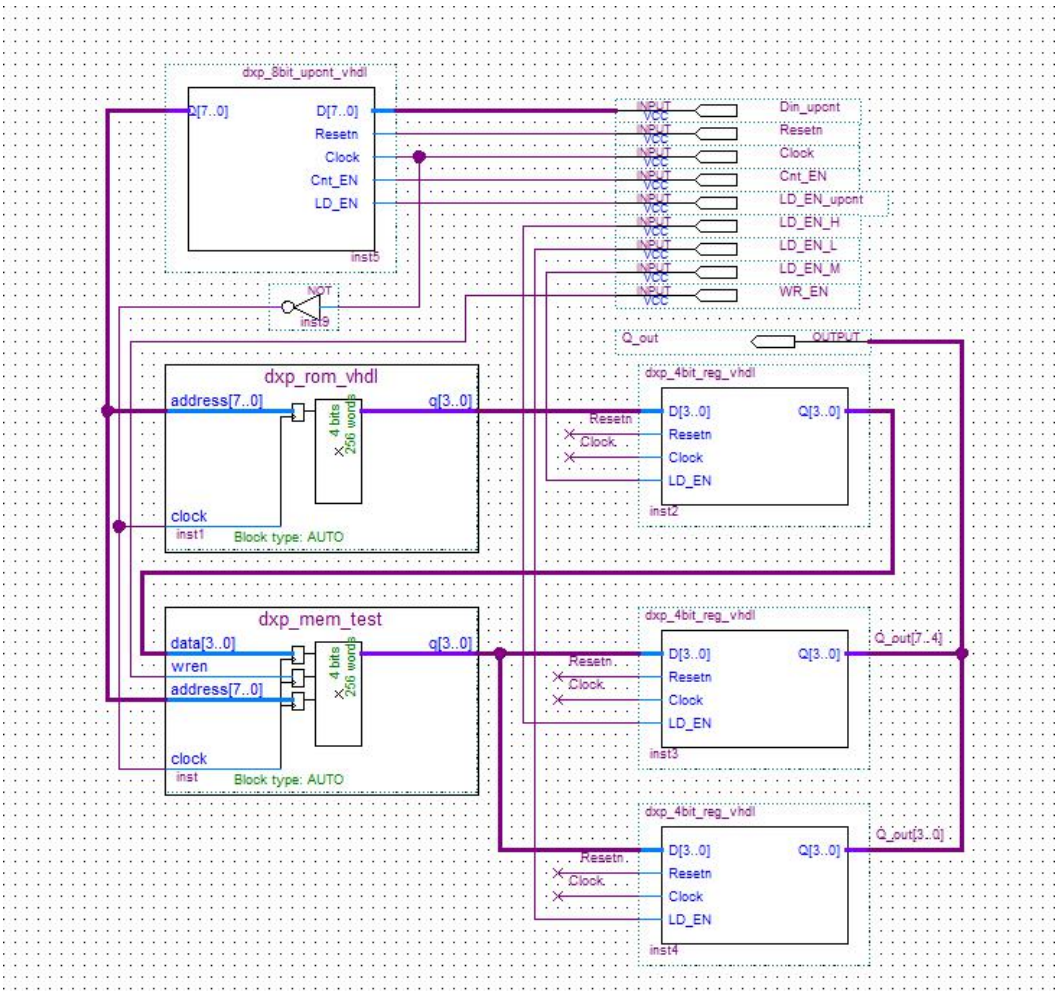
EEEE-220-Lab6

1) From mycourses download on your desktop this pdf file and all other code files up to Lab7 handout. This lab consists of six parts:
2) **Objective**: The objective of this lab exercise is to design and verify registers, counters, and memory arrays. These will be used later as functional blocks in your fml_RISC (Reduced Instruction Set Computer) design.
3) **Part 1**: Download from my courses dxp_4bit_reg_vhdl.vhd and dxp_4bit_reg_vhdl_tb.vhd.
4) Make the necessary fml changes in the code and file names.
5) Create a new project, add these two files to it, and verify it, i.e. run the testbench on the top-level entity.
6) The code is annotated with comments. Take the necessary time to read and understand it. This concludes part 1.
7) **Part 2**: Download from my courses dxp_8bit_upcnt_vhdl.vhd and dxp_8bit_upcnt_vhdl_tb.vhd.
8) Make the necessary fml changes in the code and file names.
9) Create a new project, add these two files to it, and verify it, i.e. run the testbench on the top-level entity.
10) The code is annotated with comments. Take the necessary time to read and understand it. This concludes part 2.
11) **Part 3**: Download from my courses dxp_4bit_reg_v.v and dxp_4bit_reg_v_tb.v.
12) Make the necessary fml changes in the code and file names.
13) Create a new project, add these two files to it, and verify it, i.e. run the testbench on the top-level entity.
14) The code is annotated with comments. Take the necessary time to read and understand it. This concludes part 3.
15) **Part 4**: Download from my courses dxp_8bit_upcnt_v.v and dxp_8bit_upcnt_v_tb.v.
16) Make the necessary fml changes in the code and file names.
17) Create a new project, add these two files to it, and verify it, i.e. run the testbench on the top-level entity.
18) The code is annotated with comments. Take the necessary time to read and understand it. This concludes part 4.
19) **Part 5**: Create a new VHDL project called fml_mem_vhdl with a top entity of the same name.
20) Add via Quartus II the files fml_4bit_reg_vhdl.vhdl and fml_8bit_upcnt_vhdl.vhdl to this project.
21) In the top level-entity of this project you will instantiate the circuit shown below:

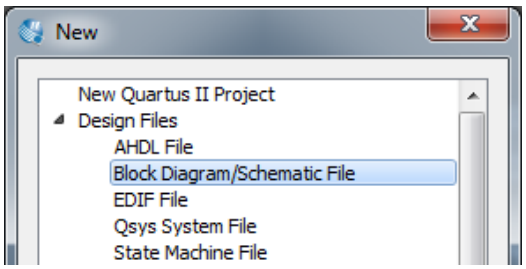


22) In this circuit we use the 8bit counter to generate a continuous sequence of 8bit address values. The ROM (Read Only Memory) will contain the ASCII codes of your first, middle and last names characters in sequence.
23) An ASCII code table you can find in your textbook on page 304 (VHDL edition) or on page 15 (Verilog edition). Alternatively, you can find similar tables on the web (Wikipedia comes to mind). Take the time to read about the ASCII code.
24) In your report include a table with your fml characters in a column, and the corresponding ASCII code in binary in two other columns to the right. Bit 7 is always 0. See an example below:

Character	High Nibble	Low Nibble
SPACE	0010	0000
D	0100	0100
o	0110	1111
r	0111	0010
i	0110	1001

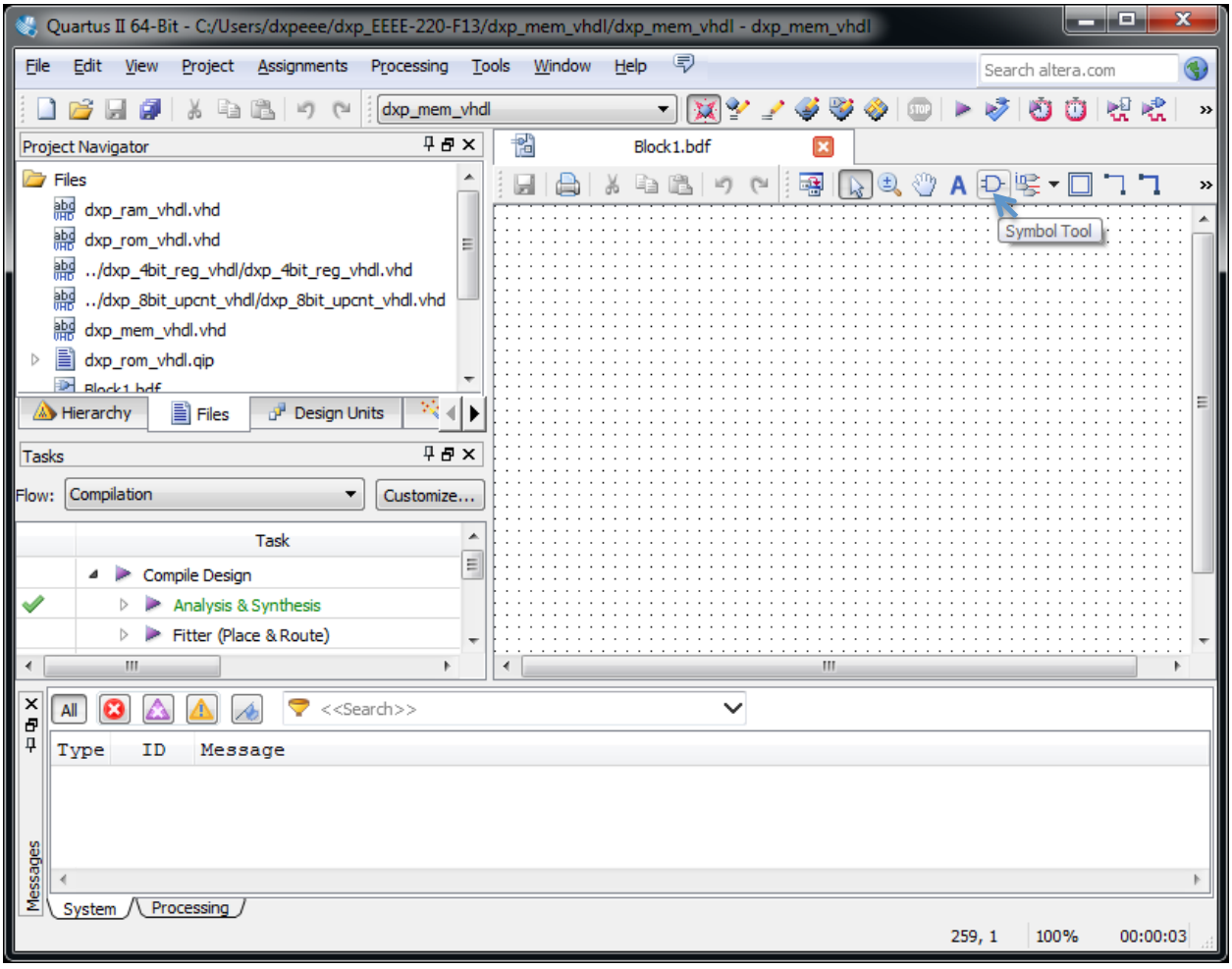
n	0110	1110
SPACE	0010	0000
P	0101	0000
a	0110	0001
t	0111	0100
r	0111	0010
u	0111	0101

