EE 371, Lab 2

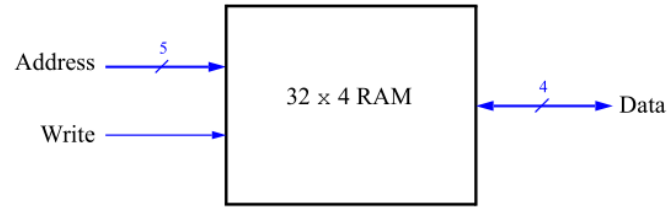
Memory Blocks

Lab Objectives

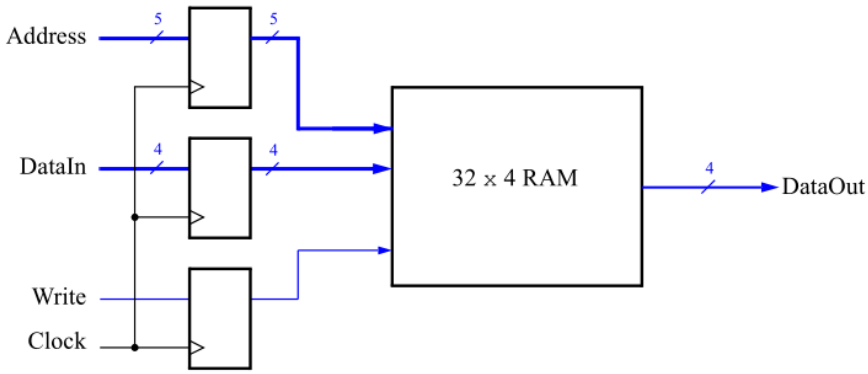
In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology, it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. In this lab we will examine the general issues involved in implementing such memory.

Introduction

A diagram of the Random Access Memory (RAM) module that we will implement is shown in Figure 1a. It contains 32 four-bit words (rows), which are accessed using a five-bit address port, a four-bit data port, and a write control input.



(*a*) RAM organization



(b) RAM implementation

Figure 1. 32x4 RAM module

The FPGA included on the DE1-SoC board provides dedicated memory resources and has M10K blocks, where each M10K block contains 10240 memory bits. The M10k blocks can be configured to implement memories of various sizes. A common term used to specify the size of a memory is its aspect ratio, which gives the depth in words and the width in bits (depth x width). In this lab we will use an aspect ratio that is four bits wide, and we will use only the first 32 words in the memory.

There are two important features of the M10K blocks. First, they include registers that can be used to synchronize all the input and output signals to a clock input. Second, the blocks have separate ports for data being written to the memory and data being read from the memory. Given these requirements, we will implement the modified 32 x 4 RAM module shown in Figure 1b. It includes registers for the address, data input, and write ports, and uses a separate unregistered data output port.

Task 1:

Commonly used logic structures, such as adders, registers, counters and memories, can be implemented in an FPGA chip by using prebuilt modules that are provided in libraries. In this exercise we will use such a module to implement the memory shown in Figure 1b.

1. Create a new Quartus project to implement the memory module.
2. To open the IP Catalog in the Quartus software click on Tools > IP Catalog.
    a. In the IP Catalog window choose the RAM: 1-PORT module, which is found under the Basic Functions > On Chip Memory category.
    b. Select SystemVerilog HDL as the type of output file to create, give the file the name ram32x4.sv, and click OK. If SystemVerilog is not available as an option, create the file in Verilog, and name it ram32x4.v.
    c. In the configuration window, specify a memory size of 32 four-bit words. Select M10K. Also on this screen accept the default setting to use a single clock for the memory's registers, and then advance to the next page.
    d. On this page deselect the setting called 'q' output port under the category *Which ports should be registered?* This setting creates a RAM module that matches the structure in Figure 1b, with registered input ports and unregistered output ports.
    e. Accept defaults for the rest of the settings in the Wizard and click the Finish button to exit from this tool.

    Examine the ram32x4.sv (or ram32x4.v) file which defines the following module:

```
module ram32x4 (address, clock, data, wren, q);
        input   [4:0]  address;
        input     clock;
        input   [3:0]  data;
        input     wren;
        output  [3:0]  q;
```

3. Create a new top-level SystemVerilog file and instantiate the ram32x4 module, using appropriate input and output signals for the memory ports given in Figure 1b. Compile the circuit. Observe in the Compilation Report, under "Total block memory bits," that the Quartus Compiler uses 128 bits in one of the FPGA memory blocks to implement the RAM circuit.
4. Simulate the behavior of your circuit in ModelSim and ensure that you can read and write data in the memory.
    a. You may encounter simulation errors. Try the following:
        i. *Module 'ram_testbench' does not have a timeunit/timeprecision specification in effect, but other modules do.*
        Solution: Add the following line above your testbench module declaration
        `timescale 1 ps / 1 ps
        ii. *Instantiation of 'altsyncram' failed. The design unit was not found.*

Solution: Go to “Start Simulation…” under “Simulate” and under the libraries tab, add “altera_mf_ver” under “Search Libraries First (-Lf).” Then, select your testbench module under the “Design” tab and select “OK”

Task 2:

Now, we want to realize the memory circuit in the FPGA on the DE1-SoC board and use slide switches to load some data into the created memory. We also want to display the contents of the RAM on the 7-segment displays.

