Cache Controller

Computer Architecture Course

Sepideh Bayati

Ahmad Anvari

Supervised by

Dr. Hamed Farbeh

20 March 2018

Contents

Background

Most computers are designed to operate with different types of memory organized in a memory hierarchy. In such hierarchies, as the distance from the processor increases, so does the volume of the memory; likewise, as the distance from the processor increases, so too does the access time for each memory. This structure, along with appropriate operating mechanisms, allows processors to efficiently use slower (cheaper) memory effectively; by taking advantage of temporal and spatial locality, this hierarchy can mitigate most of the delay associated with slow, high-volume memories. Closest to the CPU is the Cache Memory. Cache memory is fast but quite small; it is used to store small amounts of data that have been accessed recently and are likely to be accessed again soon in the future. Data is stored here in blocks, each containing a number of words. To keep track of which blocks are currently stored in Cache, and how they relate to the rest of the memory, the Cache Controller stores identifiers for the blocks currently stored in Cache. These include the index, tag, valid and dirty bits, associated with a whole block of data. To access an individual word inside a block of data, a block offset is used as an address into the block itself. Using these identifiers, the Cache Controller can respond to read and write requests issued by the CPU, by reading and writing data to specific blocks, or by fetching or writing out whole blocks to the larger, slower Main Memory. Figure 1 shows a block diagram for a simple memory hierarchy consisting of CPU, Cache (including the Cache controller and the small, fast memory used for data storage), Main Memory Controller and Main Memory proper.

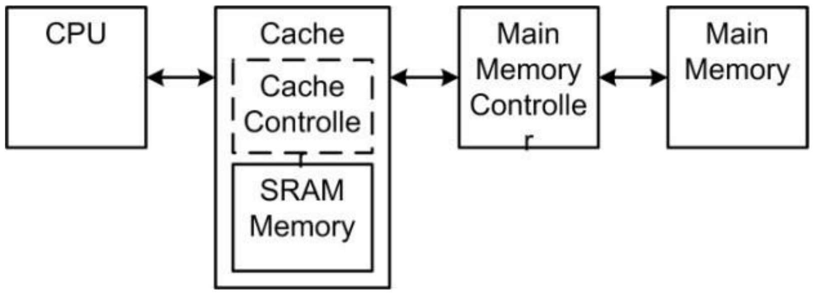

Figure 1: Memory Hierarchy Block Diagram

Application: Simple Cache Controller

General Description

The aim of this project is to design and implement a simplified Cache for a hypothetical computer system. The Cache for this computer system will store a total of 256 bytes, organized into blocks of 32 words each; each word will have 1 byte.   Figure 2 shows the organization of the Cache.

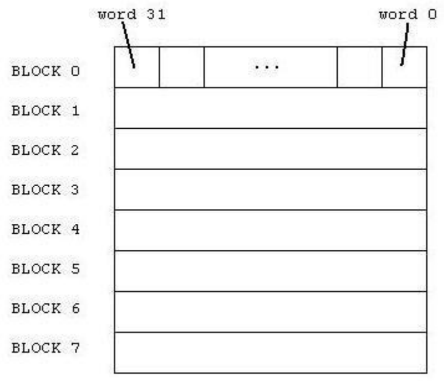

Figure 2: Organization of the Cache (implemented using Block RAM)

As part of its operation, the CPU will issue read and write requests to the memory system. As shown in Figure 1, the CPU only interacts directly with the Cache. The Cache Controller must determine what action needs to be performed based on the current contents of the Cache Memory, as well as the identifiers associated with each cache block.

The complete structure for the target system is shown in Figure 3 below. The system consists of the CPU, an SDRAM controller, and the cache acting as the intermediary between the two. The cache itself consists of a controller and a local SRAM block; the SRAM block is connected as shown. The project aim is to design the controller, SRAM then assemble a Cache using them and connect the cache to Block RAM memory module and CPU module.