25) Download and add via Windows Explorer to your project the file dxp_rom_vhdl.mif. You can open it with a normal text editor such as Notepad++ or within Quartus II. Edit its contents according to your character sequence.
26) This *.mif file contains the values that will be initialized in the ROM. You will use a similar one to initialize your program later on. Its syntax is self-explanatory.
27) Returning to the circuit, the values read from the ROM are stored temporarily in the middle 4bit register. During the next cycle this value will be written in the RAM.
28) As you will see shortly, in the testbench first reads all values of the ROM in sequence (counter generates sequential addresses), and writes them in the RAM during the next clock cycle.
29) Finally, the values are read out of the RAM and stored in two 4bit registers. Because the ASCII code is 8bits wide, we store the high value in one register and the low value in another. The testbench output signal is one 8bit bus. In the waveform window we will change the display radix to ASCII and read out your name.
30) The input clock to both memories uses the complement of the original clock, i.e. the other phase of the clock. This is done to save one cycle. We will use this trick later on in the processor too.
31) Now, to generate the two memories we could instantiate them as large arrays of registers. To physically implement these the compiler would use the available distributed memory in the FPGA, i.e. the SRAM cells embedded in the LUTs. This would consume resources, which could better be used to implement heterogeneous combinational and sequential logic.
32) Alternatively, we can have Quartus II generate the necessary code to create instances of memories that the compiler instantiates in block RAM. These are clusters/pockets of RAM cell arrays integrated for this purpose.
33) To generate these memories, create first a new block diagram/schematic file:

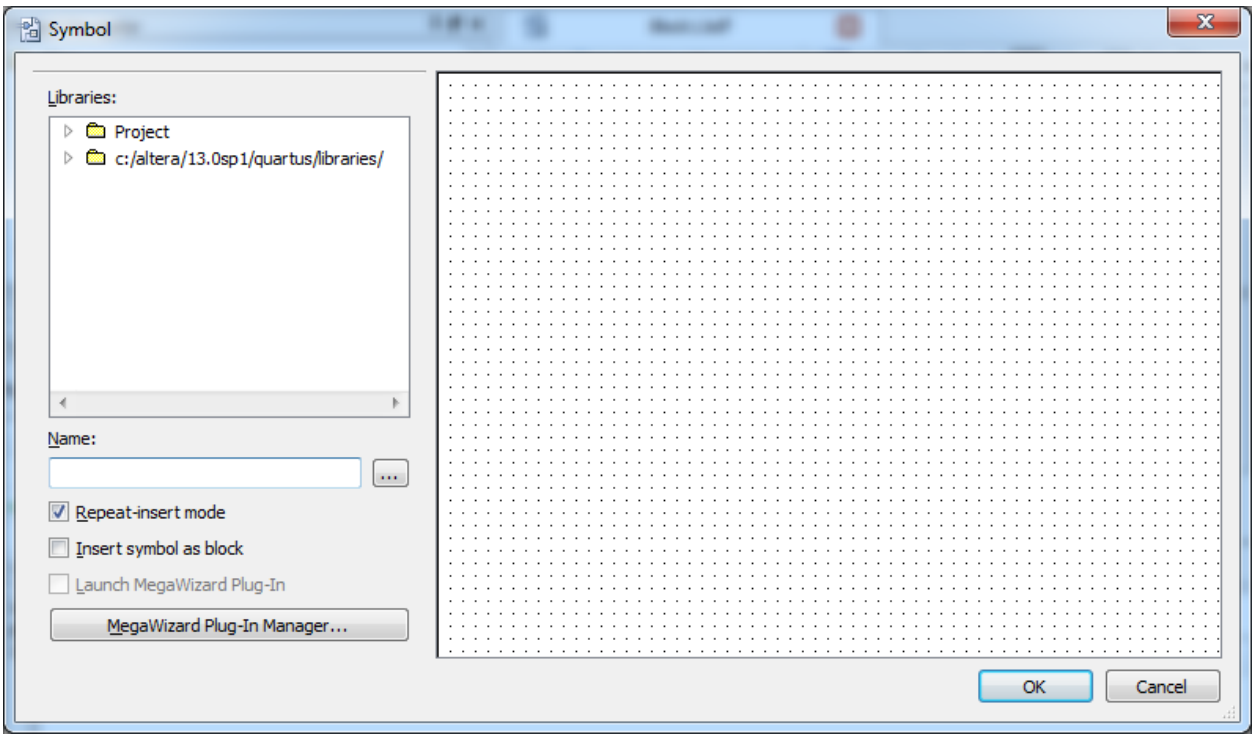


34) You can save it as Block1 or any other name. This is just a placeholder for the memory block symbols. You will be able to delete it after we have created the memories.

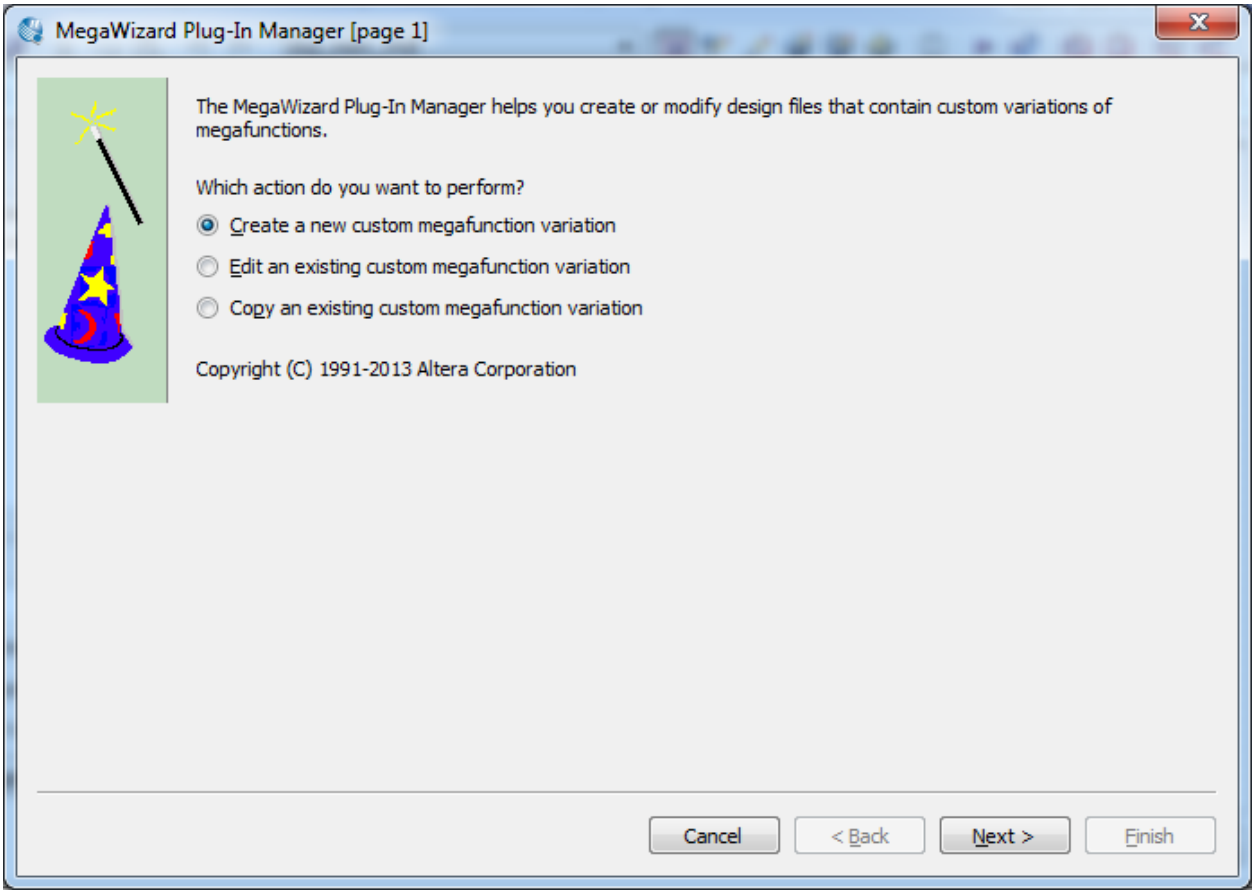
35) Once you have opened the *.bdf file, click on the symbol tool in the upper toolbar (and gate icon):



36) The window below pops up:

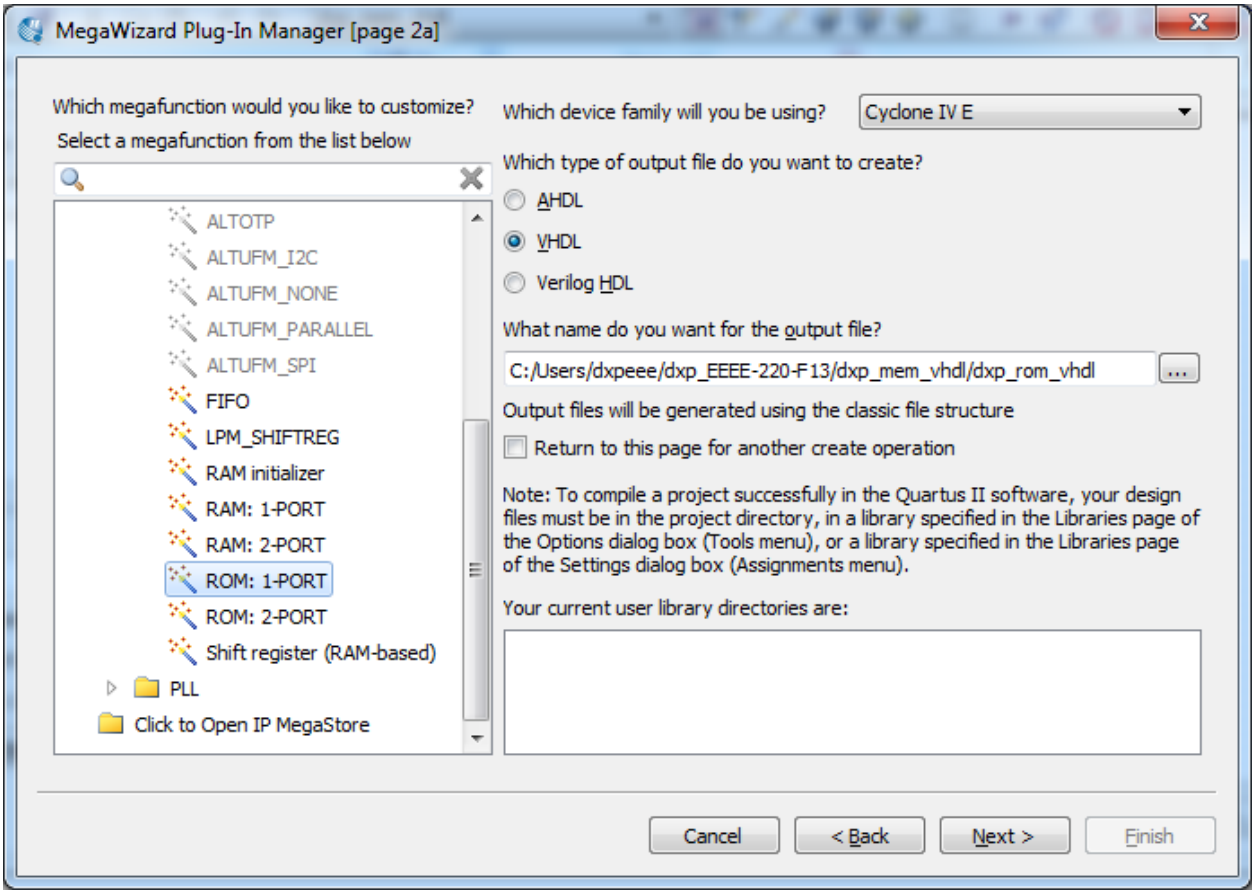


37) Click on the Mega-Wizard Plug-In Manager:

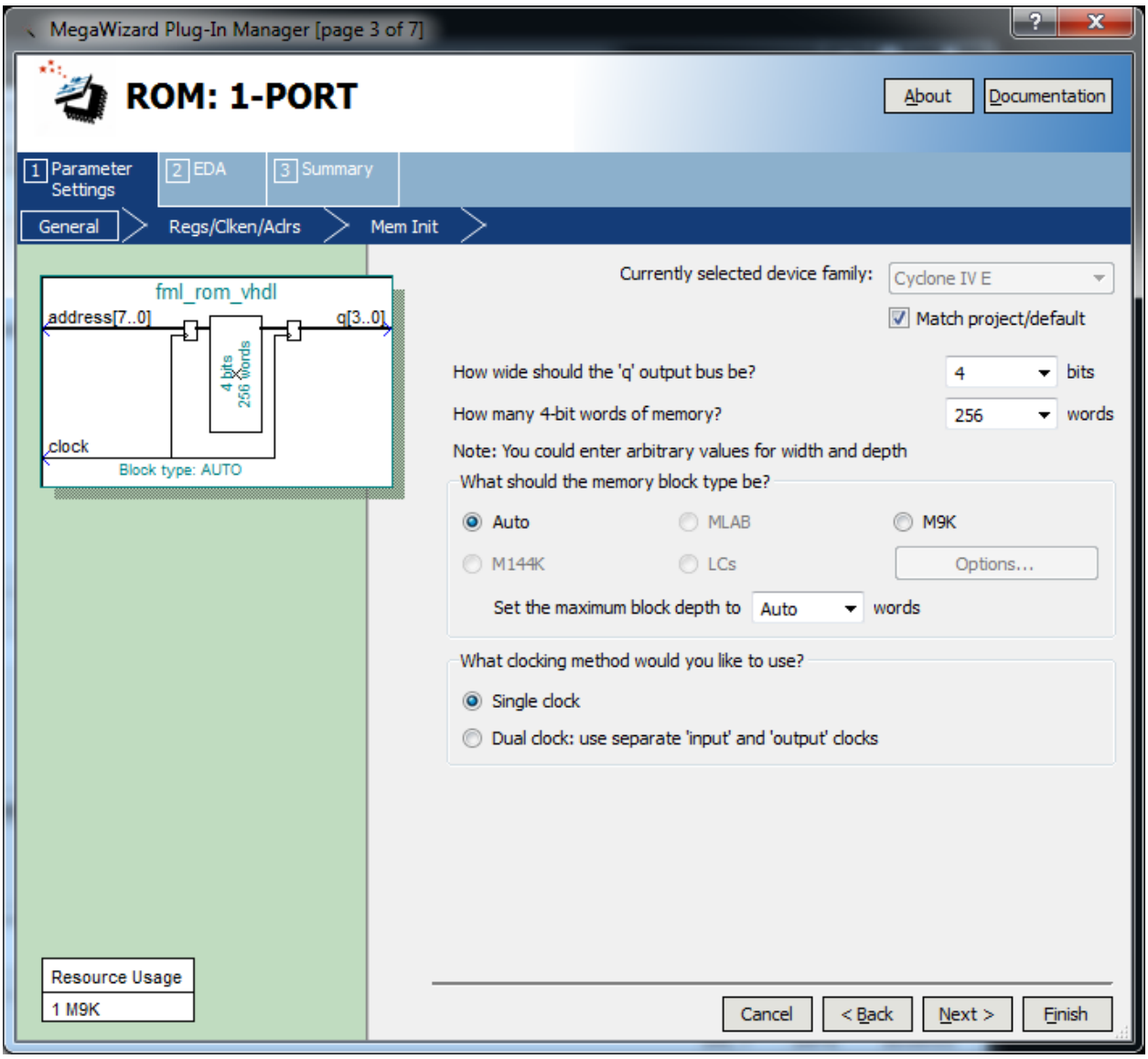


38) Keep the selection to create a new custom mega function.

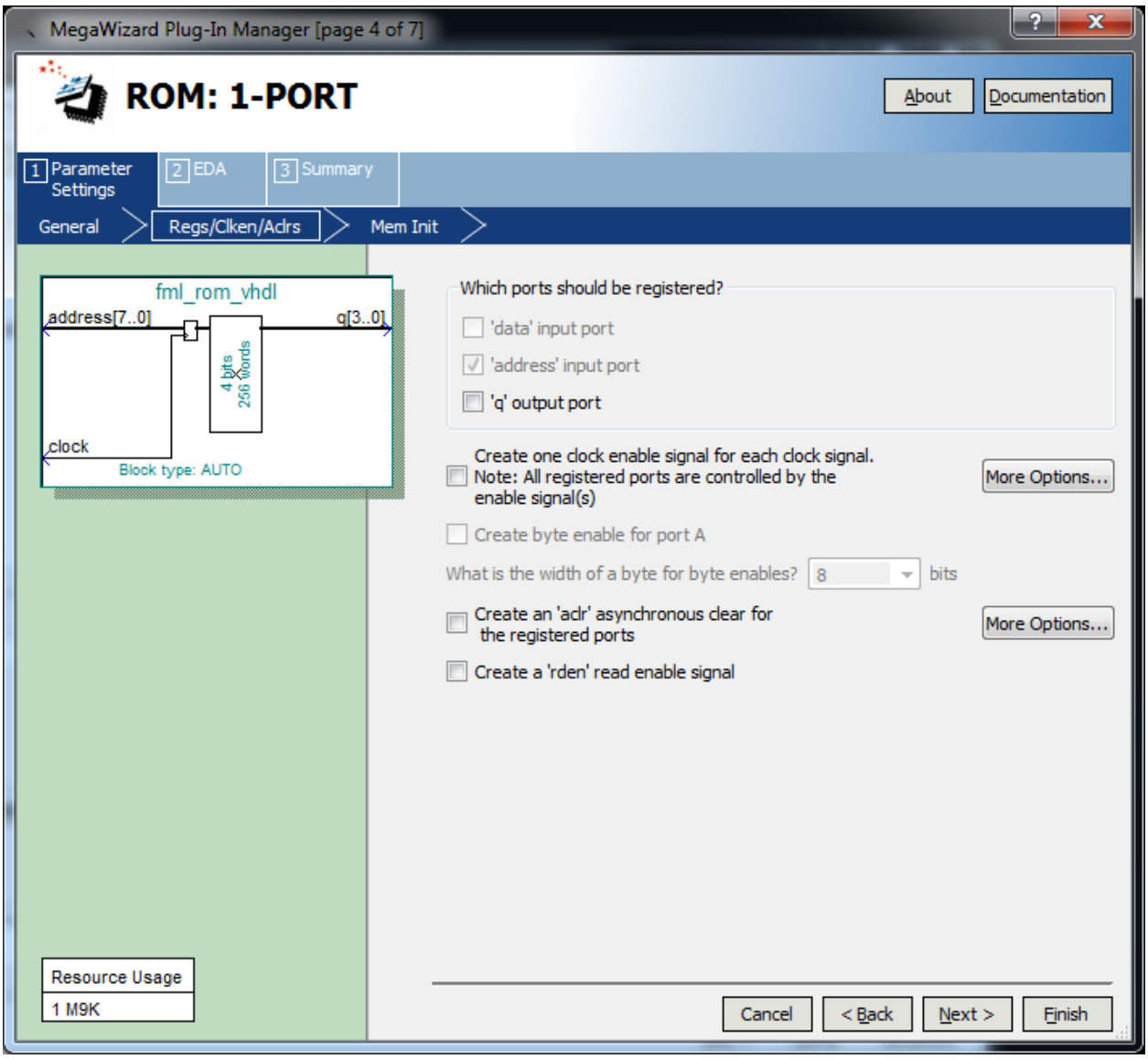
39) In the next window keep the default selections and give it the name: fml_rom_vhdl. Choose the mega-function ROM: 1-PORT. Click Next.



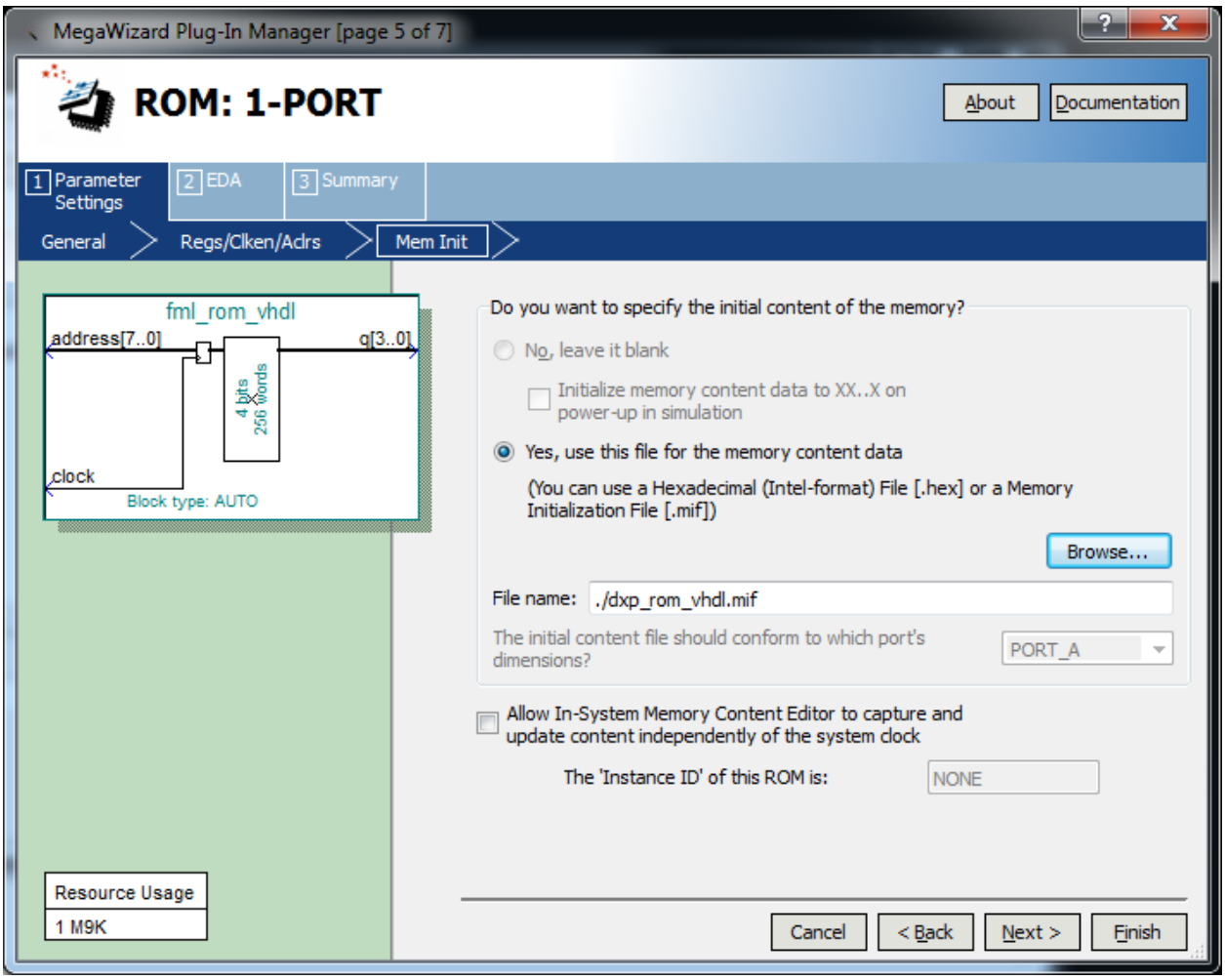
40) In the next window choose the q output bus to be 4bits wide. Leave all else unchanged. With these settings the memory block will contain 256 locations, each 4 bits wide.



