty** bit must be set low (0).

4. Target

The target platform / device is Xilinx Spartan-3E FPGA

5. Results

The following parameters have to be determined based on hardware emulation:

N	Cache performance parameter	Time in nS
1	Hit / Miss determination time	
2	Data access time	
3	Block replacement time	
4	Hit time (Case 1 and 2)	
5	Miss penalty for Case 3 (when D-bit = 0)	
6	Miss penalty for Case 4 (when D-bit = 1)	

Milestones

Week	Outline	What is Due?
3	Project #1 description and specs.	Tutorial 1, 2 and 3.
4	VHDL code and compilation	The symbol and block diagram.
5	Testing and VHDL corrections.	VHDL code and correct compilation.
6	Hardware emulation.	
7	Project #1 demonstration.	Project #1 Final report submission
8	Project #2 description and specs.	