# **Exercise Manual**

for

# **Quartus II Software Design Series:** Foundation

Software and hardware Requirements to complete all exercises

**Software Requirements:** Quartus II software version 7.1 **Hardware Requirements:** USB-Blaster<sup>™</sup> driver or ByteBlaster<sup>™</sup> & any Nios<sup>®</sup> or DSP development kit







#### **Objectives:**

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



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

- Unzip the lab project files. Double-click the executable file found in the .zip file you downloaded. In the WinZip dialog box, click Unzip to automatically extract the files in place to a new folder named
   C:\altera\_trn\Quartus\_II\_Software\_Design\_Series\_Foundation\QIIF7\_1. Close WinZip.
- Start the Quartus II software. In the Windows Start menu from the All Programs list, go to the Altera folder and then the Quartus II 7.1 folder. Click Quartus II 7.1 (32-Bit) to start the program.
- 3. Start the New Project Wizard. Under File, Select New Project Wizard.... A new window appears. If the Introduction screen appears, read it and click Next.
- 4. Complete the New Project Wizard to create the project. Page 1 of the wizard should be completed with the information from Table 1 below:

working directory for this project	Select only <b>one</b> of the following three directories, depending on the type of design entry you want to use throughout the lab exercises:	
	<lab_install_directory>\QIIF7_1\Ex1\VHDL</lab_install_directory>	
	<lab_install_directory>\QIIF7_1\Ex1\Verilog</lab_install_directory>	
	<lab_install_directory>\QIIF7_1\Ex1\Schematic</lab_install_directory>	
name of project	pipemult	
top-level design entity	pipemult	

Table 1. Settings for page 1 of New Project Wizard



#### **Quartus II Software Design Series: Foundation**

Ne	w Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5] 🛛 🛛 🔀
	What is the working directory for this project?
	Challens herkOverhall Coffman Design Codes Franchation/OUE7 1/Fra11/adice
	What is the name of this project?
	pipemult
	What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
	pipemult
	Use Existing Project Settings
	< Back Next > Finish Cancel

Page 1 of the New Project Wizard should look similar to the above.

- 5. Click **Next** to advance to page 2.
- 6. On page 2, click the browse button and select the top-level file **pipemult** (.v, .vhd, or .bdf, depending on the design entry method you chose in step 4). It should already be located in the project directory. After clicking **Open**, click **Add** to add the file to the project. Click **Next**.



Family:       Cyclone II         Target device       Package:         Auto device selected by the Fitter       Pin count:         Specific device selected in 'Available devices' list       Speed grade:         Fastest       Show advanced devices         HardCopy compatible only       Autoilable devices:						
Name	Core v	LEs	User I/	Memor	Embed	PLL
EP2C5F256C6	1.2V	4608	158	119808	26	2
EP2C8F256C6 EP2C15AF256C6 EP2C20F256C6	1.2V 1.2V 1.2V	8256 14448 18752	182 152 152	165888 239616 239616	36 52 52	2 4 4
<						
Companion device HardCopy II:	ardCopy II de	evice resour	ces			

7. On page 3, select **Cyclone II** 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 **EP2C5F256C6** device from the **Available devices:** window. Click **Next**.



New Project Wizard: EDA Tool Settings [page 4 of 5]				
Specify the other EDA tools in addit	ion to the Quartus II software used with the project.			
EDA design entry/synthesis tool:	Format:	<b>*</b>		
EDA simulation tool	Format: Verilog	4		
EDA timing analysis tool:	Format:	¥		
	< Back Next > Finish	Cancel		

8. On page 4 (above), 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 continue.

Project directory:	
C:/altera_trn/Quartus_II_Sol	ftware_Design_Series_Foundation/QIIF7_1/Ex1/Verilog/
Project name:	pipemult
Top-level design entity:	pipemult
Number of files added:	1
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone II
Device:	EP2C5F256C6
EDA tools:	
Design entry/synthesis:	<none></none>
Simulation:	<none></none>
Timing analysis:	<none></none>



9. The summary screen appears as shown. Click Finish. The project is now created.

#### **Exercise Summary**

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

## **END OF EXERCISE 1**







### **Objectives**:

- Create a multiplier and RAM block using the MegaWizard Plug-in Manager
- 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



For exercises 2-6, you can either continue working in the **Ex1** directory, or you can open the project (**File** menu  $\Rightarrow$  **Open Project**  $\Rightarrow$  Select **pipemult.qpf** and click **Open**) found in the **Ex#** directory. The **Ex#** directory contains a project completed up to that point in the exercise manual. 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 6.

# Step 1: Build an 8x8 multiplier using the MegaWizard<sup>®</sup> Plug-in Manager

1. Choose Tools  $\Rightarrow$  MegaWizard Plug-In Manager. In the window that appears, select Create a new custom megafunction variation. Click Next.

- 2. Select the megafunction to create. On **page 2a** (shown above), expand the **Arithmetic** folder and select **LPM\_MULT**.
- 3. In the drop-down menu, make sure the Cyclone II 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.



- 4. 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.
- 5. For the name of the output file, type **mult**.
- 6. Click Next.

MegaWizard Plug-In Manager - LP	M_MULT [page 3 of 7]
Version 7.1	T
1 Parameter 2 EDA 3 Summa Settings	ary
General Ceneral 2 Pipelining	>
mult	Currently selected device family: Cyclone II
dataa[70] result[150] unsigned multiplication	Multiplier configuration           Image: Multiply 'dataa' input by 'datab' input           Multiply 'dataa' input by itself (squaring operation)
	How wide should the 'dataa' input bus be? 8 💌 bits
	How wide should the 'datab' input bus be? 8 🔽 bits
	Create a 'sum' input bus with a width of
	How should the width of the 'result' output be determined?
	<ul> <li>Automatically calculate the width</li> </ul>
	○ Restrict the width to 16 v bits
Resource Usage	
	Cancel < <u>B</u> ack <u>N</u> ext > <u>F</u> inish

7. On page 3 (General), set the width of the dataa and datab buses to 8 bits. For the remaining settings in this window, use the defaults that appear. Click Next.



MegaWizard Plug-In Manag	er - LPM_MULT [page 4 of 7]
Version 7.	MULT .1 <u>About</u> Documentation
1 Parameter Settings General 2 Pipe	Summary
mult dataa[70] datab[70] Consigned multiplicati	Does the 'datab' input bus have a constant value?         • Nig         • Yes, the value is         • Which type of multiplication do you want?         • Unsigned         • Signed         Which multiplier implementation should be used?         • Use the default implementation         • Use dedicated multiplier circuitry (Not available for all families)         • Use logic elements
Resource Usage 1 dsp_9bit	Cancel < <u>B</u> ack <u>N</u> ext > <u>F</u> inish

8. On **page 4** (**General 2**), use all the default settings (i.e. <u>datab</u> input does <u>NOT</u> have a constant value, use unsigned multiplication, and select the default multiplier implementation). Click **Next.** 

MegaWizard Plug-In Mana	ger - LPM_MULT [page 5 of 7]
2 LPM_ Version 7	MULT .1 <u>About</u> <u>Documentation</u>
1 Parameter 2 EDA Settings General 2 Pipe	3 Summary
clock dataa[70] datab[70] result[1 Unsigned multiplicat	Do you want to pipeline the function? No Yes, I want an output latency of 2 clock cycles Create an asynchronous Clear input Create a Clock Enable input
	Which type of optimization do you want? Default     Speed     Area
1 dsp_9bit	Cancel < Back Next > Finish

9. On page 5 (Pipelining), choose Yes, I want an output latency of <u>2</u> clock cycles. Click Next.



MegaWizard Plug-In Manage	- LPM_MULT [page 6 of 7] EDA
LPM_M Version 7.1	ULT
1 Parameter 2 EDA 3 Settings	ummary
dataa(70) datab(70) resutt(15 Unsigned multiplication	Simulation Libraries To properly simulate the generated design files, the following simulation model file(s) are needed  File Description pm LPM megafunction simulation library pm LPM megafunctin simulation library pm LPM megafunction simulation librar
Resource Usage 1 dsp_9bit	Generate a netlist for synthesis area and timing estimation Cancel < Back Next > Einish

10. You are now on **page 6** (section 2 of the **MegaWizard** called **EDA**). This tab indicates the simulation model file needed to simulate **LPM\_MULT** in an EDA simulation tool (e.g. the **ModelSim-Altera** tool). The **lpm** simulation model file should be indicated as shown above. You also have the option of generating an early netlist for use by a 3<sup>rd</sup>-party synthesis tool. We are not using any third-party tools, so just click **Next**.



MegaWizard	Plug-In Manager -	LPM_MULT [page 7 of 7] -	- Summary	
2	LPM_MU Version 7.1	ILT	About Documer	ntation
1 Parameter Settings	2 EDA 3 Sun	nmary		
clock dataa[70] datab[70]	mult result[150] Unsigned multiplication	Turn on the files you wish to automatically generated, and Finish to generate the select subsequent MegaWizard Plug The MegaWizard Plug-In Mar directory:	generate. A gray checkmark indicates a file t d a red checkmark indicates an optional file. C ed files. The state of each checkbox is mainta g-In Manager sessions. Mager creates the selected files in the followin fitware. Decign Series Foundation/OTTET 115	nat is lick ined in g
		File	Description	
		✓ mult.v     mult.inc     mult.emp     ✓ mult.bsf     ✓ mult_inst.v     mult_bb.v     mult_bb.v     mult_waveforms.html        mult_wave*.jpg	Variation file AHDL Include file VHDL component declaration file Quartus II symbol file Instantiation template file Verilog HDL black-box file Sample waveforms in summary Sample waveform file(s)	
Resource U	sage		Cancel < <u>B</u> ack <u>N</u> ext >	Einish

11. On **page 7**, the following check boxes should be enabled to generate output files according to Table 2 below:

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

Table 2. MegaWizard files to generate

12. Click **Finish** to create the megafunction.

The multiplier is built.

