TCES 330
Homework Exercise 5

LPM Memory Blocks

May, 2016

In FPGA systems it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. The FPGA on the DE2-115 boards has 3,981,312 bits of memory available that can be configured in a variety of ways. In this exercise we will configure this memory as a read only memory (ROM) and a random access memory (RAM) using Altera's LPM technology. If you are building a project that requires more memory than is available on the FPGA you will need to add external RAM or ROM to your project. We will not explore these possibilities in this exercise, but Altera's Laboratory Exercise 8 does.

In particular, memory blocks are key components in our programmable processor project. So we should spend some effort to get to know them.

To prepare for this exercise read the tutorials SAD Project Setup (Power Point slides) and SAD Project ModelSim (Power Point Slides).

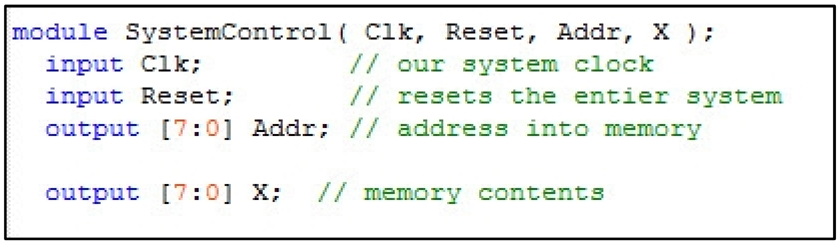
Part 1

In this part we will be building a ROM.

1. Set up an empty Quartus project called Part1 in a folder called Part1. Don't try to compile yet.
2. Have Quartus create a new mif file containing 256 8-bit words. Use the Quartus mif editor to fill the memory as follows. A memory cell in the address range 0 to 128 inclusive should contain the same value as its address. So, for instance, the cell at address 8 should contain an 8. From address 129 to 255 the cell contents should count down from 127 to 1. Cell 136, for instance, should contain 120. Hint: to accomplish this right click on the memory cell area in the editor and look at the cell fill options. Save this memory as Part1.mif.
3. Create a top-level file called Part1.v containing module Part1().

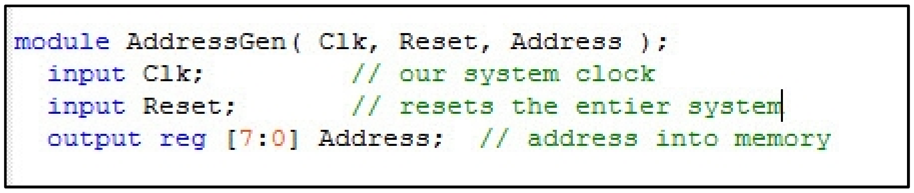
4. We are going to generate and display memory addresses (explained below) as well as the memory contents at each of these addresses. Display the memory contents on HEX1 and HEX0 in hexadecimal form. Display the current memory address on HEX5 and HEX4 in hexadecimal form. KEY0 will reset the address generator back to address 0.
5. The module Part1 will instantiate another module called SystemControl() that will do all the work. See Figure 1 for the signature of this module. SystemControl will instantiate two modules: AddressGen to generate the addresses (as described below) and Part1ROM, an LPM that we will construct using the Wizard (also described below). AddressGen has the signature shown in Figure 2. To instantiate the Part1ROM copy the contents of Par1ROM_inst.v that you will instruct the Wizard to generate for you. Then fill in the values inside the parenthesizes with the appropriate local SystemControl variables. Mine is shown in Figure 3. Keep reading for more information.
6. AddressGen() will instantiate another module that produces a pulse approximately four times per second. Call this timer module anything you want, but make it a separate module. Figure 4 shows the RTL for my AddressGen; yours should look similar, but will undoubtedly have a different name for your timer. AddressGen also contains an 8-bit address register (counter) that
    a. Resets itself to 0 if Reset is true.
    b. Increments if the pulse out of the timer is true. So the address increments about four times per second.
    c. Not counting ‘begin and ‘end’s this counter shouldn’t be any more that 5 lines long.
7. Follow the steps in the SAD Project Setup tutorial to generate your ROM. Name this module Part1ROM. The Wizard Page 3 should look like **Error! Reference source not found.** . On the Wizard Page 5 check the box to generate the Part1ROM_inst.v; see Figure 6. Note that the input Address is registered.
8. Finish setting up Part 1 with any additional files you need.
9. Compile, take care of unused pins (input tri-stated), import pins, etc.
10. Run TimeQuest and include the resulting sdc file in your project.
11. My file list in Quartus looks like Figure 7

12. Look at your RTL. Mine looks like Figure 8. Continue looking through your RTL. Make sure every register is clocked with the system clock (50 MHz CLOCK_50). There will be a minimum 10% penalty if you use derived clocks.
13. Upload this project to your DE2-115 board.
14. Continuing with the SAD tutorial, open the In System Memory Content Editor and view your ROM contents. Take a screen shot to commemorate this occasion and include it in your Part1 folder.
15. Verify that your system counts through the addresses 00 to FF and that your memory contents are displayed correctly.
16. Now use the memory content editor to enter your surname in ASCII at the very end of memory and upload this to your board. Take another screen shot of the memory content editor at this point; mine looks like Figure 9. Verify that the ASCII values you entered show up at the appropriate addresses. Note: you'll just see the ASCII values in hex, not the actual letters.
17. Now refer to the SAD Project ModelSim tutorial. Set up for a ModelSim simulation using my testbench, TestSystemConrtol. In the Settings/Simulation series of dialogs the final one of these should look like Figure 10 and the Test Benches dialog should end up looking like Figure 11. , Run a ModelSim RTL simulation on this circuit and take a screen shot of the Transcript window when the simulation ends. Yours should look like mine (Figure 12). Why are there two outputs for each value of Address? Hint: how far apart in time are these two outputs?



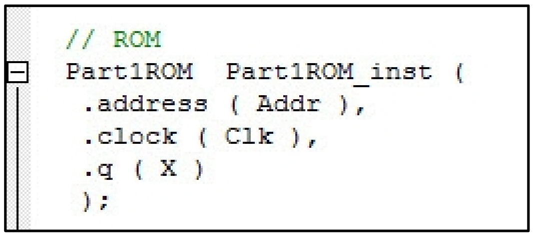
```
module SystemControl( Clk, Reset, Addr, X );
  input Clk;          // our system clock
  input Reset;        // resets the entier system
  output [7:0] Addr;  // address into memory

  output [7:0] X;   // memory contents
```

Figure 1. System Control



```
module AddressGen( Clk, Reset, Address );
  input Clk;          // our system clock
  input Reset;        // resets the entier system
  output reg [7:0] Address;  // address into memory
```

Figure 2. AddressGen



```
  // ROM
  Part1ROM  Part1ROM_inst (
   .address ( Addr ),
   .clock ( Clk ),
   .q ( X )
   );
```

