

**Exercise Manual**  
*for*  
**Quartus II Software Design Series:  
Foundation**

**Software Requirements to complete all exercises**

Quartus<sup>®</sup> II software version 12.1

**Link to the Quartus II Handbook:**

[http://www.altera.com/literature/hb/qts/quartusii\\_handbook.pdf](http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf)

# Exercise 1

## Exercise 1

### Objectives:

- *Create a project using the New Project Wizard*
- *Name the project*
- *Pick a device*


*Note: In these exercises, you'll create a brand new project and complete an existing design. You'll have the choice of creating the design using three different types of design entry: Verilog, VHDL, or as a Quartus II schematic. Where noted, be sure to only follow the instructions appropriate for your choice of design entry method. By the end of the class, you'll have a final, optimized design, ready for programming into a Cyclone® IV E FPGA device.*

*Be sure to completely read the instructions for each step and sub-step in this lab manual. Each step first summarizes what you'll be doing in that step before providing complete instructions. Use the lines next to each step (\_\_\_\_) to keep track of your progress or to check off completed steps in the exercises.*

*If you have any questions or problems, please ask the instructor for assistance.*

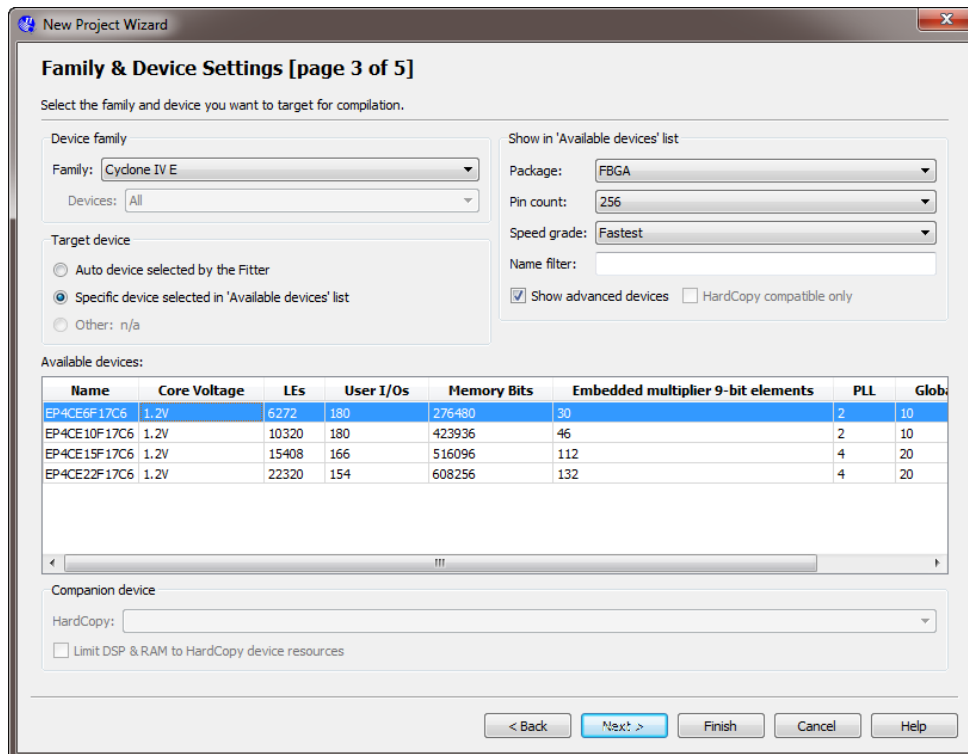
**Step 1: Create new project for use in the lab exercises**

- \_\_\_\_ 1. Unzip the lab project files. In an Explorer window, go to **C:\altera\_trn\Quartus\_II\_Software\_Design\_Series\_Foundation**. The name of the directory may be shortened to **Quartus\_II\_Foundation** or similar on some machines. This will be your lab installation directory. **Delete** any old lab file folders that may already exist there labeled **QIIF\***. Double-click the executable file found in that location or the file you downloaded from the link in your class registration email. If you still cannot find this file, ask your instructor for assistance. In the WinZip dialog box, simply click **Unzip** to automatically extract to the current directory. Close WinZip.
- \_\_\_\_ 2. Start the Quartus II software. In the Windows **Start** menu from the **All Programs** list, go to the **Altera 12.1 Build 177** folder and then the **Quartus II 12.1** folder. Depending on your operating system, click **Quartus II 12.1 (64-bit)** or **Quartus II 12.1 (32-bit)** to start the program. Check with your instructor if unsure. There may also be a shortcut on the desktop.
- \_\_\_\_ 3. Start the New Project Wizard. You can open it from the **Getting Started With Quartus II Software** welcome dialog that appears. If you've closed this window, in the **Tasks** window on the left side of the Quartus II interface, expand the **Start Project** folder and double-click **Open New Project Wizard**. You can also select **New Project Wizard...** from the **File** menu. The New Project Wizard appears. If the **Introduction** screen appears, click **Next**.
- \_\_\_\_ 4. Select one of the working directories shown below depending on the type of design entry you want to use. Once you've make the choice of design entry method, it will be your entry method for **all** of the exercises.
  - <lab\_install\_directory>\QIIF12\_1\VHDL
  - <lab\_install\_directory>\QIIF12\_1\Verilog
  - <lab\_install\_directory>\QIIF12\_1\Schematic
- \_\_\_\_ 5. Name the project **pipemult** and leave the top level entity name **pipemult**.
- \_\_\_\_ 6. Click **Next** to advance to page 2.

7. On page 2, add the top level file to the project:
- Click the browse button . Navigate to the project directory as the **Select File** dialog box may not automatically be pointing there.
  - Select the top-level file **pipemult** (.v, .vhd, or .bdf, depending on the design entry method you chose in #4).
  - Click **Open**.
  - Click **Add** to add the file to the project.
  - Click **Next**.

*Note that this step isn't really necessary since the design file is already located in the project working directory. The new project would automatically include the design file as part of the project. Files or file directories (libraries) only need to be added on page 2 of the New Project Wizard if they are not located in the project directory. Adding the file to the project removes the warning that the file has not been added.*

8. On page 3, select **Cyclone IV E** as the **Family**. In the **Show in 'Available device' list** section, set **Package** to **FBGA**, **Pin count** to **256**, and **Speed grade** to **Fastest**. This filters the list of available devices. Select the **EP4CE6F17C6** device from the **Available devices** window.



**Family & Device Settings [page 3 of 5]**

Select the family and device you want to target for compilation.

Device family  
 Family: Cyclone IV E  
 Devices: All

Target device  
☐ Auto device selected by the Fitter  
☒ Specific device selected in 'Available devices' list  
☐ Other: n/a

Show in 'Available devices' list  
 Package: FBGA  
 Pin count: 256  
 Speed grade: Fastest  
 Name filter:  
☒ Show advanced devices ☐ HardCopy compatible only

Available devices:

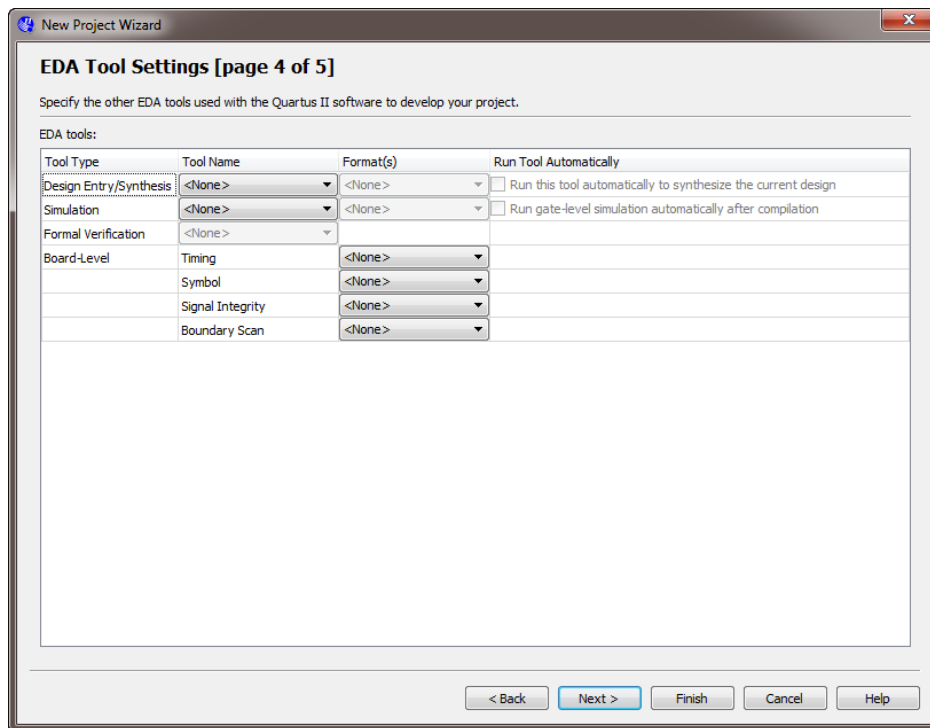
Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL	Global
EP4CE6F17C6	1.2V	6272	180	276480	30	2	10
EP4CE10F17C6	1.2V	10320	180	423936	46	2	10
EP4CE15F17C6	1.2V	15408	166	516096	112	4	20
EP4CE22F17C6	1.2V	22320	154	608256	132	4	20

Companion device  
 HardCopy: None  
☐ Limit DSP & RAM to HardCopy device resources

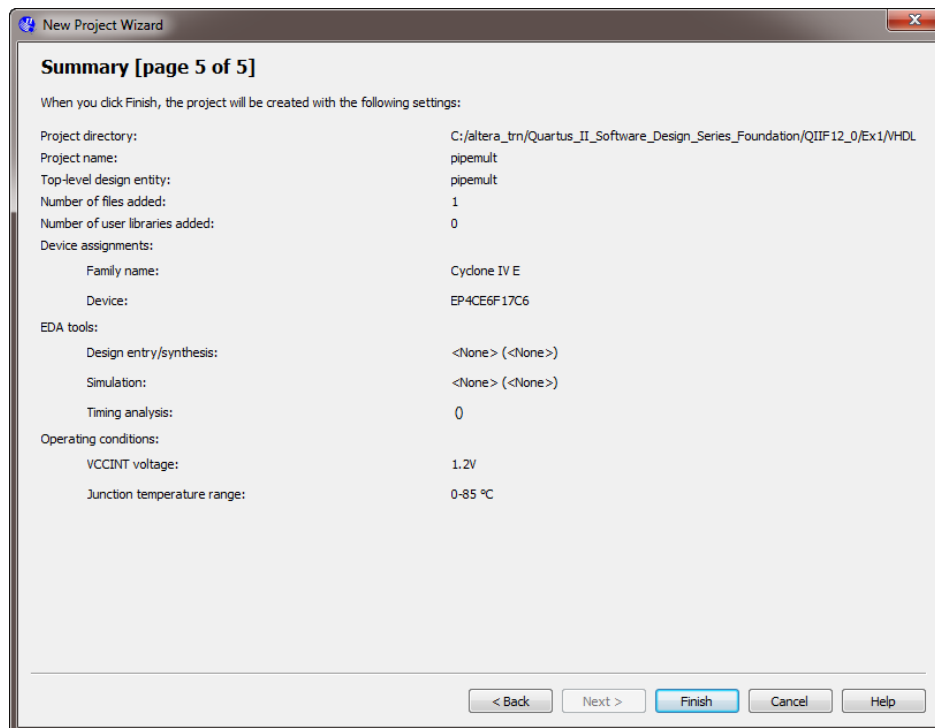
< Back Next > Finish Cancel Help

9. Click **Next**.

10. On page 4 (shown below), you can specify third-party EDA tools you may be using. Since these exercises will be done entirely within the Quartus II software without any other tools, click **Next** to skip this step.



11. The summary screen appears.



\_\_\_\_ 12. Click **Finish**.

*The project is now created.*

*Keep the project open as you continue through the exercises. There is no need to close the project. If you do close the project for some reason, be sure to select **Open Project** instead of just **Open** from the **File** menu (or **Open Existing Project** from the **Tasks** window). The **Open** command is used to simply open a single file instead of a project, preventing the ability to perform many project-based operations, such as compilation.*

### Exercise Summary

- Created a project using the New Project Wizard
  - *Named the project*
  - *Picked a device*

## END OF EXERCISE 1

# Exercise 2



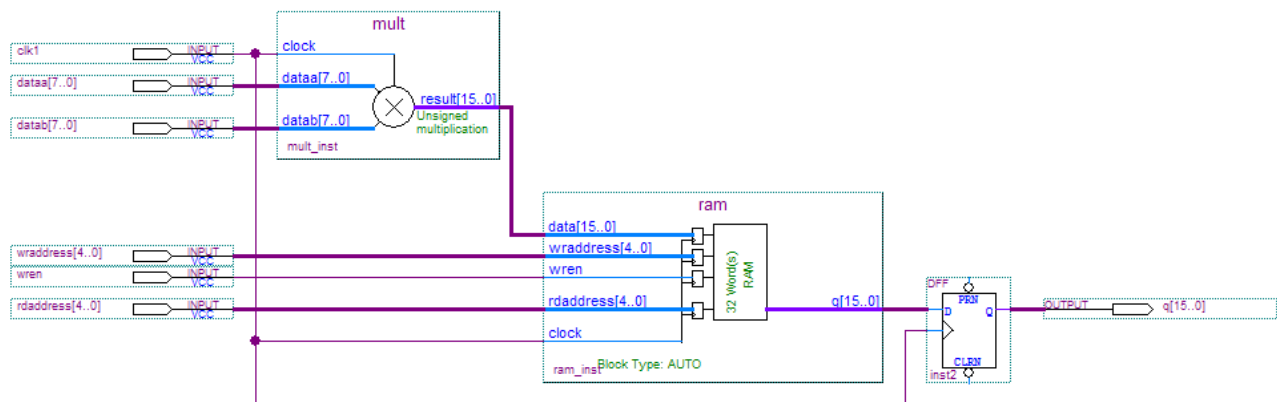
## Exercise 2

### Objectives:

- Create a multiplier and RAM block using the MegaWizard® Plug-in Manager to complete the design
- Create a HEX file to initialize the RAM block using the Memory Editor
- Analyze and elaborate the design to check for errors

### Pipelined Multiplier Design

Figure 1 shows a schematic representation of the top-level design file you will be using today. It consists of a multiplier and a RAM block. Data is fed to the multiplier from an external source and stored in the RAM block, which is also controlled externally. The data is then read out of the RAM block by a separate address control.