#### Step 2: Create a 32x16 RAM using the MegaWizard Plug-In Manager

 Open the MegaWizard Plug-In Manager again (Tools ⇒ MegaWizard Plug-In Manager). Select to Create a new custom megafunction variation, and click Next.



#### **Quartus II Software Design Series: Foundation**

MegaWizard Plug-In Manager [page 2a]	
MegaWizard Plug-In Manager [page 2a] Which megafunction would you like to customize? Select a megafunction from the list below Minstalled Plug-Ins Altera SOPC Builder Markatoric Select a megafunctions Select a megafunction for the list below Select a meg	Which device family will you be Cyclone II  Which type of output file do you want to create? AHDL VHDL VHat name do you want for the output file? Browse rant Beturn to this page for another create operation
Hindows     JTAG-accessible Extensions     Memory Compiler     CAM     CAM     CAM     FIFO     FIFO     FIFO     FIFO partitioner     Flash Memory     AM: 1-PORT     AM: 3-PORT     RAM: 3-PORT     AM: 3-PORT	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 the global user libraries specified in the Uptions dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu). Your current user library directories are:
<ul> <li>A Shift register (RAM-based)</li> <li></li></ul>	Cancel < Back Next > Finish

- 2. Select the megafunction to create. On **page 2a** (shown above), expand the **Memory Compiler** folder and select **RAM: 2-PORT**.
- 3. As before with the multiplier, choose the **Cyclone II** device and **VHDL** or **Verilog HDL**.
- 4. For the name of the **output file**, type **ram**.
- 5. Click Next.

MegaWizard Plug-In Manager - RAM: 2-PO	RT [page 3 of 12]
RAM: 2-PORT	About Documentation
Parameter     Settings     General     Widths/Blk Type     Clks/Rd, Byte En	Regs/Clikens/Acirs > Output1 > Mem Init >
data(70) wraddress[40] rdaddress[40] rdaddress[40] clock Block Type: AUTO	Currently selected device family: Cyclone II  How will you be using the dual port ram? With one read port and one write port With two read/write ports
Resource Usage 1 M4K	How do you want to specify the memory size?   As a number of words  As a number of bits  Cancel < Back Next > Einish



6. On **page 3**, select **With one read port and one write port** in the section for how you will be using the dual port ram. For the remaining setting in this window, use the default (memory size specified as number of words). Click **Next**.

MegaWizard Plug-In Manager - RAM: 2-PORT [page 4 of 12]					
RAM: 2-PORT	About Documentation				
1 Parameter 2 EDA 3 Summary Settings					
General > Widths/Blk Type > Clks/Rd, Byte En	Regs/Clkens/Adrs > Output1 > Mem Init >				
ram data[150] wraddress[40] rdaddress[40] rdaddress[40] clock Block Type: AUTO	How many 16-bit words of memory? Use different data widths on different ports Read/Write Ports How wide should the 'q_a' output bus be? How wide should the 'data_a' input bus be? How wide should the 'q' output bus be? How wide should the 'q' output bus be?				
	What should the memory block type be?				
	Auto     O M512     O M4K				
	O M-RAM O LCs Options				
	Set the maximum block depth to Auto vords				
Resource Usage 1 M4K	Cancel < <u>B</u> ack <u>N</u> ext > <u>E</u> inish				

 On page 4 (Widths/Blk Type), choose 32 as the number of 16-bit words of memory. (Note: This will read "<u>8-bit words of memory</u>" until you change the width of the read/write ports next.) The width of the Read/Write Ports for the data\_a input should be set to 16. Leave the memory block type selection set to its default of Auto. Click Next 2 times.

MegaWizard Plug-In Manager - RAM: 2-PORT [page 7 of 12]						
RAM: 2-PORT	About Documentation					
1 Parameter 2 EDA 3 Summary Settings						
General $ ightarrow$ Widths/Blk Type $ ightarrow$ Clks/Rd, Byte Er	n 🔪 Regs/Clkens/Aclrs 🔪 Output1 🔪 Mem Init 🔪					
	Which ports should be registered?					
ram data[150] wraddress[40] wren rdaddress[40] clock Block Type: AUTO	<ul> <li>Write input ports         'data', 'wraddress', and 'wren'</li> <li>Read input ports         'rdaddress' and 'rden'         <u>Read output port(s)         'q'         'q'         </u></li> </ul>					
	Create one clock enable signal for each clock signal More Options					
Resource Usage 1 M4K	Create an 'actr' asynchronous clear for More Options  Cancel < Back Next > Einish					

8. On page 7 (Regs/Clkens/Aclrs), <u>disable</u> the option to register the Read output port(s) 'q'. Accept the remaining default settings, and click Next 2 times.



MegaWizard Plug-In Manager - RAM: 2-PORT [page	ge 10 of 12] 🛛 🛛 🗙
RAM: 2-PORT Version 7.1	About Documentation
1 Parameter     Settings     General > Widths/Blk Type > Clks/Rd, Byte En > Reg	js/Clkens/Actrs > Output1 > [Mem Init] >
data(15.0) wraddress(4.0) wreaddress(4.0) rdaddress(4.0) clock Block Tunor (410)	Do you want to specify the initial content of the memory?     No, leave it blank     Initialize memory content data to XXX 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
Block Type: A010	File name:       ram.hex         The initial content file should conform to which port's dimensions?       PORT_B
	Cancel < <u>B</u> ack Next > Finish

- 9. On page 10 (Mem Init), click Yes that you have a file to use for the memory content data. Once enabled, type in the file name ram.hex. Click Next.
- 10. 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.
- 11. Choose the same file extensions for ram as you selected for mult earlier (Step 1, #11). Click Finish.

You have now created the two components needed for this design. Now you will create the HEX file needed to initialize the contents of the RAM.

#### **Step 3: Create HEX file using Memory Editor**

1. From the **File** menu, select **New** or click in the toolbar.

Device Design Files Uther Files          AHDL Include File         Block Symbol File         Chain Description File         Hexadecimal (Intel-Format) File         In-System Sources and Probes Editor File         Logic Analyzer Interface File         Memory Initialization File         Synopsys Design Constraints File         Text File         Vector Waveform File	New	ou su l		2
OK Cancel	Device Design Files AHDL Include File Block Symbol File Chain Description Fil Hexadecimal (Intelf In-System Sources a Logic Analyzer Interf Memory Initialization SignalT ap II Logic A Symopsys Design Cc T cl Script File Text File Vector Waveform Fil	Other Files e ormat) File ind Probes Editor ace File File nalyzer File instraints File e	File	
			ОК	Cancel

#### **Quartus II Software Design Series: Foundation**

2. In the New dialog box, click the Other Files tab. Select Hexadecimal (Intel-Format) File. Click OK.

Number of Words & Word Size					
Number of words:	32				
Word size:	16				
ОК	Cancel				

3. In the memory size dialog box, choose **32** as the **number of words** and **16** as the **word size**. Click **OK**.

😤 Qua	🖀 Quartus II - C:/altera_trn/Quartus_II_Software_Design_Series_Foundation/QIIF7_1/Ex2/Schematic 🔳 🗖 🔀															
File Ed	lit View	Projec	t Tools	Windo	W											
Addr	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
16	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

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

- 4. Highlight all of the memory locations in your memory space. Right-click and select **Custom Fill Cells**.
- 5. Use the **Custom Fill Cells** dialog box to enter your own values to initialize your memory. Do one of the following:
  - a. <u>Repeating Sequence</u>: Enter a series of numbers separated by commas or spaces to be repeated in memory
  - b. <u>Incrementing/Decrementing</u>: Enter a start value and another value by which to increment or decrement the start value
- 6. Save the file as **ram.hex**. Close ram.hex.



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

*Choose* <u>*ONE*</u> *of the following procedures based on your design entry method (VHDL, Verilog, or Schematic).* 

## **VHDL**

Open pipemult.vhd. You can use the Open command from the File menu, click the
 toolbar button, or double-click the entity in the Project Navigator.

This is the top-level file for the design. Normally, you would have to instantiate <u>both</u> ram and mult and connect 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 (line 22).
- 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 (line 51). Change the following signal names in the instantiation:

clock_sig	to	clk1
dataa_sig	to	dataa
datab_sig	to	datab
result_sig	to	mult_to_ram

- 5. Close mult\_inst.vhd.
- 6. Save **pipemult.vhd**.

### Verilog

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.

This is the top-level file for the design. Normally, you would have to instantiate <u>both</u> **ram** and **mult** and connect 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 (line 24). Change the following signal names in the instantiation:

clock\_sig to clk1

#### Quartus II Software Design Series: Foundation

dataa_sig	to	dataa
datab_sig	to	datab
result_sig	to	mult_to_ram

- 3. Close mult\_inst.v.
- 4. Save **pipemult.v**.

#### **Schematic**

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.

