Laboratory Exercise 8

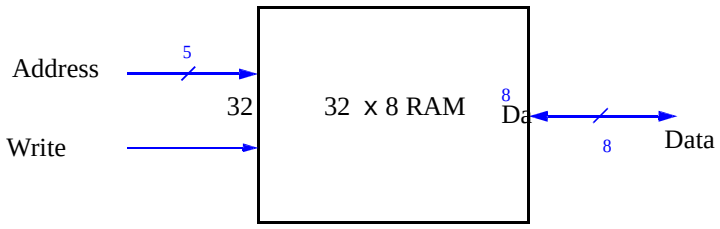
(Partial)

Memory Blocks

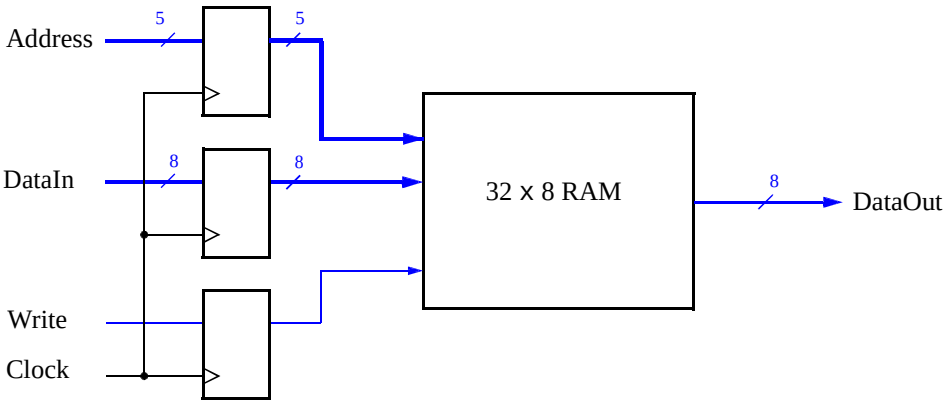
In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. If additional memory is needed, it has to be implemented by connecting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

A diagram of the random access memory (RAM) module that we will implement is shown in Figure 1*a*. It contains 32 eight-bit words (rows), which are accessed using a five-bit *address* port, an eight-bit *data* port, and a *write* control input. We will consider two different ways of implementing this memory: using dedicated memory blocks in an FPGA device, and using a separate memory chip.

The Cyclone II 2C35 FPGA that is included on the DE2 board provides dedicated memory resources called *M4K blocks*. Each M4K block contains 4096 memory bits, which 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). Some aspect ratios supported by the M4K block are 4K x 1, 2K x 2, 1K x 4, and 512 x 8. We will utilize the 512 x 8 mode in this exercise, using only the first 32 words in the memory. We should also mention that many other modes of operation are supported in an M4K block, but we will not discuss them here.



(*a*) RAM organization



(b) RAM implementation

Figure 1. A 32 x 8 RAM module.

There are two important features of the M4K block that have to be mentioned. First, it includes registers that can be used to synchronize all of the input and output signals to a clock input. The registers on the input ports must always be used, and the registers on the output ports are optional. Second, the M4K block has 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 8 RAM module shown in Figure 1*b*. It includes registers for the *address*, *data input*, and *write* ports, and uses a separate unregistered *data output* port.

Part I

Commonly used logic structures, such as adders, registers, counters and memories, can be implemented in an FPGA chip by using LPM modules from the Quartus II Library of Parameterized Modules. Altera recommends that a RAM module be implemented by using the *RAM* LPMs. In this exercise you are to use one of these LPMs to implement the memory module in Figure 1*b*.

1. Create a new Quartus II project to implement the memory module. Select as the target chip the Cyclone II EP2C35F672C6, which is the FPGA chip on the Altera DE2 board.

2. You can learn how the MegaWizard Plug-in Manager is used to generate a desired LPM module by reading the tutorial *Using Library Modules in Verilog Designs*. This tutorial is provided in the University Program section of Altera's web site. In the first screen of the MegaWizard Plug-in Manager choose the *RAM: 1-PORT* LPM, which is found under the Memory Compiler category. As indicated in Figure 2, select Verilog HDL as the type of output file to create, and give the file the name *ramlpm.v*. On the next page of the Wizard specify a memory size of 32 eight-bit words, and select M4K as the type of RAM block. Accept the default settings to use a single clock for the RAM's registers, and then advance to the page shown in Figure 3. 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 1*b*, with registered input ports and unregistered output ports. Accept defaults for the rest of the settings in the Wizard.

