Designing the Memory Access Interface

ECE/CS 3710 - Computer Design Lab
Lab 3 - Testing the memory interface
Due Date for FPGA demo: Tue Sept 29
Due Date for a short report: Tue Oct 6

In this lab, we will develop a memory access interface using the on-chip Block-RAM. If you are interested in using the cellular (pseudo-static DRAM), then I first suggest to get the Block-RAM interface operational, and then move on to the cellular 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, write it back, read it again, to verify correct operation of the memory. Your infrastructure would probably look like in Fig. 2.

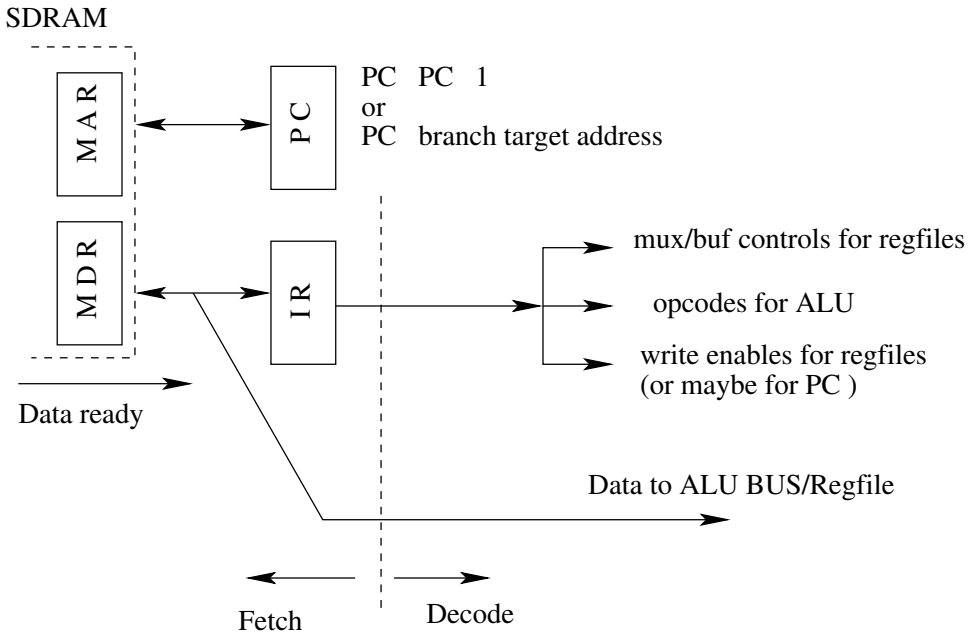

Fig. 1. Block Diagram of the Memory-CPU Interface for fetch/decode stages

About the Block-RAM on Spartan 6: First of all, please go through the Spartan-6 data-sheet (ds160), and the Block-RAM resources users guide (ug383) which can be found on the Xilinx as well as the class website. You will notice that our FPGA, XC6LS16, has **32 blocks of RAM**, and each block is a configurable **18K-bits** memory cells. Since we are working on 16-bit words, we can configure each block as 16-bit wide, and 1K (1024) words. Actually, internally, the block-RAM will be configured as 18-bit words, where the two MSBs are configured as parity and are unused in our case.

This means that with 32 blocks, we will have a total of 32K words, each 16-bit wide, to play with. This should be enough memory to play with. If you need more memory, then you will have to use the Cellular RAM. 32K words can be addressed by 15-bit addresses, so a 16-bit program counter should work for most of you.

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

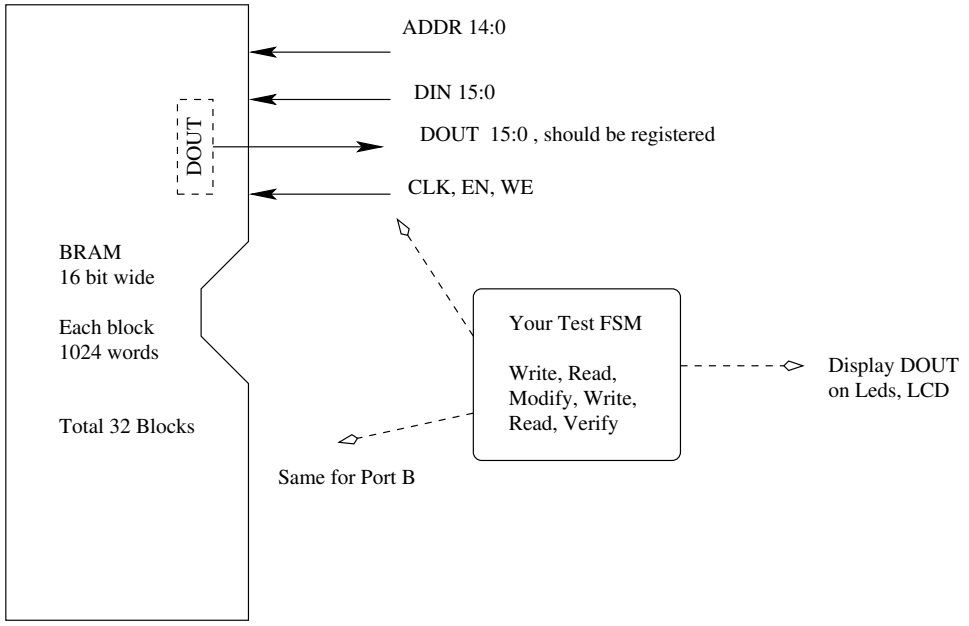

Fig. 2. Block Diagram of the Memory Access Interface for Lab 3.

- You are asked to design a synchronous (clocked) 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 15-bit addresses to access the entire block-RAM resource on-chip.
- Design a FSM wrapper that: i) Writes the data in various locations in memory; ii) re-reads the data from memory; iii) verifies that the correct data was read; and iv) display the fetched data also on the LEDs/7-Seg/LCDs.
- You may use the Fibonacci sequence generator, as used in the previous lab. Try at least two different test-benches for on-board test.
- This should, of course, be simulated and validated at RTL.
- You also need to learn how to initialize the memory during compile time (cf. $readmemh and $readmemb Verilog tasks). This way, your application program can be stored on the Block-RAM
- 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 Tuesday, 29 Sept.
- A short report can be uploaded on Canvas by Tue Oct 6.

How to proceed: Well, there are three ways of using the block-RAM:

1) Use the Core Generator that comes with the Xilinx tools. On Tuesday, in class, every group was able to invoke the core generator. Now use it properly to generate the final design.
2) The other way is to access (copy, cut, paste) the Block-RAM macro from the tool. In the ISE's menu, "Edit $\rightarrow$ Language Templates $\rightarrow$ Verilog $\rightarrow$ Device Macro/Primitive Instance" will lead you to the Verilog source macro provided by Xilinx. You can try to instantiate it.
3) A third easy way is to use the coding example (memory.v) uploaded on the class website and synthesize it. Xilinx will infer the code-style as memory and map it to Block-RAM.

Some things to be careful about. Go through the Block-RAM document provided by Xilinx, and understand the

various write-modes of the Block-RAM (Write first, Read First, No Change). There are certain cases in which the data can get invalidated, particularly when a write to and a read from the same location is accessed simultaneously on two ports. Perhaps you should experiment with various write-mode configurations of the memory (i.e. write-first, read-first, etc.).