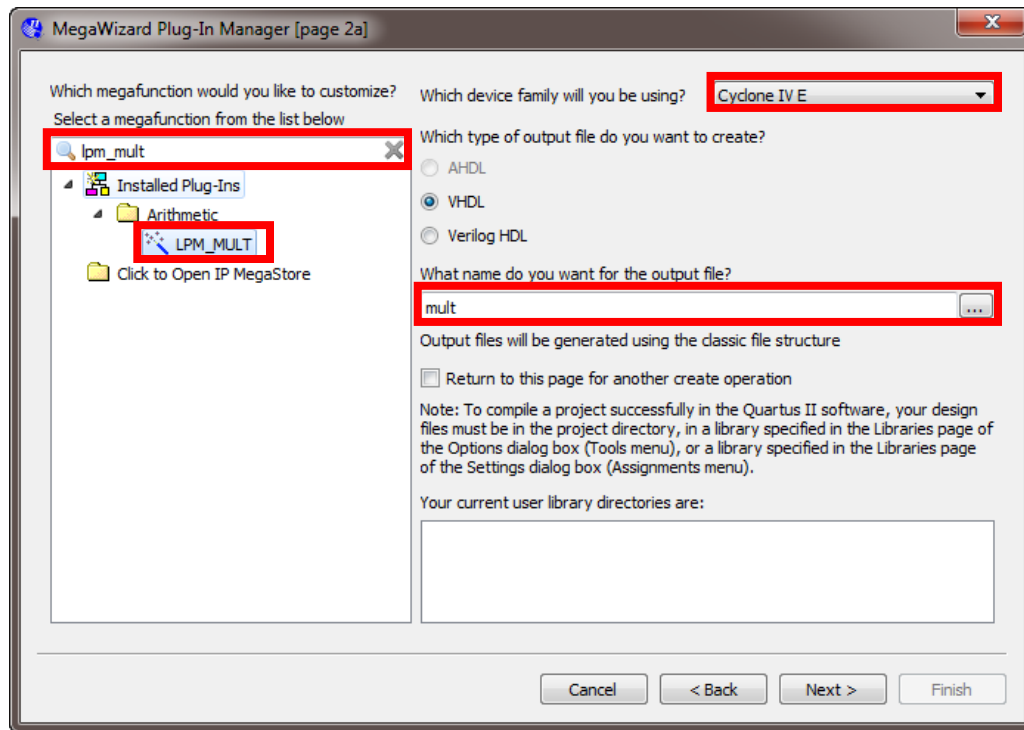
**Figure 1**

**IMPORTANT NOTE:** For exercises 2-5, you should continue working from the files of the previous exercise. The **Solutions** directory contains a Word document with the answers to questions asked in the exercises as well as the final project as it would be set up at the end of exercise 5.

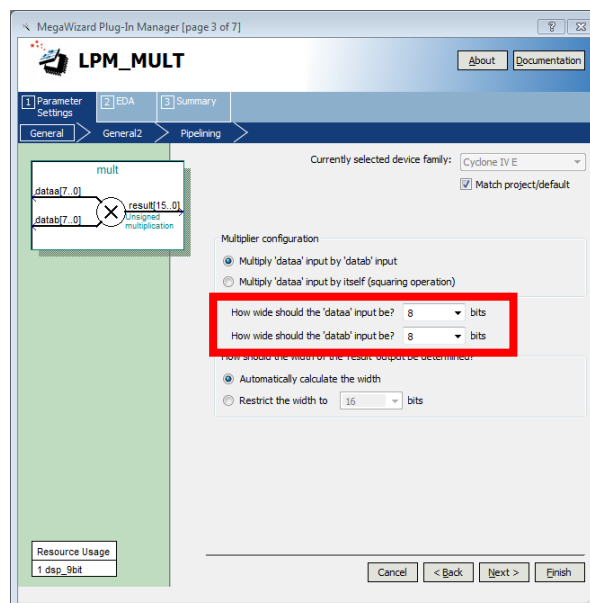
**Step 1: Build an 8x8 multiplier using the MegaWizard Plug-in Manager**

- \_\_\_\_ 1. Start the MegaWizard plug-in manager and create a new megafunction:
  - a. Choose **Tools** ⇒ **MegaWizard Plug-In Manager** or double-click **MegaWizard Plug-In Manager** in the **Create Design** folder of the **Tasks** window.
  - b. In the window that appears, select **Create a new custom megafunction variation**.
  - c. Click **Next**.
- \_\_\_\_ 2. Select the megafunction to create. On **page 2a** (shown below), do the following:
  - a. Type **LPM\_MULT** into the search bar.
  - b. Click on **LPM\_MULT** under **Arithmetic**.
  - c. In the drop-down menu, make sure the **Cyclone IV E** device family is selected.

*The selection of a device family here lets the MegaWizard Plug-In Manager know what device resources are available as the megafunction is created. You could change the device family if you wanted to create the same megafunction but for a different project that uses a different device.*
  - d. Choose **VHDL** or **Verilog HDL** output depending on your choice of HDL and exercise directory. If you are using the **Schematic** exercise, choose either VHDL or Verilog.
  - e. For the name of the output file, type **mult**. You can add this to the end of the directory path or erase the entire path to automatically place the generated megafunction files in the project directory.



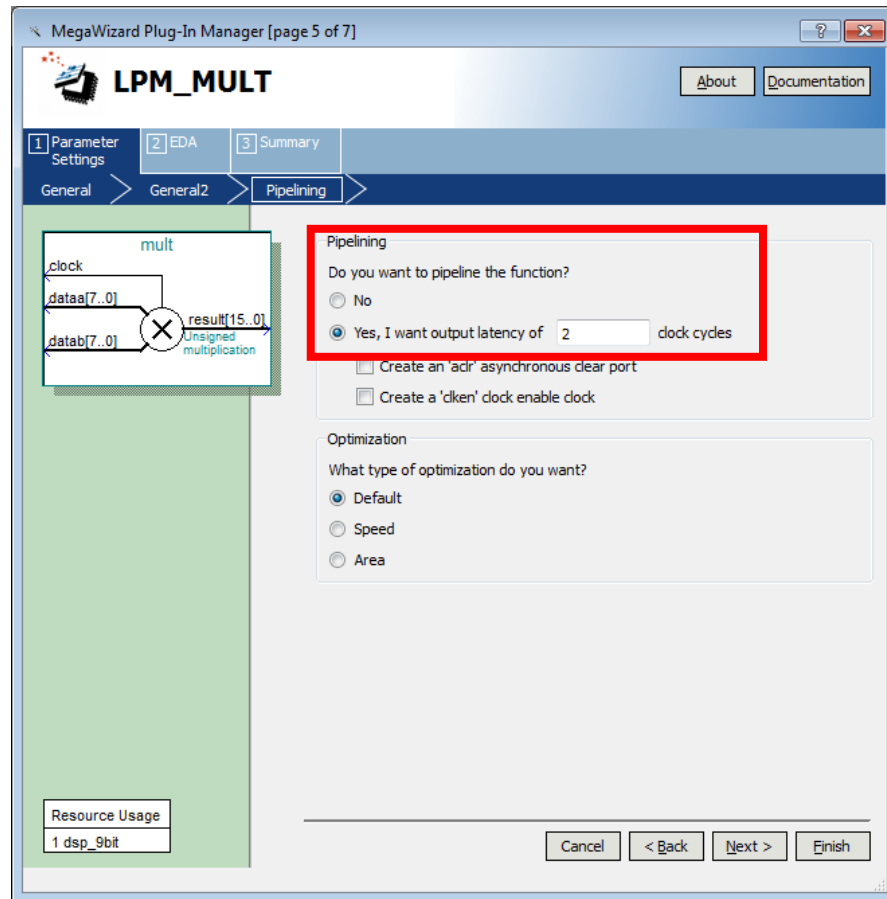
- \_\_\_ 3. Click **Next**.
- \_\_\_ 4. On **page 3 (General)**, set the width of the **dataa** and **datab** buses to **8** bits if they are not already set. For the remaining settings in this window, use the defaults that appear.



- \_\_\_ 5. Click **Next**.
- \_\_\_ 6. On **page 4 (General 2)**, use all the default settings (i.e. **datab** input does NOT have a constant value, use unsigned multiplication, and select the default multiplier implementation).

\_\_\_\_ 7. Click **Next**.

\_\_\_\_ 8. On **page 5 (Pipelining)**, choose **Yes, I want an output latency of 2 clock cycles**.

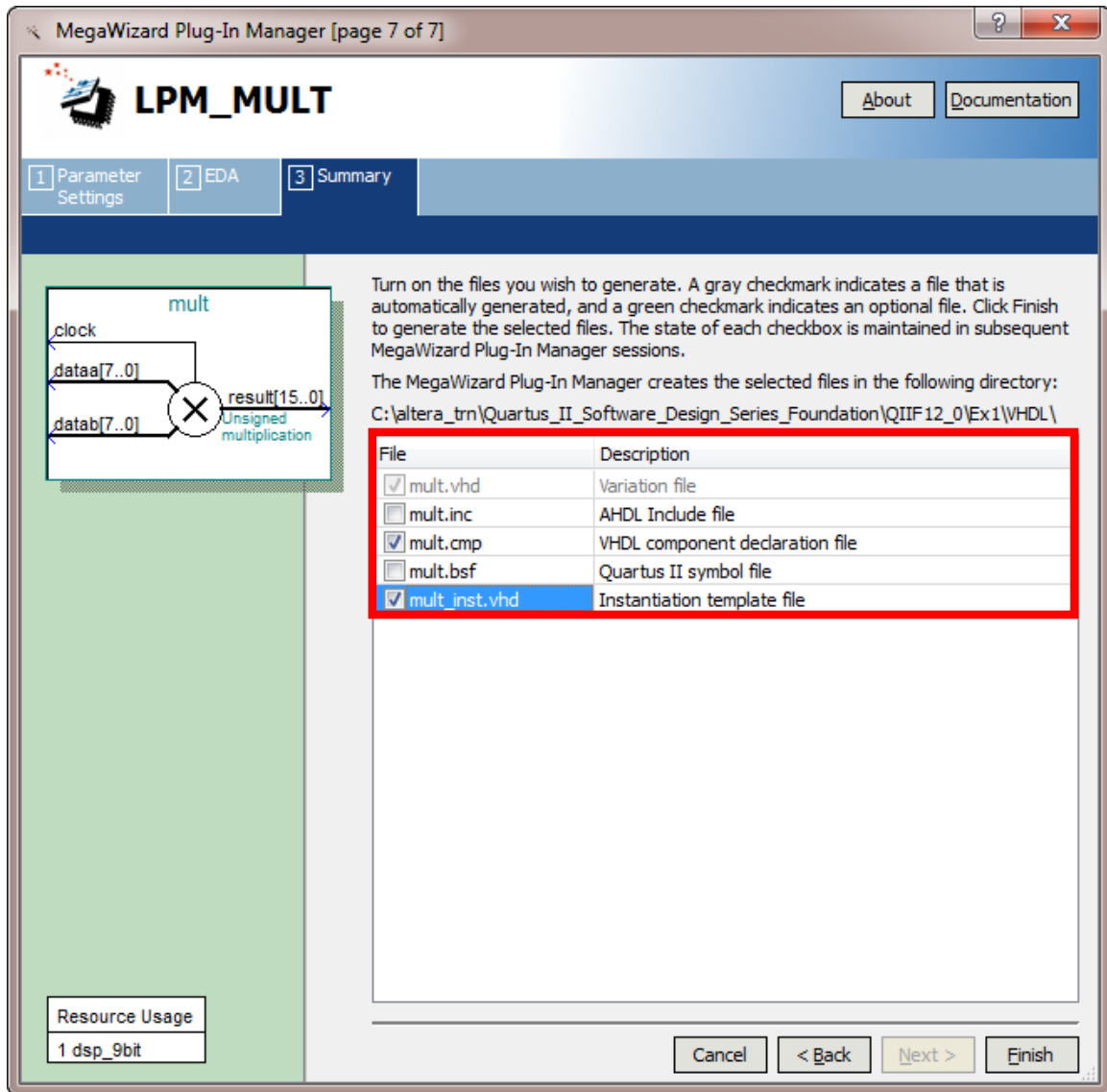


\_\_\_\_ 9. Click **Next**.

You should now be on **page 6** (section 2 of the **MegaWizard Plug-In Manager** called **EDA**). This tab indicates the simulation model file needed to simulate **LPM\_MULT** in an EDA simulation tool (e.g. **ModelSim simulator** or some other 3<sup>rd</sup>-party simulation tool). The **lpm** simulation model file should be indicated as shown above. You also have the option of generating a timing and resource estimation netlist for use by a 3<sup>rd</sup>-party synthesis tools.

\_\_\_\_ 10. We are not using any third-party tools, so just click **Next**.

11. On **page 7**, using **Table 2** below check the appropriate boxes depending on the **Design Entry Method** selected.



Design Entry Method	Files to Enable in MegaWizard Plug-In Manager
VHDL	mult_inst.vhd & mult.cmp
Verilog	mult_inst.v
Schematic	mult.bsf


Table 2. MegaWizard files to generate

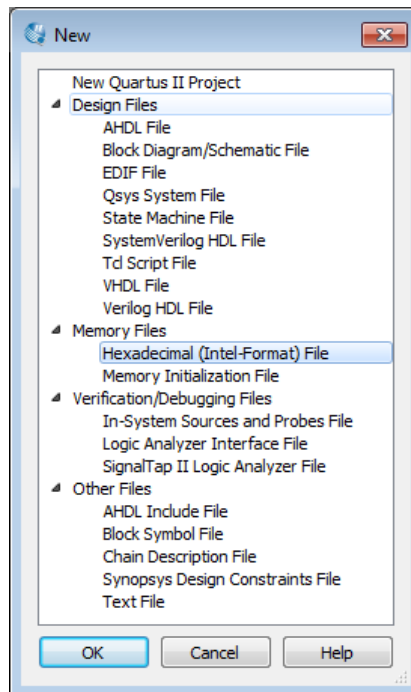
12. Click **Finish** to create the megafunction. If a dialog box appears asking if you want to add the QIP file to the Quartus II project, click **Yes**.

*The multiplier is built.*

*If for some reason your megafunction is incorrect or you forgot or missed a checkbox for generating all the required output files, open the MegaWizard Plug-In Manager again from the **Tools** menu or **Tasks** window. Select to edit an existing megafunction. Then select the main variation file for the megafunction (mult.v or mult.vhd), and go through the pages of the MegaWizard again (skip around with the tabs at the top) to fix your mistakes or generate the missing file(s). Click **Finish** to update the megafunction files.*

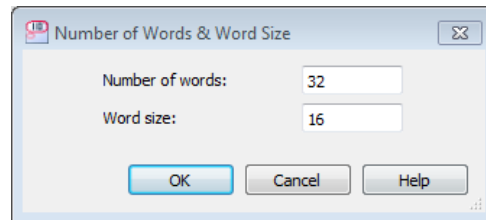
## Step 2: Create HEX file using the Memory Editor

1. From the **Tasks** window, in the **Create Design** folder, double-click **Create New Design File**. (You may have to change the Tasks Flow to **Full Design**.) You could also go to the **File** menu and select **New** or click  in the toolbar.
2. In the **New** dialog box, expand the **Memory Files** category and select **Hexadecimal (Intel-Format) File**. (This hex file will be used to initialize a dual port RAM we will be creating in the next step)

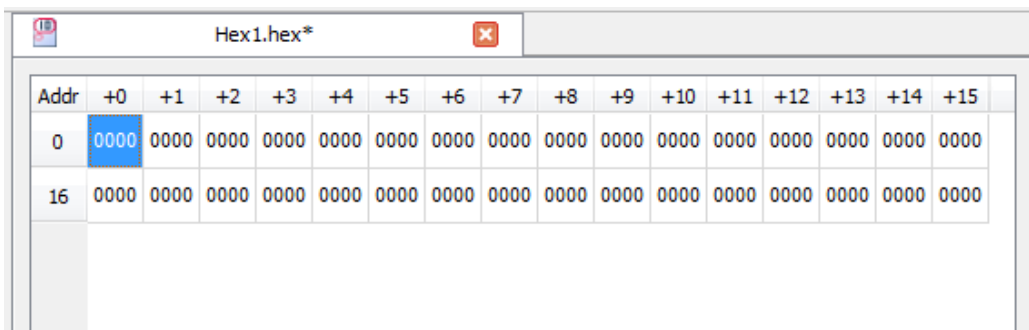


3. Click **OK**.

- \_\_\_\_ 4. In the memory size dialog box, choose **32** as the **number of words** and **16** as the **word size**.



- \_\_\_\_ 5. Click **OK**.

