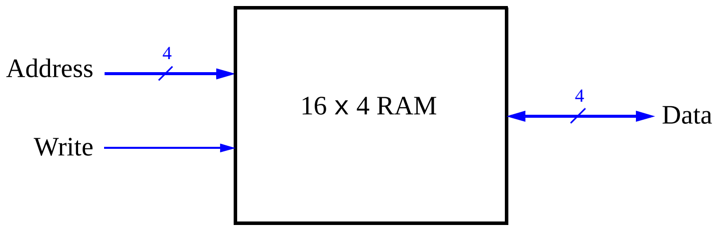
Laboratory Exercise 8

Memory Blocks

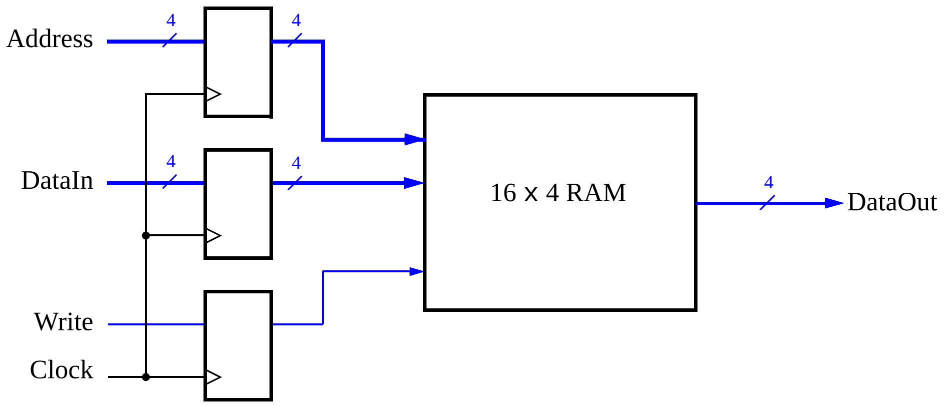
In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. If additional memory is needed, it has to be implemented by connecting 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 16 four-bit words (rows), which are accessed using a four-bit *address* port, an four-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 III 3C16 FPGA that is included on the DE0 board provides dedicated memory resources called *M9K blocks*. Each M9K block contains 9216 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 M9K block are 8K x 1, 4K x 1, 2K x 4, 1K x 9, 512 x 18, and 256 x 36. We will utilize the 2K x 4 mode in this exercise, using only the first 16 words in the memory. We should also mention that many other modes of operation are supported in an M9K block, but we will not discuss them here.



(*a*) RAM organization



(b) RAM implementation

Figure 1: A 16 x 4 RAM module.

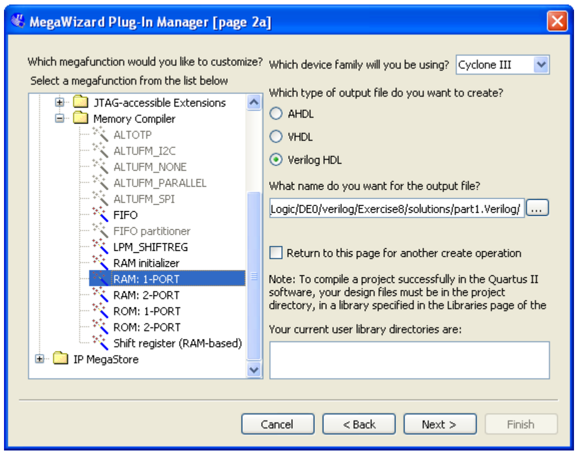
There are two important features of the M9K block that have to be mentioned. First, it includes registers that can be used to synchronize all of the input and output signals to a clock input. The registers on the input ports must always be used, and the registers on the output ports are optional. Second, the M9K 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 16 x 4 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 III EP3C16F484C6, which is the FPGA chip on the Altera DE0 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 16 four-bit words, and select M9K as the type of RAM block. Advance to the subsequent page and accept the default settings to use a single clock for the RAM's registers, and then advance again 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, and then instantiate in your top-level Verilog file the module generated in *ramlpm.v*. Include appropriate input and output signals in your Verilog code for the memory ports given in Figure 1*b*.