41) In the next window uncheck the registration of the output. This means that ones the address is registered, the memory location content to which it points to will be "immediately" available.

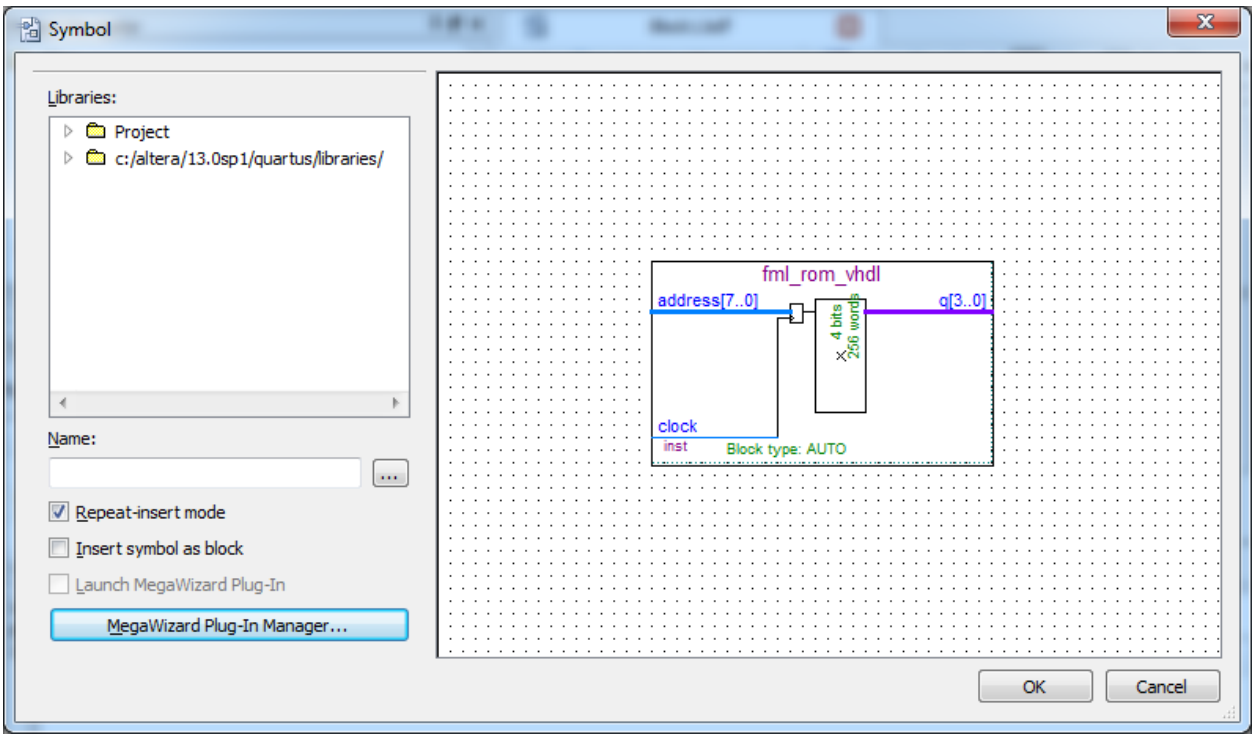


42) The ROM needs to be initialized. Browse to and use as initialization file the one you just created earlier: fml_rom_vhdl.mif. You can change it at any time later.

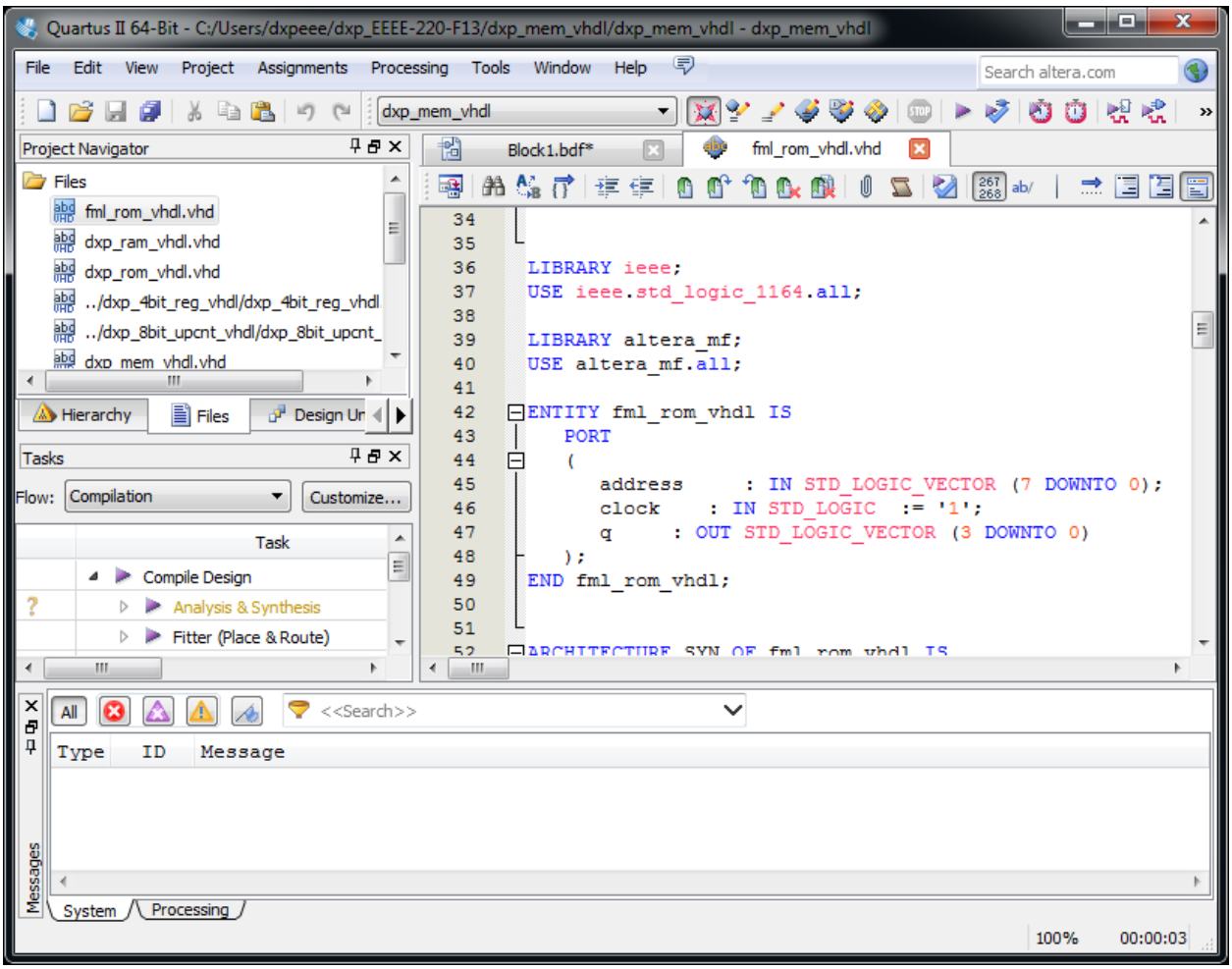


43) Click next two times and then finish. Click yes in the IP window too.

44) Now, you have the symbol of the memory block, which you can instantiate in the *.bdf file schematic.