This is the top-level schematic file for the design. Normally, you would have to instantiate both **ram** and **mult** subdesigns and connect 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].

In the schematic file, double-click any empty area so that the Symbol window appears. Inside the Symbol window, click 
 *i* to expand the symbols defined in the Project folder. Double-click the mult symbol. Click the left mouse button to place the symbol inside the schematic file where indicated.

<u>Note</u>: 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.

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

<u>Note</u>: 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, 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**).

olk1         NEVT           dataa[7.0]         NEVT           datab[7.0]         NEVT	dataal701 databl701 multiplication multiplication		
wraddress[40]         ND IT           wren         ND IT           rdaddress[40]         ND IT           rdaddress[40]         ND IT		data[15.0] wraddress[4.0] wren rdaddress[4.0] clock ram_ins <sup>Block</sup> Type: AUTO	QUTEUT q[150]

Your resulting schematic should look like the above figure.

9. Save pipemult.bdf.

#### **Step 5: Save and check the schematic**

1. From the **Processing** menu, select **Start**  $\Rightarrow$  **Start Analysis & Elaboration**.

Analysis and elaboration checks that all the design files are present and connections have been made correctly.

2. Click **OK** when analysis and elaboration is completed. If there are any errors, check your connections or return to the MegaWizard Plug-In Manager for either megafunction to fix the problem.



#### **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







#### **Objectives**:

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



	Compilation Report
Device Name	EP2C5F256C6
Total Design	
Total Logic Elements	
Total Memory Bits	
Total Embedded Multiplier 9-Bit Elements	
Total Pins	
mult subdesign	
Logic Cells (mult)	
Dedicated Logic Registers (mult)	
Memory Bits (mult)	
M4Ks (mult)	
DSP Elements (mult)	
ram subdesign	
Logic Cells (ram)	
Dedicated Logic Registers (mult)	
Memory Bits (ram)	
M4Ks (ram)	
DSP Elements (ram)	
Control signals & for out	
Control signals & fan-out	



#### Step 1: Compile the design

- 1. Select **Start Compilation** from the **Processing** menu or **click** located on the toolbar to perform a full compilation of the design. A dialog box will appear when the compilation is complete.
- 2. Click OK.

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

The Compilation Report provides <u>all</u> 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. By default, the **Compilation Report** opens when any processing begins and displays the Flow Summary Section when that process finishes.

1. From the Flow Summary section of the Compilation Report, record the Total logic elements, total memory bits, total embedded multiplier 9-bit elements and total pins in the table at the beginning of this exercise.

From these results you can see that this design is currently using mostly dedicated resources (i.e. embedded memory, embedded multipliers) and hardly any logic.

- 2. Expand the **Fitter** folder in the **Compilation Report**. Locate the **Resource Section** folder. From the **Resource Utilization by Entity** table, record in the exercise table the requested resource counts for the **mult** and **ram** subdesigns.
- 3. From the **Control Signals** table, record the control signals found and their fan-out.

Though the clock in this design obviously drives more than 3 registers, the fan-out of 3 refers to the number of architectural blocks it drives: 1 memory block, 1 embedded multiplier, and 1 logic cell.

In the next few steps, you will take a look at some additional ways to analyze the results of your compilation.

#### 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 should see the diagram shown above. 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** subdesigns, and an additional set of output registers. Notice the registers are external to the memory block per the original design.

2. Select the **ram** subdesign to highlight it. 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** subdesign again if it was deselected. Right-click and select **Group Related Nodes**.

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

4. Double-click on the **ram** subdesign.





You have now descended the hierarchy into the **ram** subdesign. As a result, the view changes to the above or similar image (may appear differently for VHDL). This shows that the ram subdesign 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.

5. 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.

6. Select the ram subdesign again. Right-click and select Display Content.



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



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

Descending the hierarchy again, the internal logic of the **altsyncram** module (**altsyncram\_coq1**) is displayed including module I/O that are not connected.

8. Select the altsyncram\_coq1 module, right-click, and select Display Content.



Moving down into the **altsyncram\_coq1** 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 be implemented in the target **Cyclone II** device.





Notice also in the **RTL Viewer** that the **Hierarchy List** shows how we have descended down into the **ram** subdesign. For each hierarchical level, you can also see the names of its pins and nets.
#### Step 4: Explore the ram subdesign 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.

 In the RTL Viewer, locate and highlight the innermost hierarchical ring for the ram function (altsyncram\_coq1). Right-click and select Locate ⇒ Locate in Technology Map Viewer.



The **Technology Map Viewer** opens and displays the **altsyncram\_coq1** module.

2. Descend the hierarchy into the **altsyncram\_coq1** 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 as well as the output. The output? Weren't the output registers outside of the memory block in the **RTL Viewer**? The fitter has moved those registers into the memory block to improve memory block performance and design density. This behavior is called register packing, and the Fitter does this to help save device resources. There is a fitter-generated "Extra Info" message in the **Message** window (possibly in the **Suppressed** tab) that also indicates this optimization was performed.

*This could be further verified by viewing ram\_block1a0 in the Resource Property Editor.* 

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 subdesign

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** 



#### Quartus II Software Design Series: Foundation

**Planner** can be used for manually locating logic, we currently only want to evaluate results.

- In the Technology Map Viewer, select the RAM block (ram\_block1a0) to highlight it. Right click and select Locate ⇒ Locate in Chip Planner (Floorplan & Chip Editor).
- 2. In the **Chip Planner**, use the zoom tool (Chip Planner toolbar) and right-click to zoom out a few times.



You should now see the ram subdesign (highlighed in blue) and where it has been placed in the **Cyclone II** device. The other shaded areas in the **Chip Planner** are where the remaining logic has been placed. Since we don't have any constraints applied to the design, the fitter was free to place logic anywhere. This will change later.

3. With the memory block highlighted in the **Chip Planner**, click the **Generate Fan-In Connections** toolbar button

The **Chip Planner** now displays the device resources that are feeding into the **ram** subdesign (device I/O and the multiplier) along with the delays incurred between them.

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





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

- 5. Turn off the fan-in/fan-out by clicking on any non-highlighted part of the device and then clicking the **Clear Unselected Connections/Paths** toolbar button **Selected Connections**.
- 6. 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 analyzed design processing results in different ways using the RTL Viewer, Technology Map Viewer & Chip Planner

# END OF EXERCISE 3









٦

Г

#### Exercises

# **Exercise 4**

# **Objectives:**

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

## Table 2

	Compilation Report
Device Name	EP2C5F256C6
Total Design	
Total Logic Elements	
Total Memory Bits	
Total Embedded Multiplier 9-Bit Elements	
Total Pins	
mult subdesign	
Logic Cells (mult)	
Dedicated Logic Registers (mult)	
Memory Bits (mult)	
M4Ks (mult)	
DSP Elements (mult)	
ram subdesign	
Logic Cells (ram)	
Dedicated Logic Registers (mult)	
Memory Bits (ram)	
M4Ks (ram)	
DSP Elements (ram)	
Control signals & fan out	
Control signals & fail-out	



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

In order to try different constraint options and see how they affect your results, the Quartus II software has support for creating revisions, with each revision building a new QSF file. You can also quickly compare the results of your various revisions.

- 1. From the **Project** menu, select **Revisions**.
- 2. In the **Revisions** dialog box, click on the **Create** button.

Create Revision					
Specify a name and do You can base the revision as	escription for the new revision. sion on an existing revision, and the current revision.				
Revision name:	pipemult_lc				
Based on revision:	pipemult 🗨				
Description:					
Created on: Monday, May 07, 2007 Based on : pipemult					
<ul> <li>Copy database</li> <li>Set as current revi</li> </ul>	sion OK Cancel				

- 3. Type in **pipemult\_lc** as the **Revision name**. Leave all other defaults and click **OK**.
- 4. Click **OK** to close the **Revisions** dialog box.

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

Cyclone II embedded multipliers are a valuable resource for implementing multiply operations in the FPGA. They provide a better usage of resources for multiplication over logic elements. But, embedded multipliers are limited in number. 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** flow, or it can be done on a multiplierby-multiplier basis using the Assignment Editor logic option.

1. In the Project Navigator, expand pipemult.





- Right-click on the mult entity in the hierarchy and choose Locate ⇒ Locate in Assignment Editor. The mult entity is listed in its own row in the Assignment Editor, ready for assignment creation.
- 3. Double-click the cell in the Assignment Name column for mult:mult\_inst (row 1) and select DSP Block Balancing from the drop-down menu.
- 4. Double-click the cell in the **Value** column for the **DSP Block Balancing** option and select **Logic Elements** from the drop-down menu.
- 5. Double-click the cell on the second row (created automatically after completing the assignment in the first row) in the **Assignment Name** column for **mult:mult\_inst** and select **Optimization Technique**.
- 6. Double-click the cell in the Value column for Optimization Technique and select Speed.
- 7. Save the Assignment Editor file.



#### **Quartus II Software Design Series: Foundation**

🎸 Assig	nment Ed	litor						
<b></b>	<u> </u>	× Category: All Category: All S Category: All S Category: All S Category: S C						
0 🔿	×	Show assignments for s	pecific nodes:					
	R	✓ ⓒ mult:mult_inst	:			Check All		
	臣					Uncheck All		
T#T	ter:					Delete All		
सं स								
	± Ir	formation: Allows you	to view and edit assign	nments for specific nodes ar	nd entites. Specify one o	r more node or entity nam		
٤	Edit:							
	7	From	То	Assignment Name	Value	Enabled		
	1		mult:mult_inst	DSP Block Balancing	Logic Elements	Yes		
	2 Optimization Techniq Speed Yes							
	3		mult:mult_inst			Yes		

Your Assignment Editor window should look similar to above.