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?
Select a megafunction from the list below

Which device family will you be using? Cyclone III

Which type of output file do you want to create?
AHDL
VHDL
[✓ Verilog HDL]

What name do you want for the output file?
Logic/DE0/verilog/Exercise8/solutions/part1.Verilog/

Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of the

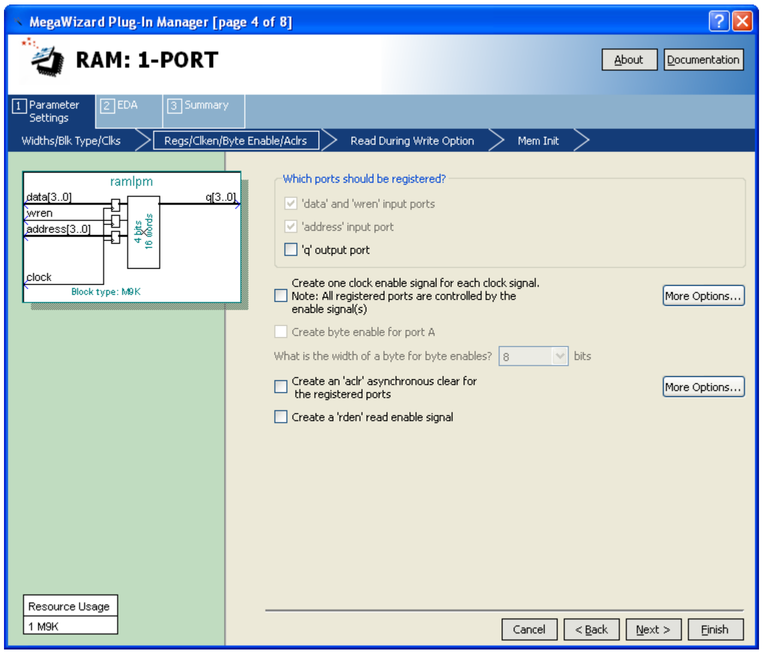
Your current user library directories are:

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

3. Compile the circuit. Observe in the Compilation Report that the Quartus II Compiler uses 64 bits in one of the M9K memory blocks to implement the RAM circuit.

4. Simulate the behavior of your circuit and ensure that you can read and write data in the memory.

Part II



Which ports should be registered?

[✓] 'data' and 'wren' input ports

[✓] 'address' input port

[ ] 'q' output port

[ ] Create one clock enable signal for each clock signal. Note: All registered ports are controlled by the enable signal(s)

[ ] Create byte enable for port A

What is the width of a byte for byte enables? 8 bits

[ ] Create an 'aclr' asynchronous clear for the registered ports

[ ] Create a 'rden' read enable signal

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 DE0 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.

1. Make a new Quartus II project which will be used to implement the desired circuit on the DE0 board.

2. Create another Verilog file that instantiates the *ramlpm* module and that includes the required input and output pins on the DE0 board. Use toggle switches SW_{3-0} to input a byte of data into the RAM location identified by a 4-bit address specified with toggle switches SW_{7-4}. Use SW_9 as the *Write* signal and use KEY_0 as the *Clock* input. Display the value of the *Write* signal on $LEDG_0$. Show the address value on the 7-segment display *HEX3*, show the data being input to the memory on *HEX1*, and show the data read out of the memory on *HEX0*.

3. Test your circuit and make sure that all 16 locations can be loaded properly.

Part III

Instead of directly instantiating the LPM module, we can implement the required memory by specifying its structure in the Verilog code. In a Verilog-specified design it is possible to define the memory as a multidimensional array. A 16 x 4 array, which has 16 words with 4 bits per word, can be declared by the statement

```
reg [3:0]  memory_array [15:0];
```

In the Cyclone III FPGA, such an array can be implemented either by using the flip-flops that each logic element contains or, more efficiently, by using the M9K blocks. There are two ways of ensuring that the M9K blocks will be used. One is to use an LPM module from the Library of Parameterized Modules, as we saw in Part I. The other is to define the memory requirement by using a suitable style of Verilog code from which the Quartus II compiler can infer that a memory block should be used. Quartus II Help shows how this may be done with examples of Verilog code (search in the Help for "Inferred memory").