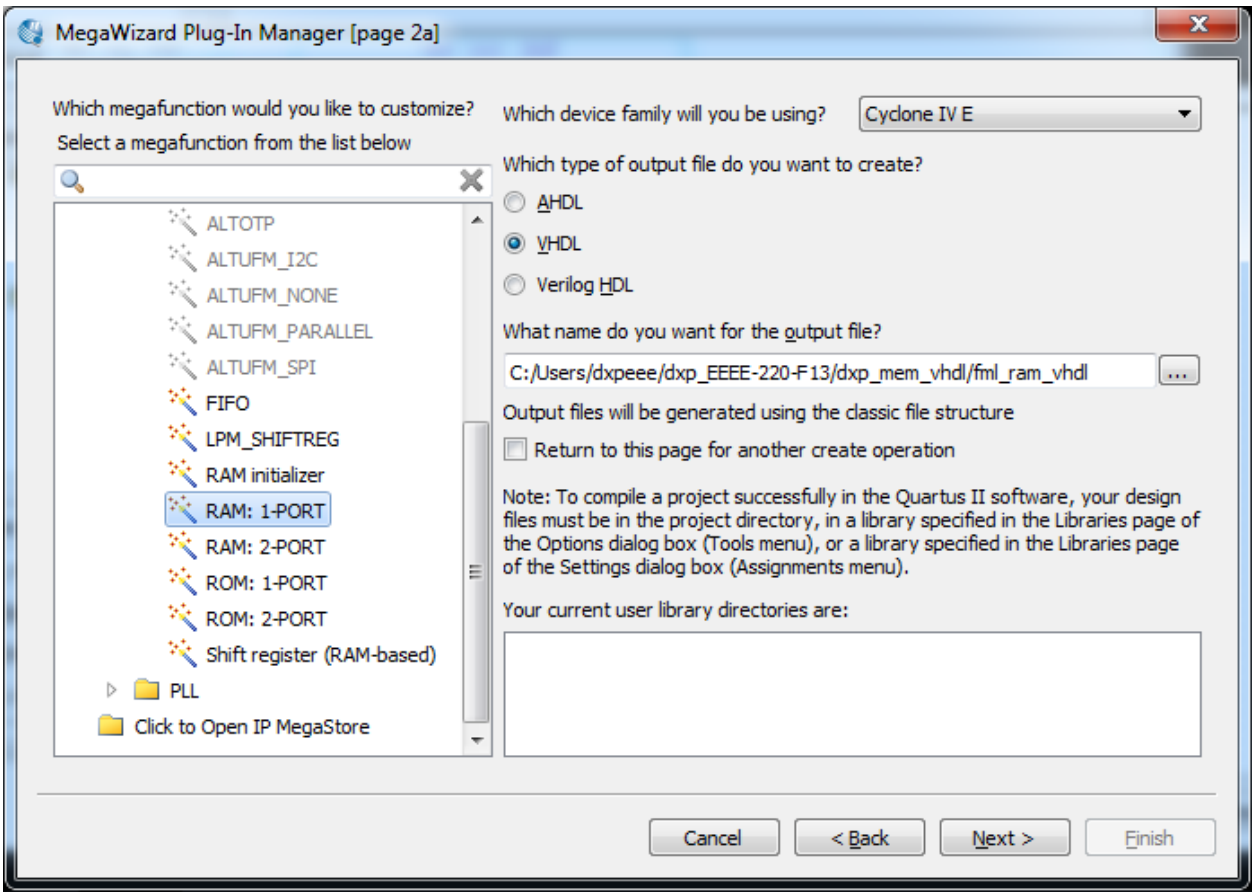


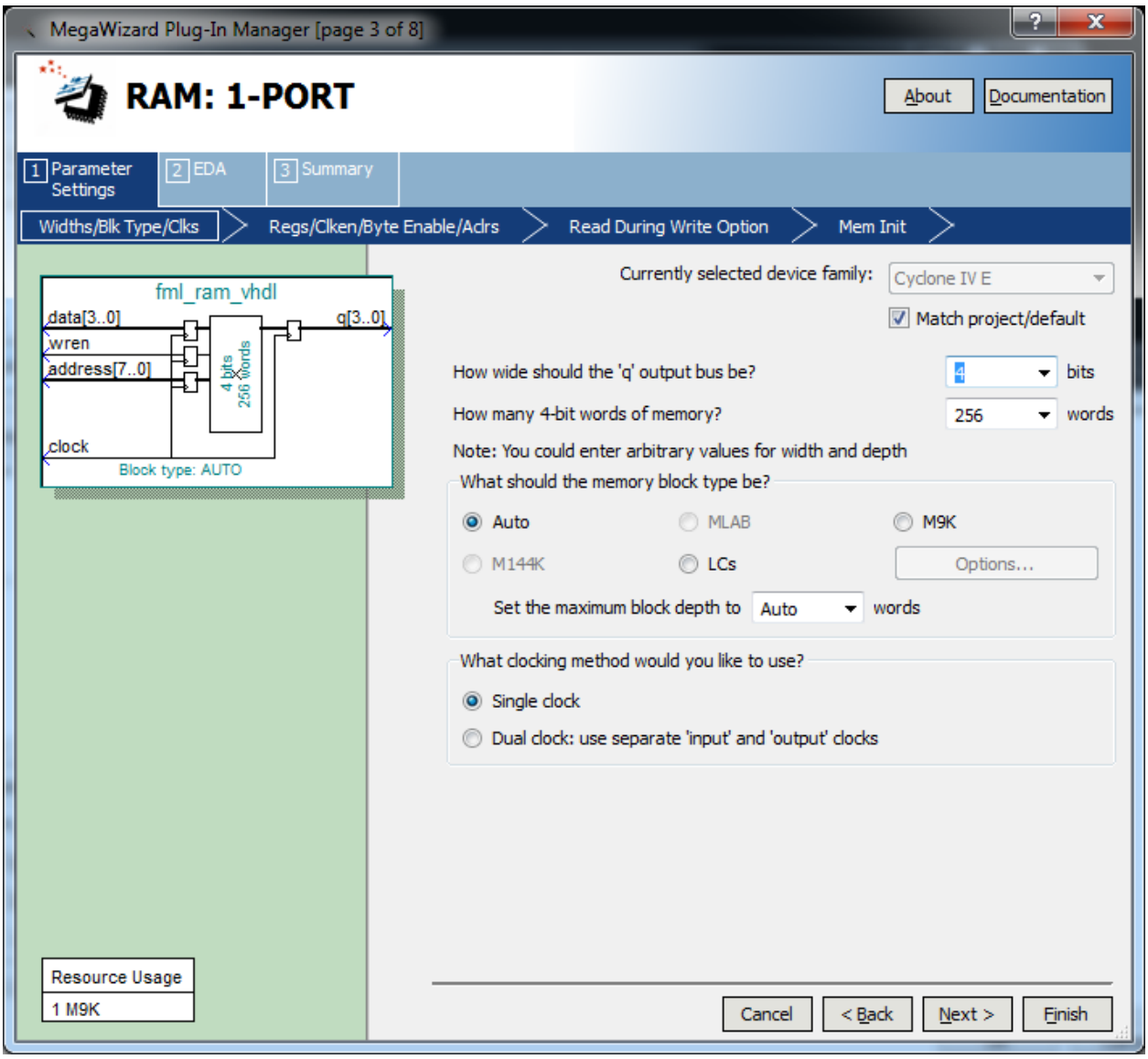
45) Drop an instance of it in the schematic. We will not use it, but just to have as a reference.
46) Now, go to Project > Add/Remove Files in Project. Browse to the project folder, where you'll find the file fml_rom_vhdl.vhd. Select and add it to the project files.
47) Open it now within Quartus II.

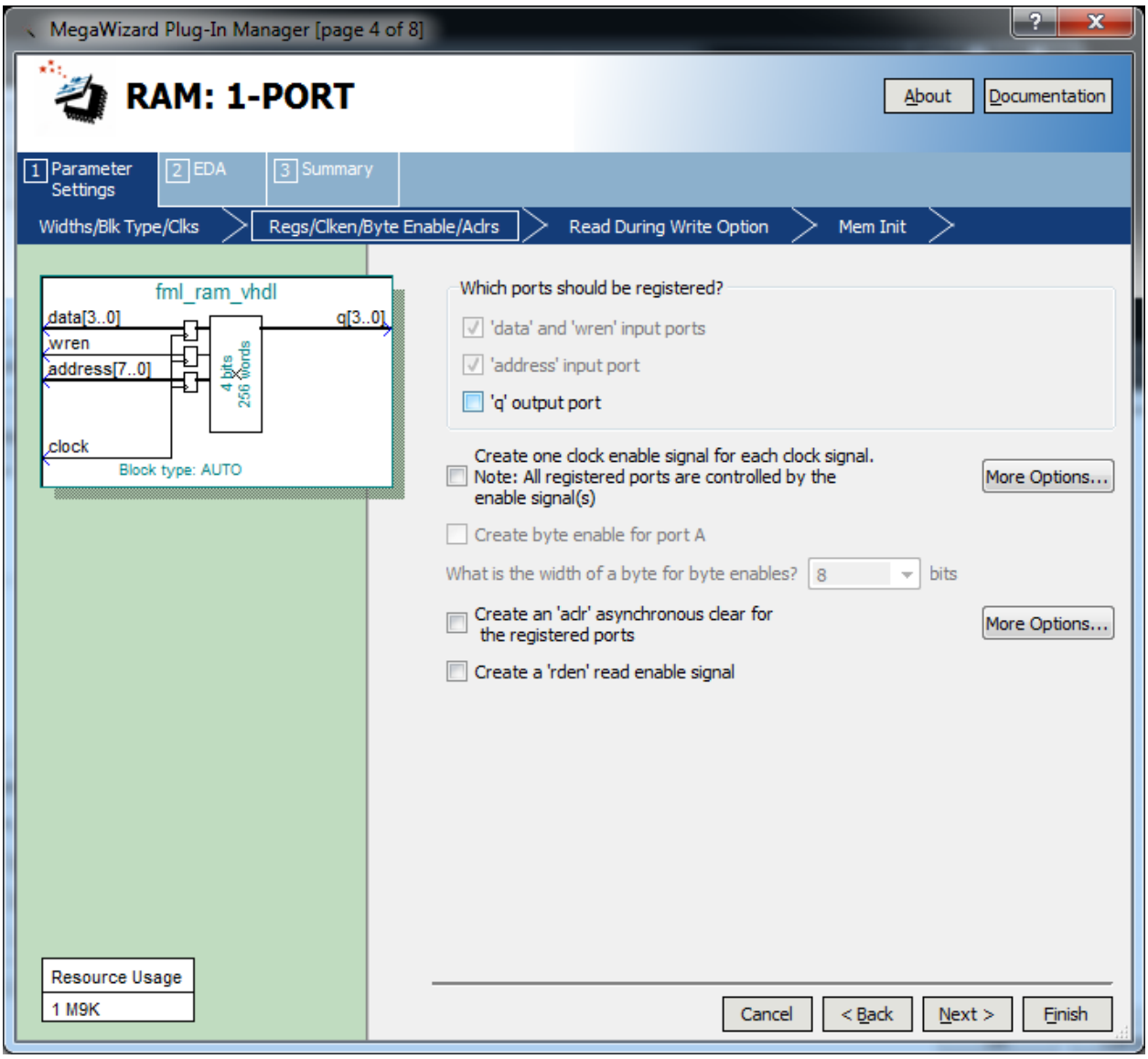


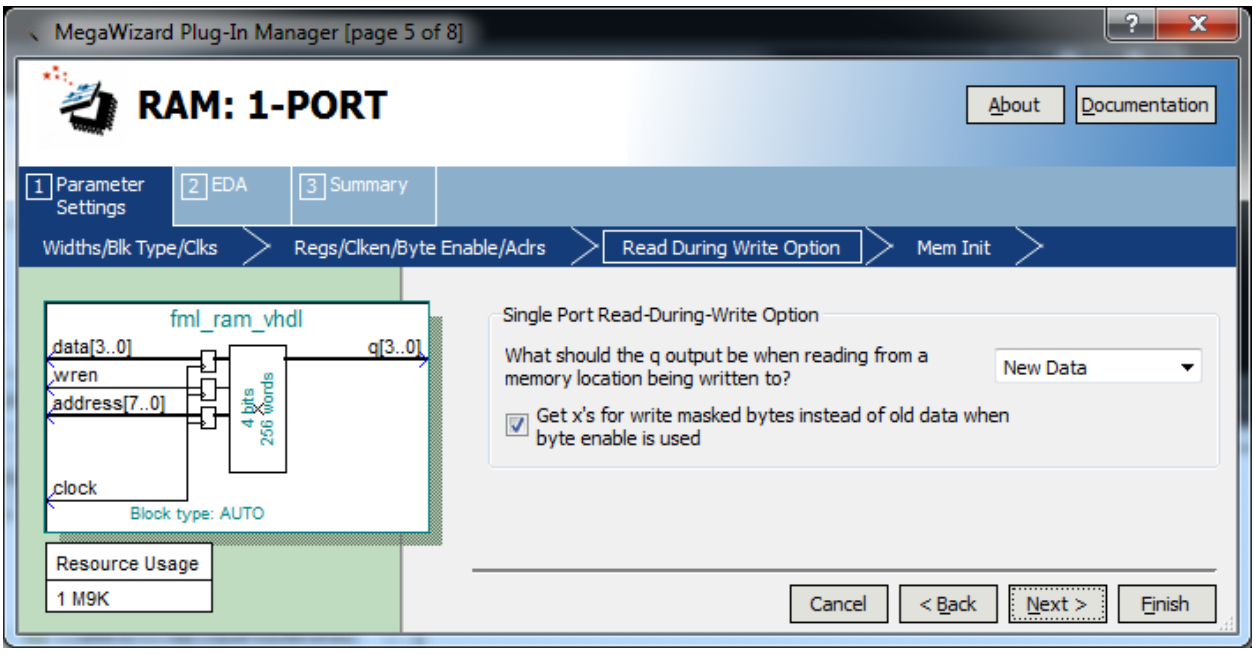
48) As you can see, it contains the entity of your custom ROM memory block. It has been generated automatically from parameterized code (remember the Generate command in VHDL). In the process of customization of the code, the compiler has taken into account the target FPGA family. Can't get a more efficient memory block code than that from this (high) level of the design hierarchy.