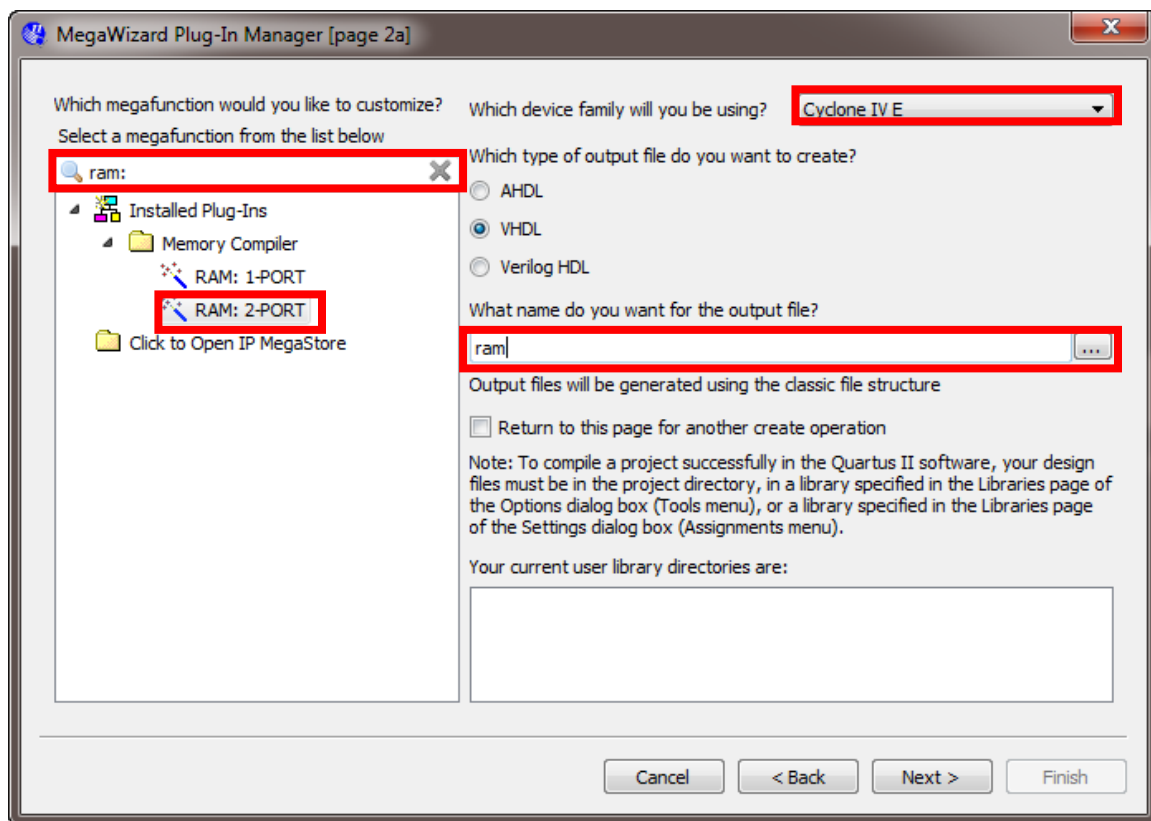


The **Memory Editor** now displays your memory space. If your memory space is not displayed exactly as above, you can change the number of cells per row (**View** menu) to **16**, the memory radix (**View** menu) to **Hexadecimal**, and the address radix to **Decimal**.

- \_\_\_\_ 6. Highlight all of the memory locations in your memory space. Right-click and select **Custom Fill Cells**.
- \_\_\_\_ 7. Use the **Custom Fill Cells** dialog box to enter your own values to initialize your memory. You can enter any values you want. Do one of the following:
- Repeating Sequence: Enter a series of numbers separated by commas or spaces to be repeated in memory.
  - Incrementing/Decrementing: Enter a start value and another value by which to increment or decrement the start value.
- \_\_\_\_ 8. Save the file as **ram.hex** in the project directory (you may have to navigate to the project directory again before you save).
- \_\_\_\_ 9. Close ram.hex.

**Step 3: Create a 32x16 RAM using the MegaWizard Plug-In Manager**

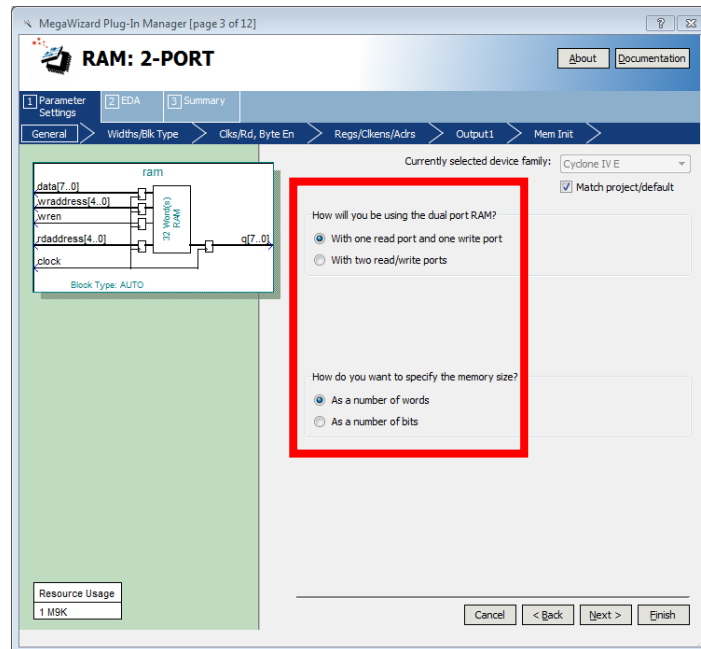
1. Open the MegaWizard Plug-In Manager again (**Tools** or **Tasks** window ⇒ **MegaWizard Plug-In Manager**). Select to **Create a new custom megafunction variation**, and click **Next**.
2. Select the megafunction to create. On **page 2a** (shown below), do the following:
  - a. In the search bar type **ram:** (be sure to include the colon).
  - b. Select **RAM: 2-PORT** under **Memory Compiler**.
  - c. As before, choose the **Cyclone IV E** device family and **VHDL** or **Verilog HDL**.
  - d. For the name of the **output file**, enter **ram**.



3. Click **Next**.



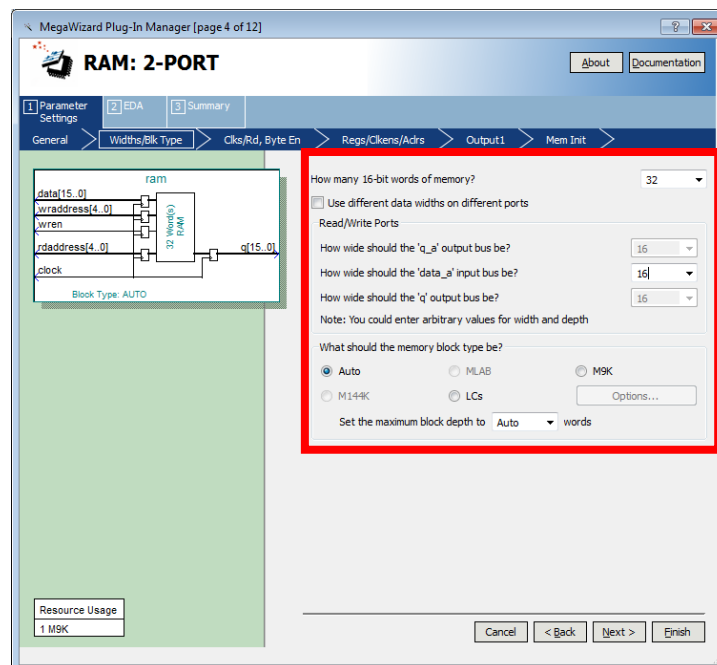
4. On **page 3**, select **With one read port and one write port** for the **Dual-port RAM** mode. Select memory size by **words**.



5. Click **Next**.

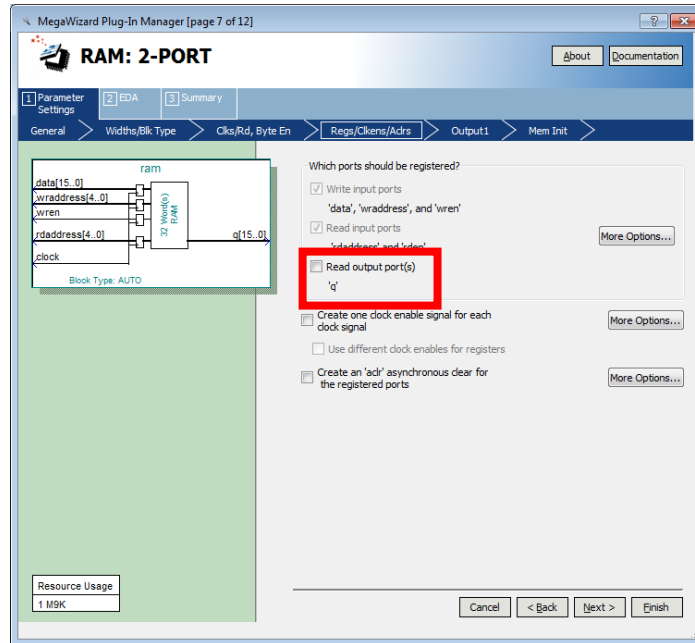
6. On **page 4 (Widths/Blk Type)**, set the width of the **data\_a** bus to **16** and the number of 16-bit words to **32**.

7. Select the **Memory block type** to **Auto** and **Max depth** to **Auto**.

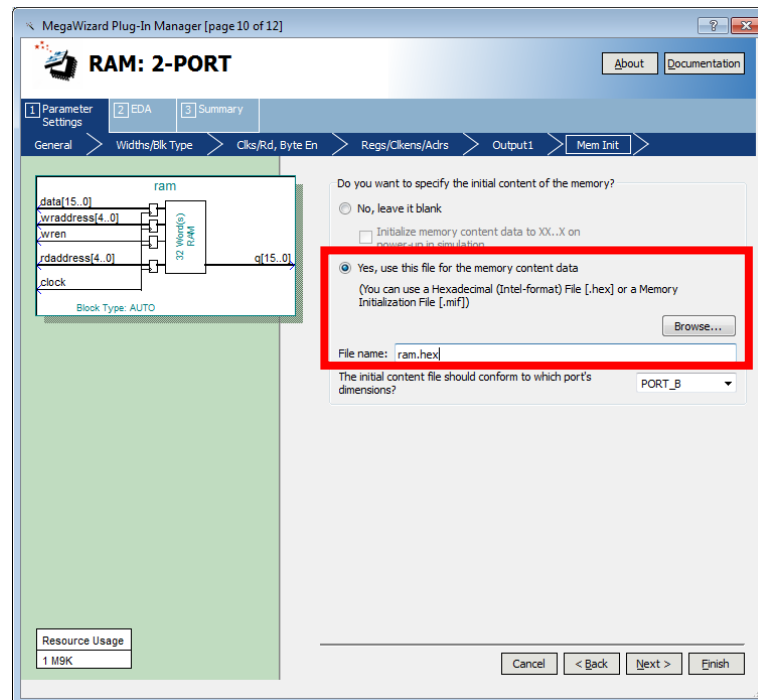


8. Click **Next** twice.

9. On **page 7 (Regs/Clkens/Aclrs)**, disable the option to register the **Read output port(s) 'q'**. Accept the remaining default settings.



10. Click **Next** 2 times.
11. On **page 10 (Mem Init)**, click **Yes**, use **this file for the memory content data**.
12. Once enabled, **type** in the file name **ram.hex**. This is the file you created in the previous step.



13. Click **Next**.

- \_\_\_\_ 14. On **page 11**, the **altera\_mf** simulation model file is displayed as being needed to simulate this function in a 3<sup>rd</sup>-party EDA simulation tool. Click **Next**.
- \_\_\_\_ 15. On page 12, choose the same files to generate for **ram** as you selected for **mult** earlier (Step 1, #11).
- \_\_\_\_ 16. Click **Finish** to close the wizard.
- \_\_\_\_ 17. If asked, click **Yes** to add the Quartus II IP file to the project.


*You have now created the two components needed for this design.*

#### Step 4: Instantiate and connect design blocks according to design entry method

Choose **ONE** of the following procedures based on your design entry method (VHDL, Verilog, or Schematic). Follow only the directions for your selected design entry method and then proceed to **Step 5**.

##### VHDL

*Perform these instructions only if you are using VHDL.*

- \_\_\_\_ 1. Open **pipemult.vhd**. You can use the **Open** command from the **File** menu, click the  toolbar button, or double-click the top-level entity in the Project Navigator. You can also double-click **Open Existing Design File** in the **Create Design** folder of the **Tasks** window.

*This is the top-level file for the design. Normally, you would have to instantiate both **ram** and **mult** and connect them together. In the interest of time, the file has been almost completed for you, but it is missing the instantiation of the multiplier.*


- \_\_\_\_ 2. **Open** the file **mult.cmp**. Copy the **component** declaration from **mult.cmp** and paste it into the architecture declaration section of **pipemult.vhd** where indicated.
- \_\_\_\_ 3. Close **mult.cmp**.
- \_\_\_\_ 4. Open the file **mult\_inst.vhd**. Copy the contents of **mult\_inst.vhd** (the component instantiation) and paste into the architecture body of **pipemult.vhd** where indicated. Change the following signal names in the instantiation:

<b>clock_sig</b>	to	<b>clk1</b>
<b>dataa_sig</b>	to	<b>dataa</b>
<b>datab_sig</b>	to	<b>datab</b>
<b>result_sig</b>	to	<b>mult_to_ram</b>

- \_\_\_\_ 5. Save **pipemult.vhd**.
- \_\_\_\_ 6. Close **mult\_inst.vhd**.
- \_\_\_\_ 7. Continue to **Step 5: Check the design**.

## Verilog

*Perform these instructions only if you are using Verilog entry.*

- \_\_\_\_ 1. Open **pipemult.v**. You can use the **Open** command from the **File** menu, click the  toolbar button, or double-click the entity in the Project Navigator. You can also double-click **Open Existing Design File** in the **Create Design** folder of the **Tasks** window.

*This is the top-level file for the design. Normally, you would have to instantiate **both ram** and **mult** and connect them together. In the interest of time, the file has been almost completed for you, but it is missing the instantiation of the multiplier.*


- \_\_\_\_ 2. Open the file **mult\_inst.v**. Copy the contents of **mult\_inst.v** (the component instantiation) and paste into the body of **pipemult.v** where indicated. Change the following signal names in the instantiation: (notice how a pop up window shows you the available signal names that have already been defined)

<b>clock_sig</b>	to	<b>clk1</b>
<b>dataa_sig</b>	to	<b>dataa</b>
<b>datab_sig</b>	to	<b>datab</b>
<b>result_sig</b>	to	<b>mult_to_ram</b>

- \_\_\_\_ 3. Close **mult\_inst.v**.
- \_\_\_\_ 4. Save **pipemult.v**.
- \_\_\_\_ 5. Continue to **Step 5: Check the design**.

## Schematic

*Perform these instructions only if you are using schematic entry.*

- \_\_\_\_ 1. Open **pipemult.bdf**. You can use the **Open** command from the **File** menu, click the  toolbar button, or double-click the entity in the Project Navigator. You can also double-click **Open Existing Design File** in the **Create Design** folder of the **Tasks** window.

*This is the top-level schematic file for the design. Normally, you would have to instantiate both **ram** and **mult** sub-designs and connect them together manually. In the interest of time, the schematic file has been almost completed for you, but it is missing the **ram** and **mult** blocks and the output pins **q[15..0]**.*

2. In the schematic file, double-click any empty space in the schematic so that the **Symbol** window appears.
  - a. In the **Symbol** window, expand the **Project** folder.
  - b. Double-click the **mult** symbol. Click the left mouse button to place the symbol inside the schematic file where indicated.