Perform the following steps:

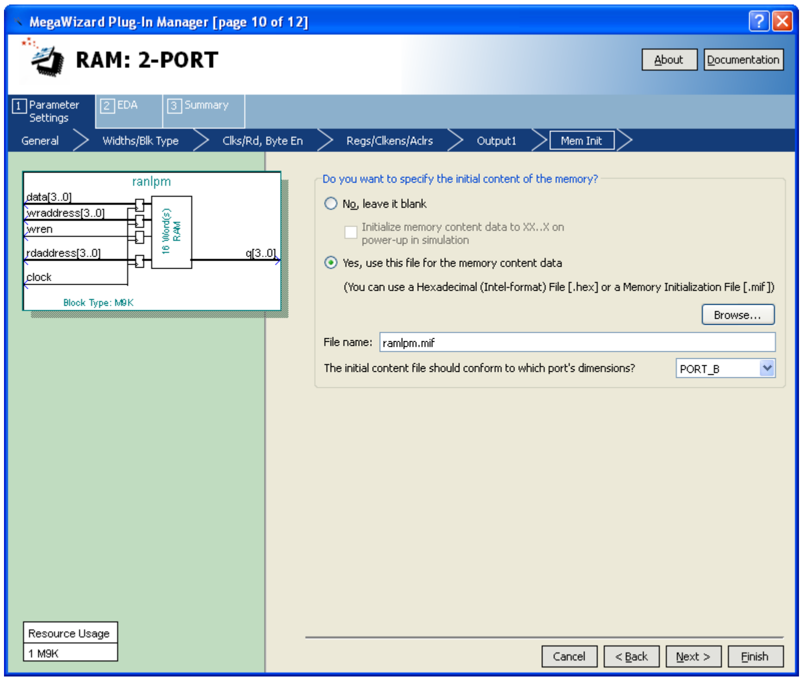
1. Create a new project which will be used to implement the desired circuit on the DE0 board.
2. Write a Verilog file that provides the necessary functionality, including the ability to load the RAM and read its contents as done in Part II.
3. Assign the pins on the FPGA to connect to the switches and the 7-segment displays.
4. Compile the circuit and download it into the FPGA chip.
5. Test the functionality of your design by applying some inputs and observing the output. Describe any differences you observe in comparison to the circuit from Part II.

Part IV

For this part you will create a 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 MegaWizard 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 II. 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 4. 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*.



Do you want to specify the initial content of the memory?

( ) No, leave it blank

[ ] Initialize memory content data to XX..X on power-up in simulation

(●) Yes, use this file for the memory content data

(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif])

File name: ramlpm.mif

The initial content file should conform to which port's dimensions? PORT_B

Figure 4: Specifying a memory initialization file (MIF).

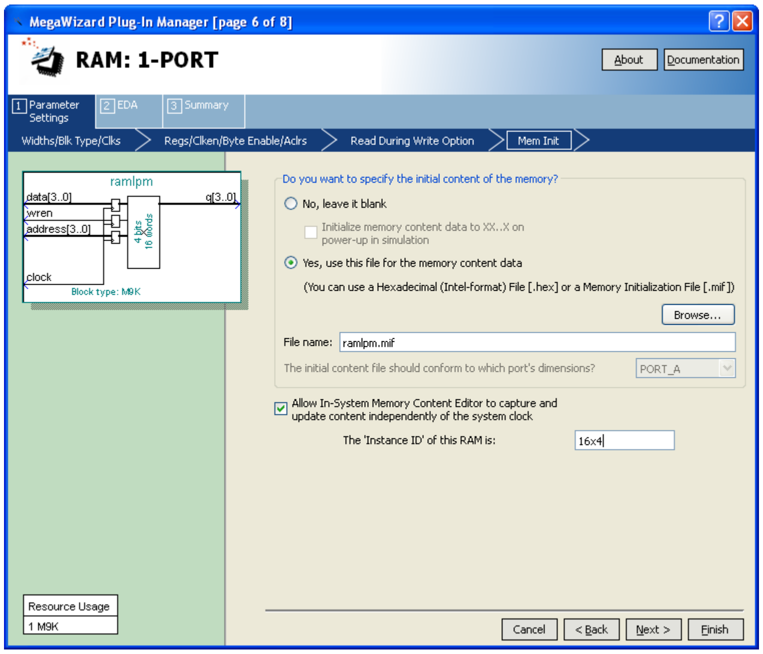
2. Write a Verilog file that instantiates your dual-port memory. To see the RAM contents, add to your design a capability to display the content of each four-bit word (in hexadecimal format) on the 7-segment display *HEX0*. Scroll through the memory locations by displaying each word for about one second. As each byte is being displayed, show its address (in hex format) on the 7-segment display *HEX2*. Use the 50 MHz clock, *CLOCK_50*, on the DE0 board, and use KEY_0 as a reset input. For the write address and corresponding data use the same switches, LEDs, and 7-segment displays as in the previous parts of this exercise. 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 V

The dual-port memory created in Part IV 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 5, specify the *ramlpm.mif* file as you did in Part IV, 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 5 we gave the RAM module the name 16x4. Complete the final steps in the Wizard.