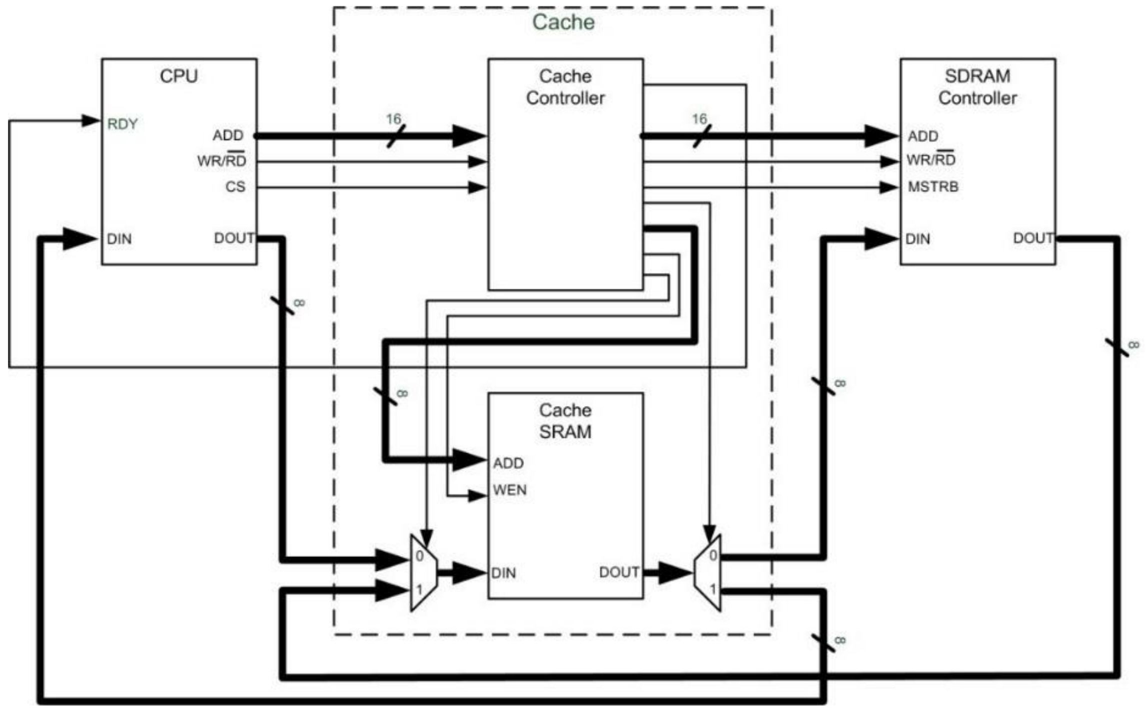

Figure 3: Complete Cache System

Specification

CPU Address Word Organization

The CPU issues 16bit address words. The address word is received by the Cache Controller and stored in the Address Word Register (AWR). The AWR consists of three fields according to the cache organization (see Figure 2): the cache memory can store 8 blocks, each of which contains 32 1-byte words. Therefore, the Index 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 3 illustrates the AWR organization.

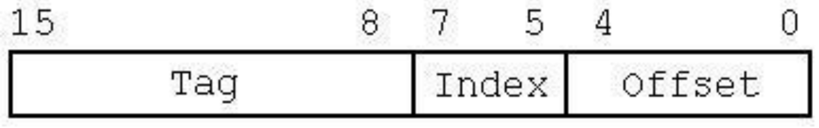

Figure 4: Address Word Register Fields

Cache Controller Behavior

The aim of the project is to design a logic circuit that responds appropriately to read and write requests made by the CPU. The potential actions available to the cache controller are reading or writing to local (cache) memory, fetching whole data blocks from main memory, or writing out blocks to main memory. The controller decides which of these actions to take based on the indicators it stores internally, which describe the status of each block in the cache memory. Specifically, the controller must compared the tag portion of the address with the tags stored for each of the eight blocks, as well as check the status of the dirty and valid bits for the block being targeted.

Behavioral Cases:

1. <u>Write a word to cache [hit]</u>
   The cache receives a write request from the CPU. 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 local SRAM as the write address for the new data. The dirty and valid bits associated with the targeted block must be set to 1, and the data must be written to the local SRAM memory.

2. <u>Read a word from cache [hit]</u>
   The cache receives a read request from the CPU. 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 local SRAM and the read data is routed back to the CPU.

3. Read/Write from/to cache [miss] and dirty bit = 0
   A read or write request is received, and the associated block is not found in cache. This requires performing a block replacement in Cache. First the dirty bit for the corresponding CPU address must be analyzed. If the dirty bit is 0 then the entire CPU address with the offset portion set to "00000" is passed to the SDRAM memory controller as the base address of the block to be read from memory, and the full block (all 32 bytes) must be read from the SDRAM memory controller, and written to the local Cache SRAM. The tag value from the CPU address needs to replace the value in the corresponding tag register, and the valid bit must be set to 1. Finally, the requested read or write operation must be performed following the procedure outlined for the hit cases above.

