# Designing the Memory Access Interface

ECE/CS 3710 - Computer Design Lab
Lab 3 - Synthesizing and testing the memory (Block-RAM) interface
Due Date for FPGA demo: Thursday, Oct 1
Due Date for a short report and source code upload: Friday, Oct 2

In this lab, we will develop a memory access interface using the on-chip Block-RAM. Once this interface is designed and tested, then in the next lab assignment, we will move on to designing the control (Fetch, Decode, Execute, Write-back) of the machine.

In general, the memory interface for our CPU will look somewhat similar to the one shown in Fig. 1. However, for this lab, we will keep things simple, and build an interface to read data from memory, modify it, write it back, and then read it again, to verify correct operation of the memory. Your infrastructure would probably look like in Fig. 2.

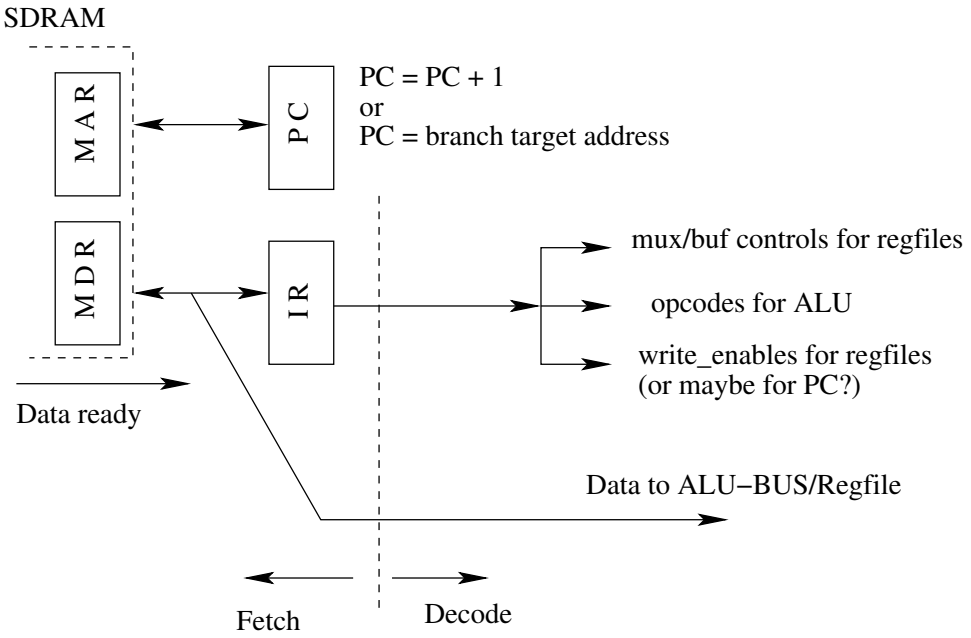


Fig. 1. A generic block diagram of the Memory-CPU Interface for fetch/decode stages

**About the Block-RAM on Cyclone V:** First of all, please go through the Cyclone V device overview (cv_51001_device_overview.pdf), device handbook (cyclone5_handbook.pdf), and go through the **M10K** Block-RAM resources, and the embedded memory user guide (ug_ram_rom.pdf), all of which can be found on the class website on Canvas. You will notice that our FPGA, Cyclone V SE A5, has **397 blocks of RAM**, and each block is a configurable block of **10K-bits** memory cells. Since we are working on 16-bit data words, we can configure each block as 16-bit wide, and 512 words. Actually, internally, the block-RAM may even be configured as 20-bit words, where the four MSBs are unused in our case.

This means that with 397 blocks, we have a total of 203,264 words, each 16-bit wide. This should be enough memory to play with for most applications that you may have in mind for the project. If you need more memory, we can address those issues later.