Figure 3 Part1ROM_inst

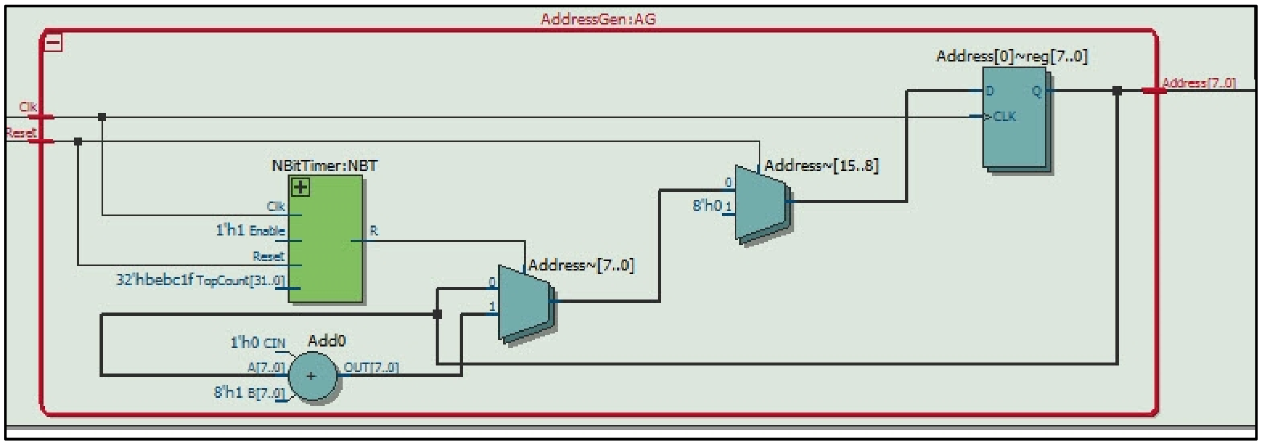
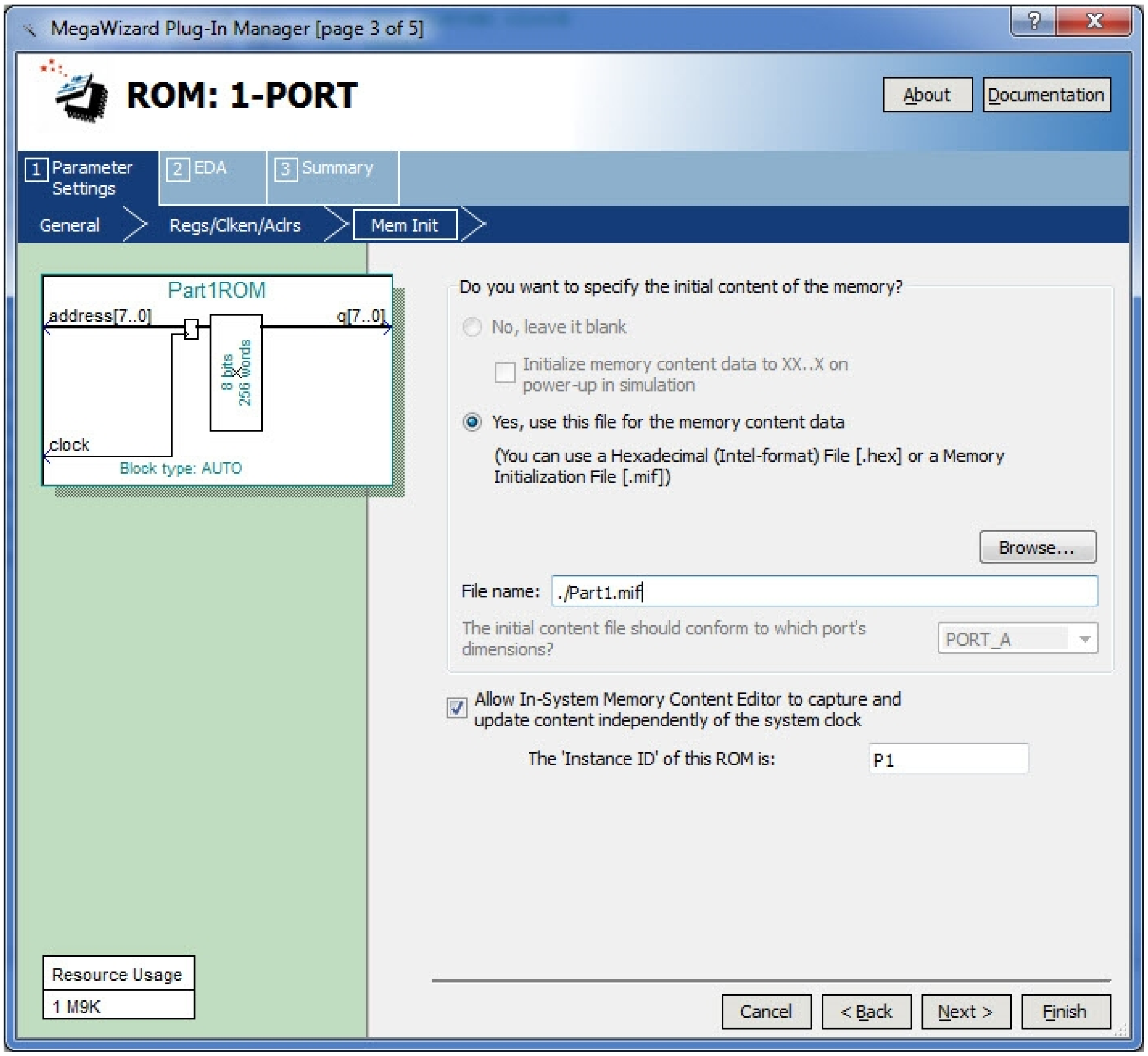


Figure 4 AddressGen RTL


MegaWizard Plug-In Manager [page 3 of 5]

ROM: 1-PORT

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])

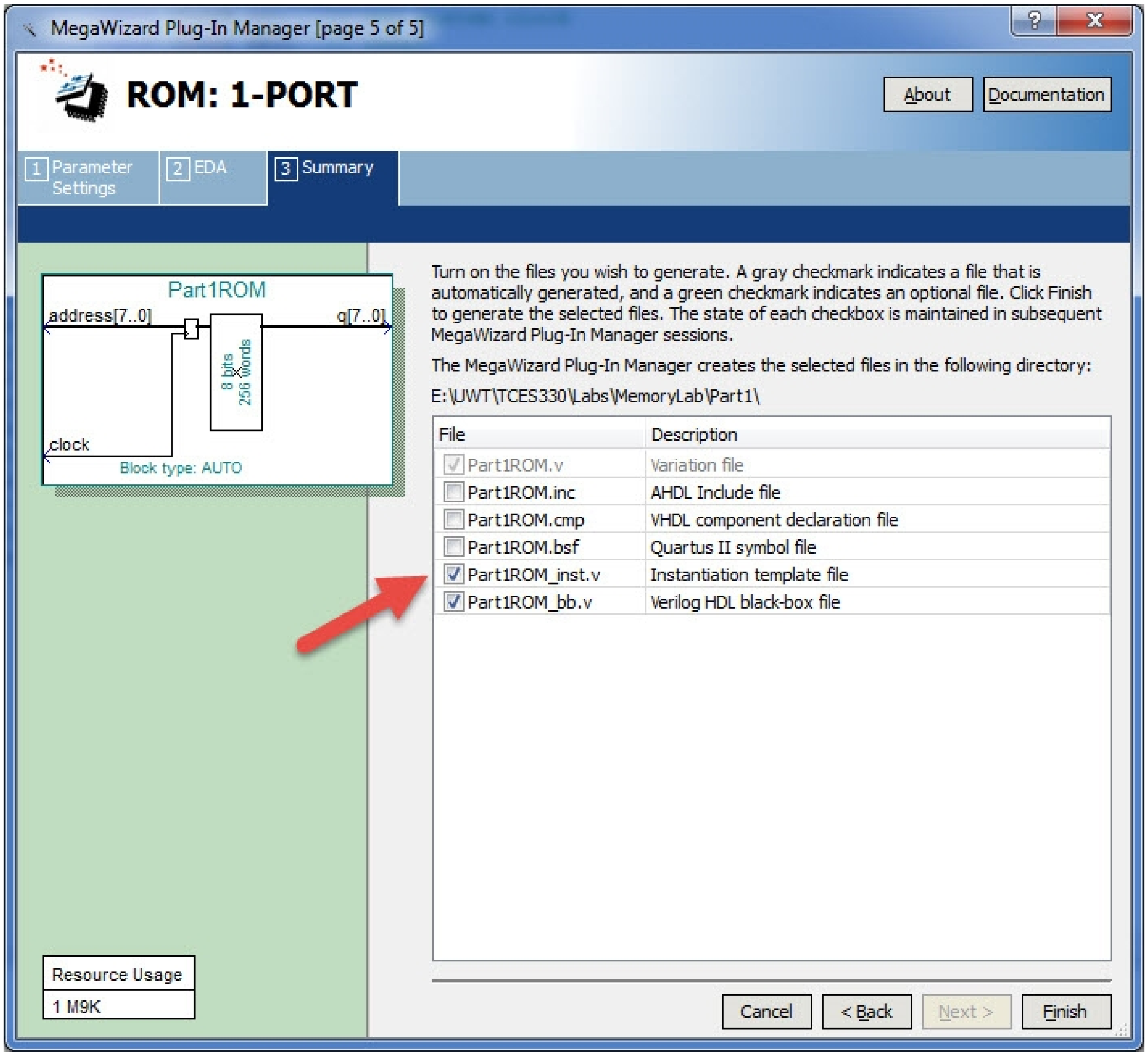
File name: ./Part1.mif

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 ROM is: P1

Figure 5. Wizard Page 3



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:

E:\UWT\TCES330\Labs\MemoryLab\Part1\

File	Description
✓ Part1ROM.v	Variation file
☐ Part1ROM.inc	AHDL Include file
☐ Part1ROM.cmp	VHDL component declaration file
☐ Part1ROM.bsf	Quartus II symbol file
✓ Part1ROM_inst.v	Instantiation template file
✓ Part1ROM_bb.v	Verilog HDL black-box file

Resource Usage
1 M9K

Figure 6. Wizard Page 5

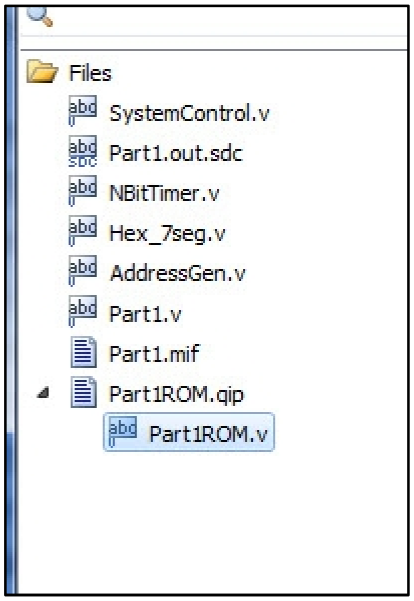

Files
SystemControl.v
Part1.out.sdc
NBitTimer.v
Hex_7seg.v
AddressGen.v
Part1.v
Part1.mif
Part1ROM.qip
Part1ROM.v

Figure 7. Part 1 File List in Quartus

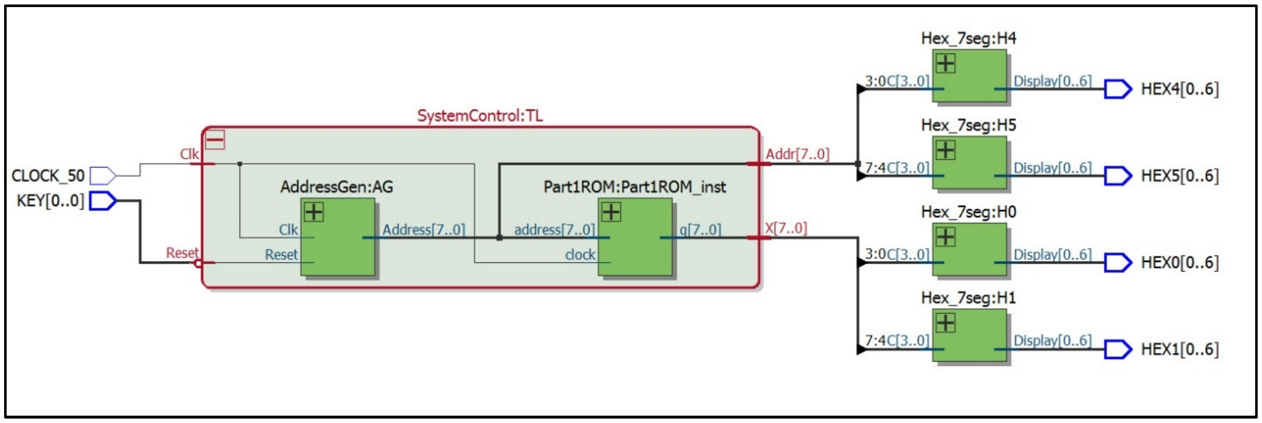
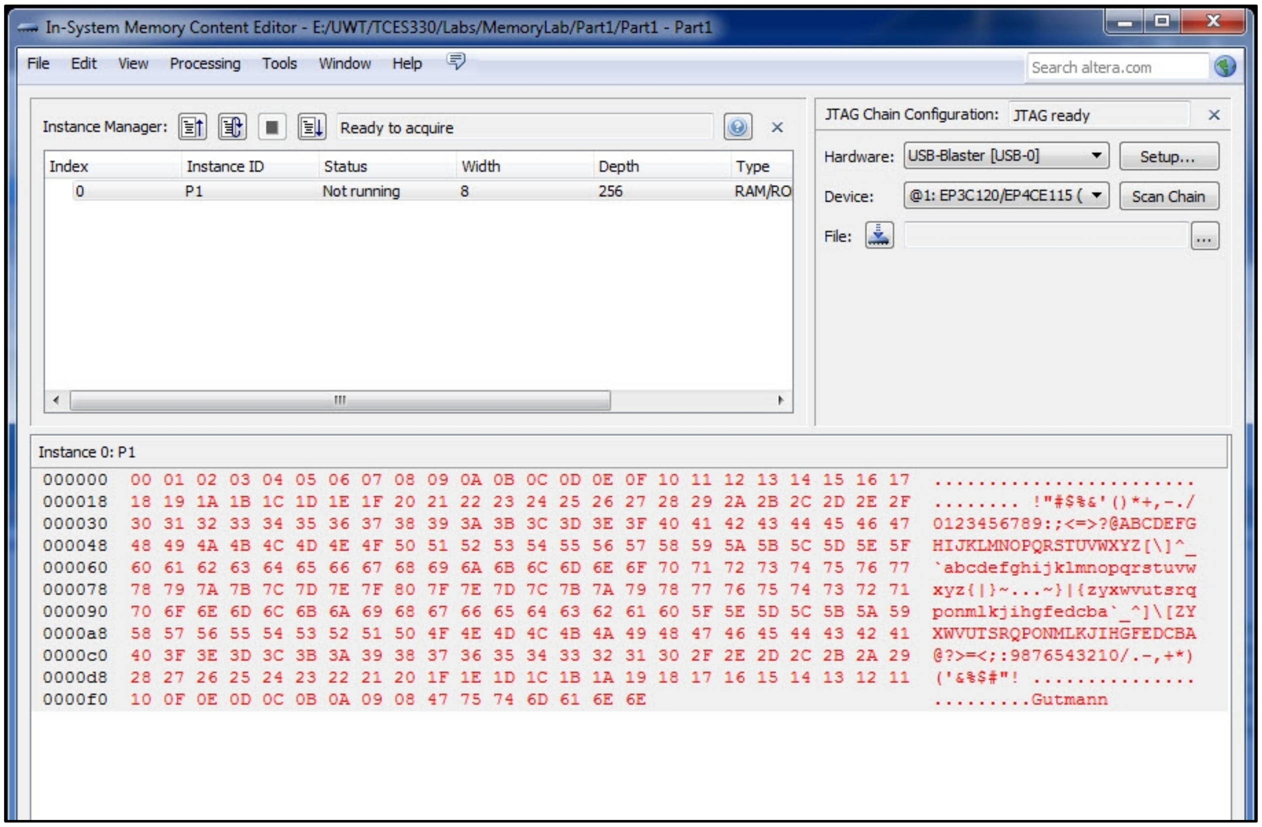


Figure 8. Part 1 RTL



In-System Memory Content Editor - E:/UWT/TCES330/Labs/MemoryLab/Part1/Part1 - Part1