4. Read/Write from/to cache [miss] and dirty bit = 1
   A read or write request is received, and the associated block is not found in cache (a miss) which leads to a block replacement operation. Again the dirty bit for the corresponding CPU address must be analyzed. If the dirty bit is 1 then the recently used (and soon to be replaced) block must be written back into main memory. The data block in local SRAM must be written to main memory via the SDRAM memory controller. The block must be written to the following base address: [Tag & Index & 00000].

   Next, the new block (based on the address requested by the CPU) must be copied into the cache. This requires reading data from the SDRAM memory controller by issuing the following address: [Tag & Index & 00000]. The Tag is now the Tag field of the AWR, requested by the CPU recently. The full block being read from the SDRAM memory controller must be written to the local cache SRAM. The original tag associated with this block must also be replaced with the new tag, as issued by the CPU. Once this process is complete, the transaction originally requested by the CPU can be completed, following the procedure outlined for the hit cases above.

Interface Specifications

Below are described the details of the three interfaces the cache controller must interact with: the CPU, the SDRAM controller, and the local SRAM (Block RAM). In each case, the type of information provided through the interface, as well as the interface timing is described.

CPU

This section is for your knowledge and understanding of this module, the related VHDL code will be given

The CPU will periodically issue read or write transaction requests. The CPU interface consists of a strobe CS, a read/write indicator WR/RD, a 16-bit address ADD, 8-bit data in and out ports DIN and DOUT, and finally a ready indicator input RDY. Finally, the CPU is assumed to be synchronous with the cache controller, and shares the same clock signal, CLK. All interface ports are shown in Figure 5.

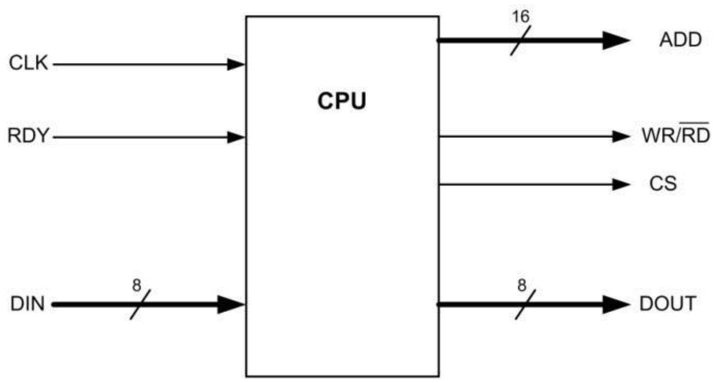

Figure 5: CPU Interface

When the CPU issues a transaction, it first sets the appropriate address on the address bus and sets the read/write indicator to the correct value; if a read is being issued, the WR/RD signal is low (0), whereas if a write is issued, the signal is high (1). Finally, if a write is performed, the appropriate data is also set on the DOUT port. Once all transaction signals are stable, the strobe CS is asserted, and stays asserted for 4 clock cycles. Figure 6 below shows an example of a write request.

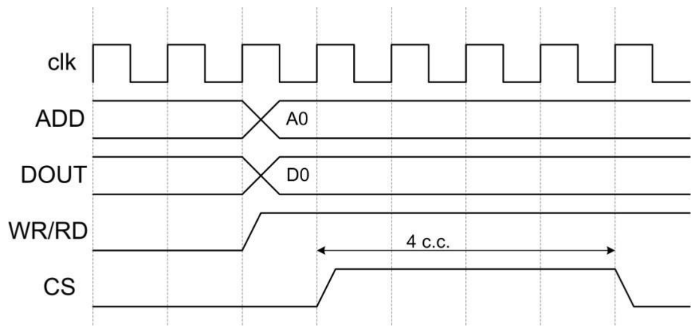

Figure 6: CPU Transaction Example

The ready indicator RDY is used by the Cache Controller to indicate to the outside world when a transaction is complete. When idle, the cache controller should leave the RDY signal asserted (thus indicating it is ready to accept transactions). Once a transaction is received, the signals should be de-asserted, and should be kept low until the requested operation is complete.

SDRAM Controller

This section is for your knowledge and understanding of this module, the related VHDL code will be given

The SDRAM Controller responds to block read and write requests issued by the Cache Controller. Its interface is shown in Figure 7. The interface consists of a 16-bit address ADD, a read/write indicator WR/RD, a strobe MEMSTRB and data in and out ports DIN and DOUT. Finally, the SDRAM Controller is synchronized with the Cache Controller, by sharing the same clock CLK.