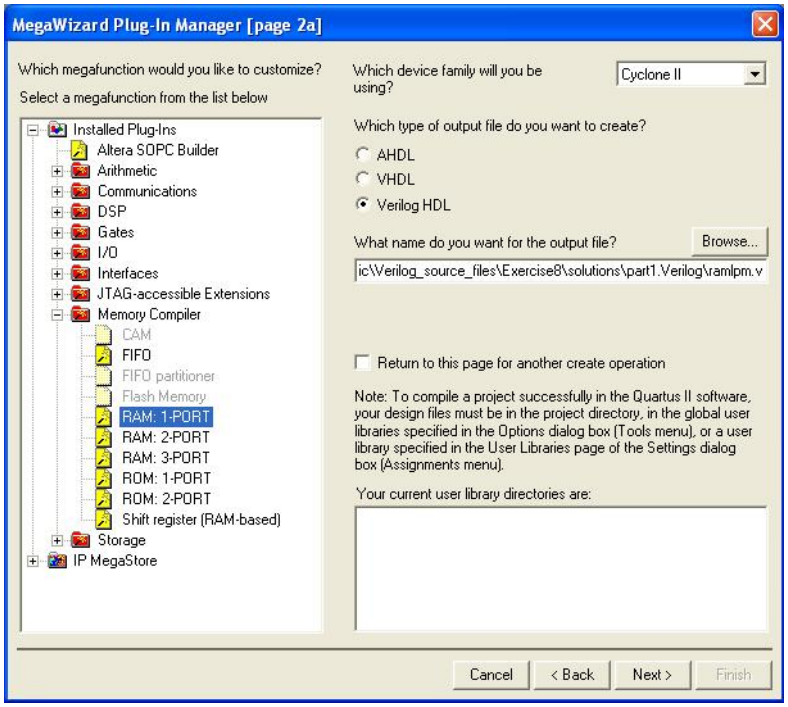


Figure 2. Choosing the *RAM: 1-PORT* LPM.

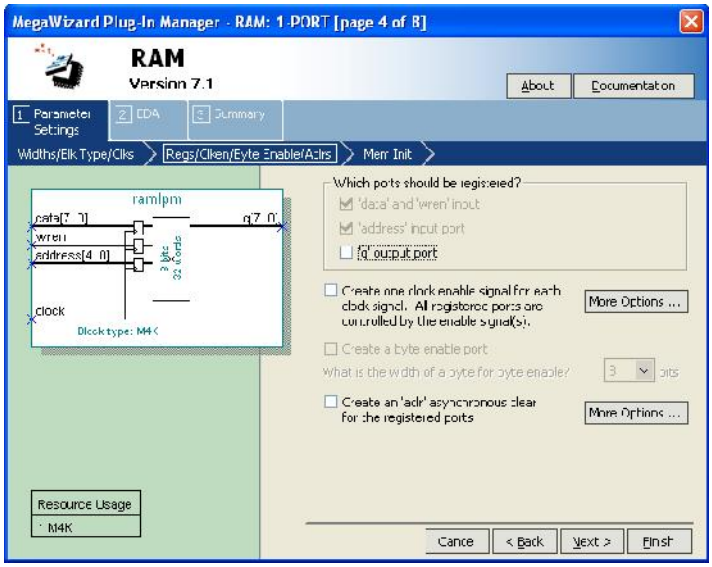


Figure 3. Configuring input and output ports on the *RAM: 1-PORT* LPM.

Now, we want to realize the memory circuit in the FPGA on the DE2 board, and use toggle switches to load some data into the created memory. We also want to display the contents of the RAM on the 7-segment displays.

3. Create another Verilog file that instantiates the *ramlpm* module and that includes the required input and output pins on the DE2 board as shown in the tables below

Inputs	**Signal**
SW[7:0]	Input Data
SW[15:11]	Address
SW[17]	Write enable signal
KEY[0]	Clock

Display	**Signal**
LEDG[0]	SW[17] (write enable signal)
HEX7, HEX6	Address
HEX5, HEX4	Data in
HEX1, HEX0	Data out
LEDR[17:0]	SW[17:0]

4. Test your circuit and make sure that all 32 locations can be loaded properly.

Part II

The SRAM block in Figure 1 has a single port that provides the address for both read and write operations. For this part 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 II project for your circuit. To generate the desired memory module open the MegaWiz- ard Plug-in Manager and select the *RAM: 2-PORT* LPM in the Memory Compiler category. On Page 3 of the Wizard choose the setting With one read port and one write port in the category called How will you be using the dual port ram? Advance through Pages 4 to 7 and make the same choices as in Part I. On Page 8 choose the setting I don't care in the category 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.