*Note: The three ports on the left side of the multiplier should line up exactly with the wires coming from the input pins (no X's). If not, you may not have specified the multiplier parameters correctly when configuring the megafunction. If this is the case, hit the **Esc** key to cancel the symbol placement, reopen the MegaWizard Plug-In Manager, and select **Edit an existing megafunction variation** to open and edit the **mult** megafunction to fix the problem and regenerate the symbol.*

3. Right-click on the **mult** symbol and choose **Properties**.
  - a. In the **Symbol Properties** dialog box, change the **Instance name:** from **inst** to **mult\_inst**.
  - b. Click **OK**.


4. Double-click an open area again to reopen the **Symbol** window. This time, select the **ram** symbol and place it in the schematic file where indicated.

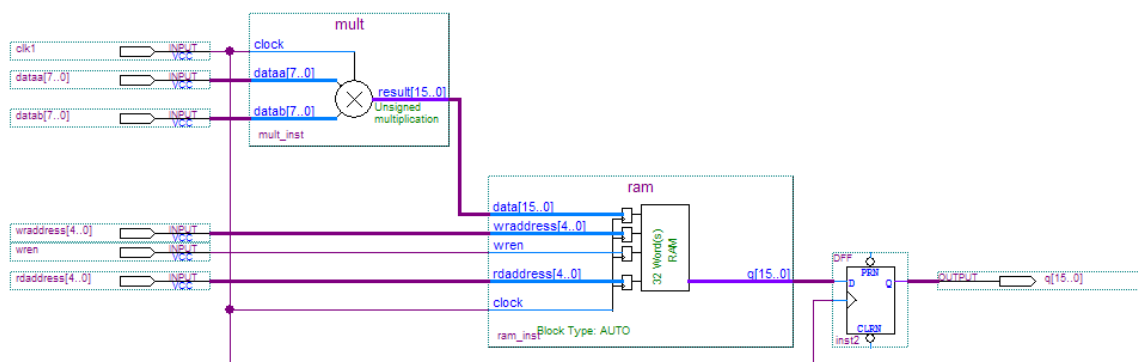
*Note: With **ram**, the lower 4 ports on the left side of the symbol should line up with the wires coming from the input pins. The data port should be unconnected.*

5. As you did with **mult**, use the **Symbol Properties** dialog box to change the name of the **ram** instantiation from **inst** to **ram\_inst**.
6. Open the **Symbol** window again and this time, as a shortcut, type **output** in the **Name:** field.

*The **Symbol** window found the output symbol automatically by name.*

7. Click **OK** and place the output pin near the output port of the **inst2** register symbol. Double-click the **pin\_name** and change it to **q[15..0]**.

8. Click on the bus drawing tool , found in the schematic editor toolbar, and draw the bus connections between **mult** (**result**) and **ram** (**data**).



*Your resulting schematic should look like the above figure.*

- \_\_\_\_ 9. Save **pipemult.bdf**.

### Step 5: Check the design

- \_\_\_\_ 1. From the **Processing** menu, select **Start ⇒ Start Analysis & Elaboration**.

*Analysis and elaboration checks that all the design files are present and connections have been made correctly. It also establishes the project hierarchy.*

- \_\_\_\_ 2. Click **OK** when analysis and elaboration is completed. If there are any errors detailed in the **Messages** window, check your connections or return to the MegaWizard Plug-In Manager for either megafunction to fix the problem (select to edit an existing megafunction instead of creating a new one). You can safely ignore any blue warnings that appear.

*Feel free to explore the Project Navigator that now contains the complete project hierarchy.*

### Exercise Summary

- Generated a multiplier and RAM using the MegaWizard Plug-In Manager and incorporated into a design
- Created a HEX file for RAM initialization using the Memory Editor
- Checked the design files using Analysis and Elaboration

## END OF EXERCISE 2

# Exercise 3

### Exercise 3

#### Objectives:


- *Perform full compilation*
- *Locate information in the Compilation Report*
- *Explore cross-probing capabilities by viewing logic in various windows*

<b>Device Name</b>	<b>EP4CE6F17C6</b>
<b>Total Design</b>	
<b>Total logic elements</b>	
<b>Total registers</b>	
<b>Total pins</b>	
<b>Total memory bits</b>	
<b>Embedded Multiplier 9-bit elements</b>	
<b>mult sub-design</b>	
<b>Logic Cells (mult)</b>	
<b>Dedicated Logic Registers (mult)</b>	
<b>DSP Elements (mult)</b>	
<b>ram sub-design</b>	
<b>Logic Cells (ram)</b>	
<b>Dedicated Logic Registers (ram)</b>	
<b>Memory Bits (ram)</b>	
<b>M9Ks (ram)</b>	
<b>Control Signals</b>	
<b>Name</b>	<b>Fanout</b>

*Table 3 – Compilation Report*




**Step 1: Compile the design**

- \_\_\_\_ 1. Select **Start Compilation** from the **Processing** menu or click  located in the toolbar to perform a full compilation of the design. You can also double-click **Compile Design** in the **Tasks** window. A dialog box will appear to indicate when the compilation is complete.
- \_\_\_\_ 2. Click **OK**.

**Step 2: Gather information from the Compilation Report)**

*The Compilation Report provides all information on design processing. You use it to understand how the compiler interpreted your design and to verify results. It is organized by compiler executables, with each one generating its own folder.*

- \_\_\_\_ 1. Open the **Compilation Report** by clicking on  from the tool bar or choose Compilation Report from the processing menu.
- \_\_\_\_ 2. From the **Flow Summary** section of the **Compilation Report**, record the **Total logic elements**, **Total memory bits**, number of **Embedded multiplier 9-bit elements** and **Total pins** in the table at the beginning of this exercise (Table 3).

*From these results, you can see that this design is currently using only dedicated device resources (i.e. embedded memory, embedded multipliers) and no logic other than the registers included within these resources.*

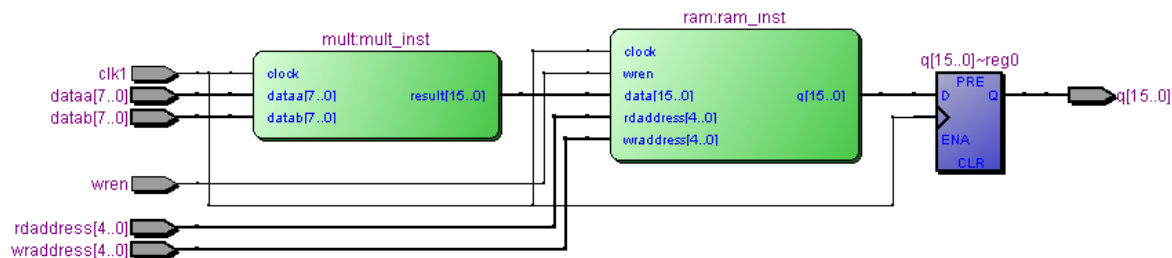
- \_\_\_\_ 3. Expand the **Fitter** folder in the **Compilation Report**.
  - a. Locate the **Resource Section** folder.
  - b. From the **Resource Utilization by Entity** table, record again in Table 3, the resource counts for the **mult** and **ram** sub-designs.
  - c. From the **Control Signals** table, also in the **Resource Section**, record the control signals found and their fan-out.

*In the next few steps, you will take a look at some additional ways to analyze the results of your compilation and determine the location of the output registers.*

### Step 3: Explore the design logically using the RTL Viewer

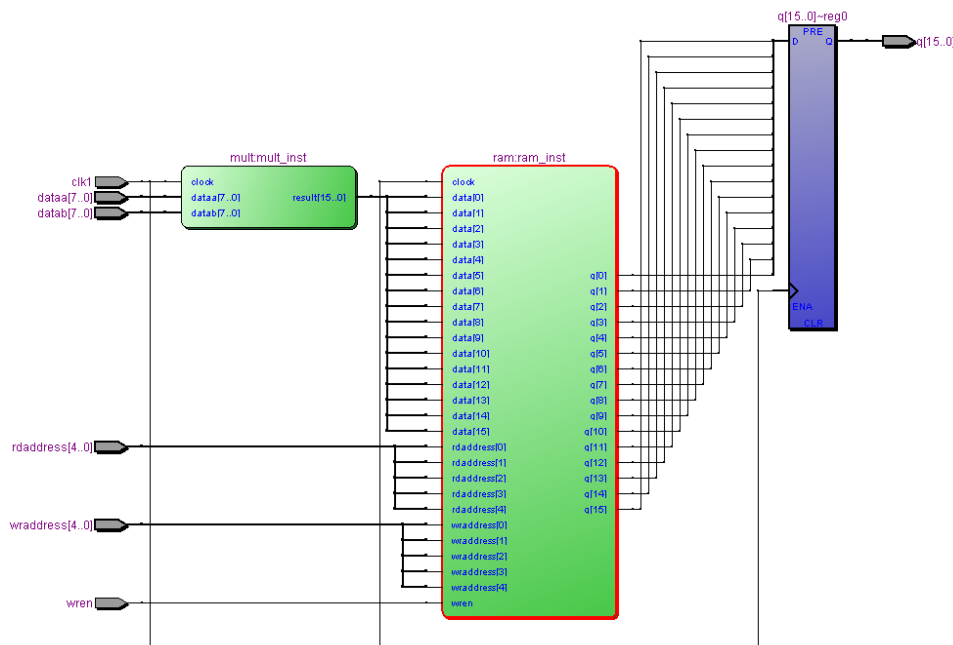
The **RTL Viewer** allows you to view a logical representation of an analyzed design graphically. It is a very helpful tool for debugging HDL synthesis results.

1. From the **Tools** menu, open the **RTL Viewer** (under **Netlist Viewers**). You can also access the RTL Viewer from the **Compile Design** section of the **Tasks** window.



You should see the diagram shown above. The output register block is named **inst2[15..0]** in the schematic version of the project, but the rest of the diagram is the same for all three versions. This is a graphical view displaying the logical representation of the design. Currently it shows the I/O, the instantiation of the **mult** and **ram** sub-designs, and an additional set of output registers. Notice the registers are external to the memory block per the original design.

2. Select the **ram** sub-design to highlight it. The outline of the block turns bright red when the block is selected. Right-click and select **Ungroup Selected Nodes**.

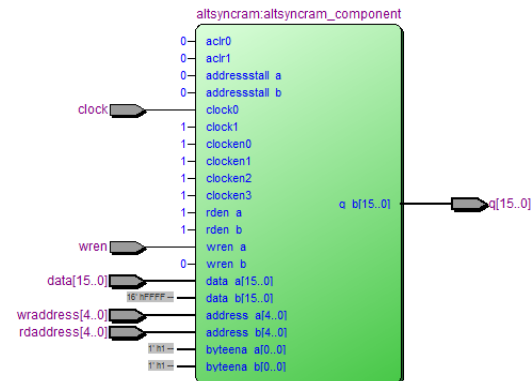


The ram block is now displayed with all of the input and output buses expanded. This operation is helpful when you want to see how individual bits are connected. This operation can be performed on internal blocks and I/O.

- \_\_\_\_ 3. Select the **ram** sub-design again if it was deselected.
- \_\_\_\_ 4. Right-click and select **Group Related Nodes**.

*This returns the **RTL Viewer** to the previous view.*

- \_\_\_\_ 5. Double-click on the **ram** sub-design block.

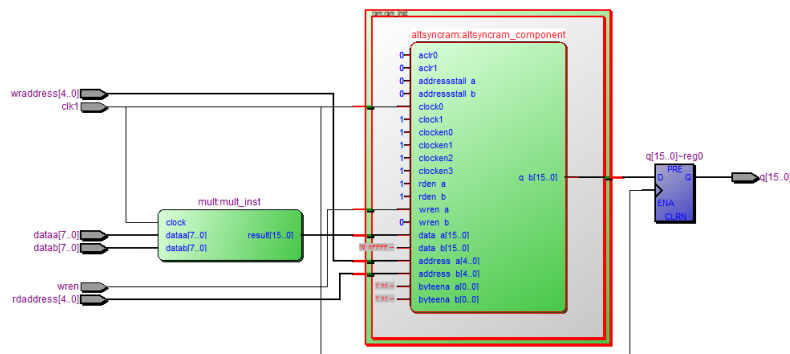


*You have now descended the hierarchy into the **ram** sub-design. As a result, the view changes to the above (or similar) image. This shows that the ram sub-design is made up of a single megafunction block called **altsyncram**. You can continue double-clicking blocks to descend the hierarchy to its lowest level: single-bit RAM functions. Let's view this lowest level of the hierarchy in a different way.*

- \_\_\_\_ 6. Double-click in any empty space.

*This returns the viewer to the top-level view of the design. If you've descended further into the hierarchy, you may need to do this a few times to return to the top.*

- \_\_\_\_ 7. Select the **ram** sub-design again.
- \_\_\_\_ 8. Right-click and select **Display Content**.



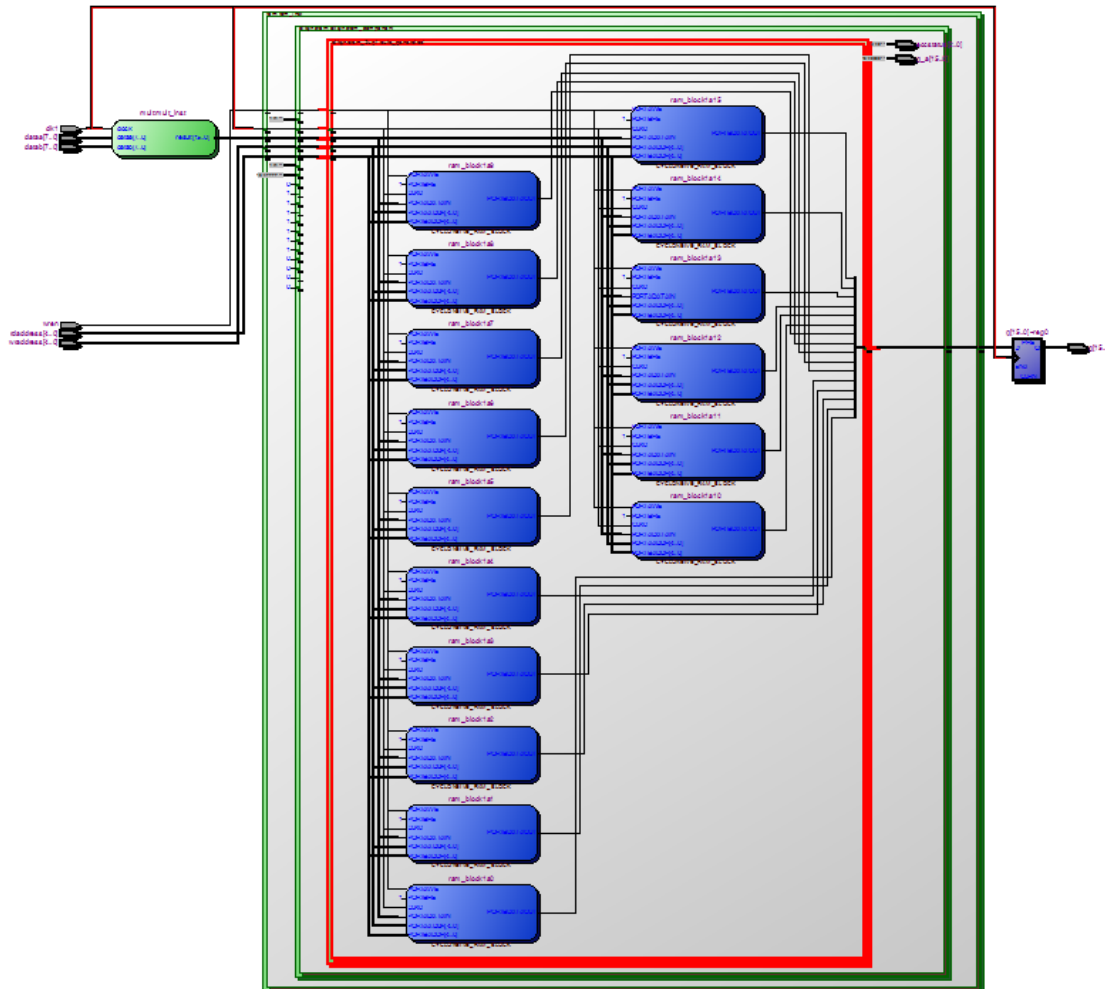
*Instead of displaying the lower hierarchical level alone, the **Display Content** option displays the lower level within the current hierarchy logic (may appear differently than above). The green box (currently highlighted in red) around the **altsyncram** sub-design indicates a hierarchical boundary.*