Do you want to specify the initial content of the memory?

No, leave it blank

Initialize memory content data to XX..X on power-up in simulation

[✓ Yes, use this file for the memory content data]

(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif])

File name: ramlpm.mif

The initial content file should conform to which port's dimensions? PORT_A

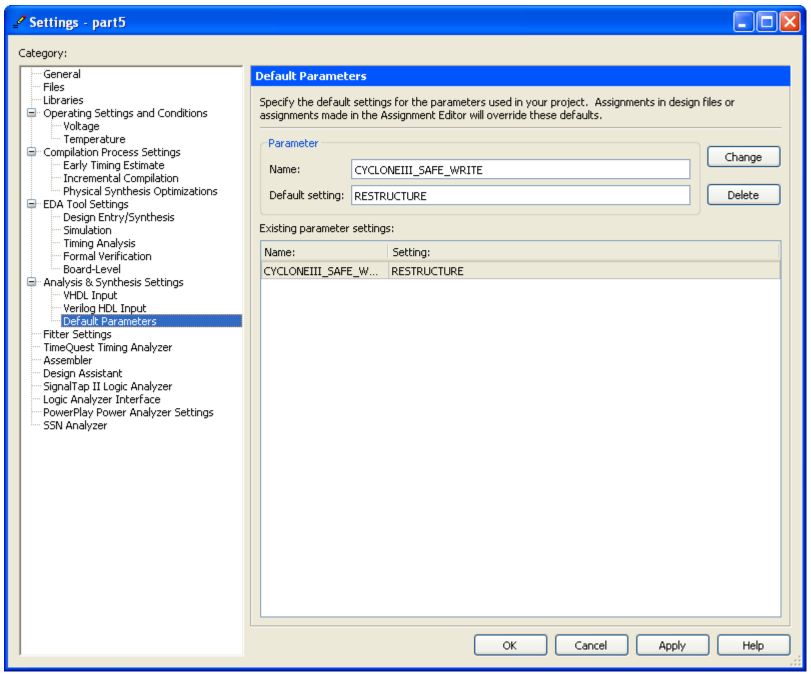
[✓] Allow In-System Memory Content Editor to capture and update content independently of the system clock

The 'Instance ID' of this RAM is: 16x4

Figure 5: 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 IV. Use the same switches, LEDs, and 7-segment displays as you did previously.

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 6, and then open the item called Default Parameters under Analysis and Synthesis Settings. As shown in the figure, type the parameter name CYCLONEIII_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.



Specify the default settings for the parameters used in your project. Assignments in design files or assignments made in the Assignment Editor will override these defaults.

Name: CYCLONEIII_SAFE_WRITE

Default setting: RESTRUCTURE

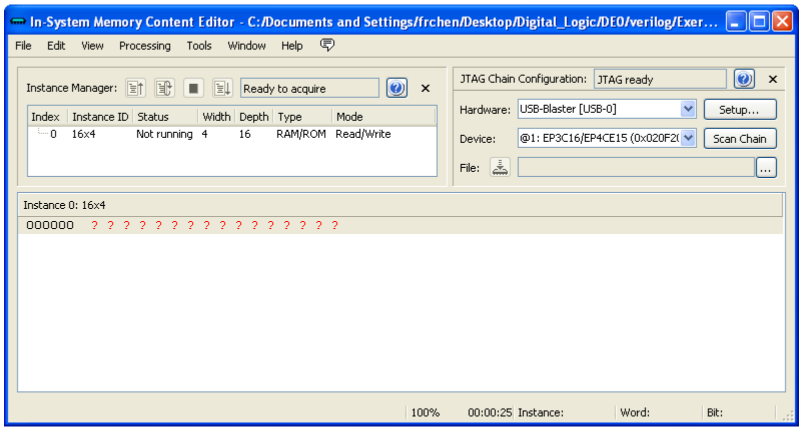
Existing parameter settings:

Name:	Setting:
CYCLONEIII_SAFE_W...	RESTRUCTURE

Figure 6: Setting the *CYCLONEIII_SAFE_WRITE* parameter.