Instance Manager: Ready to acquire

Index	Instance ID	Status	Width	Depth	Type
0	P1	Not running	8	256	RAM/RO

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

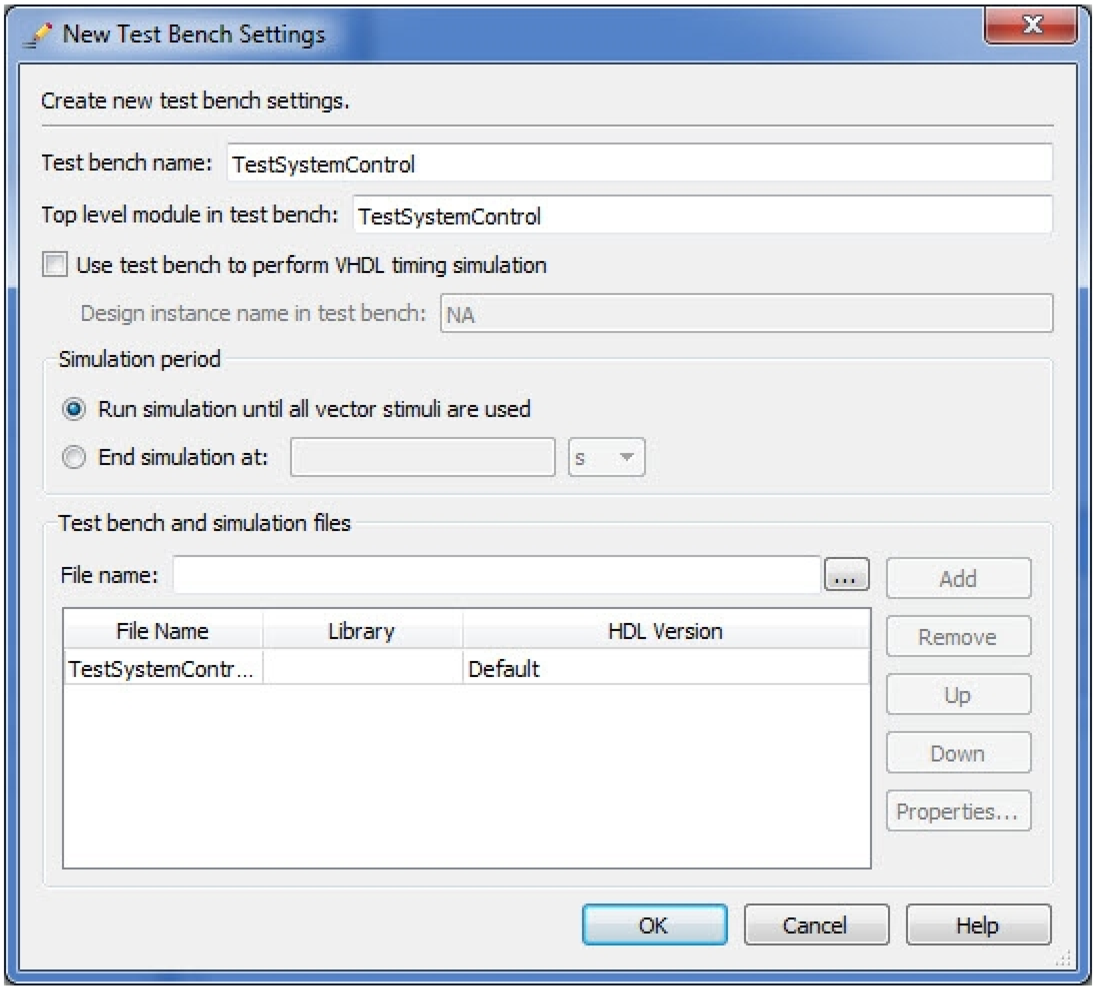
Device: @1: EP3C120/EP4CE115 (

File:

Instance 0: P1

```
000000  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17  ........................
000018  18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F  ........ !"#$%&'()*+,-./
000030  30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47  0123456789:;<=>?@ABCDEFG
000048  48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F  HIJKLMNOPQRSTUVWXYZ[\]^_
000060  60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77  `abcdefghijklmnopqrstuvw
000078  78 79 7A 7B 7C 7D 7E 7F 80 7F 7E 7D 7C 7B 7A 79 78 77 76 75 74 73 72 71  xyz{|}~...~}|{zyxwvutsrq
000090  70 6F 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60 5F 5E 5D 5C 5B 5A 59  ponmlkjihgfedcba`_^]\[ZY
0000a8  58 57 56 55 54 53 52 51 50 4F 4E 4D 4C 4B 4A 49 48 47 46 45 44 43 42 41  XWVUTSRQPONMLKJIHGFEDCBA
0000c0  40 3F 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32 31 30 2F 2E 2D 2C 2B 2A 29  @?>=<;:9876543210/.-,+*)
0000d8  28 27 26 25 24 23 22 21 20 1F 1E 1D 1C 1B 1A 19 18 17 16 15 14 13 12 11  ('&%$#"! ...............
0000f0  10 0F 0E 0D 0C 0B 0A 09 08 47 75 74 6D 61 6E 6E                          .........Gutmann
```

Figure 9. Edited Memory


New Test Bench Settings

Create new test bench settings.

Test bench name: TestSystemControl

Top level module in test bench: TestSystemControl

☐ Use test bench to perform VHDL timing simulation

Design instance name in test bench: NA

Simulation period

● Run simulation until all vector stimuli are used

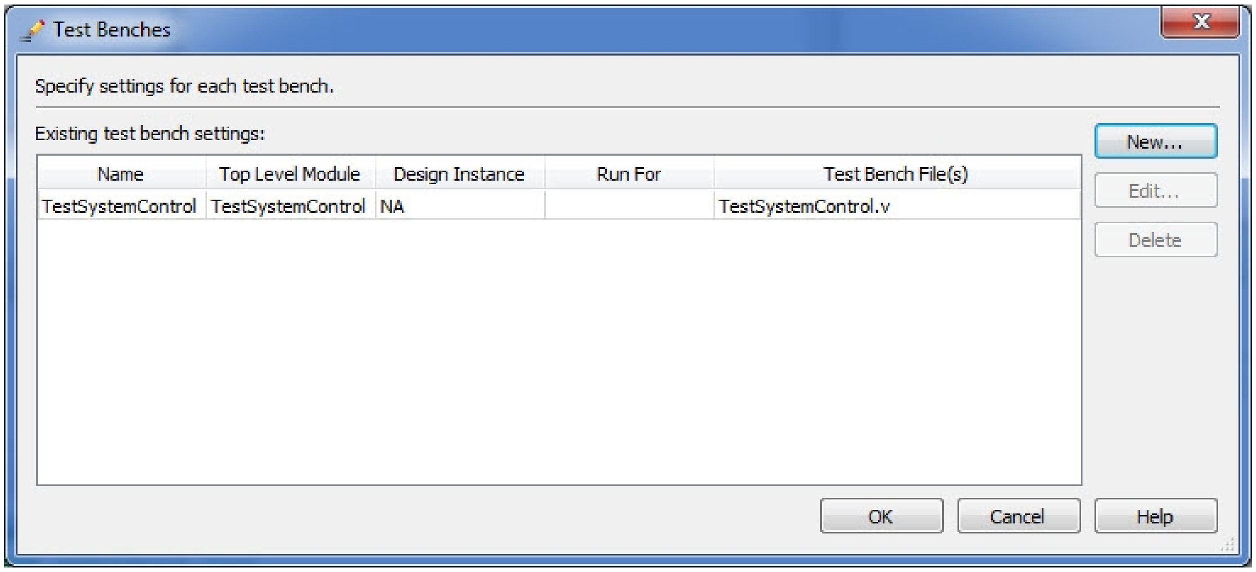
○ End simulation at: s

Test bench and simulation files

File name:

File Name	Library	HDL Version
TestSystemContr...		Default

Figure 10. New Test Bench Settings

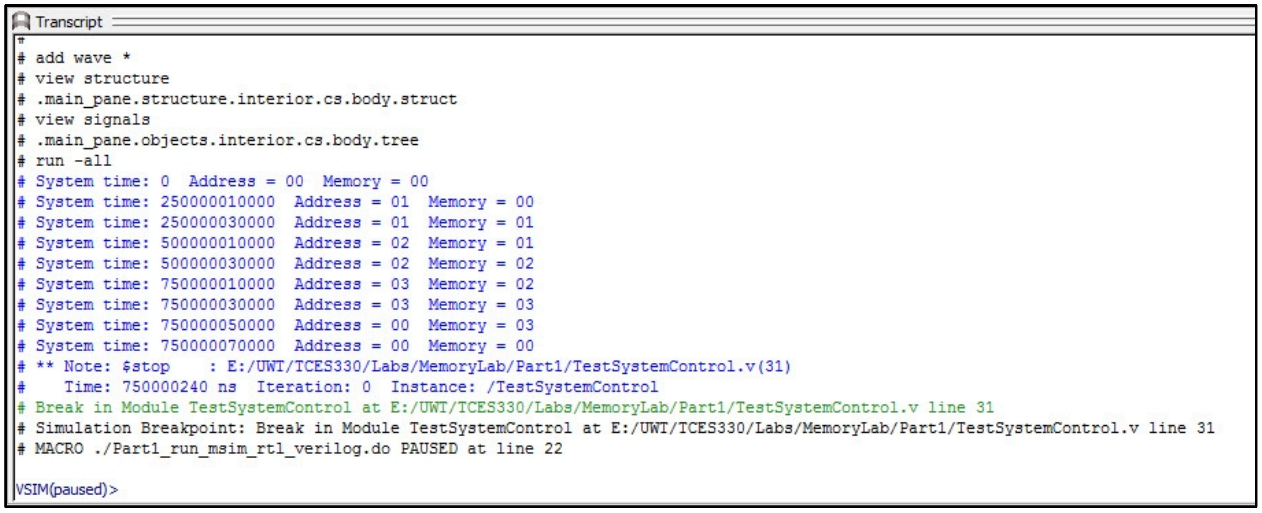


Specify settings for each test bench.

Existing test bench settings:

Name	Top Level Module	Design Instance	Run For	Test Bench File(s)
TestSystemControl	TestSystemControl	NA		TestSystemControl.v

Figure 11. Part 1 Test Benches



```
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# System time: 0  Address = 00  Memory = 00
# System time: 250000010000  Address = 01  Memory = 00
# System time: 250000030000  Address = 01  Memory = 01
# System time: 500000010000  Address = 02  Memory = 01
# System time: 500000030000  Address = 02  Memory = 02
# System time: 750000010000  Address = 03  Memory = 02
# System time: 750000030000  Address = 03  Memory = 03
# System time: 750000050000  Address = 00  Memory = 03
# System time: 750000070000  Address = 00  Memory = 00
# ** Note: $stop    : E:/UWT/TCES330/Labs/MemoryLab/Part1/TestSystemControl.v(31)
#    Time: 750000240 ns  Iteration: 0  Instance: /TestSystemControl
# Break in Module TestSystemControl at E:/UWT/TCES330/Labs/MemoryLab/Part1/TestSystemControl.v line 31
# Simulation Breakpoint: Break in Module TestSystemControl at E:/UWT/TCES330/Labs/MemoryLab/Part1/TestSystemControl.v line 31
# MACRO ./Part1_run_msim_rtl_verilog.do PAUSED at line 22

VSIM(paused)>
```

Figure 12. Part 1 Test Results

Part 2

In this part we will implement a random access memory (RAM) LPM module. A block diagram of a basic RAM is shown in Figure 13. For a RAM the ‘Write Enable’ bit governs the write operation. If it is a ‘1’ then the data on the Write Data line gets written into the memory at the Address. The Data Out line always shows the data at Address. If a write operation is taking place, you get to choose if you want to see the old data or the new data on the output. Refer to Figure 16, below. If Write Enable is ‘0’ then Data Out is simply the data at Address and no write operation takes place.

1. Create a Part2 folder and copy all the Verilog (except the Part1ROM files and the testbench) from Part 1 into the new folder. Rename Part1.v to Part2.v and change the name of the module from Part1 to Part2; other than for comments you shouldn’t need to make any further changes to this module.
2. Create a Quartus project called Part2 and add the Verilog files now in your Part2 folder. Don’t compile yet.
3. Create a mif with 256 8-bit words. Just leave it as all zeros. Save this as Part2.mif.
4. Create a single-port RAM LPM called Part2RAM using the Wizard. Screen shots for this are shown in Figure 14 through Figure 19. Once again, note that all the inputs are registered.
5. Replace the reference to the ROM in SystemControl with a reference to your Part2RAM.
6. Modify your design so that
    a. The first pass through memory you don’t change anything (i.e., you display ‘00’ for each address (Write Enable = ‘0’).
    b. The next pass (and all further passes) through memory you store (write) the address of each word in that word. So location 08 would contain a 08, etc.
    c. Display memory at about 8 Hz.
7. Complete your project by assigning pins, taking care of unused pins, running TimeQuest and so forth.
8. Once your circuit has made that second pass, bring up the In System Memory Content Editor and view the contents of your RAM. Take another screen shot and store this in your Part2 folder.

When you are done with Part 1 and Part 2, zip both projects along with the requested screen shots and submit your zip file on Canvas. Due to the large size of these files, please submit all images as jpg files.

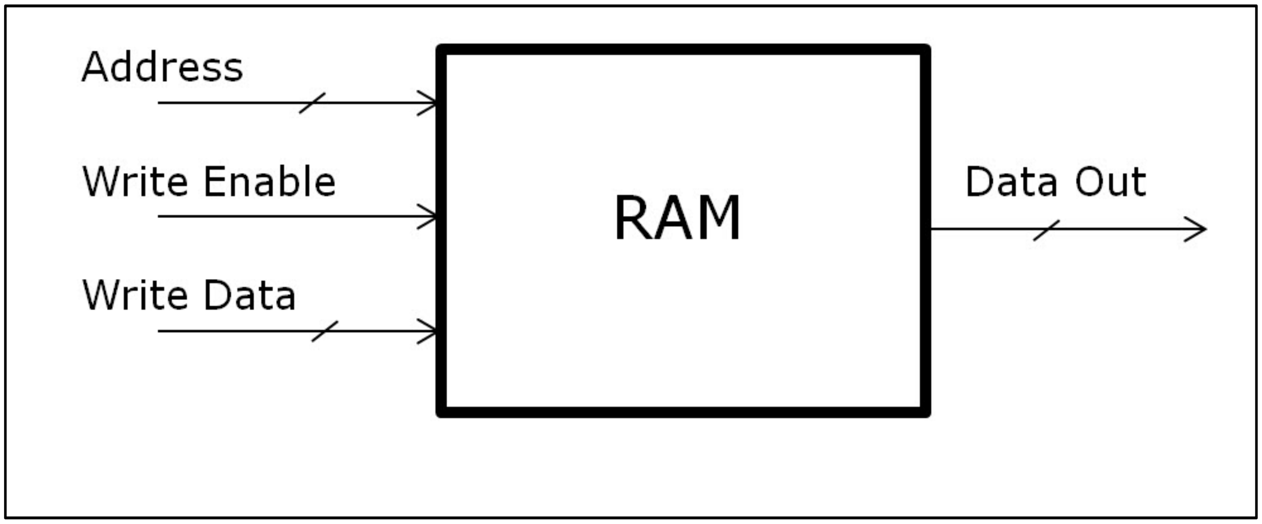
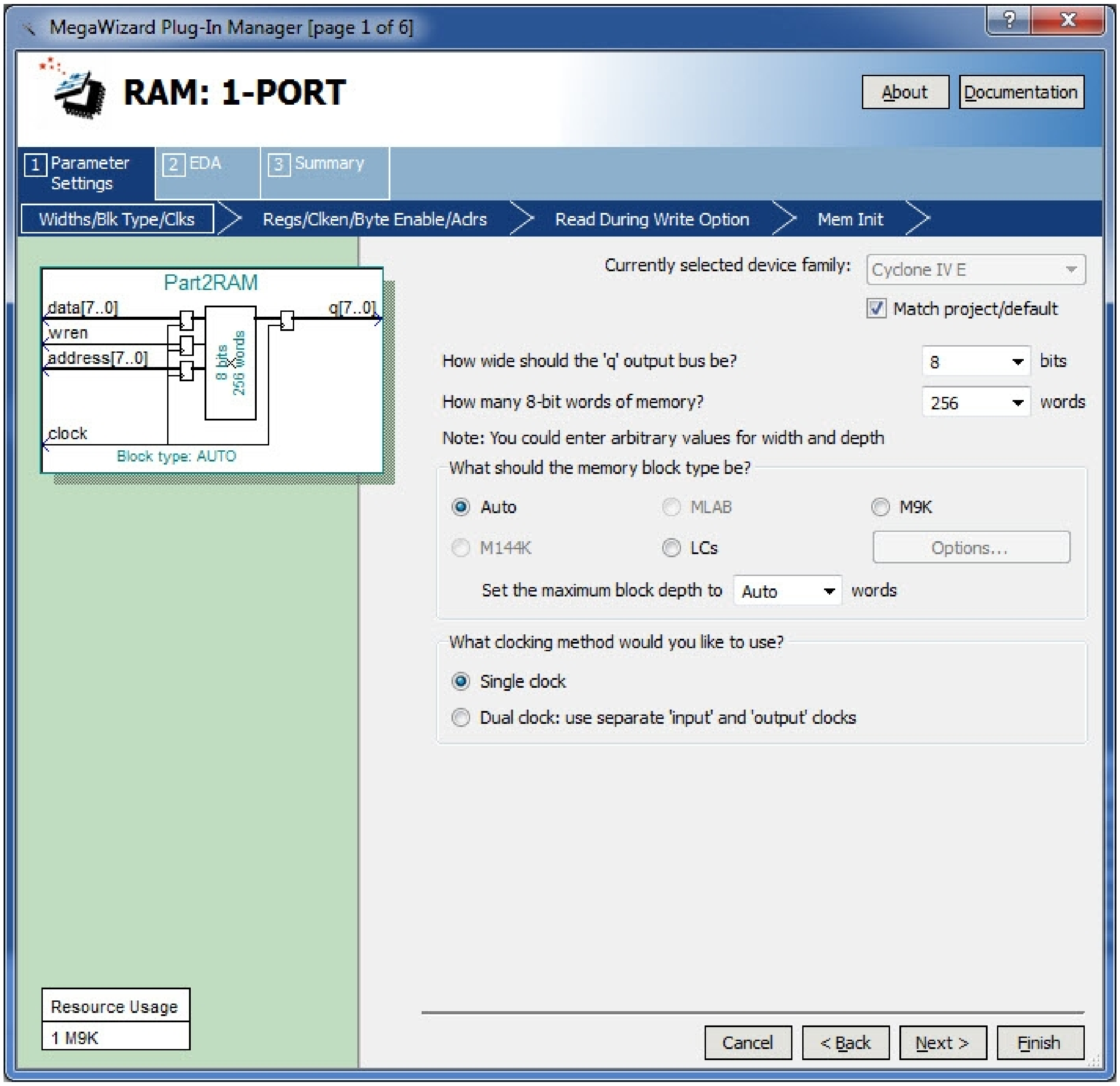