9. Select the **altsyncram** module, right-click, and select **Display Content**.

*Descending the hierarchy again, the internal logic of the **altsyncram** module (**altsyncram\_???1** The ??? denote random characters the tool may introduce to differentiate the ram blocks) is displayed including module I/O that are not connected.*

10. Select the **altsyncram\_???1** module.

11. Right-click, and select **Display Content**. Zoom out with the magnifying glass tool (right-click) if necessary.

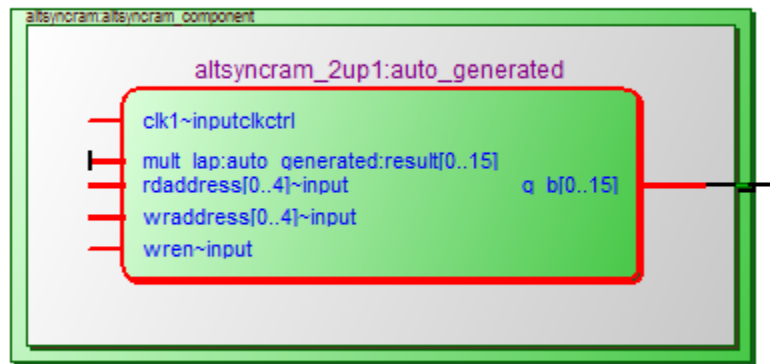


*Moving down into the **altsyncram\_???1** module, the viewer indicates that this module is represented by 16 smaller RAM functions, one function for each input/output data bit. Remember, this is a FUNCTIONAL representation of the design, not how the design will ultimately be implemented in the target **Cyclone IV E** device.*

**Step 4: Explore the ram sub-design physically using the Technology Map Viewer**

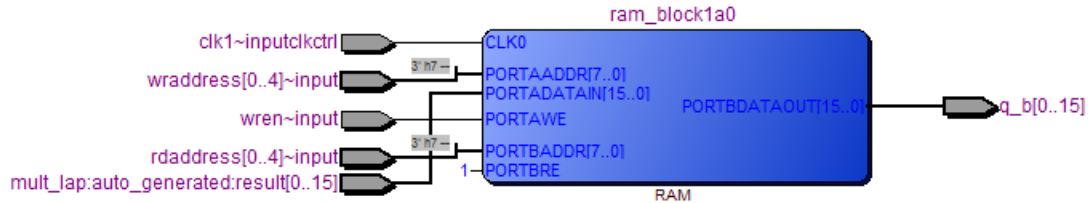
*The Technology Map Viewer allows you to see how a design is actually implemented using FPGA/CPLD resources. Use this tool as an aid during constraining and debugging to see changes in resource usage as settings and options are adjusted.*

1. Locate the ram block from the RTL Viewer into the Technology Viewer by doing the following:
  - a. In the **RTL Viewer**, switch back to the standard cursor if you switched to the magnifying glass
  - b. locate and highlight (select) the innermost green hierarchical ring for the ram function (**altsyncram\_???**).
  - c. Right-click and select **Locate** ⇒ **Locate in Technology Map Viewer**.



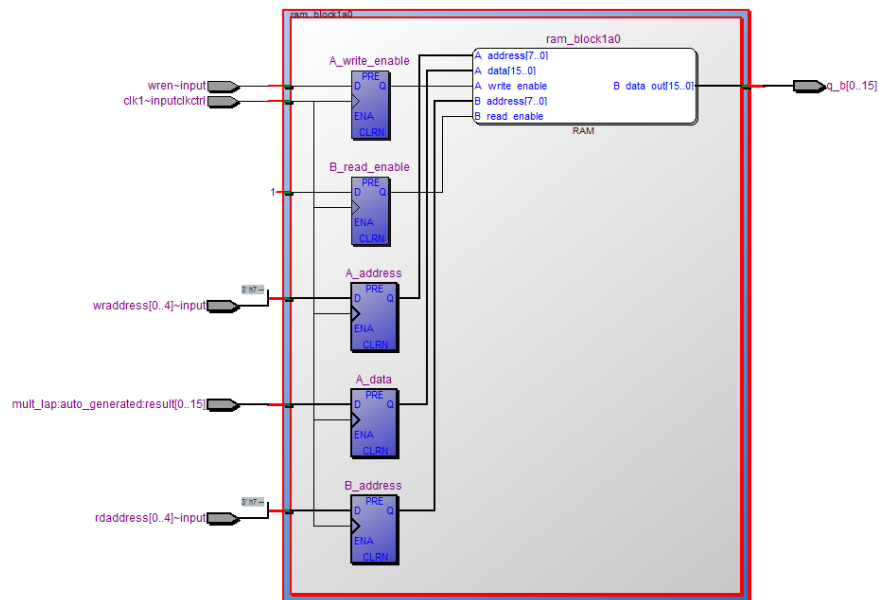
*The **Technology Map Viewer** opens and displays the **altsyncram\_???** module. Your screen may have the outputs grouped depending on the entry method you've chosen.*

- \_\_\_\_ 2. Descend the hierarchy into the **altsyncram\_???** module, using any of the methods shown so far in the exercise.



Now, instead of showing the 16 functional memory blocks as shown in the **RTL Viewer**, the **Technology Map Viewer** displays the **ACTUAL** device resource that was used, a single RAM block (**ram\_block1a0**).

- \_\_\_\_ 3. Double-click on the **ram\_block1a0** to view the detailed implementation of the memory block.




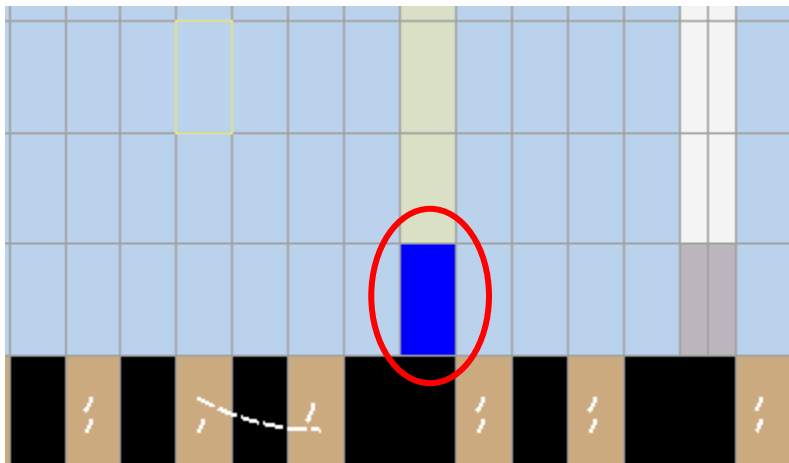
From this view you can see all the inputs to the RAM are registered.

- \_\_\_\_ 4. Click the back toolbar button  to return to the view of the RAM block.


**Step 5: Use the Chip Planner to view connections to the ram sub-design**

The **Chip Planner** will give you a sense of where logic has been placed in the design. This can be very helpful when trying to understand design performance, as proximity is the key to performance in most newer FPGAs and CPLDs. Though the **Chip Planner** can be used for manually locating and moving logic, we only want to use it to evaluate results.

- \_\_\_\_ 1. In the Technology Map Viewer, select the RAM block (ram\_block1a0) to highlight it. Right-click the block.
- \_\_\_\_ 2. From the **Locate** submenu, select **Locate in Chip Planner**.
- \_\_\_\_ 3. In the Chip Planner, select the zoom tool  (Chip Planner toolbar) and right-click to zoom out a few times.

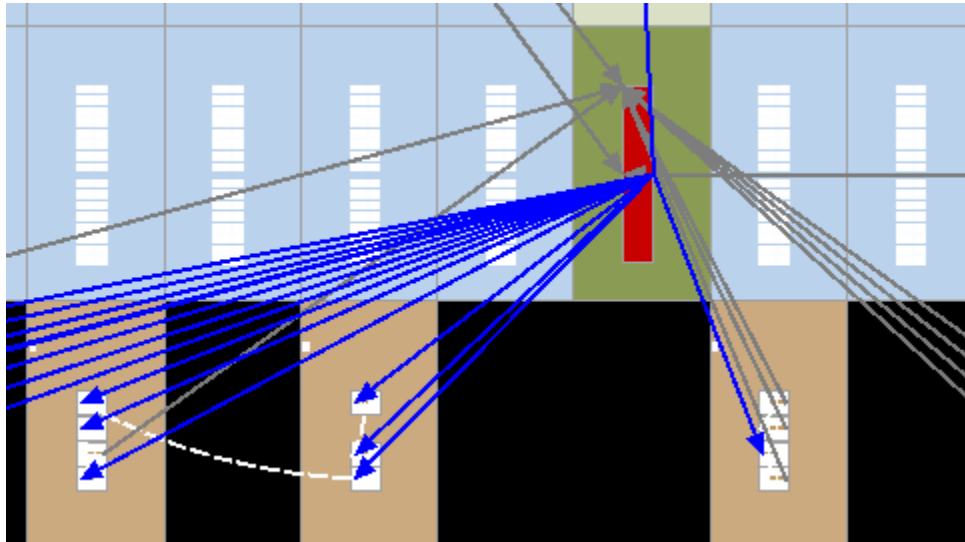


You should now see the **ram** sub-design (circled) and where it has been placed in the **Cyclone IV** device by the Fitter. (Your location may differ slightly depending on the design entry method you chose.) The other shaded areas in the **Chip Planner** are where the remaining logic and routing have been placed. Since we don't have any constraints applied to the design, the Fitter was free to place logic anywhere on the device. This will change later.


- \_\_\_\_ 4. With the memory block still highlighted in the Chip Planner, click the  **Generate Fan-In Connections** toolbar button.

*The **Chip Planner** now displays the signal connections that are feeding into the **ram** sub-design (coming from device I/O and the multiplier).*

- \_\_\_\_ 5. Select the **ram** block in the Chip Planner again (you may need to zoom back in a little), and click the **Generate Fan-Out Connections** button  .



*The **Chip Planner** now displays both fan-in and fan-out connections. The fan-out connections are displayed in blue because they are currently selected.*

- \_\_\_\_ 6. Turn off the fan-in/fan-out by first clicking on any non-highlighted part of the device. Then click the **Clear Unselected Connections/Paths** toolbar button  .
- \_\_\_\_ 7. Close the RTL Viewer, Technology Map Viewer, and Chip Planner.

### Exercise Summary

- Performed a full compilation
- Gathered information from the compilation report
- Cross-probed between windows to analyze design processing results in different ways using the RTL Viewer, Technology Map Viewer, & Chip Planner

## END OF EXERCISE 3



# Exercise 4

## Exercise 4

### Objectives:

- Create a new revision to store new constraint settings
- Make design constraints using the Assignment Editor

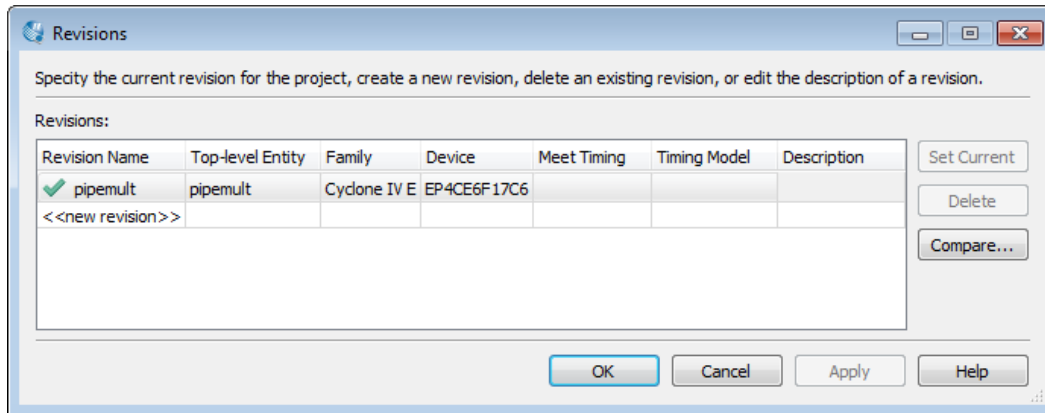
Device Name	EP4CE6F17C6
<b>Total Design</b>	
Total logic elements	
Total registers	
Total pins	
Total memory bits	
Embedded Multiplier 9-bit elements	
<b>mult sub-design</b>	
Logic Cells (mult)	
Dedicated Logic Registers (mult)	
DSP Elements (mult)	
<b>ram sub-design</b>	
Logic Cells (ram)	
Dedicated Logic Registers (ram)	
Memory Bits (ram)	
M9Ks (ram)	
<b>Control Signals</b>	
Name	Fanout

Table 4 – Compilation Report

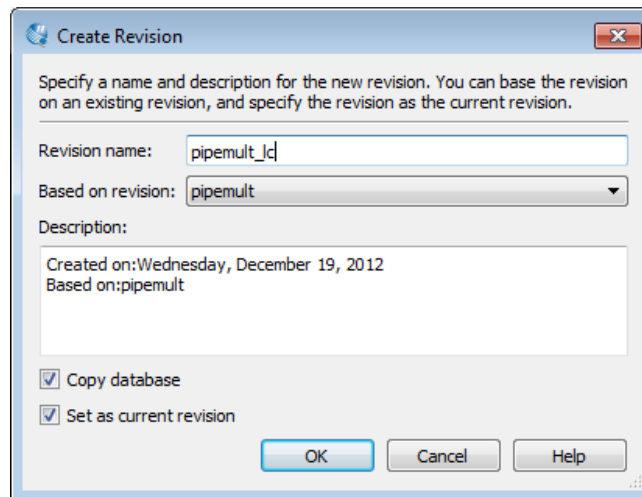
**Step 1: Create a new revision to store constraint changes**

*In order to experiment with different settings or assignments and to see how these changes affect your results, the Quartus II software has support for creating revisions, with each revision building a new QSF file. You can also compare the compilation results of your various revisions.*

- \_\_\_\_ 1. From the **Project** menu, select **Revisions**.



- \_\_\_\_ 2. In the **Revisions** dialog box, double-click **<<new revision>>**.
- \_\_\_\_ 3. Type in **pipemult\_lc** as the **Revision name**.



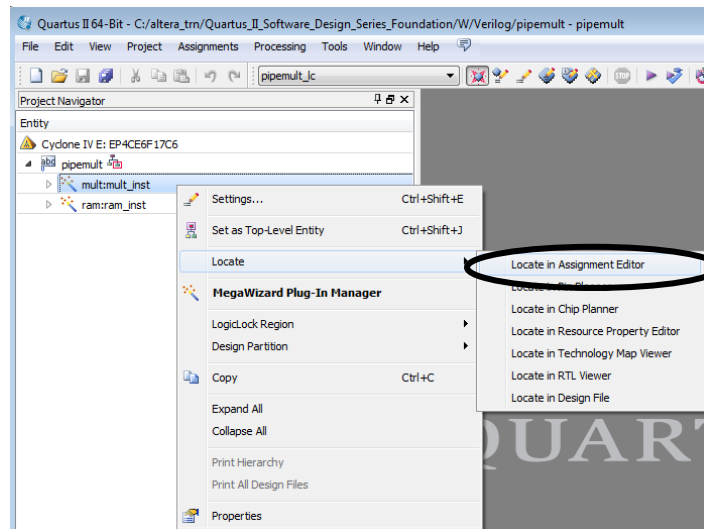
*Notice the revision is automatically based on the original revision pipemult. You could use the drop-down to base it on a blank revision (like starting a new project).*

- \_\_\_\_ 4. Leave all other defaults and click **OK**.
- \_\_\_\_ 5. Click **OK** to close the **Revisions** dialog box.