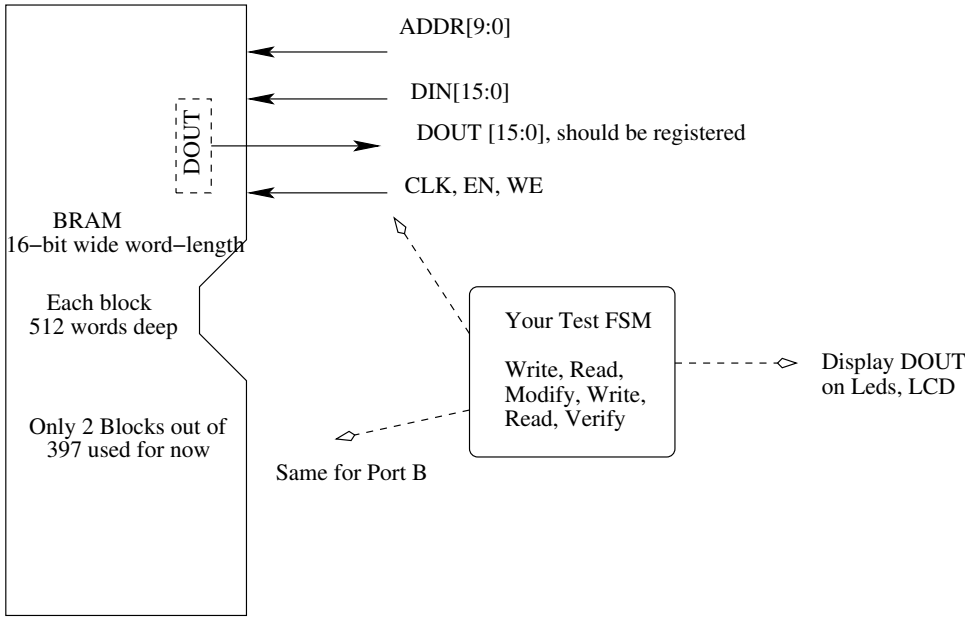


Fig. 2. Block Diagram of the Memory Access Interface for Lab 3.

**The Assignment:** Your overall task for this lab is the following:

- You are asked to design a **synchronous (clocked) true dual-port memory**. Dual ports will be needed if one port is accessed by the CPU and the other by, say, the VGA controller.
- You should be able to use 18-bit addresses to access the entire block-RAM resource on-chip. However, for this lab assignent, there is no need to access all 397 blocks. It is okay to synthesize 2 blocks of RAM for now, which would provide $512 \times 2 = 1024$ words. This corresponds to 10 address lines. Later on, as your project advances, you can decide whether a 16-bit PC suffices, or you do need 18-bits to access the whole RAM. So, for now **synthesize 2 blocks of RAM**.
- Using the `$readmemb() or $readmemh()` Verilog system tasks, initialize the memory with known data at various locations: initialize some known data at addresses 0, 1, 2, and then some data at memory addresses 510, 511, 512, 513 — to see how the data is stored correctly in the two different blocks of RAM.
- Design a FSM wrapper that:
  - Reads data from the above locations that was initialized to known values.
  - Modifies this data. E.g., you could fetch the data and increment it by a constant.
  - Re-writes the modified data in various locations (potentially the same locations) in memory;
  - Re-reads the data from memory and verifies that the correct updated data was read.
  - And finally, displays the updated data from one of those locations on the 7-Segment display.
- This should, of course, be simulated and validated at RTL.
- It is imperative that we test the correct functioning of the memory on-board. So your objective should be to be able to perform the **read-modify-write-read** operation on the FPGA. *This will ensure that we can faithfully read instructions from and write data to memory* at arbitrary locations.
- Please ensure that all operations are synchronized w.r.t. the same clock, e.g. at the *posedge* of clock cycles.
- Once this is achieved, it would imply that our memory access interface is now working correctly, and we will move onto CPU control.
- The due date for a functioning demo — read, display on 7-seg, (over-)write, display the new value on 7-seg

— is Thursday October 1.

- A short report (2-3 pages) can be uploaded on Canvas by midnight Friday, October 2.

Please go through the document that my former TA (Vikas) had prepared where he shows multiple ways to instantiate the Block-RAM (quartus_memory_manual.pdf uploaded on the class website on Canvas). The Quartus tool will give you the template which you can use yourself.

Some things to be careful about. Go through the Block-RAM documents and understand the various write-modes of the Block-RAM (Write first and then Read, or Read First and then Write, etc.). There are certain cases in which the data can get invalidated, particularly when two writes to the same location are accessed simultaneously on two different ports. Perhaps you should experiment with various write-mode configurations of the memory (i.e. write-first, read-first, etc.). This is achieved by using either a blocking or a non-blocking assignment at the output ports of the memory.