Figure 13. RAM Block Diagram



MegaWizard Plug-In Manager [page 1 of 6]

RAM: 1-PORT

Resource Usage
1 M9K

Currently selected device family: Cyclone IV E

[✓] Match project/default

How wide should the 'q' output bus be? 8 bits

How many 8-bit words of memory? 256 words

Note: You could enter arbitrary values for width and depth

What should the memory block type be?

[● Auto] MLAB M9K M144K LCs

Options...

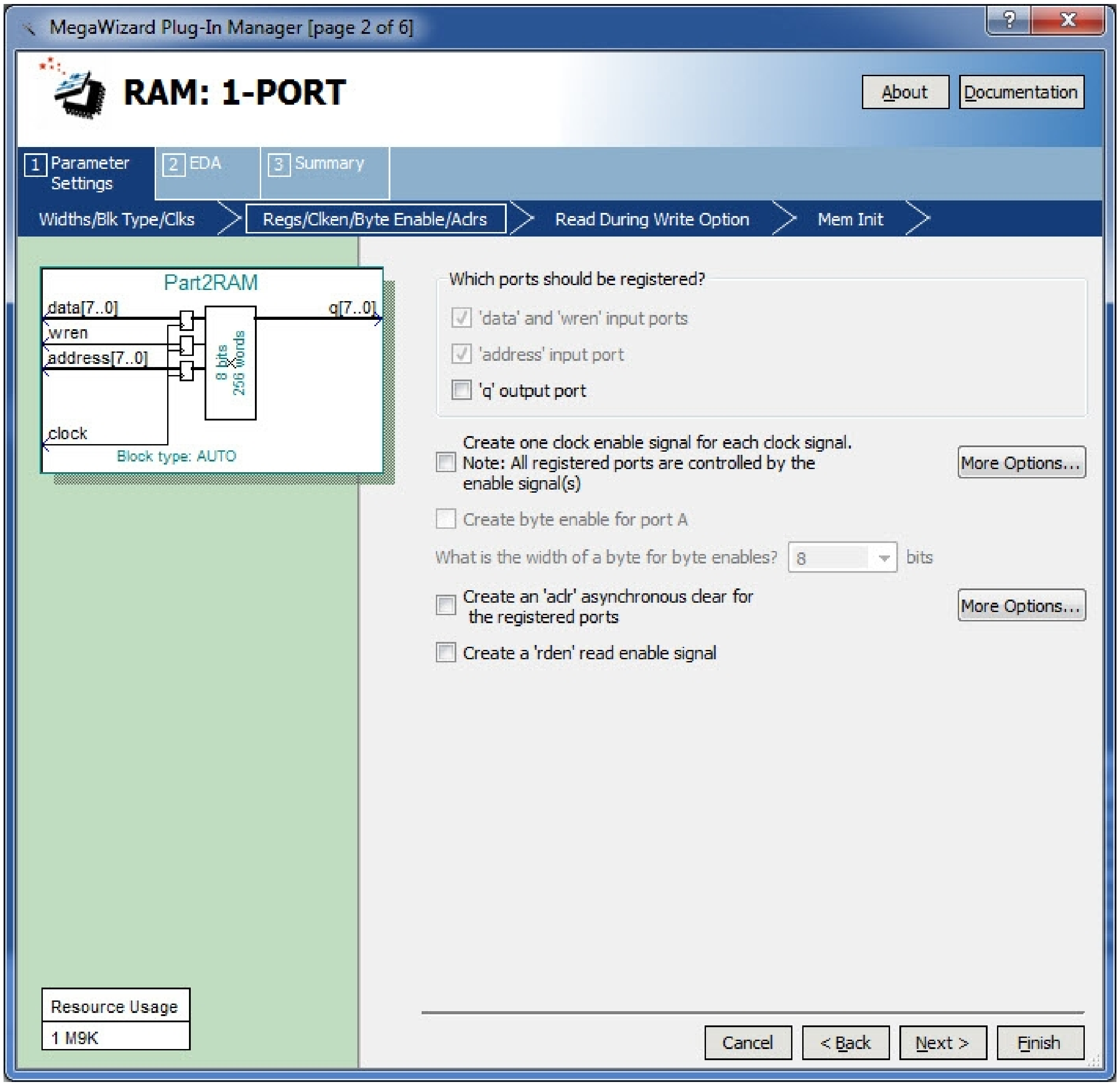
Set the maximum block depth to Auto words

What clocking method would you like to use?

[● Single clock]

Dual clock: use separate 'input' and 'output' clocks

Figure 14. RAM Page 1



Which ports should be registered?