1. Make a new Quartus project.
2. Create another SystemVerilog file that instantiates the ram32x4 module and that includes the required input and output pins on your DE1-SoC board.
   a. Use slide switches SW 3−0 to provide input data for the RAM and switches SW 8−4 to specify the address
   b. Use SW 9 as the Write signal and use KEY 0 as the Clock input
   c. Using hexadecimal, show the address value on the 7-segment displays HEX5 − 4, show the data being input to the memory on HEX2, and show the data read out of the memory on HEX0.
3. Test your circuit and make sure that data can be stored into the memory at various locations. Demonstrate the circuit to your TA.

Task 3:

Instead of creating a memory module by using the IP Catalog, we can implement the required memory by specifying its structure in SystemVerilog code. In a SystemVerilog-specified design it is possible to define the memory as a multidimensional array. A 32 x 4 array, which has 32 words with 4 bits per word, can be declared by the statement

```
logic [3:0] memory_array [31:0];
```

In the FPGAs, such an array can be implemented either by using the flip-flops that each logic element contains or, more efficiently, by using the built-in memory blocks.

Perform the following steps:

1. Create a new Quartus project.
2. Write a SystemVerilog file that provides the necessary functionality, including the ability to load the RAM and read its contents as was done in task 2.
3. Assign the pins on the FPGA to connect to the switches and the 7-segment displays.
4. Compile the circuit and download it into the DE1_SoC.
5. Test the functionality of your design by applying some inputs and observing the output. Demonstrate to your TA.

Task 4:

The RAM block in Figure1 has a single port that provides the address for both read and write operations. For this task you will create a different type of memory module, in which there is one port for supplying the address for a read operation, and a separate port that gives the address for a write operation. Perform the following steps.

1. Create a new Quartus project for your circuit. To generate the desired memory module open the IP Catalog and select the *RAM: 2-PORT* module in the *Basic Functions > On Chip Memory* category. Choose "*With one read port and one write port*" in the category called "*How will you be using the dual port ram?*"
    a. Configure the memory size, clocking method, and registered ports the same way as in task 2.
    b. Select *I do not care (The outputs will be undefined)* for *Mixed Port Read-During-Write for Single Input Clock RAM*. This setting specifies that it does not matter whether the memory outputs the new data being written, or the old data previously stored, in the case that the write and read addresses are the same during a write operation.
    c. On the following configuration window, choose the setting *Yes, use this file for the memory content data*, and specify the filename *ram32x4.mif*. This configuration window shows how the memory words can be initialized to specific values. It makes use of a feature that allows the memory module to be loaded with data when the circuit is programmed into the FPGA chip.
        i. An example of a MIF file is provided in Figure 2. You can also learn about the format of a memory initialization file (MIF) by using the Quartus Help. You will need to create a MIF file like the one in Figure 2 to test your circuit.
    d. Finish the Wizard and then examine the generated memory module in the file ram32x4.sv.

```
DEPTH = 32;
WIDTH = 4;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN
0 : 0000;
1 : 0001;
2 : 0010;
3 : 0011;
... (some lines not shown)
1E : 1110;
1F : 1111;
END;
```

Figure 2: An example memory initialization file (MIF).

2. Write a SystemVerilog file that instantiates your dual-port memory.
    a. To see the RAM contents, add to your design a capability to display the content of each four-bit word (in hexadecimal format) on the 7-segment display HEX0
    b. Use a counter as a read address and scroll through the memory locations by displaying each word for about one second. As each word is being displayed, show its address (in hex format) on the 7-segment displays HEX3−2
    c. Use the 50 MHz clock, CLOCK_50, and use KEY 0 as a reset input
    d. For the write address and corresponding data use switches SW 8−4 and SW 3−0
    e. Show the write address on HEX5−4 and show the write data on HEX1.
    f. Make sure that you properly synchronize the slide switch inputs to the 50 MHz clock signal.
3. Test your circuit and verify that the initial contents of the memory match your ram32x4.mif file. Make sure that you can independently write data to any address by using the slide switches.

4. Use the SignalTap II functionality of Quartus to verify the contents of your RAM module. Find the Intel SignalTap II tutorial on canvas (SignalTap.pdf) to get started. For your RAM module, you'll probably want to probe the read address and data output port.
5. Demonstrate to your TA.

Lab Demonstration and Submission Requirements

- Submit a lab report that includes the procedures and results obtained in the lab.
- Include any hurdles or challenges (if any) that you faced in finishing this lab and how you overcome them.
- Submit the SystemVerilog code for all tasks and include screenshots for the ModelSim waveforms of all modules.
- Submit the Flow Summary (produced during compilation) of compiling your system.
- In your report, include the number of hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).
- Submit your report and programs to Canvas. No hard copies.