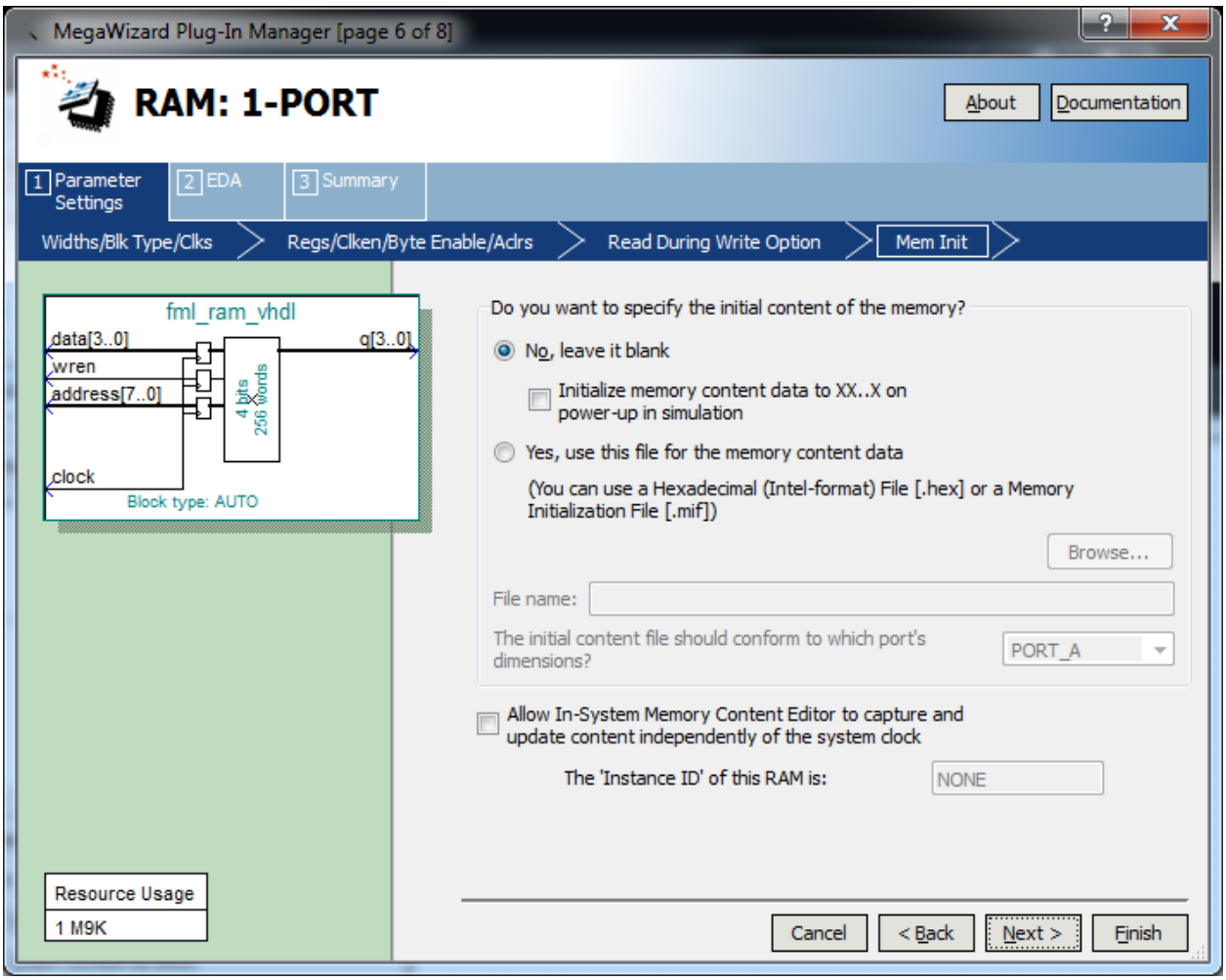
49) Now, create a similar RAM block using the mega wizard. Back to the *.bdf file, click again on the symbol tool. In the next pages I only list the relevant windows captures that also contain the necessary configuration selections.


MegaWizard Plug-In Manager [page 2a]
Which megafunction would you like to customize?
Select a megafunction from the list below
ALTOTP
ALTUFM_I2C
ALTUFM_NONE
ALTUFM_PARALLEL
ALTUFM_SPI
FIFO
LPM_SHIFTREG
RAM initializer
RAM: 1-PORT
RAM: 2-PORT
ROM: 1-PORT
ROM: 2-PORT
Shift register (RAM-based)
PLL
Click to Open IP MegaStore
Which device family will you be using?
Cyclone IV E
Which type of output file do you want to create?
AHDL
VHDL
Verilog HDL
What name do you want for the output file?
C:/Users/dxpeee/dxp_EEEE-220-F13/dxp_mem_vhdl/fml_ram_vhdl
...
Output files will be generated using the classic file structure
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 Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).
Your current user library directories are:
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager [page 3 of 8]
RAM: 1-PORT
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
Widths/Blk Type/Clks
Regs/Clken/Byte Enable/Adrs
Read During Write Option
Mem Init
fml_ram_vhdl
data[3..0]
wren
address[7..0]
clock
q[3..0]
4 bits
256 words
Block type: AUTO
Currently selected device family:
Cyclone IV E
Match project/default
How wide should the 'q' output bus be?
4
bits
How many 4-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
Resource Usage
1 M9K
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager [page 4 of 8]
RAM: 1-PORT
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
Widths/Blk Type/Clks
Regs/Clken/Byte Enable/Adrs
Read During Write Option
Mem Init
fml_ram_vhdl
data[3..0]
wren
address[7..0]
clock
q[3..0]
4 bits
256 words
Block type: AUTO
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)
More Options...
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
More Options...
Create a 'rden' read enable signal
Resource Usage
1 M9K
Cancel
< Back
Next >
Finish

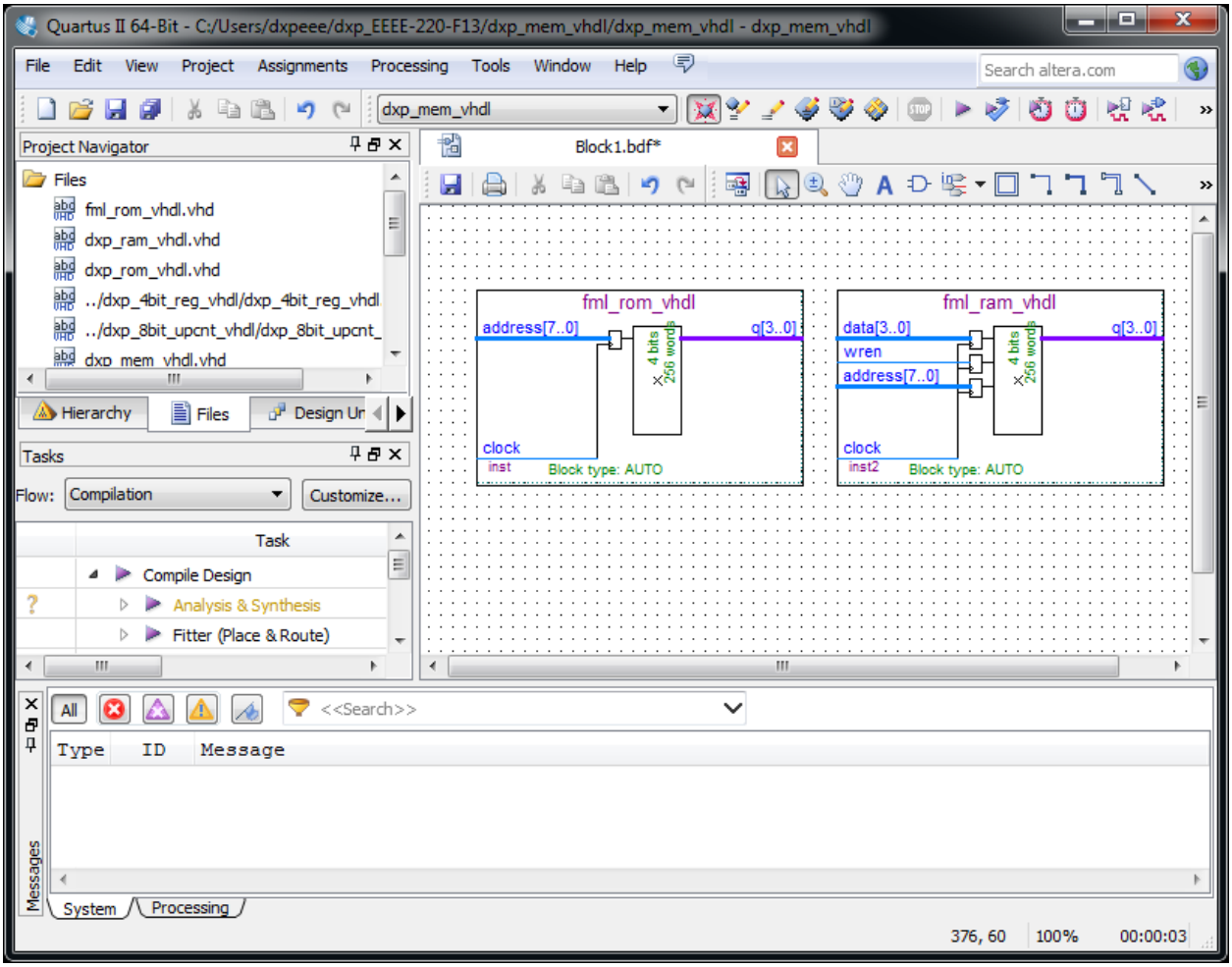

MegaWizard Plug-In Manager [page 5 of 8]
RAM: 1-PORT
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
Widths/Blk Type/Clks
Regs/Clken/Byte Enable/Adrs
Read During Write Option
Mem Init
fml_ram_vhdl
data[3..0]
wren
address[7..0]
clock
q[3..0]
4 bits
256 words
Block type: AUTO
Resource Usage
1 M9K
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
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager [page 6 of 8]
RAM: 1-PORT
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
Widths/Blk Type/Clks
Regs/Clken/Byte Enable/Adrs
Read During Write Option
Mem Init
fml_ram_vhdl
data[3..0]
wren
address[7..0]
clock
q[3..0]
4 bits
256 words
Block type: AUTO
Resource Usage
1 M9K
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])
Browse...
File name:
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:
NONE
Cancel
< Back
Next >
Finish