#### **Step 3: Recompile the design**

1. Click to compile the design.

#### **Step 4: Gather resource information from the Compilation Report**

- 1. From the Flow Summary section of the Compilation Report, record the Total logic elements, total memory bits, total embedded multiplier 9-bit elements and total pins in the table at the beginning of this exercise.
- 2. Expand the **Fitter** folder in the **Compilation Report**. Locate the **Resource Section** folder. From the **Resource Utilization by Entity** report, record in the exercise table the requested resource counts for the **mult** and **ram** subdesigns.
- 3. From the **Control Signals** table in the **Compilation Report**, record the control signals found and their fan-out.

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



# **Exercise Summary**

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

# **END OF EXERCISE 4**









# **Objectives**:

- Assign I/O pins and perform I/O Assignment Analysis
- Use the Pin Migration View to see the affect 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, open the Pin Planner.

🨻 Q File	<mark>uartus II</mark> Edit View	- C:/altera_trn/Qua Tools Window	rtus_II_Software_Design_	Series_Foundation/	QIIF7_1/Ex5/Verilog/	'pipemult - pipemu	lt_lc - [Pin 🔳	
1 S	Groups Named:	Node Name	Direction	C	- Top View Cyclone II - E	Wire Bond P2C5F256	1 6C6	
e 🖩 🗗 📜 🕈 🕈 🗤 🗤		dataa[7]         dataa[7]           dataa[7]         dataa[6]           dataa[6]         dataa[6]           dataa[3]         dataa[3]           dataa[2]         dataa[1]           dataa[0]         dataa[0]           dataa[7]         dataa[1]           dataa[0]         dataa[1]           dataa[0]         dataa[1]	Input Group Input Output Group Input Group Input Group					
📰 📰 🐻 🧱 🔤 👯 🖼		wraddress(=.Uj < <new node="">&gt;</new>	Input Group					
2	× Named:	JI .	▼ «≫ Edit: X ✓			Filte	er: Pins: all	-
24		Node Name	Direction	Location	I/O Standard	I/O Bank	Vref Group	~
5	1 2 3 4		Input Input Input Input		3.3-V LVTTL (default) 3.3-V LVTTL (default) 3.3-V LVTTL (default) 3.3-V LVTTL (default)			
	5 6 27	dataa[4]     dataa[3]     dataa[2]	Input Input Input		3.3-V LVTTL (default) 3.3-V LVTTL (default) 3.3-V LVTTL (default) 3.3-V LVTTL (default)			
	All Pir	1	The second secon	1				>

- 2. Make sure the **Top View** of the device is displayed. 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 has been enabled. If you are unsure of which button this is, you can also find this option in the **View** menu.

🨻 Q	uartus II - C:/altera_trn/Quart	us_II_Software_Desi	gn_Series_Foundation/QIIF7_1/Ex5/Verilog/pipemult - pipemult_lc - [Pin 📃 🗖 🗙
File	Edit View Tools Window		
	Groups Named:	<u>×</u>	Top View - Wire Bond
R	Node Name	Direction	
Ð,	🔂 🗉 🛛 🔂 🔤	Input Group	
		Inpuesa	
	i di	Output Group	
1		Input Group	27.000
E		Input Group	(Carsigned/3 bba)
	< <new node="">&gt;</new>		1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
-1			
ı.			^ / ∕ <b>0 9 9 9 0 0 0 0 0 0 0 0 0 0</b>
₽			▫◭▽◓๏๏๏๙×๏๏๏๏๏๙▽๙▫
1			

4. 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** into the **IOBANK\_2** box of the **Package View** (the top side of the chip).



The display for IOBANK\_2 will now change to indicate that I/O Bank 2 has 8 assigned pins out of 43 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.

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

Node Properties	X
Node name:	dataa[70]
Location:	IOBANK_2
1/0 standard:	2.5V 💌
Reserved:	•
Properties:	
Name	Value
1	
	OK Cancel

6. In the **Node Properties** dialog box, set the **I/O standard** for **dataa** to **2.5** V. Click **OK**.

×	Named:	1	<ul> <li>✓ «»</li> </ul>	Edit: 🗙 🗸		$\frown$				Filter: F	ins: all	•
			Node Name	Direction	7	Location		I/O Standard		I/O Bank	Vref Group	~
	2		dataa[7]	Input		IOBANK_2		2.5 V		2		
	3		dataa[6]	Input		IOBANK_2	V	2.5 V		2		
	4		dataa[5]	Input		IOBANK_2		2.5 V		2		
	5		dataa[4]	Input		IOBANK_2		2.5 V		2		
	6		dataa[3]	Input		IOBANK_2		2.5 V		2		
	7		dataa[2]	Input		IOBANK_2	Λ	2.5 V	1	2		
	8		dataa[1]	Input		IOBANK_2		2.5 V	1	2		
<u>1</u>	9		dataa[0]	Input		IOBANK_2		2.5 V		2		~
AIP	<							$\smile$			>	

Notice that the All Pins list reflects the I/O standard change for all of the individual dataa pins. You could have assigned the location and changed the I/O standard using the All Pins list as well.

- 7. Using either the **Package View** or the **All Pins List**, assign the **datab** bus to **IOBANK\_2**.
- 8. Set the **I/O Standard** for the **datab** bus to **1.8 V** using the **Node Properties** dialog box.
- 9. Assign the **q** output bus to **IOBANK\_3** and set the **I/O standard** to **1.8 V**.
- 10. Assign both **read** and **write addresses** to **IOBANK\_3** and set **the I/O standard** to **1.8 V**.
- 11. Using the All Pins List, assign clk1 to Pin H16 and set the I/O standard to 1.8 V.
- 12. Using the All Pins list, assign wren to IOBANK\_3 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 you have made general I/O assignments, you can check the validity of those assignments without running a full compilation. This way you can quickly and easily find I/O placement issues and correct them.

- From the Processing menu, go to Start and select Start Analysis & Synthesis or click on the solution in the main Quartus II toolbar. Click OK when complete.
- 2. From the **Processing** menu, go to **Start** and select **Start I/O Assignment Analysis** or click on the button in the Pin Planner toolbar. Click **OK** once the analysis is complete.



- Hassagas			<b>•••</b>				
Туре	Flag	Nessage	^				
🕕 🗉		Info: Fitter converted 3 user pins into dedicated programming pins					
🕀 🔥		Warning: No exact pin location assignment(s) for 43 pins of 44 total pins					
🗉 👿		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank					
🗉 😟		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank					
🗉 😟		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank					
E 😣		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal WCCIO value for the bank					
🗉 😟 💮		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank					
🗉 🗄 🖸		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank					
🗉 😟		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank	=				
🗉 🗉 🧑		Error: I/O bank 2 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank					
a de la companya de l		Error: Can't fit design in device					
Ā		Warning: The Reserve All Unused Pins setting has not been specified, and will default to 'As output driving ground'.					
🗉 🗉 👿		Error: Quartus II I/O Assignment Analysis was unsuccessful. 9 errors, 2 warnings					
-			~				
<			>				
System Proce	System A Processing / Extra Info / A Info / Warning / Critical Warning / Error / Suppressed / Flag /						
Message: 0 of 104	5		te I				

Was the analysis successful? Check the messages in the **Messages** window or the **Fitter Messages** in the **Compilation Report**. They should read as shown above.

3. 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 II** 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 II** device or general FPGA experience, then steps 4-6 will show you how to correct the errors.

4. Bring the **Pin Planner** to the foreground again.

Notice that you have assigned the **dataa** and **datab** input buses to **I/O Bank 2** but set different **VCCIO** (1.8 & 2.5) voltage levels for them. Cyclone II FPGAs, like all Altera FPGA devices, allow for only <u>one VCCIO per I/O bank</u>.

5. Using the **Groups List**, expand the **datab** group and change the **I/O standard** setting for the group and all the individual **datab** signals to **2.5** V. Make sure the **datab** group as well as each individual signal in the group is set to 2.5 V.

Tip: Try changing the I/O standard for just **datab**[7..0] in the Groups List and placing your cursor at the lower right corner of the I/O standard cell for this signal. You can use auto-fill, a standard spreadsheet operation, to fill in the changed I/O standard for the rest of the **datab** bus located below **datab**[7..0].

6. 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!

7. To see where the fitter placed your I/O, click on the 2 button in the **Pin Planner** toolbar or from the **View** menu choose **Show**  $\Rightarrow$  **Show Fitter Placements**.





You can see that fitter-selected pins appear in green in the **Pin Planner** as shown above.

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

This is the step you would use once you have produced a verified 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, select Back-Annotate Assignments to open the Back-Annotate Assignments dialog box. The dialog box may appear slightly different from the screenshot below.