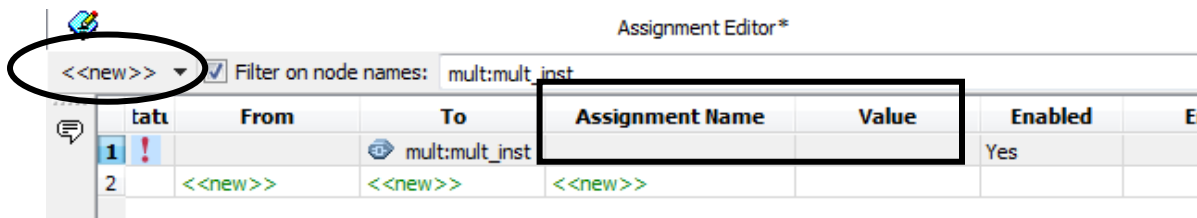
## Step 2: Implement the multiplier in logic elements instead of embedded multipliers

*Cyclone IV E embedded multipliers are a valuable resource for implementing multiply operations in the FPGA. They provide a better usage of resources for multiplication compared to using logic elements (LEs). However, embedded multipliers are a limited resource compared to the number of LEs available. If your design has many multipliers, it may be advantageous to implement smaller or non-speed critical multipliers in logic elements (or even memory blocks) instead. This can be done using an option in the MegaWizard Plug-In Manager or on a multiplier-by-multiplier basis using a logic option in the Assignment Editor.*

- \_\_\_\_ 1. In the Project Navigator, Hierarchy tab, expand the **pipemult** hierarchy.
- \_\_\_\_ 2. Right-click on the **mult** entity in the hierarchy.
- \_\_\_\_ 3. From the **Locate** submenu, select **Locate in Assignment Editor**.



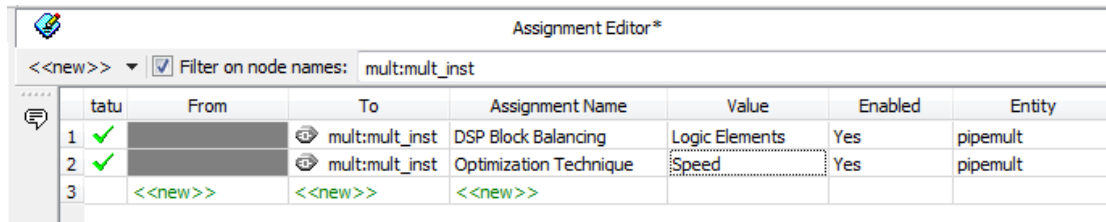
- \_\_\_\_ 4. Click the **<<new>>** button at the top left hand corner of the assignment editor.



*The name **mult:mult\_inst** should have automatically been entered in the **To** field*

- \_\_\_\_ 5. Double-click the cell in the **Assignment Name** column and select **DSP Block Balancing** from the drop-down menu.  
*Tip: By quickly typing “DSP”, you may find the option faster than scrolling.*
- \_\_\_\_ 6. Hit the **Enter** key to accept the assignment name.
- \_\_\_\_ 7. Double-click the cell in the **Value** column for the **DSP Block Balancing** option and select **Logic Elements** from the drop-down menu.

- \_\_\_\_ 8. Hit the **Enter** key to accept the assignment value.
- \_\_\_\_ 9. Click the <<new>> button again and set the Assignment Name to **Optimization Technique**.
- \_\_\_\_ 10. Double-click the cell in the **Value** column for **Optimization Technique** and select **Speed**. The Assignment Editor should look like this:



	From	To	Assignment Name	Value	Enabled	Entity
1	tatu	mult:mult_inst	DSP Block Balancing	Logic Elements	Yes	pipemult
2	tatu	mult:mult_inst	Optimization Technique	Speed	Yes	pipemult
3	<<new>>	<<new>>	<<new>>	<<new>>	<<new>>	<<new>>

- \_\_\_\_ 11. Save the **Assignment Editor** file.
- \_\_\_\_ 12. Click  to compile the design.

### Step 3: Gather resource information from the Compilation Report

- \_\_\_\_ 1. Open **Compilation Report** from the processing menu if it's not already open.
- \_\_\_\_ 2. From the **Flow Summary** section of the **Compilation Report**, record the **Total logic elements**, **Total memory bits**, number of **Embedded multiplier 9-bit elements** and **Total pins** in the table at the beginning of this exercise (Table 4).
- \_\_\_\_ 3. In the **Fitter** folder of the **Compilation Report**, do the following:
  - a. Locate the **Resource Section** folder.
  - b. From the **Resource Utilization by Entity** report, record in the exercise table the resource counts for the **mult** and **ram** sub-designs.
  - c. From the **Control Signals** table, also in the **Resource Section**, record the fan-out for control signals **clk1** and **wren**.

*From the results, you can see that in this revision, the multiplier implementation was moved from the embedded multipliers into the logic array blocks.*

*Feel free to open up the tools used in the prior exercise (RTL Viewer, Technology Map Viewer, Chip Planner) to examine the results of this compilation.*

### Exercise Summary

- Created a new revision to evaluate different constraints
- Controlled logic options (constraints) using the Assignment Editor

## END OF EXERCISE 4

# Exercise 5

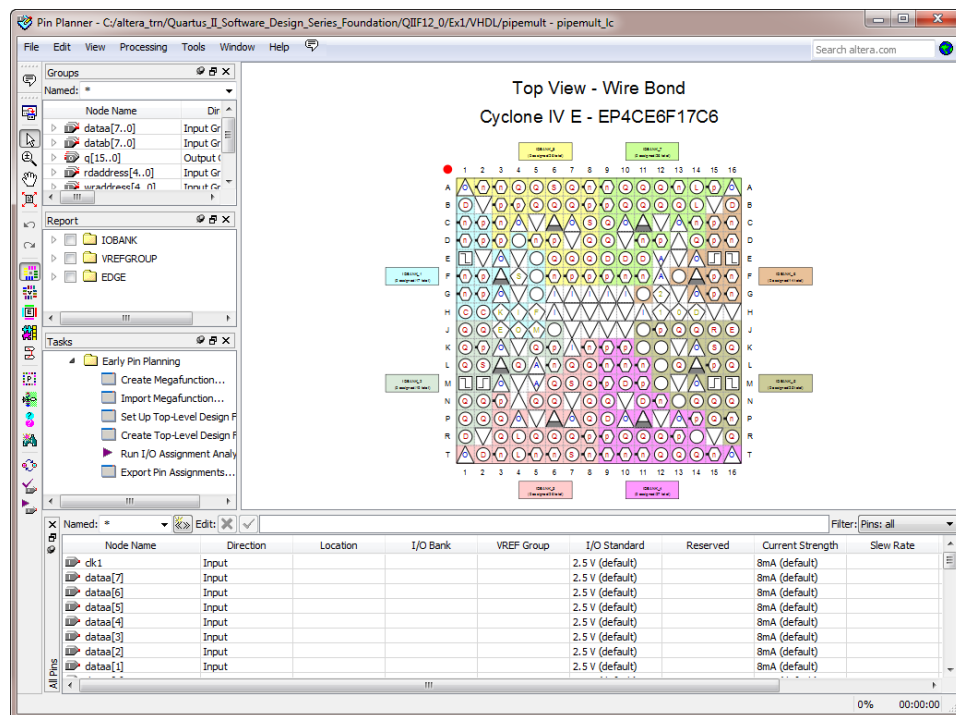
## Exercise 5

### Objectives:


- Assign I/O pins and perform I/O Assignment Analysis
- Back-annotate pin assignments to lock placement
- Use the Pin Migration View to see the effect of device migration on I/O assignments

### Step 1: Use Pin Planner to assign I/O pins & set I/O standards


1. From the **Assignments** menu or the **Assign Constraints** folder of the **Tasks** window, open the **Pin Planner**.



Remember, if not already detached, you can detach the Pin Planner from the main Quartus II application window to make it easier to work with. You can also undock the Groups and All Pins lists.

2. Make sure the **Top View** of the device is displayed, indicated by the red dot in the upper left hand corner of the package. If not, select **Package Top** from the **View** menu.
3. In the **Pin Planner** toolbar, make sure that the **Show I/O Banks** toolbar button  is selected. This is the default view in the Pin Planner. If you are unsure of which button this is, you can also find this option in the **View** menu.

4. Turn on the Live I/O checking feature to perform limited checks on the I/O assignments you'll be making in the following steps.

- Click  in the toolbar or select **Enable Live I/O Check** from the **Processing** menu in the Pin Planner.
- If a dialog appears indicating that Fitter placements will be turned off, click **Yes**.

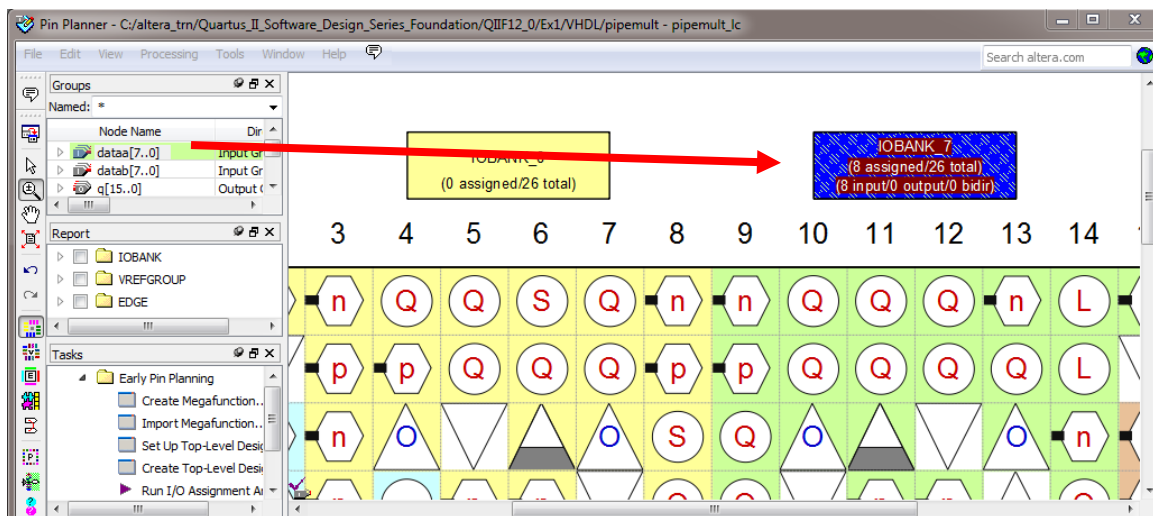
*The Live I/O Check icon will appear in the lower-left corner of the Package view to indicate the feature is turned on.*

5. From the **View** menu in the Pin Planner, select **Live I/O Check Status Window**.

*This opens a small floating window that alerts you to I/O assignment errors and warnings as you make assignments. Be sure to watch this window during the following steps. If the status indicates an error or warning, check the Messages window in the main Quartus II window for information.*

*The new window can be placed into the frames of the main pin planner interface or turned in to tabs if dragged on top of other frames.*

6. In the **Groups List** window of the **Pin Planner**,
- Locate the **dataa** input bus, and click it once to select it.
  - Then, **click and drag** the **dataa** bus from the **Groups List** window and **drop** it into the **IOBANK\_7** box of the **Package View** (the top right side of the chip).

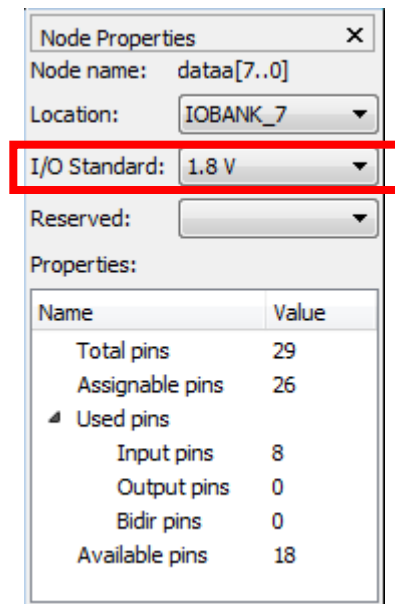


*The display for IOBANK\_7 will now change to indicate that I/O Bank 7 has 8 assigned pins out of 26 total pins. Assigning signals to an I/O bank like this gives the Fitter the freedom to place the signals anywhere within the specified I/O bank.*

7. In the **Groups List**, right-click on the **dataa** bus and select **Node Properties**.



8. In the **Node Properties** dialog box, set the **I/O standard** for **dataaa** to **1.8 V**.



Notice that the All Pins list reflects the I/O standard change for all of the individual `dataaa` pins. You can drag and drop the column headings to rearrange them or right-click anywhere in the All Pins list and select **Customize Columns** to add or remove I/O-related assignment columns.

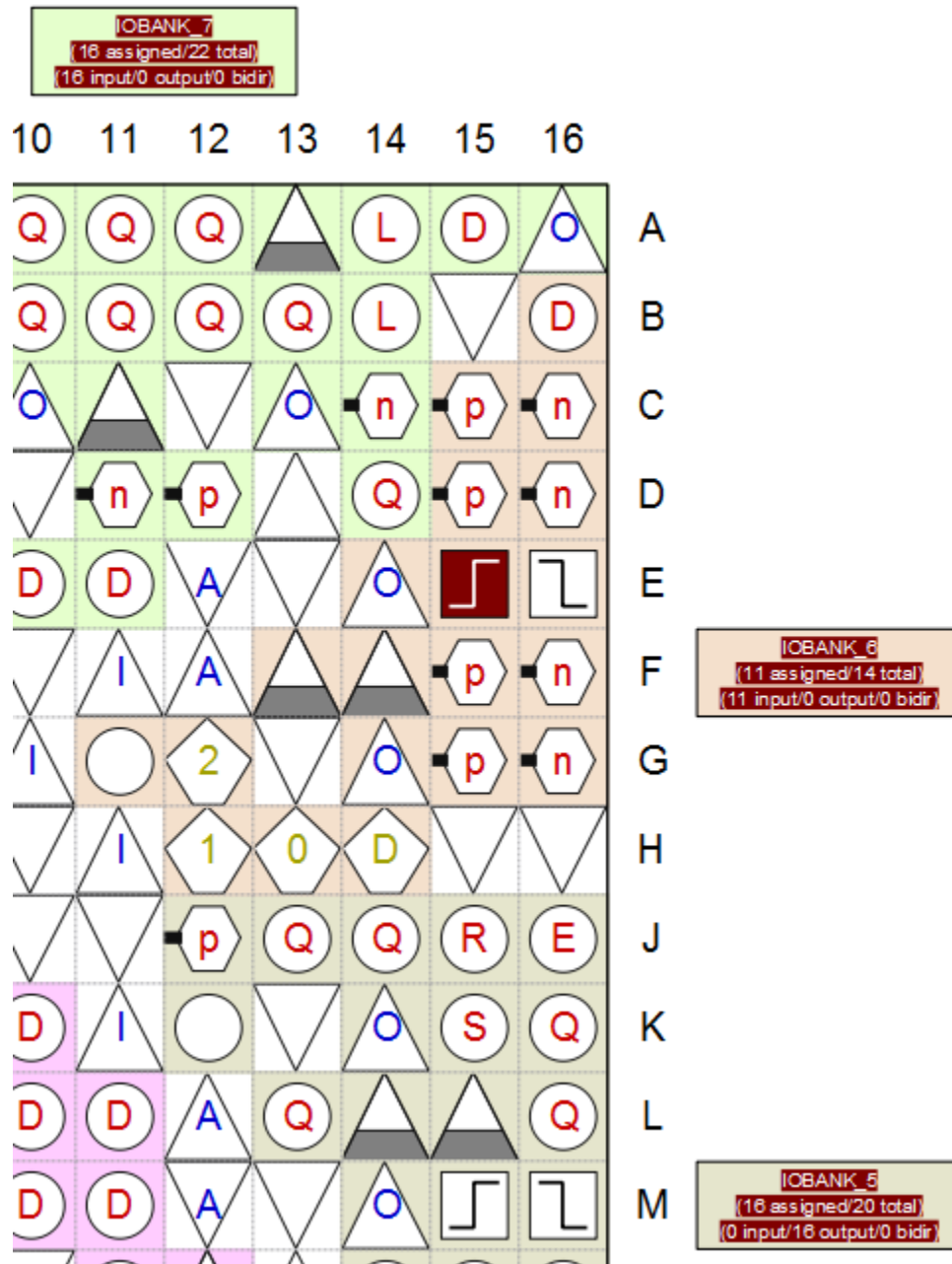
Node Name	Direction	Location	I/O Standard	I/O Bank	Vref Group
<code>dataaa[7]</code>	Input	IOBANK_7	1.8 V	7	
<code>dataaa[6]</code>	Input	IOBANK_7	1.8 V	7	
<code>dataaa[5]</code>	Input	IOBANK_7	1.8 V	7	
<code>dataaa[4]</code>	Input	IOBANK_7	1.8 V	7	
<code>dataaa[3]</code>	Input	IOBANK_7	1.8 V	7	
<code>dataaa[2]</code>	Input	IOBANK_7	1.8 V	7	
<code>dataaa[1]</code>	Input	IOBANK_7	1.8 V	7	
<code>dataaa[0]</code>	Input	IOBANK_7	1.8 V	7	

