EE/CSE 371:
Design of Digital Circuits and Systems

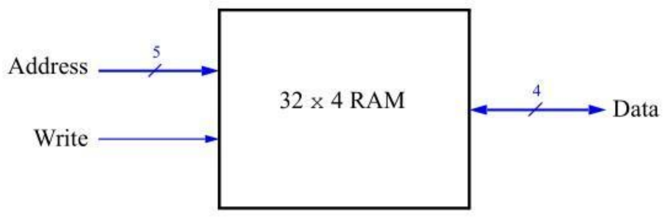
Lab2: Memory Blocks

Lab Objectives

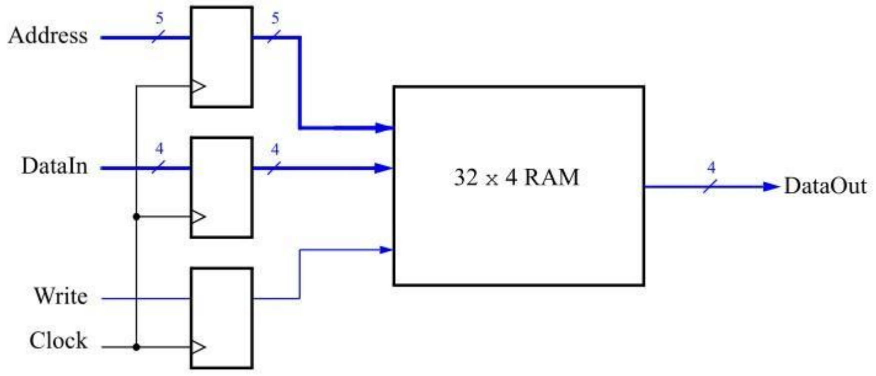
In computer systems, it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology, it is possible to provide some amount of memory by using the memory resources that exist within the FPGA. In this lab, we will examine the general issues involved in implementing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

We can implement a memory unit such as a RAM unit on an FPGA 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.

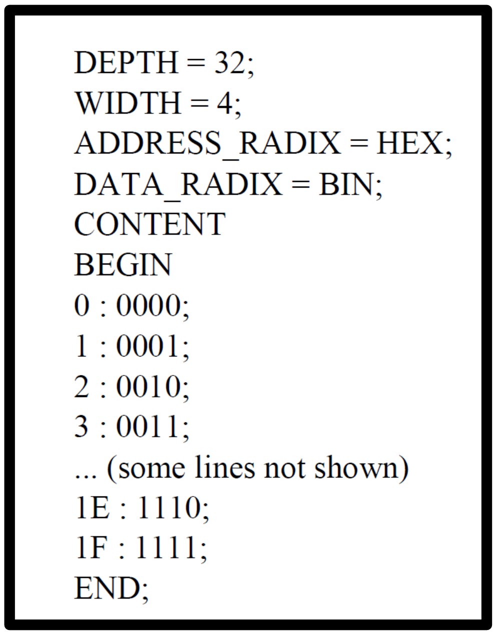
Perform the following steps:

1. Create a new Quartus project.
2. Write a SystemVerilog file that provides the expected functionality, including the ability to load the RAM and read its contents, as specified below:
   a Use 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. Create testbenches for all your modules.
4. Compile the circuit and upload it onto the LabsLand DE1_SoC board.
5. Test the functionality of your design by applying some inputs and observing the output. Record a video to demonstrate your design.

Task 2:

The RAM block in Figure 1 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. Additionally, you will use the IP Catalog to achieve this memory module. Perform the following steps.

1. Create a new Quartus project for this task on your local machine.
2. To open the IP Catalog in the Quartus software click on Tools > IP Catalog.
    a. In the IP Catalog window, under "Library" click "Basic Functions" -> "On Chip Memory" then double click the RAM: 2-PORT module.
    b. Select Verilog as the type of output file to create, give the file the name ram32x4.v, and click OK. After you click Ok, a configuration wizard will open.
    c. Choose "*With one read port and one write port*" in the category called "*How will you be using the dual port ram?*" In the same menu, select "*As a number of words*" in the category "*How do you want to specify the memory size?*" Then hit next.
    d. In the configuration window, specify a memory size of 32 four-bit words. To do this, select the value 4 for the category "*How wide should the 'data_a' input bus be?*" Make sure to then select 32 in the drop-down menu for "*How many 4-bit words of memory?*"
    e. Select M10K in the category "*What should the memory block type be?*" Now hit next.
    f. Accept the default setting to use a single clock for the memory's registers, and then advance to the next page.
    g. On this page deselect the setting called "Read output port(s)" 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. Again, click next.
    h. 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. Click next.
    i. 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.
        - An example of a MIF file is provided in Figure 2. You will need to create a MIF file like the one in Figure 2 to test your circuit.
    j. Finish the Wizard and then examine the generated memory module in the file ram32x4.v



```
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).

- To create a MIF, do the following:
  - i. Choose File ->New-> Memory Initialization file
  - ii. Specify the number of words as 32 and word size as 4
  - iii. Under the View Tab, the number of cells per row can be adjusted in addition to the address and memory radix.
  - iv. Manually fill the grid with the values you want to place in each memory address, preferably something with a pseudo-random nature for the sake of demonstration.
  - v. Save the .mif file as ram32x4.mif. If you open the file in notepad, it will look similar to Figure 2.

3. Write a SystemVerilog module that instantiates your dual-port memory. You are expected to create a testbench for all your modules and simulate them before uploading your design onto LabsLand.
   - 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. Lastly, use KEY 3 as your wr_en.
   - 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 and your wr_en KEY to the 50 MHz clock signal.
4. Upload your ram32x4.v, ram32x4.mif, and your own modules for this task to LabsLand. 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.
5. Record a video for the demonstration.

A note on simulating modules from the IP catalog (on ModelSim):

You may encounter simulation errors. Here are some errors you may come across and ways to get around them:

1. *"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
   ```