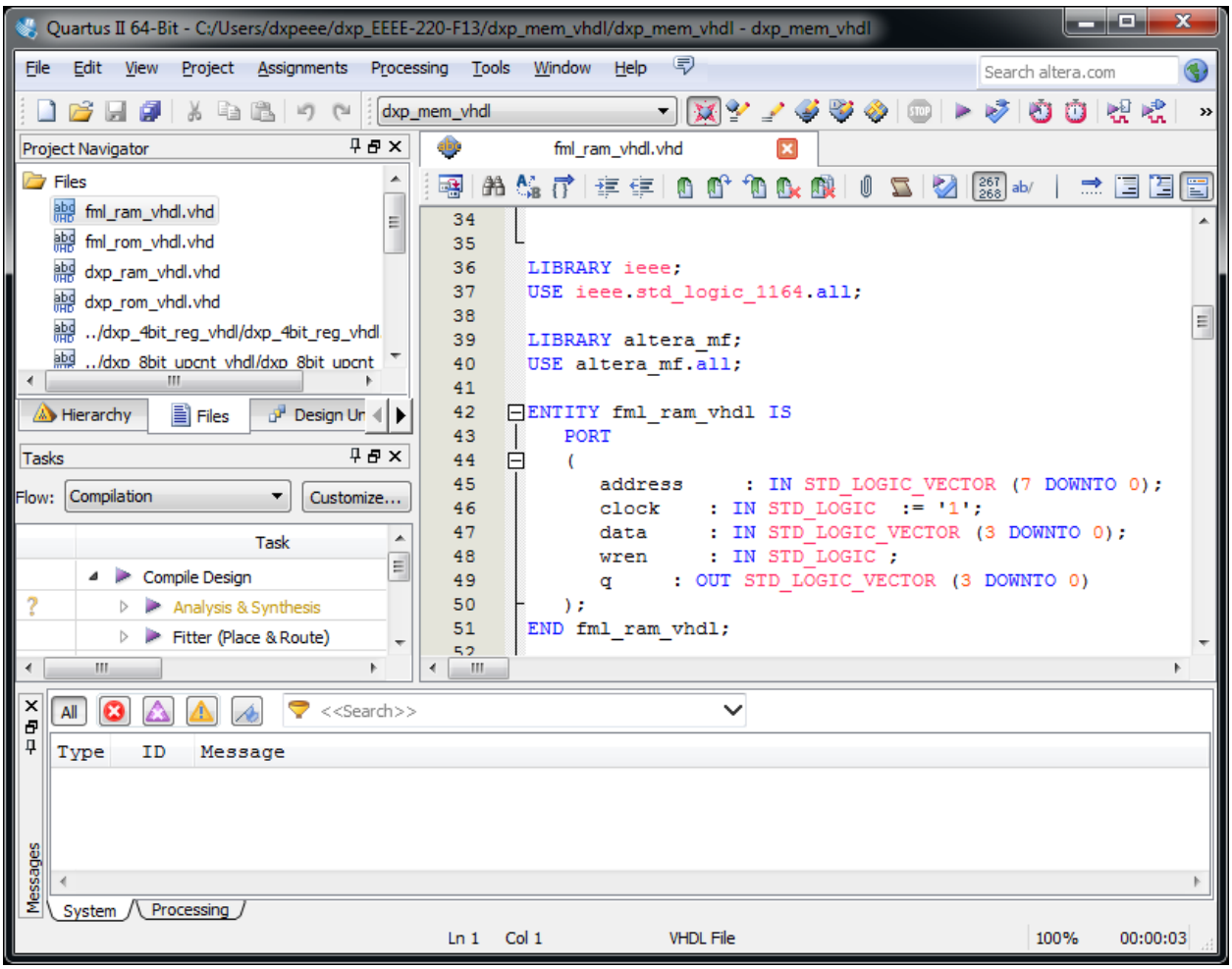
50) As you can see, this RAM we left blank, i.e. not initialized. This is the true state at which a RAM power up. However, if backed up by ROM/Flash memory cells, it can be initialized on power up. Some of you will have to do this in the processor design.

51) Add the new created fml_ram_vhdl.vhd file to the project files.

52) Your schematic may look as below now. You can save and close it. We will not need it any longer.



53) Open the fml_ram_vhdl.vhd file. Again you see the entity name fml_ram_vhdl.



54) Starting with this lab, you create a fml_package.vhd. This will contain the declaration of all useful components you have created so far. Create one and add it to the project based on the template dxp_package.vhd, which you can download from mycourses.

55) Now you are ready to create the structural design of your top-level entity fml_mem_vhdl. Follow previous structural code examples to do this.

56) It is strongly recommended to use the same port names as in the schematic above! This will ease the use of the template testbench.

57) Once the top-level entity code is complete and compiles without errors, download from mycourses the testbench: dxp_mem_vhdl_tb.vhd. Modify fml as necessary.

58) If you kept the same port names and naming convention, you should be able to run the testbench on your circuit without any other modifications. Make sure you increase the loop iteration number to a value that will read and write out all characters in your name.

59) Take the time to read the testbench code and use the comments and the code itself to make sense of it. You will need this understanding later on.

60) In the waveform, change the radix of Q_out to ASCII. Now, in the second half of it you should see your name written out. **Why does the right letter show up every other cycle? Explain this in your report.**

61) This concludes part 5.

62) Part 6: In this part we design and verify the circuit in part 5 using Verilog code.

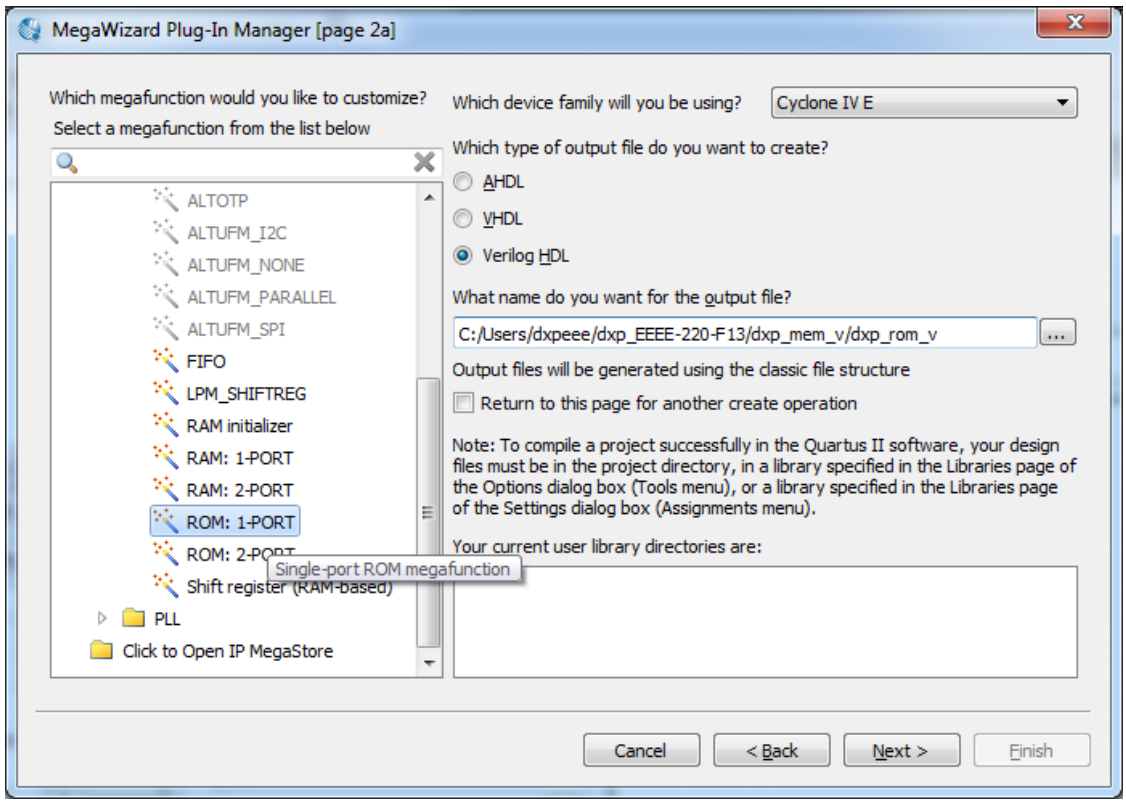
63) Create a project and to level entity with the name: fml_mem_v. Add to this project the files fml_4bit_reg_v.v and fml_8bit_upcnt_v.v.

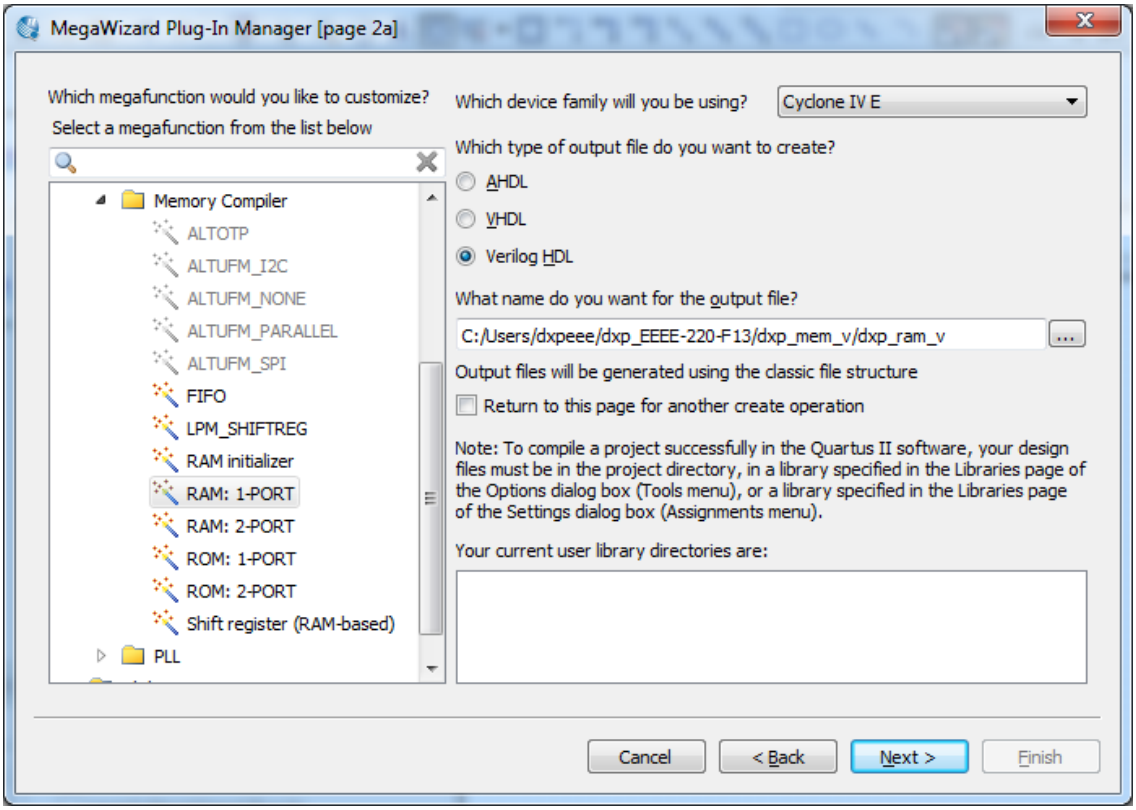
64) Copy using Windows Explorer the file fml_rom_vhdl.mif. Rename it fml_rom_v.mif. You don't need to change anything else in this file.