Back-Annotate Assignments	×
Back annotation type: Default	•
<ul> <li>Device assignment</li> <li>Pin &amp; device assignments</li> <li>Pin, cell &amp; device assignments to: LABs</li> <li>Pin, cell, routing &amp; device assignments to: LABs</li> <li>Pin, cell, routing &amp; device assignments</li> <li>PLL location assignments</li> <li>Delay chains</li> <li>DSP Balancing</li> <li>RAM Packing</li> <li>Global Signal</li> <li>Auto Packed Registers</li> </ul>	
Save intermediate synthesis results Save a node-level netlist of the entire design into a persistent source file File name:	
OK Cancel	

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 a 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). To do this, you will export the assignments as a .CSV file and import them into the original pipemult revision.

1. With the **Pin Planner** as the active window, go to the **File** menu and select **Export**. In the **Export** dialog box, type the filename **io\_assignments.csv** and click **Export**.



2. Use the drop-down menu at the top of the **Quartus II** window to change the revision back to **pipemult**.



3. From the Assignments menu, select Import Assignments.

mport Assignments				
Specify the source and categories of assignments to import. Click LogicLock Import to select LogicLock Import File(s).	rt File Assignments			
Assignment source     File name: undation/QIIF7_1/Ex1/Verilog/io_assignments.csv	Categories			
C Use LogicLock Import File Assignments	Advanced			
LogicLock Import File Assignments				
Copy existing assignments into pipemult.qsf.bak before importing				
OK	Cancel			

- 4. In the **Import Assignments** dialog box, click the browse button by **File name:** and choose **io\_assignments.csv**. Click **Open** and then **OK**.
- 5. Open the **Pin Planner** to see that the assignments were imported correctly.



Remember that the green pins are the fitter-placed locations from the previous compilation of the revision **pipemult**. The red pins are the pins imported from the **pipemult\_lc** revision, and the green and red hatched pins are intersecting



#### Quartus II Software Design Series: Foundation

assignments between the two revisions of the project. The green locations will all move during the next compile.

6. Turn off the viewing of fitter placements by clicking in the **Show** submenu of the **View** menu.

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

Now you'll see what happens to a pin when the pin's function changes 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, and reopen the Pin Planner.
- 2. In the Package View, look at pin L12 to determine whether a signal is already assigned to this pin. If the pin is white, no signal is assigned to this pin. Proceed to the next step. If a signal is already assigned to this pin, skip to 7.
- 3. In the All Pins list, click the **New Node** solution, or scroll to the bottom of the list, and click in the **<<new node>>** cell in the Node Name column.
- 4. Name the new node **reserved\_pin**.
- 5. Double-click the cell in the **Location** column for this new node, and select **PIN\_L12** from the list.
- 6. In the **Reserved** column, set the reserve type to **As input tri-stated**.

In the Package View, you should now see that pin L12 is colored blue, indicating that the pin is reserved.

7. From the View menu, select Pin Migration View.



Pin Migra	ation View					×			
	Current Device: EP2C5F256C6								
	Pin	No	Migration	1		^			
	Number	Pin Function	1/0 Bank	VREF Group					
1	PIN_A1	GND							
2	PIN_A2	VCCI02	2						
3	PIN_A3	Column I/O	2	B2_N1					
4	PIN_A4	Column I/O	2	B2_N1					
5	PIN_A5	Column I/O	2	B2_N1					
6	PIN_A6	Column I/O	2	B2_N1					
7	PIN_A7	Column I/O	2	B2_N1					
8	PIN_A8	Column I/O	2	B2_N1					
9	PIN_A9	Column I/O	2	B2_N0					
10	PIN_A10	Column I/O	2	B2_N0					
11	PIN_A11	Column I/O	2	B2_N0					
12	PIN_A12	Column I/O	2	B2_N0					
13	PIN_A13	Column I/O	2	B2_N0					
14	PIN_A14	Column I/O	2	B2_N0					
15 PIN_A15 VCCI02 2				v					
Devid	Device Pin Finder Show only highlighted pins Show migration differences								

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

- 8. Click **Device** to open the Settings dialog box to the Device category.
- 9. Click Migration Devices, and turn on the option to Show all speed grades.

Migration Devices	
Select the migration device(s) for the current device. W of the migration devices you select. Note: Specifying migration devices can reduce the likel	hen the Compiler processes your project, it will be compatible with all ihood of achieving a successful fit.
Current device: EP2C5F256C6	
Compatible migration devices:	Selected migration devices:
EP2C5F256C7 EP2C5F25608 EP2C5F25608 EP2C8F256C6 EP2C8F256C7 EP2C8F256C8 EP2C8AF25608 EP2C8AF25608	>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
🔽 Show all speed grades	
Base the Pin-Out File (.pin) and floorplan package vi	iews on the largest selected SameFrame device
	OK Cancel

- Select the EP2C8F256I8 from the list, and double-click it or click > to move it to the list of migration devices. Click OK. Click OK again to close the Settings dialog box.
- 11. In the Pin Migration View window, you can now see pin information about the migration device. Turn on the **Show migration differences** option and look for pin L12.



#### Quartus II Software Design Series: Foundation

Pin Migration View											
Current Device: EP2C5F256C6											
			JL	Migration Devices					~		
	Pin	Migration Result		EP2C5F256C6			EP2C8F256I8				
	Number	Pin Function	1/0 Bank	VREF Group	Pin Function	1/0 Bank	VREF Group	Pin Function	1/0   Bank	VREF Group	
13	PIN_G4	NC			NC			Row I/O	1	B1_N0	
14	PIN_H6	NC			NC			Row I/O	1	B1_N0	
15	PIN_J6	NC			NC			Row I/O	1	B1_N0	
16	PIN_J10	VCCINT			NC			VCCINT			
17	PIN_K6	NC			NC			Column I/O	4	B4_N1	
18	PIN_K7	NC			NC			Column I/O	4	B4_N1	
19	PIN_K8	VCCINT			NC			VCCINT			
20	PIN_K13	NC			NC			Row I/O	3	B3_N1	
21	PIN_L12				Column I/O	4	B4_N0	Row I/O	3	B3_N1	
22	PIN_N3	NC			NC			Row I/O	1	B1_N1	
23	PIN_N4	NC			NC			Row I/O	1	B1_N1	
24	PIN_N6	NC			NC			Column I/O	4	B4_N1	
25	PIN_N7	NC			NC			Column I/O	4	B4_N1	
26	PIN_P6	NC			NC			Column I/O	4	B4_N1	
27	PIN_R6	NC			NC			Column I/O	4	B4_N1	~
Devid	Device Pin Finder Show only highlighted pins 🔽 Show migration differences										

Notice that this pin moves from I/O bank 4 in the current device to I/O bank 3 in the migration device. Thus, the **Migration Result** columns are blank, indicating that a signal on this pin cannot migrate to the faster chip. Let's verify this with I/O Assignment Analysis.

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

What happened? I/O Assignment Analysis failed because the assignment on pin L12 cannot exist both in the current device and the migration device. If you now look at pin L12 in the Package View, you'll notice an X on the pin, indicating it as NC, or no connect. Even though a signal can be assigned to this pin in one device or the other, the Pin Planner prevents us from assigning it here if the faster device is used as a migration device because the functionality of the pin changes.

13. Switch the revision back to **pipemult** and close the Pin Planner.

#### **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**









# **Objectives:**

- Follow the steps to using TimeQuest
- Apply constraints to a design using TimeQuest GUI
  - create\_clock
  - *set\_input\_delay*
  - *set\_output\_delay*

# Table 3

	Worst Setup Slack	Worst Hold Slack
pipemult revision		
pipemult_lc revision		
pipemult_lc_phys_syn revision		



## **Step 1: Synthesize the design**

1. Verify that you are using the **pipemult** revision and click **v** to synthesize the design.

Though you could also perform a full compilation, performing synthesis allows you to quickly generate a netlist in order to start constraining. And even though the pipemult revision has been previously compiled, we want to highlight the recommended tool flow.

# Step 2: Start TimeQuest and setup timing netlist for analysis

1. From the main Quartus II toolbar, click <sup>(1)</sup> or, from the **Tools** menu, select **TimeQuest Timing Analyzer**. Click **No** that you do not want to generate an .SDC file from a .QSF file.



The window above opens. You will now go through the steps to using **TimeQuest**.



#### Quartus II Software Design Series: Foundation

Create Timing Netlist						
Dinput netlist	● Slow corner					
Post-map	Speed grade:					
Tcl command:       create_timing_netlist -post_map -model slow -zero_ic_del         OK       Cancel						

2. Create a timing netlist. From the **Netlist** menu, select **Create Timing Netlist** and change the **Input netlist** type to **Post-map** <u>OR</u> in the **Console** pane, type **create\_timing\_netlist –post\_map**.

A green checkmark appears next to **Create Timing Netlist** in the **Tasks** pane to indicate the command was successful. Notice there is a message in the **Console** pane (in blue) indicating that **TimeQuest** is not the default timing analysis tool. You will correct that later in the exercise. You can also double-click on **Create Timing Netlist** in the **Tasks** pane, but that would not have given you the option to use a post-map netlist, so it was not used in this case.

3. Read an SDC file. Simply type **read\_sdc** at the **tcl>** prompt in the **Console** pane.

A message appears indicating that an SDC file has not been found. This is correct. Since you did not specify a filename, **TimeQuest** automatically looked for any SDC files that were added to the project and then an SDC file sharing the same name as the current revision **pipemult**, neither of which exists.

There should now be a green checkmark next to **Read SDC File** indicating you tried to read an SDC file. Instead, you will enter SDC commands directly in the GUI.