You could have assigned the location and changed the I/O standard using the All Pins list instead of the Groups list. However, be aware that **individual** assignments made in the All Pins list (or on individual signals within groups in the Groups list) take precedence over **group** assignments made in the Groups list. The best way to ensure that all the signals in an entire bus or signal group inherit an assignment like this is to make the assignment to the group heading (i.e. `dataaa[7..0]` that you dragged and dropped) in the Groups list.

9. Using either the **Package View** or the **All Pins List**, assign the **datab** bus to **IOBANK\_7**.

You may notice something going on with the Live I/O Check status window. Be sure to check it out and look at the Messages window. For now, though, **don't** try to fix the problem. We want to see how the compiler handles this.


- \_\_\_\_ 10. Assign the **q** output bus to **IOBANK\_5** (lower I/O bank on the right side of the chip) and set the **I/O standard** to **1.8 V**.
- \_\_\_\_ 11. Assign both the **rdaddress** and **wraddress** buses to **IOBANK\_6** (upper I/O bank on the right side of the chip) and set the **I/O standard** to **1.8 V**.
- \_\_\_\_ 12. Using the **All Pins** list, assign **clk1** to **Pin E15** and set the **I/O standard** to **1.8 V**.
- \_\_\_\_ 13. From the **All Pins** list, assign **wren** to **IOBANK\_5** and set the **I/O standard** to **1.8 V**



Your Pin Planner should look similar to the above picture.


## Step 2: Analyze I/O assignments

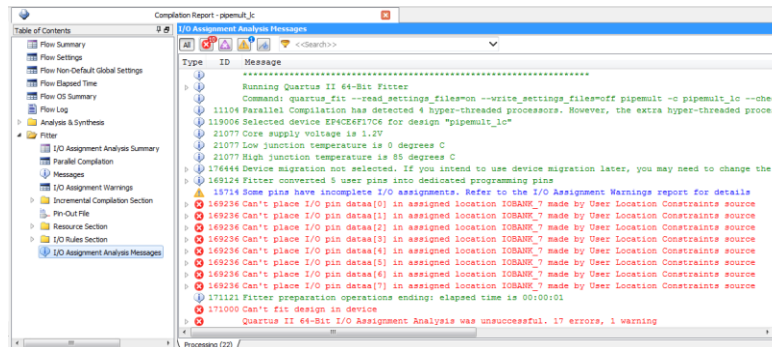
*Now that you have made general I/O assignments, you can check the validity of those assignments without running a full compilation using I/O Assignment Analysis. This way you can quickly and easily find I/O placement issues and correct them. This will check more I/O rules than the live I/O check, so be sure to always run I/O Assignment Analysis even if you are using the live I/O checking feature.*

1. Synthesize the design: From the **Processing** menu in the main Quartus II window, go to **Start** and select **Start Analysis & Synthesis** or click on the  button in the main Quartus II toolbar.

2. Click OK when complete.

*The compiler may turn off live I/O checking and then turn it back on. If the compiler ever leaves it turned off after a compilation process, simply turn it back on in the Pin Planner.*

3. From the **Processing** menu, go to **Start** and select **Start I/O Assignment Analysis** or click on the  button in the Pin Planner toolbar.
4. Click **OK** once the analysis is complete.
5. Open the **Compilation Report** from the Processing menu
6. Expand the **Fitter** folder and click on **I/O Assignment Analysis Messages**



*Was the analysis successful? Check the messages. They should read as shown above. I/O Assignment Analysis has verified the error that the live I/O check alerted us to while we were making the I/O assignments.*

7. Review the **I/O Analysis Messages** and determine the cause of the error. Expand the error messages to get more detail as to why each pin of the **dataa** bus is not being placed successfully.

*Determining the cause of the I/O placement failure requires reading the error messages carefully and having a little understanding of **Cyclone IV E** devices or general FPGA I/O blocks. See if you can understand and correct the cause of the errors on your own. If you do not have a lot of **Cyclone IV E** device or general FPGA experience, the next steps will show you how to correct the errors.*

- \_\_\_\_ 8. Bring the **Pin Planner** to the foreground and re-open the **Live I/O Check Status** window.
- \_\_\_\_ 9. Expand the **dataa** and **datab** groups in the Groups list. If necessary, scroll to the right to make sure you can see the **I/O Standard** column in the Groups list or All Pins list. If the **I/O Standard** column does not appear, right-click in the Groups list and select **Customize Columns** to make the column visible.

*Notice that you have assigned the **dataa** and **datab** input buses to **I/O Bank 7** but set different **VCCIO (1.8 & 2.5)** voltage levels for them. Cyclone IV E FPGAs, like all Altera FPGA devices, allow for only one VCCIO per I/O bank.*

- \_\_\_\_ 10. Using the Groups List, change the **I/O standard** setting for the **dataa** group to **2.5 V** (or **2.5 V (default)**). Make sure the **dataa** group as well as each individual signal in the group is set to 2.5 V.


*In the Groups list, signals have **individual** assignments as well as **group** assignments, and individual or group assignments are not automatically inherited by each other (i.e. setting all the individual signals in **dataa** to 2.5 V does not automatically set the group assignment, **dataa[7..0]**, to 2.5 V), so make sure that both the group and individual assignments are all set the same.*

*Notice the change in the Live I/O Check Status window.*

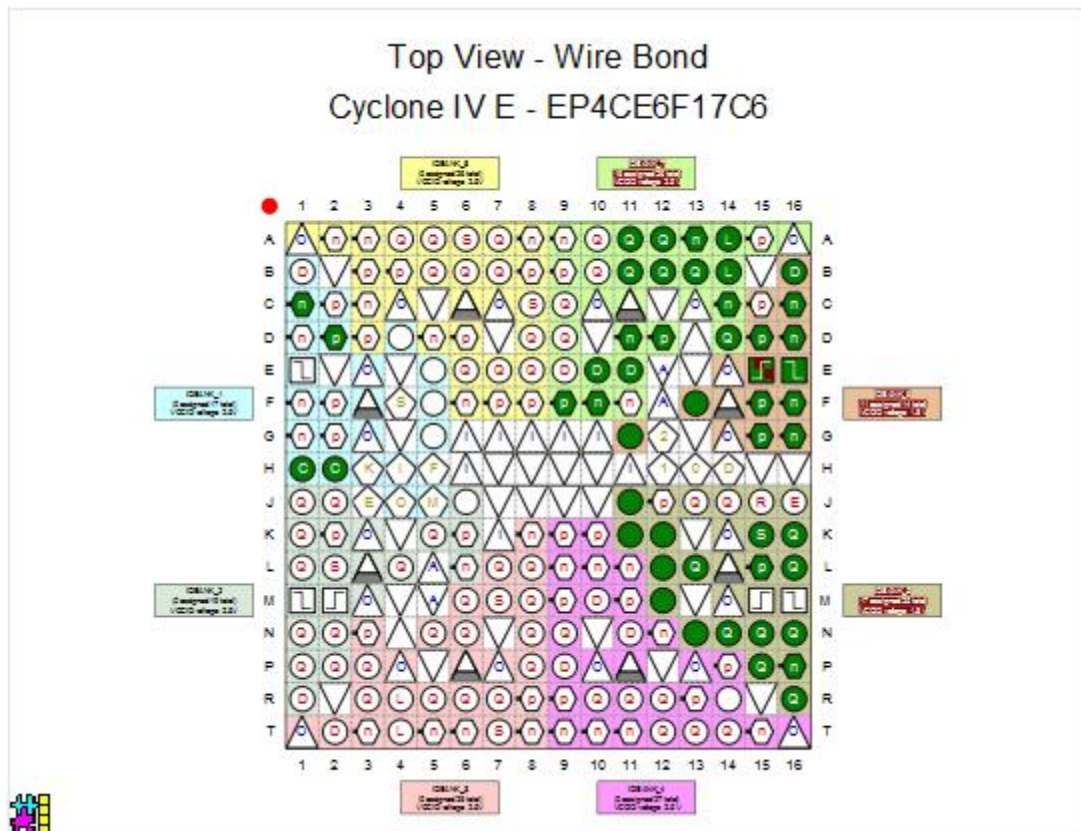
- \_\_\_\_ 11. Re-run **I/O Assignment Analysis**. Click **OK** when complete.

*See how quickly and easily you can check your I/O placement assignments without running a full compilation!*

12. To see where the Fitter placed your I/O, in the Pin Planner

- Click on the  button in the **Pin Planner** toolbar or from the **View** menu choose **Show** ⇒ **Show Fitter Placements**.
- Click **OK** to turn off live I/O checking.

*Live I/O checking must be turned off when viewing Fitter placements. The status window confirms that it is off.*

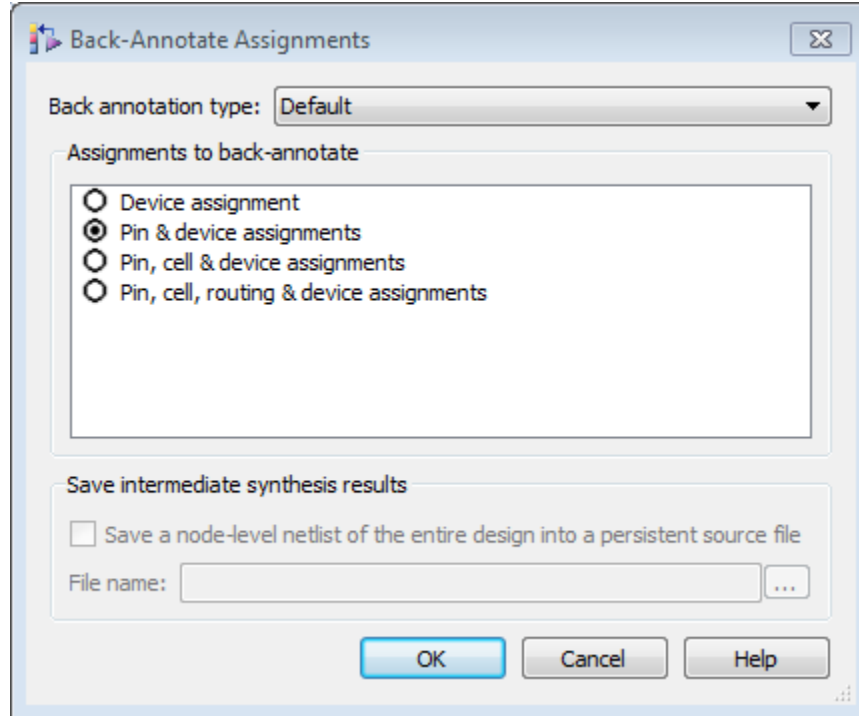


You can see that Fitter-selected pin locations appear in green in the **Pin Planner** as shown above. By adding I/O bank location assignments, you constrained the signals to be placed in the selected I/O banks. Without these assignments, the Fitter could have placed the signals anywhere on the chip. The extra signals located in **IOBANK\_1** on the left side of the device are device programming pins that the Fitter will always add. Notice that **clk1**, the only signal you manually assigned to a specific pin (E15), appears green with a red hatched pattern. Remember that this indicates that your user assignment matches up with the location assignment selected by the Fitter.

**Step 3: Back-annotate pin assignments to lock placement**

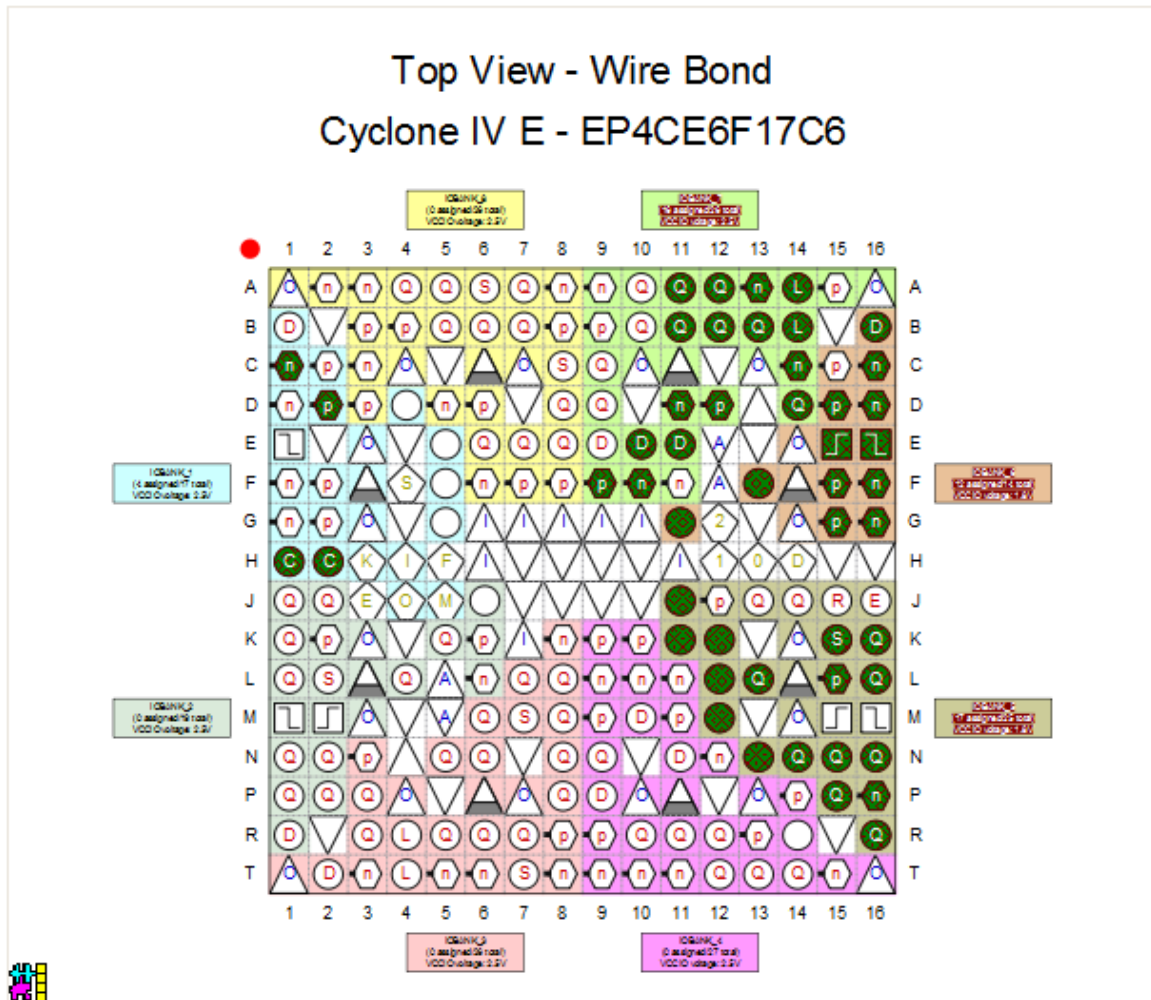
*At this point, you like the I/O location assignments made by the Fitter, and you've verified the pin-out to begin board design. Now you need to make sure that the pin locations are not moved during successive compilations.*

1. From the **Assignments** menu in the main Quartus II window, select **Back-Annotate Assignments** to open the **Back-Annotate Assignments** dialog box. The dialog box may appear slightly different from the screenshot below.