65) Create a new block diagram/schematic file as before. As before, this is used as a placeholder for the memory blocks that you will generate next.

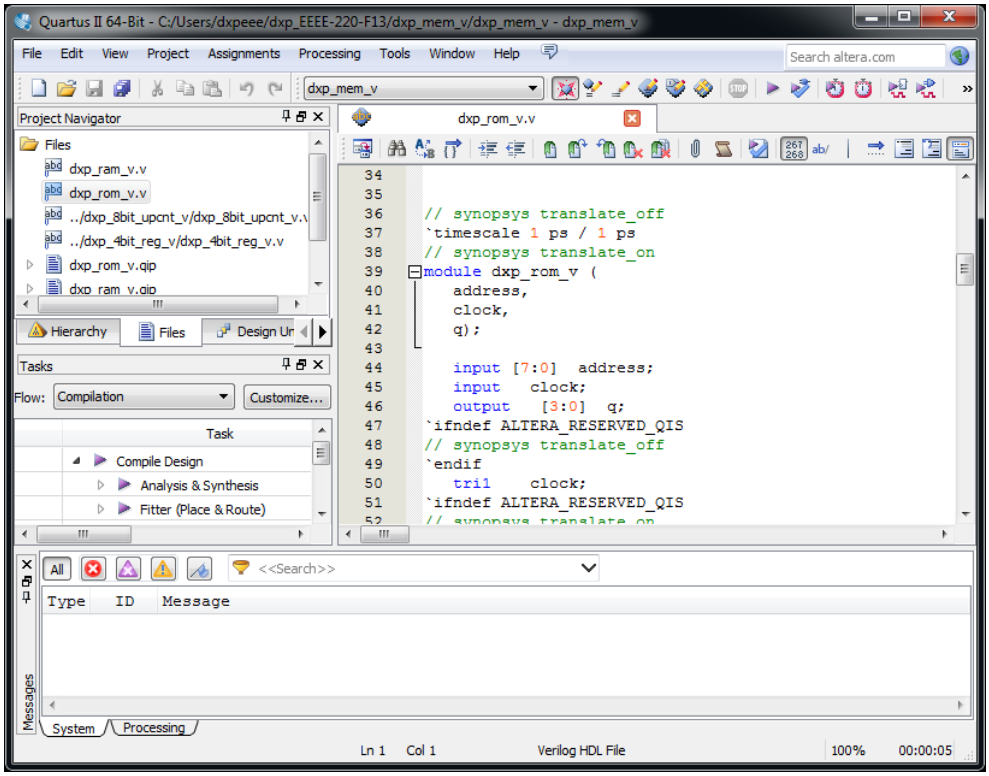
66) Following the same steps as in part 5, create and add to the project fml_rom_v.v and fml_ram_v.v.

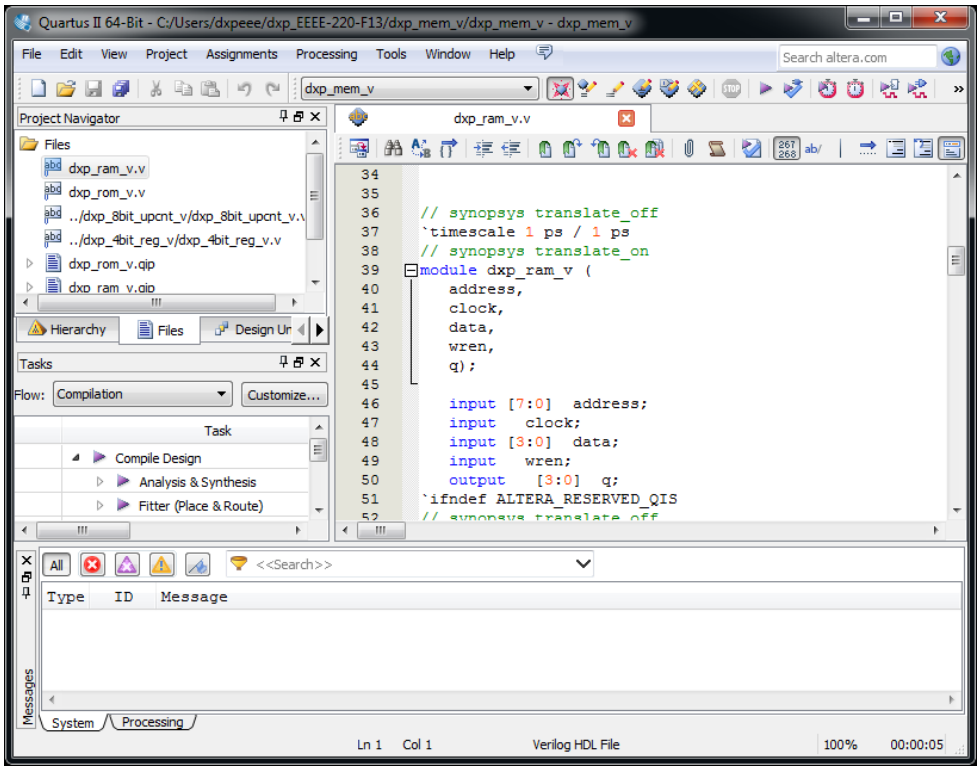
67) Very Important: select Verilog HDL as the language of the output file!


MegaWizard Plug-In Manager [page 2a]
Which megafunction would you like to customize?
Select a megafunction from the list below
ALTOTP
ALTUFM_I2C
ALTUFM_NONE
ALTUFM_PARALLEL
ALTUFM_SPI
FIFO
LPM_SHIFTREG
RAM initializer
RAM: 1-PORT
RAM: 2-PORT
ROM: 1-PORT
ROM: 2-PORT
Shift register (RAM-based)
PLL
Click to Open IP MegaStore
Single-port ROM megafunction
Which device family will you be using?
Cyclone IV E
Which type of output file do you want to create?
AHDL
VHDL
Verilog HDL
What name do you want for the output file?
C:/Users/dxpeee/dxp_EEEE-220-F13/dxp_mem_v/dxp_rom_v
Output files will be generated using the classic file structure
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 Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).
Your current user library directories are:
Cancel
< Back
Next >
Finish


MegaWizard Plug-In Manager [page 2a]
Which megafunction would you like to customize?
Select a megafunction from the list below
Memory Compiler
ALTOTP
ALTUFM_I2C
ALTUFM_NONE
ALTUFM_PARALLEL
ALTUFM_SPI
FIFO
LPM_SHIFTREG
RAM initializer
RAM: 1-PORT
RAM: 2-PORT
ROM: 1-PORT
ROM: 2-PORT
Shift register (RAM-based)
PLL
Which device family will you be using?
Cyclone IV E
Which type of output file do you want to create?
AHDL
VHDL
Verilog HDL
What name do you want for the output file?
C:/Users/dxpeee/dxp_EEEE-220-F13/dxp_mem_v/dxp_ram_v
Output files will be generated using the classic file structure
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 Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).
Your current user library directories are:
Cancel
< Back
Next >
Finish

68) Open and check the module definitions:





69) In the fml_rom_v.v file you need to change the following line to the next one below, i.e. need to provide the entire path to your fml_rom_v.mif file. Otherwise, ModelSim will not find it – ModelSim bug.

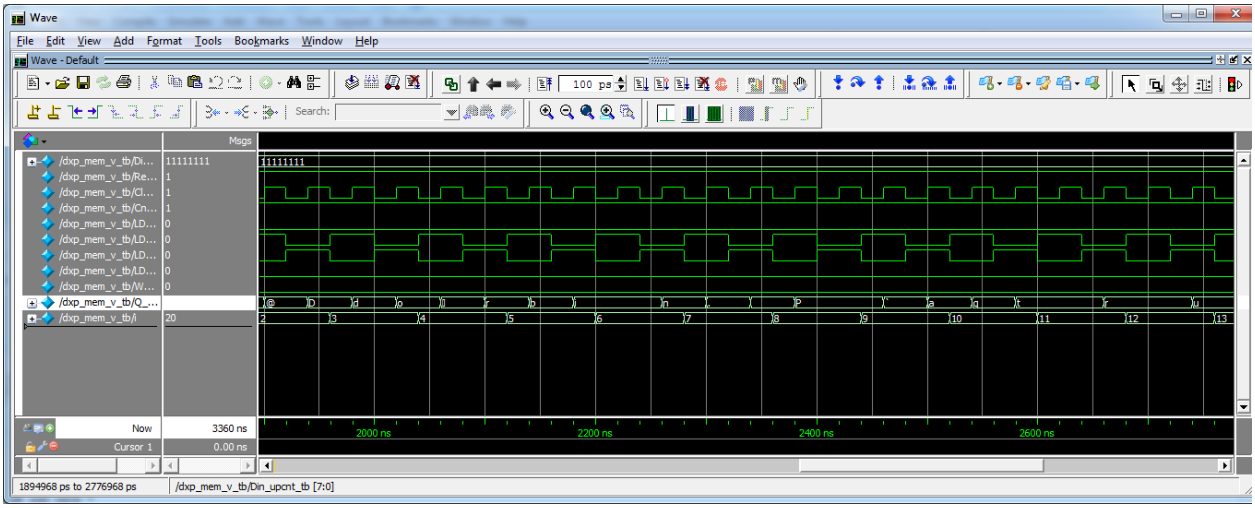
This:

altsyncram_component.init_file = "../dxp_mem_v/dxp_rom_v.mif",

Change to this:

altsyncram_component.init_file = "C:/Users/dxpeee/dxp_EEEE-220-F13/dxp_mem_v/dxp_rom_v.mif",

1) Once the top-level entity code is complete and compiles without errors, download from mycourses the testbench: dxp_mem_v_tb.v. Modify fml as necessary.
2) If you kept the same port names and naming convention, you should be able to run the testbench on your circuit without any other modifications. Make sure you increase the loop iteration number to a value that will read and write out all characters in your name.
3) Take the time to read the testbench code and use the comments and the code itself to make sense of it. You will need this understanding later on.
4) In the waveform, change the radix of Q_out to ASCII. Now, in the second half of it you should see your name written out. As before: **Why does the right letter show up every other cycle? Explain this in your report.**



5) This concludes part 6.
6) Grading:
    a. 5 points for part 1
    b. 5 points for part 2
    c. 5 points for part 3

   d. 5 points for part 4
   e. 10 points for part 5
   f. 10 points for part 6.

7) Show your working, i.e. compiled and simulated, designs to the TA. Write your report and upload it along with your project archives in the dropbox on mycourses, as described in the lab policy.
8) This concludes this week's lab.