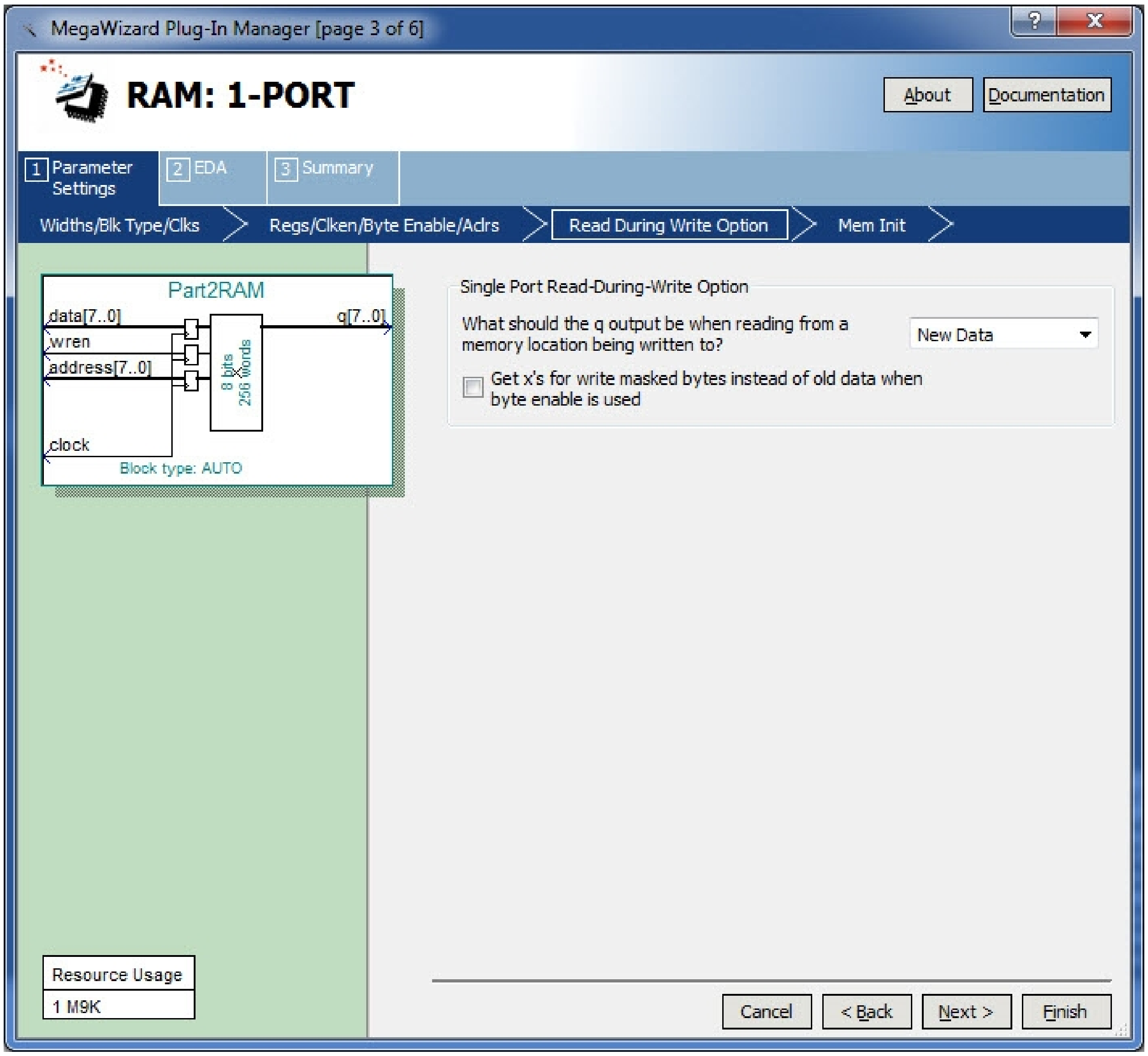
- [✓] 'data' and 'wren' input ports
- [✓] 'address' input port
- [ ] 'q' output port

- [ ] 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 15. RAM Page 2


MegaWizard Plug-In Manager [page 3 of 6]

RAM: 1-PORT

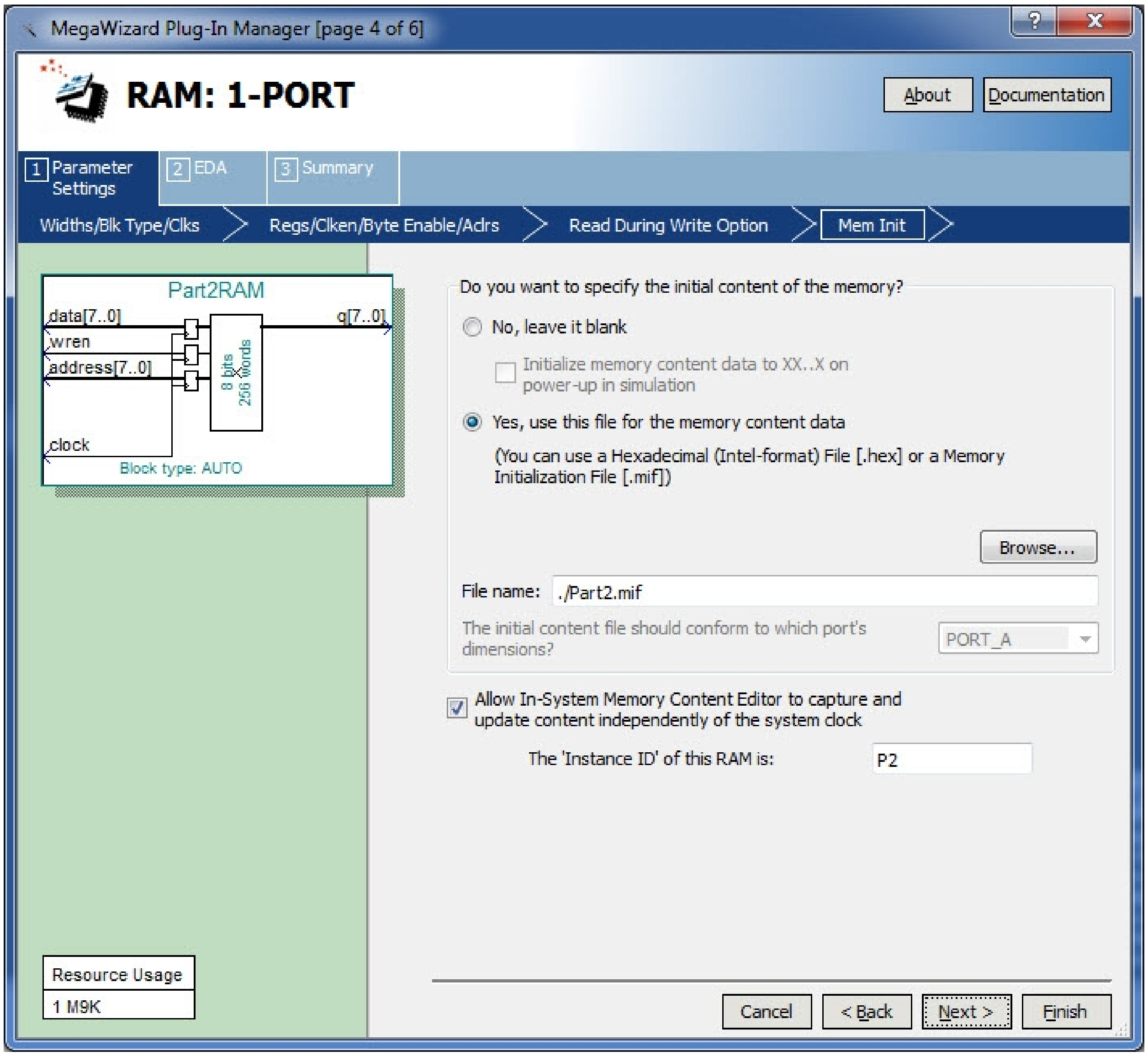
Single Port Read-During-Write Option

What should the q output be when reading from a memory location being written to? New Data

☐ Get x's for write masked bytes instead of old data when byte enable is used

Resource Usage: 1 M9K

Figure 16. RAM Page 3


MegaWizard Plug-In Manager [page 4 of 6]