2. *"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, go to the "Design" tab, select your testbench module and click "OK"

Task 3:

A FIFO in digital design is a memory mechanism that is comparable to the object "queue" in a computer-science setting. A FIFO uses a memory module to store data when written to and ejects data when read from. It also uses a control module to organize the process. We can abstractly realize a FIFO in hardware by keeping track of two memory addresses (pointers, if you will): a "read address" and a "write address." The read address "points" to the memory location that holds the "oldest" or "least recently stored" data. Whenever a "read" input is asserted, the data stored in this address is output from the FIFO. On the other hand, the write address points to a memory location that will be written to next. Whenever a "write" input is asserted, the data from the input data bus is stored in this address. The relationship between the read and write addresses is displayed below in Figure 3a. A FIFO also always has a full and an empty signal that signals the state of the FIFO (whether it is full, empty, or neither). A block diagram of a FIFO can be found in Figure 3b. The inputs and outputs associated with a FIFO are listed in Table 1 below (note that this is for a FIFO of width "n"):

Input Logic	Number of Bits	Output Logic	Number of Bits
read	1	**full**	1
write	1	**empty**	1
clk	1	**r_data**	n
reset	1		
w_data	n		

Table 1: Input/output logic of a generic FIFO of width “n”

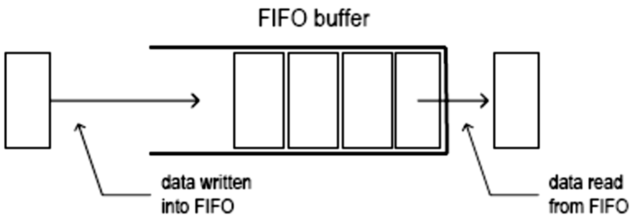


Figure 3a: FIFO Read/Write Visualization

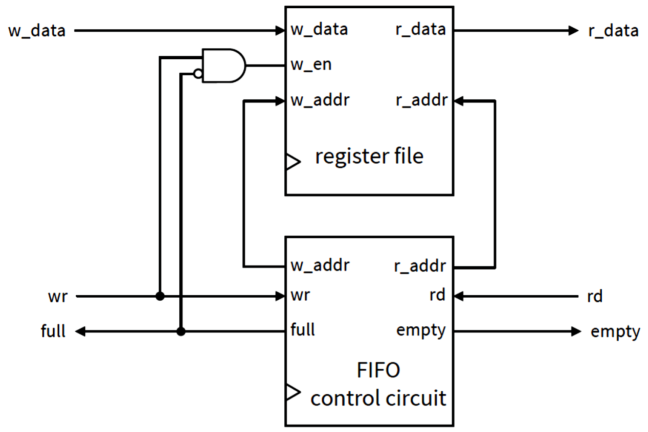


Figure 3b: FIFO Block Diagram

Your job is to design a FIFO from the ground up. We have given you a skeleton project for this task that should get you started. In it, you will find two modules: “FIFO_Control” and “FIFO.” These should set you in the right direction as to how to design your FIFO. When you are designing the FIFO_Control module, you should consider using an FSM (you may find the sample code for FSM under samples on LabsLand). You will also need to add a dual-port RAM (one address for reading, the other address for writing) to the main FIFO module. You may either use the IP catalog or your own dual-port RAM design. If you choose to design your own, your dual-port RAM **must** use memory blocks of the FPGA (as was the case in Task 1).   DO NOT INITIALIZE THE RAM OF YOUR CHOOSING. Your dual-port RAM must have a size of 16x8.   Other than the restrictions listed above, we are giving you some leeway as to the limitations of your FIFO. Note, no matter how you design your FIFO, a “read” assertion must always spit out the least recent data, and a “write” assertion must always store input data onto the RAM. Here are some questions that you need to consider in designing your FIFO:

- How can I tell when my FIFO is full? What do I do when my FIFO is full?
- How can I tell when my FIFO is empty? What do I do when my FIFO is empty?
- How do I write to my FIFO? What address in my dual-port RAM do I write to?
- How do I read from my FIFO? What address in my dual-port RAM do I read from?
- What happens once I read something from my FIFO? What happens once I write something to my FIFO?

To demonstrate the validity of your design, you will need to do the following. As always, make sure you simulate your design!

- Show, in hexadecimal, the value of the data input of your FIFO on HEX5-4. SW7-0 will represent the data input.
- Show the current data output of your FIFO on HEX1-0 (again in hexadecimal).
- Use LEDR9 to show the current value of the output "full."
- Use LEDR8 to show the current value of the output "empty."
- Use CLOCK_50 as the input clk to your FIFO.

Lab Demonstration and Submission Requirements

- Record a demo video for each of the tasks in this lab. You will need to demonstrate the soundness of your design by executing the design on the FPGA.
- Write a Lab Report, as framed by the Lab Report Outline document on Canvas. Comment your code. Follow commenting guidelines as discussed in the Commenting Code document on Canvas. Submit your lab report as a pdf file and all of the code files you created in this lab, including .v, .sv, and .mif, on Canvas.