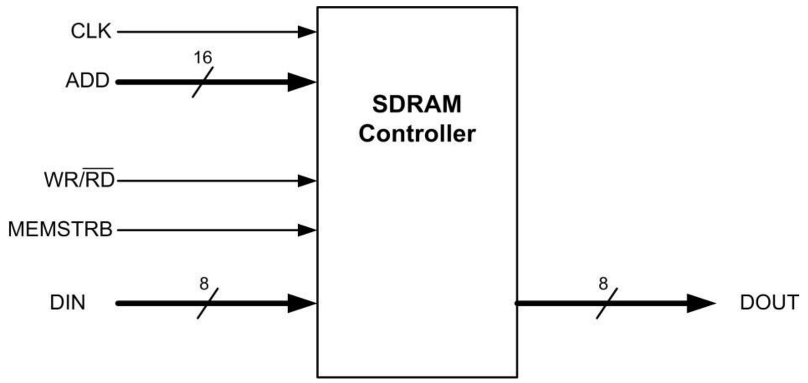

Figure 7: SDRAM Controller Interface

The SDRAM Controller responds to block read and write requests from the Cache Controller. To write a word to main memory, the correct base address is set on the ADD port by the Cache Controller, and the WR/RD signal is asserted (set to 1) to indicate a write operation. Finally, once all other signals are stable (one clock cycle later), the strobe MEMSTRB is asserted for one clock cycle. To write a full block to memory, this process is repeated 32 times, with the address and data changing accordingly. To read a full block from memory, the same process is used, with the exception that WR/RD is held low (0), and no data needs to be placed on the DIN port; rather, the data will appear on the DOUT port. Figures 8 and 9 below show read and write processes for a full block of data.

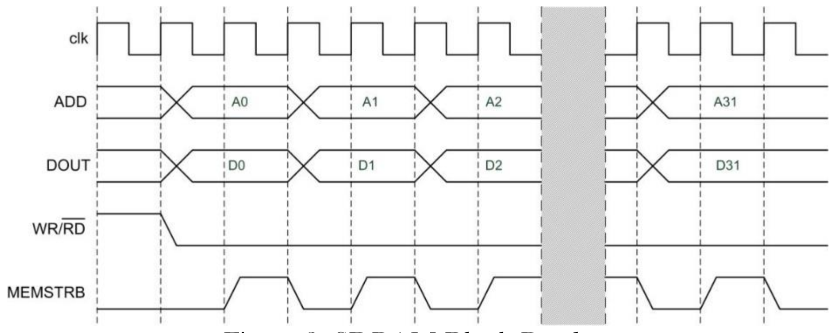

Figure 8: SDRAM Block Read

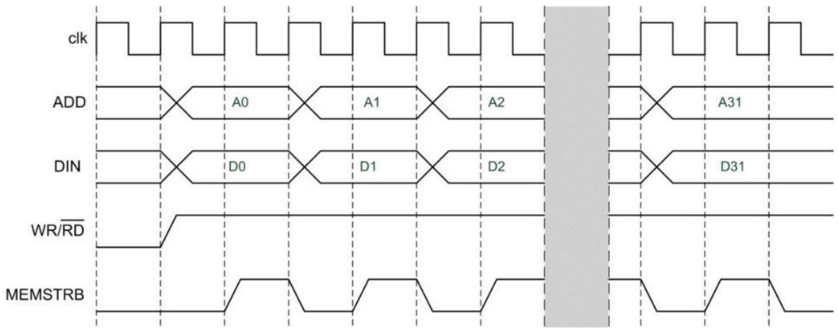

Figure 9: SDRAM Block Write

Local SRAM

To implement the VHDL code of this part, the sample code will be given but the completion of it is all yours

The Block RAM memory used to implement the local memory for the cache has the interface shown in Figure 10. It consists of an 8-bit address ADD, 8-bit data input and output DIN and DOUT, and a write enable signal WEN. Finally, the Block RAM shares the same clock CLK as the Cache Controller.