4. Compile your code and download the circuit onto the DE0 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 IV.

5. Select Tools > In-System Memory Content Editor, which opens the window in Figure 7. To specify the connection to your DE0 board click on the Setup button on the right side of the screen. In the window in Figure 8 select the USB-Blaster hardware, and then close the Hardware Setup dialog.


In-System Memory Content Editor - C:/Documents and Settings/frchen/Desktop/Digital_Logic/DE0/verilog/Exer...
File
Edit
View
Processing
Tools
Window
Help
Setup...
Scan Chain

Instance Manager: Ready to acquire

Index	Instance ID	Status	Width	Depth	Type	Mode
0	16x4	Not running	4	16	RAM/ROM	Read/Write

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: EP3C16/EP4CE15 (0x020F2(

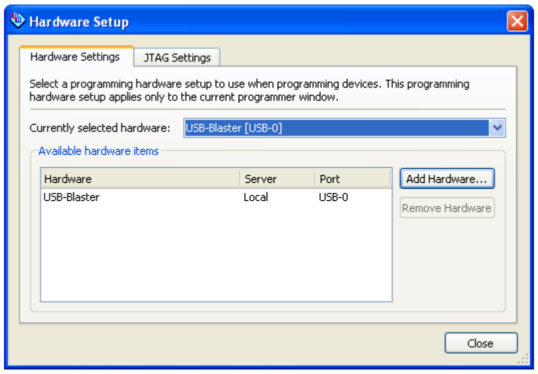
File:

Instance 0: 16x4

```
000000  ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
```

100% 00:00:25 Instance: Word: Bit:

Figure 7: The In-System Memory Content Editor window.



Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-0]

Available hardware items

Hardware	Server	Port
USB-Blaster	Local	USB-0

Figure 8: 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 16x4 memory module, as indicated in Figure 9, 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 16x4 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 DE0 board and in the In-System Memory Content Editor window.

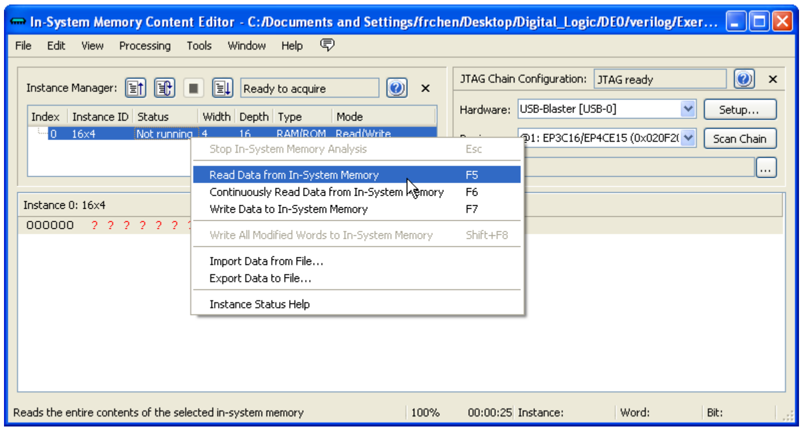

In-System Memory Content Editor - C:/Documents and Settings/frchen/Desktop/Digital_Logic/DE0/verilog/Exer...
File Edit View Processing Tools Window Help
Instance Manager:
Ready to acquire
JTAG Chain Configuration:
JTAG ready
Hardware:
USB-Blaster [USB-0]
Setup...
@1: EP3C16/EP4CE15 (0x020F2(
Scan Chain
Index Instance ID Status Width Depth Type Mode
0 16x4 Not running 4 16 RAM/ROM Read/Write
Stop In-System Memory Analysis Esc
Read Data from In-System Memory F5
Continuously Read Data from In-System Memory F6
Write Data to In-System Memory F7
Write All Modified Words to In-System Memory Shift+F8
Import Data from File...
Export Data to File...
Instance Status Help
Instance 0: 16x4
000000 ? ? ? ? ? ? ?
100%
00:00:25
Instance:
Word:
Bit:
Reads the entire contents of the selected in-system memory

Figure 9: Using the In-System Memory Content Editor tool.