While entering the following commands, double-check that you've entered the command correctly. Otherwise, you may have to recreate the netlist and start over. The use of an SDC file would prevent this problem since all commands are stored and can be edited at any point. However, here we want to examine the most basic use of TimeQuest by entering commands directly.





Your **Tasks** pane should look like the above.

# Step 3: Manually add clock constraint

- 1. Use the Create Clock command to add a 6 ns clock constraint to the clk1 input.
  - a. From the Constraints menu in TimeQuest, select Create Clock...
  - **b.** In the **Create Clock** dialog box, type **clk1** as the clock name.

We are using the same name as the clock node but you can name the clock whatever you want.

**c.** In the **period** field, type **6**.

Notice you can enter the waveform edges to create a non-50% duty-cycle clock. We will leave that blank since this clock's duty cycle is 50%.

**d.** In the **targets** field, click on the browse button

This opens the Name Finder window.

- e. In the Name Finder, choose get\_ports from the Collection drop-down menu. In the Matches section, click List.
- **f.** Double-click on **clk1** in the list of matches.



#### Quartus II Software Design Series: Foundation

Name Finder					X
Collection:	get_ports	Filter:			
Options Case-inse Hierarchic Compatibi	nsitive :al lity mode				
Matches List	]				
44 matches clk1 dataa[0] dataa[1] dataa[2] dataa[3] dataa[4] dataa[5] dataa[6] dataa[7] datab[0] datab[1] datab[2] datab[3] datab[4]	ound	>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>	elected name		
SDC command:	[get_ports {clk1}]				
			ОК	Cancel	Help

Notice the bottom of the dialog box shows the SDC command for the target you selected.

g. Click OK.

Create Clock						X
Clock name:	clk1					
Period:	6	ns			٦	
-Waveform edges-						
Rising:		ns				
Falling:		ns	0.0	0	3.00	6.00
Targets:	[get_ports {clk1}]					
SDC command: create_clock -period 6 -name clk1 [get_ports {clk1}]						
				ОК	Cancel	Help

The **Create Clock** dialog box should appear as shown above. Again at the bottom, the SDC equivalent command for the options set is displayed.

h. Click OK.

Your clock constraint has now been added. At this point, you could update the timing netlist. But, we will add the I/O constraints first.



### Step 4: Manually add I/O constraints

- 1. Use the **Set Input Delay** command to set a **maximum** input delay of **3.25** ns to both 8-bit input buses **dataa** and **datab**, with respect to **clk1**.
  - a. From the **Constraints** menu in **TimeQuest**, select **Set Input Delay**.
  - b. Fill in the window with the following information:

Clock name	=	clk1 (use drop-down)
Input Delay Options	=	Maximum (Leave Rise/Fall to Both)
Delay Value	=	3.25
Targets	=	[get_ports data*]

You could have used the browse button and Name Finder to locate all of the data inputs (like you did with **clk1**), but sometimes, especially with buses, it is just easier to type in a command directly and use wildcards. If you didn't know the name of the bus, you could still use the Name Finder and add the wildcard in the Name Finder SDC command field after selecting the signals that make up the bus.

Set Input Delay 🛛 🔀						
Clock name:	Cik1					
Input delay optic	ns					
<ul><li>C Minimum</li><li>● Maximum</li><li>○ Both</li></ul>	<ul> <li>C Rise</li> <li>C Fall</li> <li>C Both</li> </ul>					
Delay value:	3.25 ns 🗆 Add delay					
Targets:	[get_ports data*]					
SDC command: set_input_delay -clock clk1 -max 3.25 [get_ports data*]						

Your Set Input Delay dialog box should look like the above.

- c. Click OK.
- Use the Set Input Delay command to set a minimum input delay of 1.75 ns to both 8-bit input buses dataa and datab, with respect to clk1. To do this, simply click on the tcl> prompt in the Console pane and hit the up arrow on your keyboard. The last command entered (the *set\_input\_delay -max* constraint) will appear. Edit the line to change -max to -min and 3.25 to 1.75, and hit the Enter key.


#### Quartus II Software Design Series: Foundation

- 3. Similar to the above, use the **Set Output Delay** command (**Constraints** menu or typing the command) to set a **maximum** output delay of **0.7 ns** to the 16-bit **q** output bus, with respect to **clk1**. Using a wildcard here will be very handy.
- 4. Use the **Set Output Delay** command again to set a **minimum** output delay of **0.0** ns to the 16-bit **q** output bus, with respect to **clk1**.
- 5. Use the **Set Input Delay** command from the **Constraints** menu to constrain all **rdaddress** and **wraddress** inputs to a **maximum** delay of **2.5 ns** and a **minimum** of **1.0 ns**, with respect to **clk1**.

#### **Step 5: Update the timing netlist**

1. Update the timing netlist. In the **Tasks** pane of the TimeQuest GUI, double-click **Update Timing Netlist**.

This takes any constraints that were entered (stored in memory) and applies them to the current timing netlist.

#### Step 6: Use TimeQuest reports to verify all constraints entered correctly

Now that the netlist has been updated, you can begin generating various reports. The first ones you want to run are to check the timing constraints you entered.

1. In the **Tasks** pane, double-click **Report SDC**.

In the **Report** pane, a new folder called **SDC** Assignments appears containing three reports called **Create Clock**, **Set Input Delay** and **Set Output Delay**. Do these look like all of the constraints that you entered?

2. In the Tasks pane, double-click on Report Clocks.

Use this report to verify that your clock(s) have been entered correctly and applied to the correct ports or pins.

3. In the Tasks pane, double-click on Report Ignored Constraints.

This report will list any constraints that you entered that were ignored by TimeQuest. For example, if you typed an incorrect port name that caused your SDC command to be ignored when you entered it, then it would appear in the Ignored Constraints folder. Are any of your constraints showing up as ignored that should not be?

4. In the **Tasks** pane, double-click on **Report Unconstrained Paths**.

Use this report to ensure you have a fully constrained design. Are there any missing constraints? If so, what is missing?

5. Look through the reports in the **Unconstrained Paths** report folder to see what is missing.



The Unconstrained Paths Summary (in red) should indicate to you that 1 input port and 1 input port path have been left unconstrained. Open the Setup Analysis & Hold Analysis folders to verify which port/path they are.

Do you see that the **wren** input port is not constrained? Because of this, the path from the **wren** port to the **write enable input register** for **ram** is also unconstrained.

6. Use the **Set Input Delay** command from the **Constraints** menu to constrain **wren** to a **maximum** delay of **2.5** ns and a **minimum** of **1.0** ns, with respect to **clk1**.

Notice when you enter new constraints that all of your previous reports are displayed in yellow and indicate that they are "Out of Date." This is because the current reports do not reflect the new constraint you just entered. To fix this, you would need to update your timing netlist again and re-run all reports. In the GUI, there's an even easier way to do this.

7. In the **Report** pane, right-click on any report and select **Regenerate All Out of Date**.

Is your design fully constrained now?

#### Step 7: Write SDC file

Now you want to save all of your timing constraints into an SDC file so that you can use them to guide the fitter during compilation. To do this, you will have TimeQuest write out an SDC file based on all current constraints.

1. Write an output SDC file. In the **Tasks** pane of the **TimeQuest GUI**, double-click **Write SDC File** and name the file **pipemult.sdc**.

#### Step 8: Run compilation using SDC file

- 1. Bring the **Quartus II** software to the foreground.
- 2. From the Assignments menu, choose Timing Analysis Settings.

The Settings dialog box opens with the Timing Analysis Settings category selected.

Settings - filtref		k
Category: General Files Libraries Device Deprating Settings and Conditions Compilation Process Settings EDA Tool Settings Analysis & Synthesis Settings Fitter Settings Timing Analysis Settings	Timing Analysis Settings         Specify whether to use the TimeQuest Timing Analyzer or the Classic Timing Analyzer as the default timing analysis tool. The TimeQuest analyzer requires a Synopsys Design Constraints File (SDC) containing timing constraints or exceptions.         Timing analysis processing         Image: Specify the Specify Constraints or exceptions.         Timing analysis processing         Image: Use TimeQuest Timing Analyzer during compilation         Image: Use Classic Timing Analyzer during compilation	

3. Enable Use TimeQuest Timing Analyzer during compilation as shown above.



#### **Quartus II Software Design Series: Foundation**

Settings - pipemult		×
Category: General Files Libraries Device Device Deprating Settings and Conditions Compilation Process Settings EDA Tool Settings Analysis & Synthesis Settings Fitter Settings Timing Analysis Settings Timing Analysis Settings Classic Timing Analyzer Classic Timing Analyzer Settings Assembler Design Assistant	TimeQuest Timing Analyzer         Specify TimeQuest Timing Analyzer options.         SDC files to include in the project         SDC filename:         File name         pipemult.sdc         Synopsys Desi         Up         Down	
SignalTap II Logic Analyzer Logic Analyzer Interface E Simulator Settings	Enable Advanced I/O Timing	

- 4. Add your **pipemult.sdc** file to the project. In the **Settings** dialog box, click on the **TimeQuest Timing Analyzer** category (under **Timing Analysis Settings**). Use the browse button to locate the file **pipemult.sdc**, click **OK**, and then click **Add**.
- 5. Click **OK** to close the **Settings** dialog box.
- 6. Click on or select **Start Compilation** from the **Processing** menu.
- 7. When compilation is complete, open the **TimeQuest Timing Analyzer** folder in the **Compilation Report**.