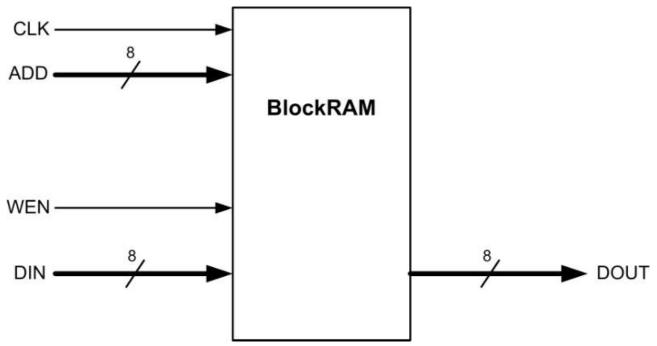

Figure 10: Block RAM Interface

All operations issued to the cache controller are synchronized on the rising edge of the clock. Read operations are performed by setting the appropriate address on the address bus. The addressed data will propagate to the output DOUT after the next rising edge of the clock. Write operations are performed by setting a specific address and data on the appropriate ports, and then asserting the write enable signal WEN. On the next rising edge, the data will be written to the specified address. Read and write operations are shown in Figures 11 and 12 below.

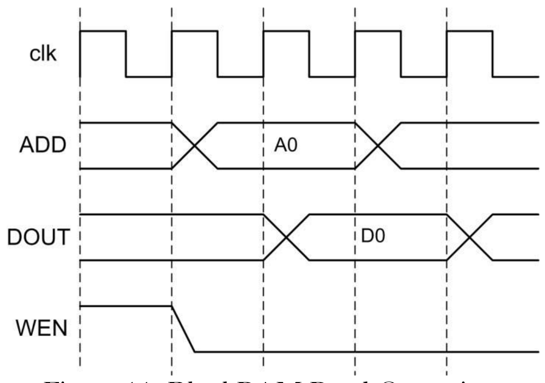

Figure 11: BlockRAM Read Operation

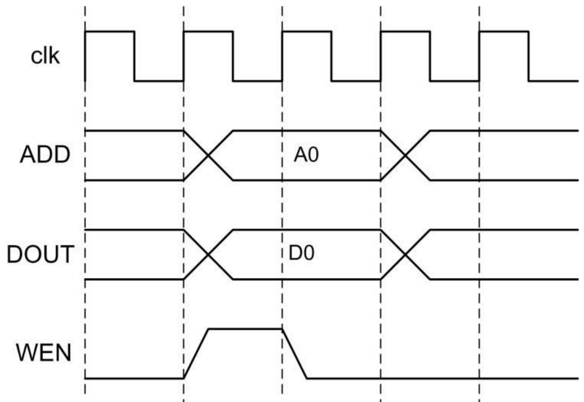

Figure 12: Block RAM Write Operation

Handling Instructions

1. All your project should be implemented by VHDL not Verilog or ...
2. About 10% of your score would be devoted to the report you deliver within your projects code. In this report you would explain the whole job and anything special you have done.
3. In addition to bonus sections mentioned beforehand, implementation of this project on Altera DE2 FPGA boards will result in extra mark as well.
4. You can simulate your code with softwares like: Modelsim, Altera Quartus, GHDL, Xilinx ISE, Active HDL, Xilinx Vivado Design Suite,. . . (note that Modelsim and Quartus has got free versions and GHDL is totally free so it's better to obey Copyright) but the TA's laptop is only equipped with Modelsim and in case of delivery you should be able of running your code in that environment.
5. All questions would be answered by the course's email: computerarchitecture2018@gmail.com .
6. Place all your modules and report into a .zip named as "StudentID_FirstName_LastName.zip" before upload. if any other module is used in your implementation but hasn't been mentioned in this document place it in its proper place next to modules within the same hierarchy.
7. Deadline of the project is
8. In case of delivery, your code will be downloaded by the responsible TA from Moodle, so the only way to convey your code is Moodle and in if you need to reform your code please upload it when possible to be used in the due date.

Cheating alert

a. This project should be done by individuals.
b. All your codes would be processed, both manually and automatically and in case of any similarity by any means, both of individuals involved, would be fined by getting -35 percent of the project's score (note that if pushing your code to Github or any other VCS, exposing your code to a friend or ... results in unexpected similarities of others, you ALL, are responsible!).
c. Any cooperation beyond members of the group is forbidden!
d. The only source you are allowed to copy form, is github.com/AUT-CEIT/Arch2018 repo which has been devoted to this course, copying any other source from the Internet, ... would be considered just like from another classmate.
e. By the way, in case of any similarity with any of the previous students of this course, you blame the same.

Remember that, any HDL (Hardware Design Language) is a piece of art, and could be really enjoyable, if you try your best to understand what's going on instead of just doing it to make it end.

Good Luck