RAM: 1-PORT

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: ./Part2.mif

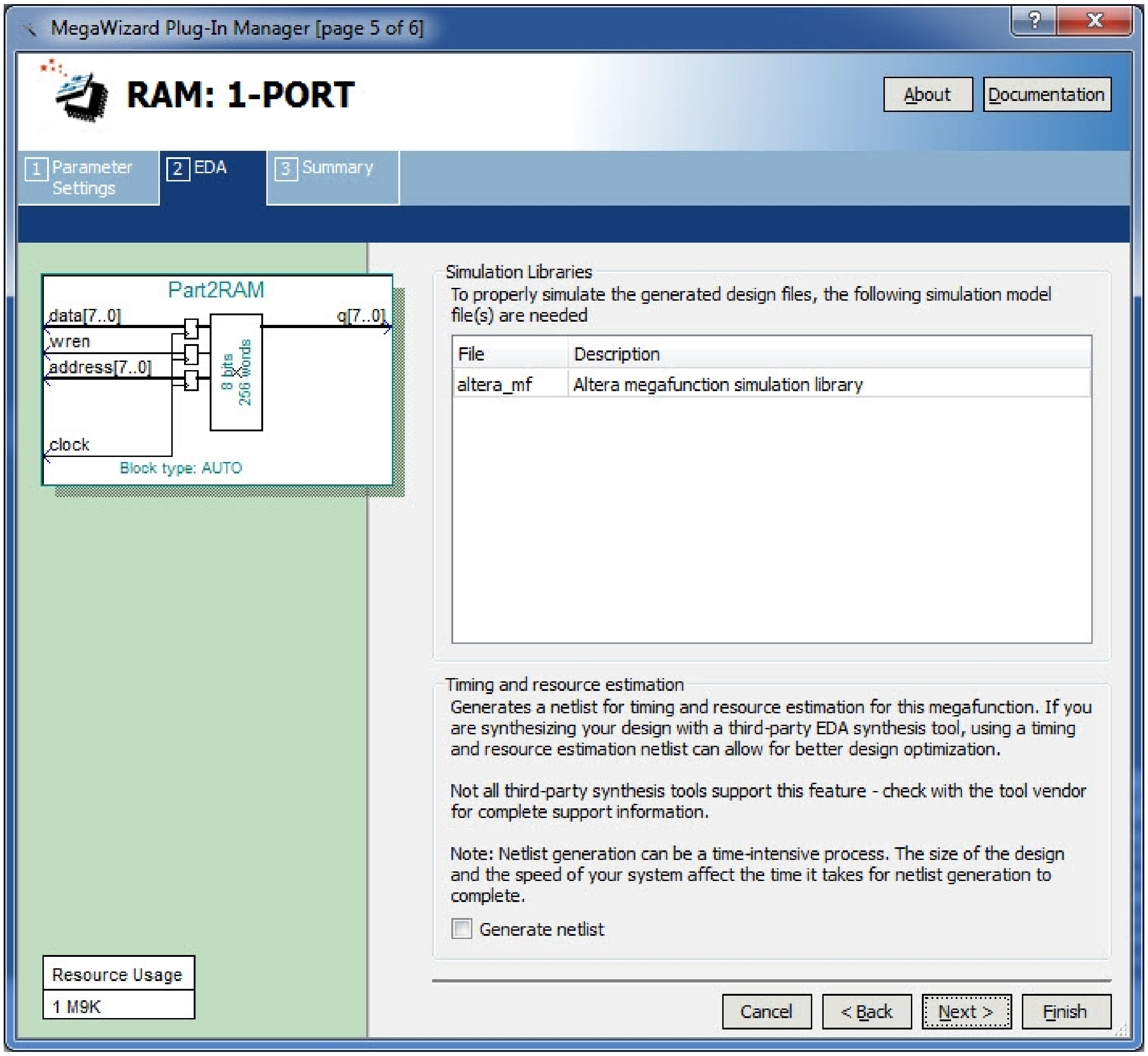
The initial content file should conform to which port's dimensions? PORT_A

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

The 'Instance ID' of this RAM is: P2

Resource Usage
1 M9K

Figure 17. RAM Page 4



Simulation Libraries

To properly simulate the generated design files, the following simulation model file(s) are needed

File	Description
altera_mf	Altera megafunction simulation library

Timing and resource estimation

Generates a netlist for timing and resource estimation for this megafunction. If you are synthesizing your design with a third-party EDA synthesis tool, using a timing and resource estimation netlist can allow for better design optimization.

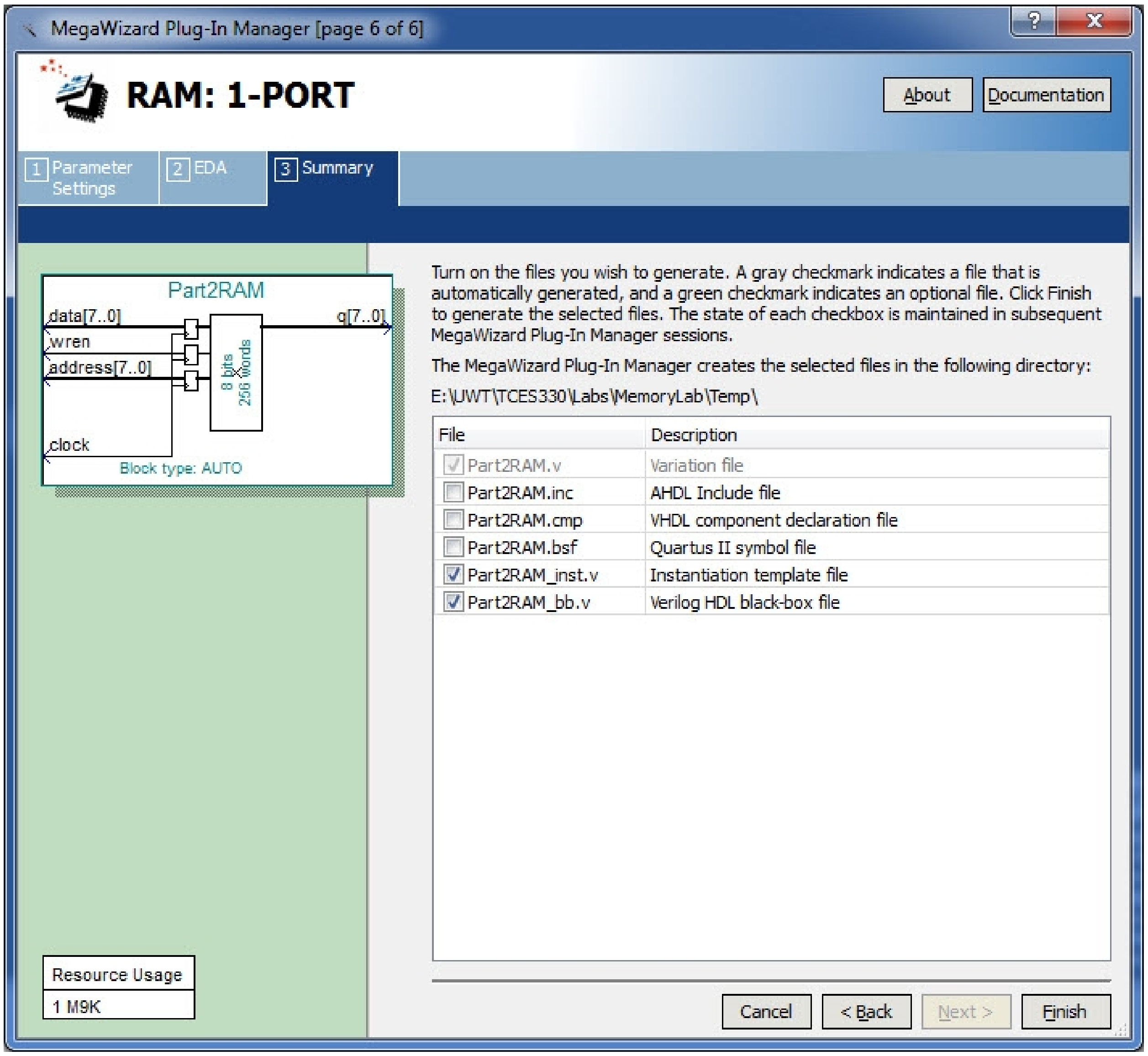
Not all third-party synthesis tools support this feature - check with the tool vendor for complete support information.

Note: Netlist generation can be a time-intensive process. The size of the design and the speed of your system affect the time it takes for netlist generation to complete.

☐ Generate netlist

Resource Usage
1 M9K

Figure 18. RAM Page 5



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:

E:\UWT\TCES330\Labs\MemoryLab\Temp\

File	Description
✓ Part2RAM.v	Variation file
☐ Part2RAM.inc	AHDL Include file
☐ Part2RAM.cmp	VHDL component declaration file
☐ Part2RAM.bsf	Quartus II symbol file
✓ Part2RAM_inst.v	Instantiation template file
✓ Part2RAM_bb.v	Verilog HDL black-box file

Figure 19. RAM Page 6