Are you meeting or missing timing? A quick glance of the summary reports will tell you. Are any shown in red? If not, then your timing has been validated. You do not need to check any further.

8. Record the setup and hold slack in **Table 3** (beginning of exercise).

#### Step 9: Apply SDC file to pipemult\_lc revision

*Now you can use your SDC file to check timing on the pipemult\_lc revision.* 

- 1. Use the drop-down menu at the top of the **Quartus II** window to change the revision back to **pipemult\_lc**.
- 2. Enable **TimeQuest** to be used during compilation (**Assignments** ⇒ **Timing Analysis Settings**).
- 3. Add **pipemult.sdc** to the project as the timing file.
- 4. Perform a full compilation of the **pipemult\_lc** revision.

Is this revision meeting or missing timing?

5. In the **Compilation Report**, look at the **Setup Summary** and **Hold Summary** reports. Record the setup and hold slack values in Table 3.



*Here you will see that clk1 is missing setup timing by over 0.6 ns. Let's investigate further.* 

#### 6. Open TimeQuest.

This time, since you know you have an SDC file that fully constrains your design and you have added this SDC file to the project, you can use the **Tasks** pane to quickly generate reports.

7. Generate a **Setup Summary** report. In the **Tasks** pane, double-click **Report Setup Summary**.

Notice the green check marks next to **Create Timing Netlist**, **Read SDC File** and **Update Timing Netlist**. By using the **Tasks** pane to run a report, it has automatically executed the steps needed to generate that report. This is a handy shortcut the GUI provides. You could perform the same shortcut by executing these individual commands from a script file.

In the **Report** pane, the **Summary** (Setup) report is shown in red to indicate a failure (as we already know).

8. Generate reports with more detail. In the **Summary** (**Setup**) report, highlight **clk1**. Right-click and choose **Report Timing**.

Report Timing
Clocks From clock:
To clock: {clk1}
Targets
From:
Through:
Τα.
Analysis type Paths
Setup C Recovery Report number of paths: 20
C Hold C Removal Maximum slack limit: ns
Output
Detail level: Summary Set Default
I Report panel name: Setup: clk1 Summary
File name:
File options
Console
Tcl command: report_timing -to_clock {clk1} -setup -npaths 20 -detail summary -pa
-ja Report Timing Close Help



In the Report Timing dialog box, in the Paths section, change Report number of paths to 20. In the Output section, use the Detail level: drop-down menu to select Summary. Click the Report Timing button.

The **Setup: clk1 Summary** report appears. Notice that you have at least 20 failing paths. What's common about these paths? Notice the names of the source nodes and destination nodes.

All of your multiplier logic has been placed between two register stages, one input stage and one output stage. As a result of implementing this multiplier using logic cells, there is now too much logic in between these registers which is causing a timing failure. If you want to verify this, you can use the **Technology Map Viewer** to see the resources used by the filter. Do the following:

- a. Switch back to the Quartus II software, and open the Technology Map Viewer from the Tools menu.
- b. Right-click on any block and choose Viewer Options. In the Filtering section, disable Number of filter levels. Click OK.
- c. Descend two levels down into the **mult** subdesign.
- d. Highlight the output pin **OUT1**. Right-click and choose **Filter** ⇒ **Sources**. You will see the register named **output\_reg[0]**. This is the multiplier output register. So there is no logic between the multiplier output register and the multiplier output pin.
- e. Highlight output\_reg[0], right-click, and choose Filter  $\Rightarrow$  Sources again.

Now you will see two levels of logic (part of the multiplier function) and 3 source registers.

f. Highlight one of the 3 source registers, right-click and choose  $Filter \Rightarrow Sources$  again.

Now you see the source register's data input (SDATA) is fed directly by an input pin. Thus, synthesis has placed all of the multiplier logic in between a bank of input and output registers. You can try these same steps on any of the multiplier output pins and you will see the same.

#### Step 10: Enable physical synthesis on a new revision

Before you use the physical synthesis option to try to further improve your clock timing, create another new revision. This is helpful because if your design takes a long time to compile and you don't like the results of your physical synthesis, you can quickly change back to the previous version without recompiling the entire design again.

- 1. From the **Project** menu in the Quartus II software, select **Revisions**.
- 2. In the **Revisions** dialog box, click on the **Create** button.





- 3. Type in **pipemult\_lc\_phys\_syn** as the **Revision name**. Leave all other defaults and click **OK**.
- 4. Click **OK** to close the **Revisions** dialog box.
- 5. From the Assignments menu, select Settings. Go to Physical Synthesis Optimizations found in the Fitter Settings category.
- 6. In the **Fitter optimizations** box, enable **Perform register retiming**. Leave the **Physical synthesis effort** set to **Normal**.

Settings - pipemult_lc_phys_syn		×
Settings - pipemult_lc_phys_syn Category: General Files Device Operating Settings and Conditions Compilation Process Settings EDA Tool Settings Analysis & Synthesis Settings Filter Settings Filter Settings Filter Settings Timing Analysis Settings Timing Analysis Settings Compilation Physical Synthesis Optimizations Timing Analysis Settings Compilation Physical Synthesis Coptimizations Timing Analysis Settings Compilation Physical Synthesis Optimizations Compilation Physical Synthesis Optimizations Compilation Physical Synthesis Optimizations Filter Settings Category: Category: Compilation Physical Synthesis Optimizations Compilation C	Physical Synthesis Optimizations         Specify options for performing physical synthesis optimizations during fitting.         Physical synthesis for performance         Perform physical synthesis for combinational logic         Perform automatic asynchronous signal pipelining         Physical synthesis for registers         Perform register duplication         Image: Perform register retiming         Physical synthesis for fitting	
<ul> <li>Logic Analyzer Interface</li> <li>Simulator Settings</li> <li>Simulation Verification</li> <li>Simulation Uput Files</li> <li>PowerPlay Power Analyzer Settings</li> </ul>	Perform logic to memory mapping     Physical synthesis effort     Normal (default; increases compilation time two to three times)     Extra (should improve design performance; increases compilation time)     Fast (may reduce performance gains; decreases compilation time)	

This feature will now to try to balance the combinatorial logic between registers to improve the system performance.

- 7. Click **OK** to close the **Settings** dialog box.
- 8. Click to perform a full compilation.

By creating a new revision based on pipemult\_lc, all of the settings for pipemult\_lc were carried over (e.g. SDC file, I/O assignments), so there is no need to specify them.



#### Quartus II Software Design Series: Foundation

9. When compilation is complete, check to see if your timing has been met. Record the setup and hold slack values in Table 3.

You now have two revisions that meet your timing specifications. Remember that if your design uses multipliers, the default settings (using embedded multipliers) should be your first option as the design will perform faster and use less logical resources. If you run out of embedded multipliers, then the logic array may still be an option with a little optimization.

#### Step 11: Compare the original results versus the revision results

- 1. From the **Project** menu, select **Revisions**.
- 2. In the **Revisions** dialog box, click **Compare**.

A table now displays comparing the assignments and results of the 4 revisions you have created. Across the 4 revisions, compare:

- Total number of logic elements (Fitter section)
- Total number of registers (Fitter or Analysis & Synthesis section)
- Total embedded multiplier 9-bit elements (Fitter section)
- Setup & hold slack (TimeQuest section)



## **Exercise Summary**

- Practiced basic steps for using TimeQuest
- Entered SDC timing constraints for analysis

# **END OF EXERCISE 6**



# (Nios II or DSP Development Board Programming Lab)





### <u>Objectives</u>:

- Create a chain description file (CDF) to use in programming a JTAG chain
- Use the Quartus II Programmer to configure a device



## Step 1: Connect development board & open Quartus II project

- 1. Take out your **Nios II** or **DSP** (usually indicated in one corner) development board. Connect power to the board (DSP boards must also be switched on) and connect the download cable as follows:
  - USB-Blaster cable: Connect to a powered USB port on your PC or laptop. If the Windows New Hardware Wizard opens, choose to manually find the USB Blaster driver (Have Disk... option). The driver is located in C:\altera\<Quartus\_II\_install\_version>\quartus\drivers\usb-blaster\x32\.
  - **ByteBlasterII or ByteBlasterMV cable:** Connect to the parallel port of your PC or laptop.
- 2. Connect the 10-pin female connector of the download cable to the 10-pin JTAG header on your development board (it will be marked). The "tail" of the JTAG cable should hang off the board, not across it, matching pin 1 of the cable with pin 1 of the board header. Ask your instructor if you are having trouble locating the header or figuring out the orientation.
- Open the project top\_counter.qpf located in the <lab\_install\_directory>\QIIF7\_1\Ex7\counter\_nios\ or <lab\_install\_directory>\QIIF7\_1\Ex7\counter\_dsp\ directory, depending on which development kit your are using.



This project contains a simple 2-digit decimal counter.

4. At the top of the Quartus II window (example above), use the drop-down menu to choose the correct project revision for your **Nios II** or **DSP** development board. (Look at the part number on or near the device or ask your instructor).

You may also change revisions by opening the **Revisions** dialog box from the **Project** menu.



#### Step 2: Open and set up Programmer to configure device

1. From the **Tools** menu, select **Programmer** (or click <sup>V</sup> in the toolbar).

This will open a **CDF** file. The **CDF** file lists all of the devices in your configuration or JTAG chain along any associated programming or configuration files.