2. In **Assignments to back-annotate**, enable **Pin & device assignments** (default setting) as shown above. Click **OK**.



*Notice how all the I/O pins that were green earlier have changed to the green and red hatch pattern. This indicates that the locations that were Fitter-assigned I/O are now user-assigned I/O and have been written into your .QSF file as constraints.*

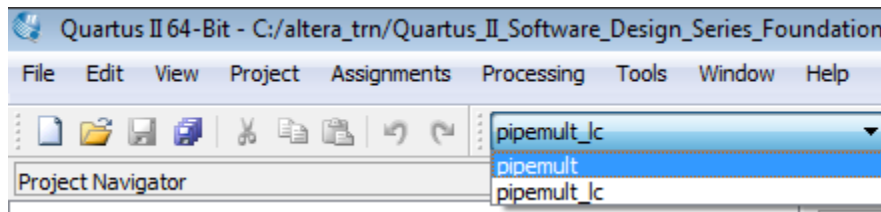
**Step 4: Transfer pin assignments to original revision**

Now you should carry these pin assignments from your current design revision to the original revision (in case you decide to choose that revision later). Remember that all assignments, including I/O-related assignments, are unique to a particular project revision, so the assignments must be copied over if you want them to match between the two revisions. To do this, you will export the assignments as a .tcl script and import them into the original pipemult revision by running the script.

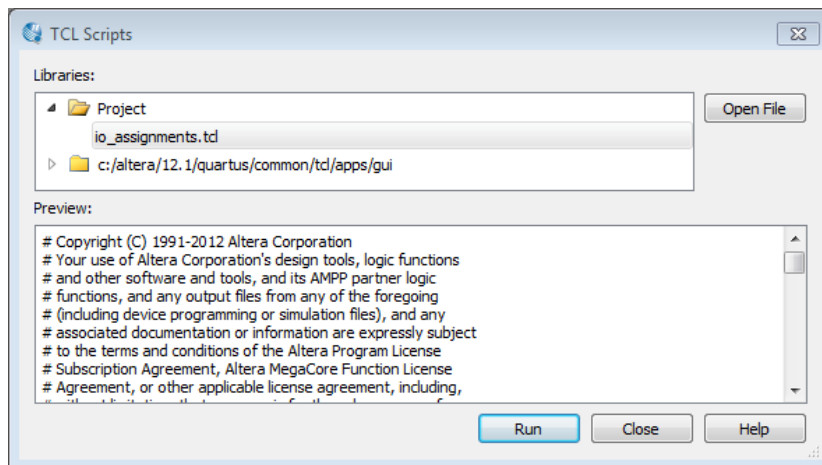
- \_\_\_\_ 1. With the **Pin Planner** active, go to the **File** menu and select **Export**.
- \_\_\_\_ 2. In the **Export** dialog box, choose Save as type **Tcl Script File**. Type the filename **io\_assignments.tcl** and click **Export**.



- \_\_\_\_ 3. Go back to the main Quartus II window. Use the drop-down menu at the top of the Quartus II window to change the revision back to **pipemult**.



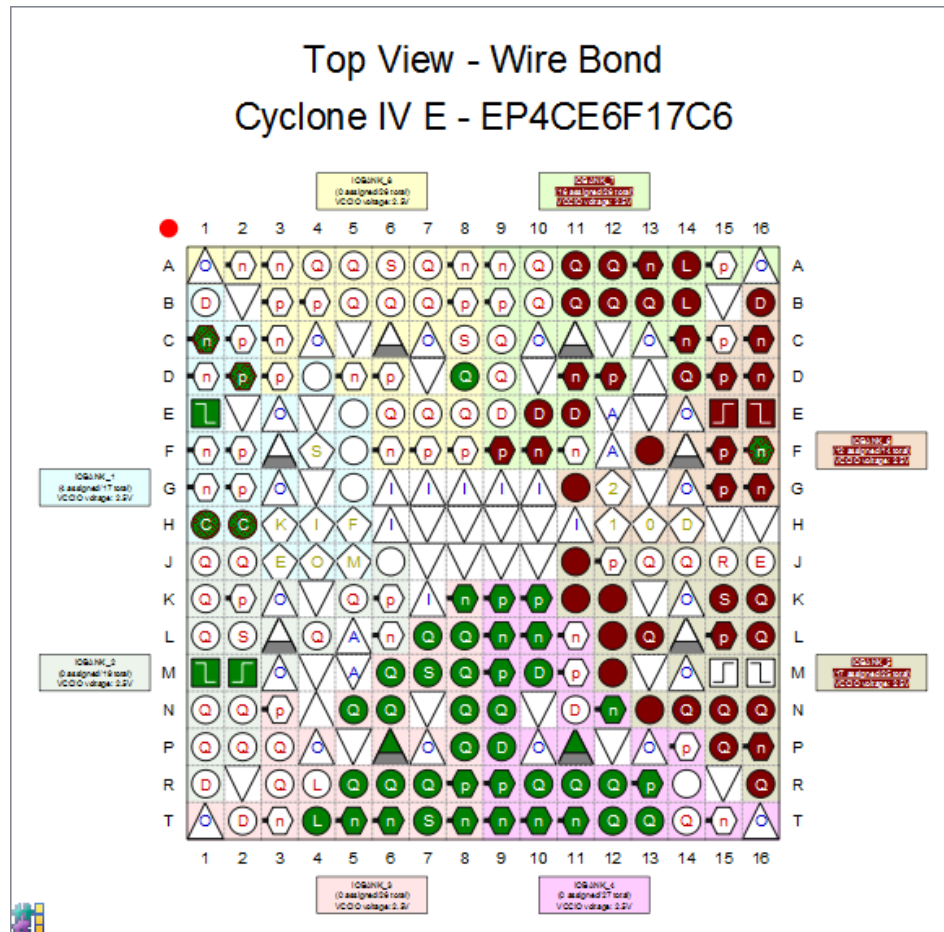
- \_\_\_\_ 4. From the **Tools** menu, select **Tcl Scripts...**
- \_\_\_\_ 5. Highlight **io\_assignments.tcl** file from the Project folder. Click **Run**.




- \_\_\_\_ 6. Click **OK** when done.



7. Open the **Pin Planner** to see that the assignments were imported correctly. If **Show Fitter Placements** was turned off, turn it back on.




Remember that the green pins are the Fitter-placed locations from the previous compilation of the revision **pipemult**. The reddish brown pins are the pins imported from the **pipemult\_lc** revision, and the green and red hatched pins are intersecting assignments (think of the intersecting area of a Venn diagram) between the two revisions of the project. The green locations would all be removed if this revision were compiled again since the user-assigned I/O (in red) have precedence.

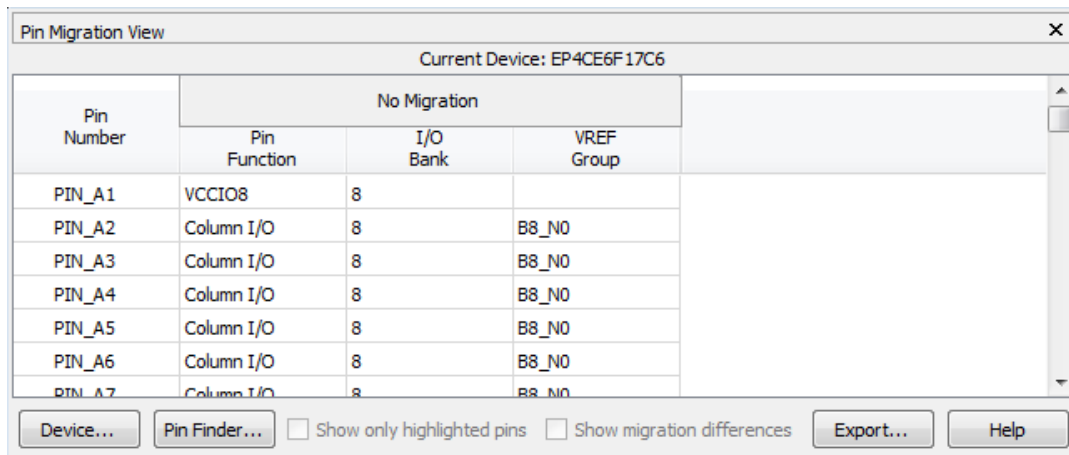
8. Turn off the viewing of Fitter placements by clicking  or by turning off the option in the **Show** submenu of the **View** menu.

You can see that the locations match the back-annotated locations from the **pipemult\_lc** revision.

**Step 5: Add a migration device and validate cross-migration I/O**

*Now you'll see what happens to pins when the pins' functions change when moving a design to a migration device. You can add migration devices to a project at any point if you expect to later move the design to another device. The Pin Planner Pin Migration View keeps track of possible issues you may have with your I/O assignments during a migration.*

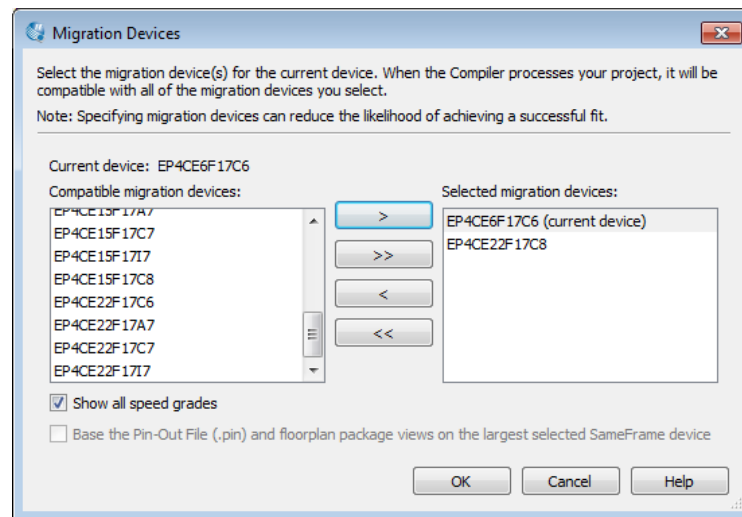
- \_\_\_\_ 1. Create a new revision of the project called **pipemult\_migration**, based off of the **pipemult** revision. Switch to the new revision in either the Revisions dialog box or the menu in the toolbar.
- \_\_\_\_ 2. Reopen the Pin Planner.
- \_\_\_\_ 3. **Turn off the Show Fitter Placement Button**  **in the Pin Planner if it is on**
- \_\_\_\_ 4. From the **View** menu (or by right-clicking in the Package view), select **Pin Migration Window**.



*The Pin Migration View appears. It only displays pin information for the current device because you have not selected any migration devices yet.*

- \_\_\_\_ 5. Click **Device** to open the Device dialog box.
- \_\_\_\_ 6. Click **Migration Devices**, and turn on the option to **Show all speed grades**.


7. Select the **EP4CE22F17C8** (it should be at or near the bottom of the list; double-check that you've selected the correct device; others have very similar designations) from the list, and double-click it or click > to move it to the list of migration devices.



8. Click **OK**.
9. Click **OK** again to close the **Device** dialog box.
10. In the Pin Migration View window, you can now see pin information about the migration device. Turn on the **Show migration differences** option at the bottom and look for pin **F10**.

Pin Migration View									
Current Device: EP4CE6F17C6									
Pin Number	Migration Result			Migration Devices					
	Pin Function	I/O Bank	VREF Group	Pin Function	I/O Bank	VREF Group	Pin Function	I/O Bank	VREF Group
PIN_A8	Column I/O	8	B8_N0	Column I/O	8	B8_N0	Dedicated Clock	8	B8_N0
PIN_A9	Column I/O	7	B7_N0	Column I/O	7	B7_N0	Dedicated Clock	7	B7_N0
PIN_B8	Column I/O	8	B8_N0	Column I/O	8	B8_N0	Dedicated Clock	8	B8_N0
PIN_B9	Column I/O	7	B7_N0	Column I/O	7	B7_N0	Dedicated Clock	7	B7_N0
PIN_D4	VCCD_PLL3			Row I/O	1	B1_N0	VCCD_PLL3		
PIN_E5	GND A3			Row I/O	1	B1_N0	GND A3		
PIN_F5	VCCA3			Row I/O	1	B1_N0	VCCA3		
PIN_F6	GND			Column I/O	8	B8_N0	GND		
PIN_F7	VCCINT			Column I/O	8	B8_N0	VCCINT		
PIN_F10	GND			Column I/O	7	B7_N0	GND		
PIN_F11	VCCINT			Column I/O	7	B7_N0	VCCINT		
PIN_G11	GND			Row I/O	6	B6_N0	GND		
PIN_J6	VCCINT			Row I/O	2	B2_N0	VCCINT		

Notice that this pin, used for one of our **dataa** signals in the design, changes function from a user I/O in I/O bank 7 in the current device to a ground pin in the migration device. Thus, the **Migration Result** columns show the pin function to be the ground pin, indicating that a signal on this pin cannot migrate to the same pin on the migration device. In a migration, power pins “win” over user I/O to prevent signals from being assigned to a pin that cannot be controlled and must be powered or grounded in the migration device. Let’s verify this with I/O Assignment Analysis.

- \_\_\_ 11. Close the Pin Migration View.
- \_\_\_ 12. From the **Processing** menu, go to **Start** and select **Start I/O Assignment Analysis** or click on the  button in the Pin Planner toolbar.
- \_\_\_ 13. Click **OK** once the analysis is complete.

*What happened? I/O Assignment Analysis failed because several pin assignments cannot exist both in the current device and the migration device. If you now look at pin F10 in the Package View, you’ll notice the pin symbol has changed to an upside-down triangle, indicating it as ground. Even though a signal could be assigned to this pin in the original device, the Pin Planner prevents us from assigning it or any other signal here if the **EP4CE22F17C8** is used as a migration device because the functionality of the pin changes.*

*If we fixed these I/O placement issues, we would be ready to perform a full compile. Once the compilation is done completed, we would be able to perform static timing analysis to verify all timing requirements are met and then be able to program the device using the generated bitstream.*

### Exercise Summary

- Assigned pin locations using the Pin Planner
- Back-annotated pin locations from prior compilation
- Verified I/O placement and constraints using I/O assignment analysis
- Used the Pin Migration View to see the affect of device migration on an I/O assignment

## END OF EXERCISE 5