   Page 10 of the Wizard is displayed in Figure 5. It makes use of a feature that allows the memory module to be loaded with initial data when the circuit is programmed into the FPGA chip. As shown in the figure, choose the setting Yes, use this file for the memory content data, and specify the filename *ramlpm.mif*. To learn about the format of a *memory initialization file* (MIF), see the Quartus II Help. You will need to create this file and specify some data values to be stored in the memory. Finish the Wizard and then examine the generated memory module in the file *ramlpm.v*.

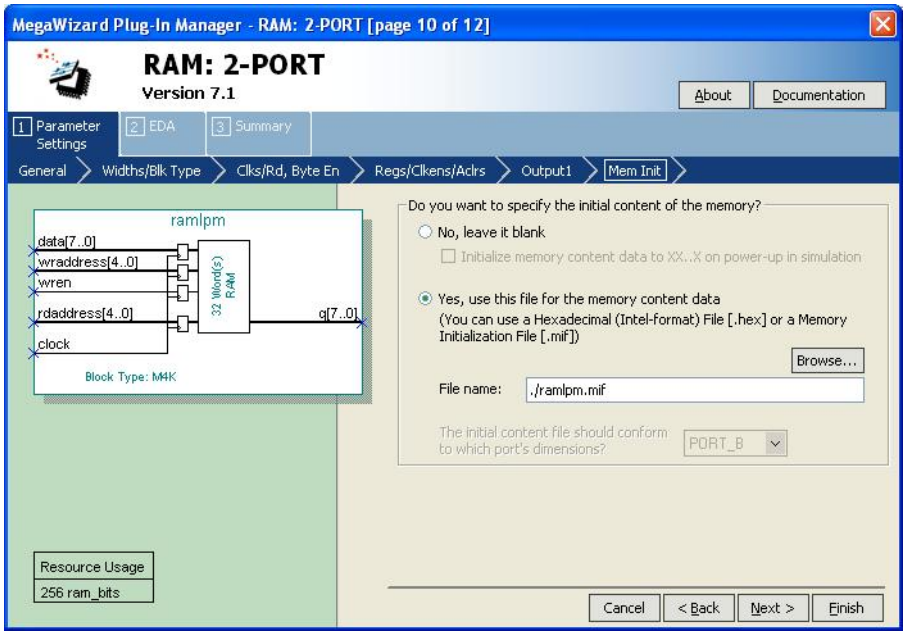


Figure 5. Specifying a memory initialization file (MIF).

2. Write a Verilog file that instantiates your dual-port memory. Use the I/O shown in the tables below.

Inputs	**Signal**
SW[7:0]	Input Data
SW[15:11]	Write Address
SW[17]	Write enable signal
~KEY[0]	Reset
CLOCK_50	Clock

Display	**Signal**
LEDG[0]	SW[17] (write enable signal)
HEX7, HEX6	Write Address
HEX5, HEX4	Data in
HEX3, HEX2	Read Address
HEX1, HEX0	Data out
LEDR[17:0]	SW[17:0]

You generate the read address using a counter. Make sure that you properly synchronize the toggle switch inputs to the 50 MHz clock signal.

3. Test your circuit and verify that the initial contents of the memory match your *ramlpm.mif* file. Make sure that you can independently write data to any address by using the toggle switches.

Part III

The dual-port memory created in Part II allows simultaneous read and write operations to occur, because it has two address ports. In this part of the exercise you should create a similar capability, but using a single-port RAM. Since there will be only one address port you will need to use multiplexing to select either a read or write address at any specific time. Perform the following steps.

1. Create a new Quartus II project for your circuit, and use the MegaWizard Plug-in Manager to again create a *RAM: 1-PORT* LPM. For Pages 3 to 5 of the Wizard use the same settings as in Part I. On Page 6, shown in Figure 6, specify the *ramlpm.mif* file as you did in Part II, but also make the setting Allow In-System Memory Content Editor to capture and update content independently of the system clock. This option allows you to use a feature of the Quartus II CAD system called the In-System Memory Content Editor to view and manipulate the contents of the created RAM module. When using this tool you can optionally specify a four-character 'Instance ID' that serves as a name for the memory; in Figure 7 we gave the RAM module the name 32x8. Complete the final steps in the Wizard.