- 2. Save the CDF file. Use the default name.
- 3. Locate the **Hardware Setup** button at the top of the **CDF** file. The field to the right of this button should display the name of the programming cable you are using: **USB-Blaster, ByteBlasterII**, or **ByteBlasterMV**.

If this window reads anything else or displays the wrong cable, click the **Hardware** Setup button and select USB-Blaster, ByteBlasterII, <u>or</u> ByteBlasterMV from the drop-down menu of Currently selected hardware. Click Close.

If the **download** cable you are using is not listed in the **Available Hardware** window, please let the instructor know as the drivers may not have been loaded correctly onto your PC.

- 4. Back in the **CDF** file, use the drop-down **Mode** field to choose **JTAG**, if it is not already selected.
- 5. In the main programming window, you should see the **counter\_<device>.sof** (configuration) file listed along with its target device.

If you do not see the **counter\_<device>.sof** configuration file listed in the programmer window, click the **Add File** button and select it.



6. Enable the **Program/Configure** option for the configuration file and target device as shown above.

*Remember if you want to bypass any devices in your JTAG chain, you can simply leave this option unchecked for those devices.* 

7. Click the **Start** button to begin configuration.



#### Quartus II Software Design Series: Foundation

The LED display should begin counting from 0 to 99 when configuration is complete. The counter's reset signal is controlled by:

SW8-CPU Reset (Nios boards) SW6 (Cyclone II DSP board) SW4 (Stratix II DSP boards) SW0 (Stratix DSP boards)

## **Exercise Summary**

- Created a Chain Description File (CDF)
- Programmed an FPGA device

# **END OF EXERCISE 7**





# Optional Exercise (Quartus II Simulation)





# **Optional Exercise**

### <u>Objectives</u>:

- Create a Vector Waveform (.cvwf) input stimulus file
- Run a functional simulation
- Examine the simulation output for verification of the design



#### Step 1: Set up Simulator

1. Open the **pipemult.qpf** project located in the **QIIF7\_1\Opt Ex\** directory.

Any of the revisions wll work for this exercise.

- 2. From the Assignments menu select Settings.
- 3. In the **Simulator Settings** category, select **Functional** from the **Simulation mode** drop-down menu.

Settings - pipemult		×
Settings - pipemult Category: General Files Libraries Device Operating Settings and Conditions Compilation Process Settings EDA Tool Settings Analysis & Synthesis Settings Filter Settings Filter Settings Timing Analysis Settings Timing Analysis Settings Assembler Design Assistant SignalTap II Logic Analyzer Logic Analyzer Interface Simulation Settings Simulation Verification Simulation Verification Simulation Output Files PowerPlay Power Analyzer Settings	Simulator Settings Select simulation options. Simulation mode: Functional Simulation input: pipemult.cvwf Add Multiple Files Simulation until all vector stimuli are used Function at: Glitch filtering options: Auto More Settings	
<ul> <li>SignalT ap II Logic Analyzer</li> <li>Logic Analyzer Interface</li> <li>Simulator Settings</li> <li>Simulation Verification</li> <li>Simulation Output Files</li> <li>PowerPlay Power Analyzer Settings</li> </ul>	Glitch filtering options:	
	OK Cancel	

4. For simulation input, type **pipemult.cvwf** (shown above). Click **OK**.

#### Step 2: Create .vwf file

1. From the **File** menu, select **New**. From the **New** dialog box, select the **Other Files** tab. In the **Other Files** tab, select **Vector Waveform File** and click **OK**.



#### Step 3: Enter signals

- 1. From the **Edit** menu, go to the **Insert** submenu and select **Insert Node or Bus**. In the **Insert Node or Bus** dialog box, click the **Node Finder** Button.
- 2. In the **Node Finder**, go to the **Filter** box and select **Pins: all** from the drop down menu. Now click the **List** button.
- 3. Select clk1, dataa, datab, wraddress, rdaddress, wren, and q. Click on the > button to copy these pins to the Selected Nodes area. Click OK. Click OK again in the Insert Node or Bus dialog box.
- 4. In the .vwf file, highlight **dataa**, **datab**, **wraddress**, **rdaddress** and **q**. Right-click and select **Properties**. Change the radix from **Binary** to **Hexadecimal**. Click **OK**.
- 5. From the **Edit** menu, select **End Time**. In the **End Time** dialog box, set the **Time** to **100 ns**. Make sure the units are set correctly. Click **OK**.
- 6. Click the **Zoom** tool to select it. Right-click anywhere in the waveforms to zoom out until the entire 100 ns of the simulation is visible. Change back to the normal select tool.

#### **Step 4: Set stimulus**

1. The .vwf file should contain all the nodes you have selected. Enter the input waveforms for **clk1**, **dataa**, **datab**, **wraddress**, **rdaddress** and **wren** as shown below.

To edit a waveform (as shown in the presentation), highlight the waveform or a section of the waveform and edit (overwrite) with the correct value from the toolbar or Edit menu. You can use the Insert Clock and Insert Count Value tools as shortcuts when entering the waveforms.

🗗 Qua	🖸 Quartus II - C:/altera_trn/Quartus_II_Software_Design_Series_Foundation/QIIF7_1/Opt Ex/Schematic/pipemult - pipemult - [pipemult.cvwf]																		
File Edi	file Edit View Project Tools Window																		
-	Master T	ime Bar:	0 p:	s	• • Poin	ter:	41.46 n	s	Interva	al:	41.46 r	ns	Start				End:		
A			Value	0 ps	10.0 ns	20.0 r	IS	30.0 ns		40.0 ns	50	1.0 ns	60.	Dins	70.0 ns	8	0.0 ns	90.01	ns 100.0 ms
æ€		Name	0 p	0 ps J															
	₽0	clk1	BC									1							
88 A.2	<b>P</b> 1	🗉 dataa	НO	00	X	01 X	02	X	03		04	<u>X 05</u>	<u> </u>	06	X	07	Х	08 X	)
<b>699</b>	<b>1</b> 0	🗉 datab	НO	00	X	01 X	02	X	03	_X	04	X 05	<u> </u>	06	X	07	Х	08 X	)
ेख ॐ	<b>P</b> 19	<ul> <li>wraddress</li> </ul>	НO	0	X	<u>01 X</u>	02	X	03	_X	04	<u>X 05</u>	<u> </u>	06	_X_	07	_X	<u>08 X</u>	)
0, 1	₽25	🛨 rdaddress	НO	<u> </u>	00	X_	01	X	02	_X	03	<u>X 04</u>	<u>ــــــــــــــــــــــــــــــــــــ</u>	05	X	06	_X	<u>07 X</u>	
Z VII	<b>■</b> 31	wren	B 1										8						
	<b>i 3</b> 2	₽ E	н∞	<u> </u>						_	×	₩							)
신문사면																			
$\langle 2 \rangle \langle B \rangle$																			
Pr: Z∔	<		>																



#### **Step 5: Save and simulate stimulus file**

- 1. Save the simulation file as pipemult.cvwf (a compressed vector waveform file).
- 2. From the **Processing** menu select **Start Simulation**

*Did you get an error message? Do you remember what you must do before you can perform a <u>functional</u> simulation?* 

- 3. From the **Processing** menu select Generate Functional Simulation Netlist.
- 4. Once again, **Start Simulation**. Once the simulation is complete, a box appears "Simulation was Successful." Click **OK**.

#### **Step 6: Examine results**

1. The **Simulation Report** automatically opens when simulation begins.

4	🖻 Qu	artus II	- C:/altera_t	rn/Quart	us_II_Softw	are_Desig	n_Series_	Foundation/Q	IIF7_1/0	t Ex/Sche	matic/pipe	nult - pip	oemult -	[Simulati	ion R 📒	
F	File E	Edit View	) Project Tool	s Window												
	Simu	lation W	aveforms													
	Simulation mode: Functional														<u>^</u>	
																~
	<b>₽</b>	Master T	ime Bar:	0 ps	I F	ointer:	46.67 ns	Interva	: 4	5.67 ns	Start:			End:		
	Α			Value at	0 ps 1	0.0 ns	20.0 ns	30.0 ns	40.0 ns	50.0 ns	60.0 n	s 70.	0 ns	80.0 ns	90.0 ns	100.0 ns
	₩ ⊕		Name	0 ps	0 ps J											
	9	▶0	clk1	ВO												
		1	🛨 dataa	H 00	00	χ 01	X 02	. <u>χ</u> 03		14 X	05 X	06	07	_X	08 X	09 )
	<b>AA</b>	10	🛨 datab	H 00	00	X 01	X 02	: X 03	(	14 X	05 X	06	07	_X	08 X	09 )
	8°98	19	🛨 wraddress	H 00	00	X 01	X 02	: X 03	<u> </u>	14 X	05 X	06	07	_X	08 X	09 )
	+	25	🛨 rdaddress	H 00		00	X 01	X 02	X (	з 🗶	04 X	05	06	_X	07 X	08 )
	<b>→</b>	<b>3</b> 1	wren	B 1									1000			
	82	32	⊞ q	H 0000	0000	) X	XXXX X		0000		X 0001	( OC	)04 X	0009	χ 0010	<u>(0019</u> )
	₽↓	<		>												

2. The **Simulation Waveform** in the simulator report should appear similar to the image above.



#### **Exercise Summary**

- Created a .CVWF file
- Performed functional simulation
- Used simulation report to view simulation results

# END OF OPTIONAL EXERCISE