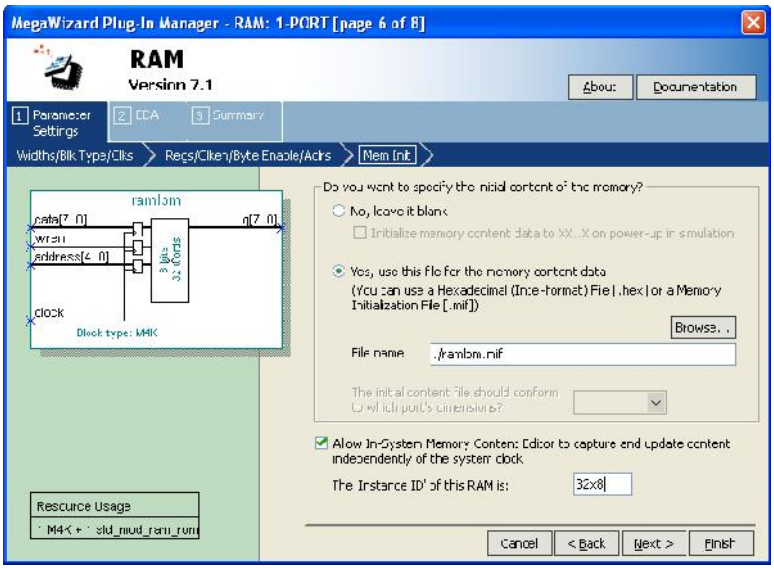


Figure 6. Configuring *RAM: 1-PORT* for use with the In-System Memory Content Editor.

2. Write a Verilog file that instantiates your memory module. Include in your design the ability to scroll through the memory locations as in Part II. Use the same switches, LEDs, and 7-segment displays as you did previously.

Inputs	**Signal**
SW[7:0]	Input Data
SW[15:11]	Write Address
SW[17]	Write enable signal
~KEY[0]	Reset
CLOCK_50	Clock

Display	**Signal**
LEDG[0]	SW[17] (write enable signal)
HEX7, HEX6	Write Address
HEX5, HEX4	Data in
HEX3, HEX2	Read Address
HEX1, HEX0	Data out
LEDR[17:0]	SW[17:0]

3. Before you can use the In-System Memory Content Editor tool, one additional setting has to be made. In the Quartus II software select Assignments > Settings to open the window in Figure 7, and then open the item called Default Parameters under Analysis and Synthesis Settings. As shown in the figure, type the parameter name CYCLONEII SAFE WRITE and assign the value RESTRUCTURE. This parameter allows the Quartus II synthesis tools to modify the single-port RAM as needed to allow reading and writing of the memory by the In-System Memory Content Editor tool. Click OK to exit from the Settings window.

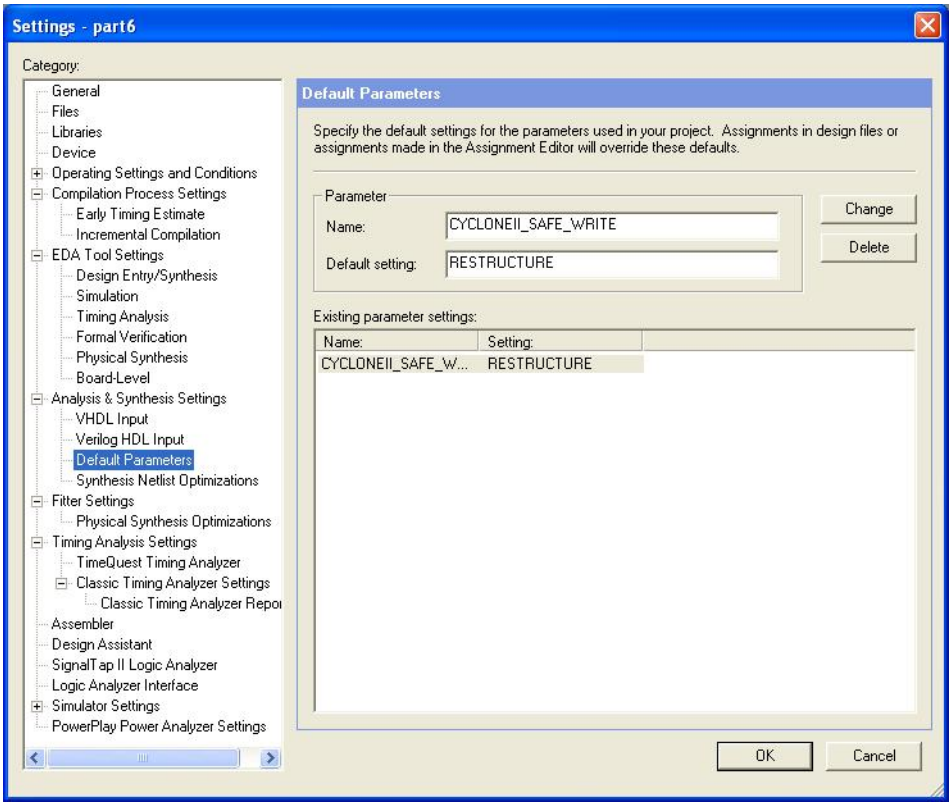


Figure 7. Setting the *CYCLONEII SAFE WRITE* parameter.

4. Compile your code and download the circuit onto the DE2 board. Test the circuit's operation and ensure that read and write operations work properly. Describe any differences you observe from the behavior of the circuit in Part V.

5. Select Tools > In-System Memory Content Editor, which opens the window in Figure 8. To specify the connection to your DE2 board click on the Setup button on the right side of the screen. In the window in Figure 9 select the USB-Blaster hardware, and then close the Hardware Setup dialog.

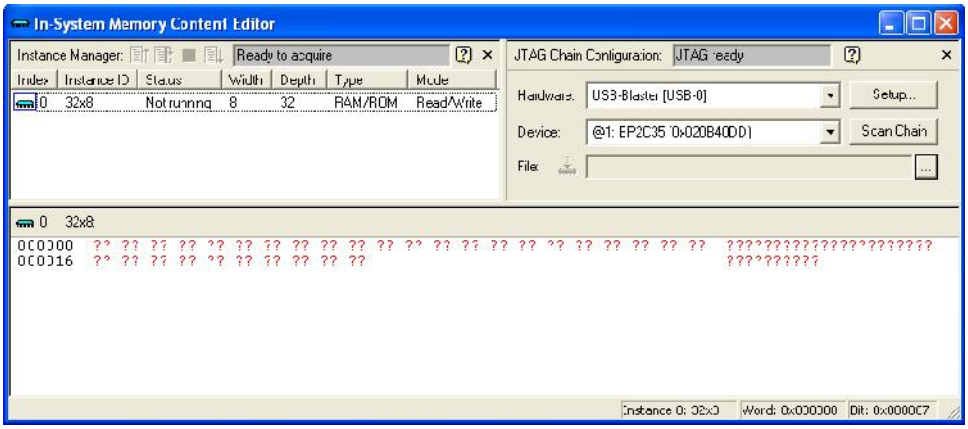


Figure 8. The In-System Memory Content Editor window.

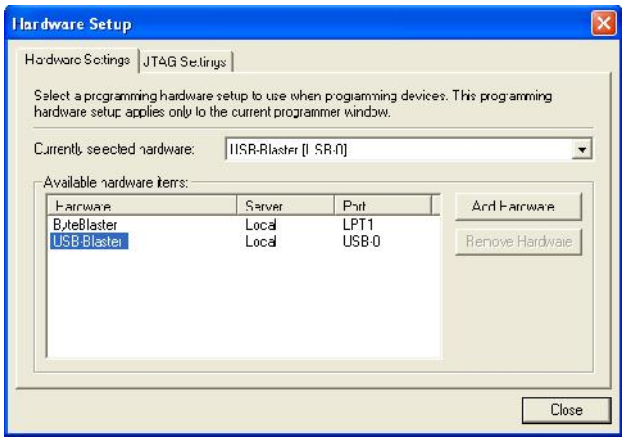


Figure 9. The Hardware Setup window.

Instructions for using the In-System Memory Content Editor tool can be found in the Quartus II Help. A simple operation is to right-click on the 32x8 memory module, as indicated in Figure 10, and select Read Data from In-System Memory. This action causes the contents of the memory to be displayed in the bottom part of the window. You can then edit any of the displayed values by typing over them. To actually write the new value to the RAM, right click again on the 32x8 memory module and select Write All Modified Words to In-System Memory.

Experiment by changing some memory values and observing that the data is properly displayed both on the 7-segment displays on the DE2 board and in the In-System Memory Content Editor window.

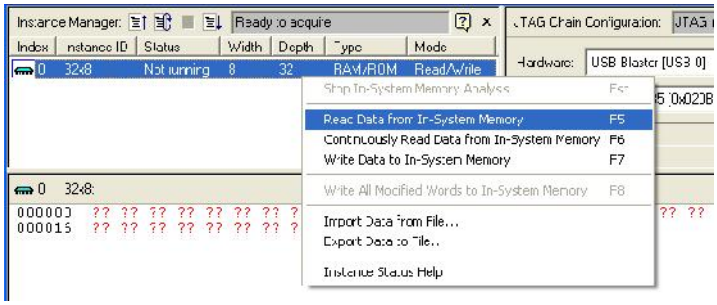


Figure 10. Using the In-System Memory Content Editor tool.