lpm_divide Megafunction

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Software Version: 7.1
Document Version: 2.3
Document Date: June 2007

UG-MF82704-2.3

Contents

About this User Guide v
Revision History v
Referenced Documents v
How to Contact Altera v
Typographic Conventions vi

Chapter 1. About this Megafunction
Device Family Support 1–1
Introduction 1–1
Features 1–2
General Description 1–2
Resource Utilization 1–2

Chapter 2. Getting Started
Software and System Requirements 2–1
MegaWizard Customization 2–1
MegaWizard Page Descriptions 2–1
Inferring Megafunctions from HDL Code 2–5
Instantiating Megafunctions in HDL Code 2–5
Identifying a Megafunction after Compilation 2–5
Simulation 2–5
Quartus II Simulation 2–6
EDA Simulation 2–6
SignalTap II Embedded Logic Analyzer 2–6
Design Example: 8-Bit Divider 2–7
Design Files 2–7
Example 2–7
Generate an 8-Bit lpm_divide Megafunction 2–7
Implement the lpm_divide Megafunction 2–12
Functional Results—Simulate the lpm_divide Design in Quartus II 2–14
Functional Results—Simulate the 8-Bit Multiplier-Adder Design in ModelSim-Altera 2–16
Conclusion 2–17

Chapter 3. Specifications
Ports and Parameters 3–1

About this User Guide

Revision History

The following table displays the revision history for this user guide.

Date and Document Version	Changes Made	Summary of Changes
June 2007 v2.3	Added Arria™ GX and Statix® III device to list of supported devices.	Updated for Quartus® II version 7.1 by adding support for Arria GX and Stratix III devices.
March 2007 v2.2	Added Cyclone® III device to list of supported devices.	Updated for Quartus® II version 7.0 by adding support for Cyclone III device.
December 2006 v2.1	Added Stratix III device support to Table 1–1.	
June 2006 v2.0	Updated for Quartus II 6.0 software release.	
September 2006 v1.0	Initial release	

Referenced Documents

This document references the following documents:

- *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*
- *Quartus II Handbook*

How to Contact Altera

For the most up-to-date information about Altera® products, go to the Altera website at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown in the following table.

Contact *(1)*	Contact Method	Address
Technical support	Website	www.altera.com/mysupport/

Contact *(1)*	Contact Method	Address
Technical training	Website	https://mysupport.altera.com/etraining
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Altera literature services	Email	literature@altera.com
Non-technical support (General)	Email	nacomp@altera.com
	Email	authorization@altera.com

Note to table:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown in the following table.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.

Courier type	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.

Chapter 1. About this Megafunction

Device Family Support

The lpm_divide megafunction supports the following target Altera® device families:

- Arria™ GX
- Stratix® III
- Stratix II
- Stratix II GX
- Stratix
- Stratix GX
- Cyclone® III
- Cyclone II
- Cyclone
- HardCopy® II
- HardCopy Stratix
- MAX® II
- MAX 7000AE
- MAX 7000B
- MAX 7000S
- MAX 3000A
- APEX™ II
- APEX 20KC
- APEX 20KE
- APEX 20K
- ACEX 1K®
- FLEX® 10KE
- FLEX 10KA
- FLEX 6000

Introduction

As design complexities increase, use of vendor-specific intellectual property (IP) blocks has become a common design methodology. Altera provides parameterizable megafunctions that are optimized for Altera device architectures. Using megafunctions instead of coding your own logic saves valuable design time. Additionally, the Altera-provided functions may offer more efficient logic synthesis and device implementation. You can scale the megafunction's size by simply setting parameters.

Features

The lpm_divide megafunction offers additional features, which include:

- Parameterizable input data widths for the numerator and denominator values
- Support for both signed and unsigned data representation for the numerator and denominator
- Ability to specify positive remainder output
- Support for pipelining with parameterizable output latency
- Active high asynchronous clear and clock enable control inputs
- Support for area versus speed trade-off

General Description

The lpm_divide megafunction is one of the arithmetic megafunctions provided in the Quartus® II software MegaWizard® Plug-In Manager.

The basic function of a divider is to divide a numerator input value by a denominator input value to produce a quotient and a remainder. A divider performs basic mathematical functions often found in digital signal processing (DSP) applications. The lpm_divide megafunction lets you implement a divider efficiently in Altera devices.

Resource Utilization

Table 1–1 summarizes the resource usage for an lpm_divide function.

Table 1–1. lpm_divide Megafunction Resource Usage *Note (1)*

Device Family	Width	Optimization	Resource Usage	
			ALUT	LE
Stratix III	8	Speed	84	—
		Balanced	82	—
		Area	81	—
Cyclone III	8	Speed	—	88
		Balanced	—	86
		Area	—	85

Note to Table 1–1:

(1) The information in this table is valid and accurate in Quartus II version 7.1 using default settings.

Chapter 2. Getting Started

Software and System Requirements

The instructions in this section require the following software:

- Quartus® II software
- For OS support information, refer to: www.altera.com/support/software/os_support/oss-index.html

MegaWizard Customization

The MegaWizard® Plug-In Manager allows you to create and modify design files that contain custom megafunction variations, which you can then instantiate in a design file.

With the MegaWizard Plug-In Manager, you can specify custom megafunction variation options. The MegaWizard Plug-In Manager provides a wizard that allows you to set parameter values and select optional ports.

You can use the MegaWizard Plug-In Manager to set the lpm_divide megafunction features for each divider in the design.

Start the MegaWizard Plug-In Manager in one of the following ways:

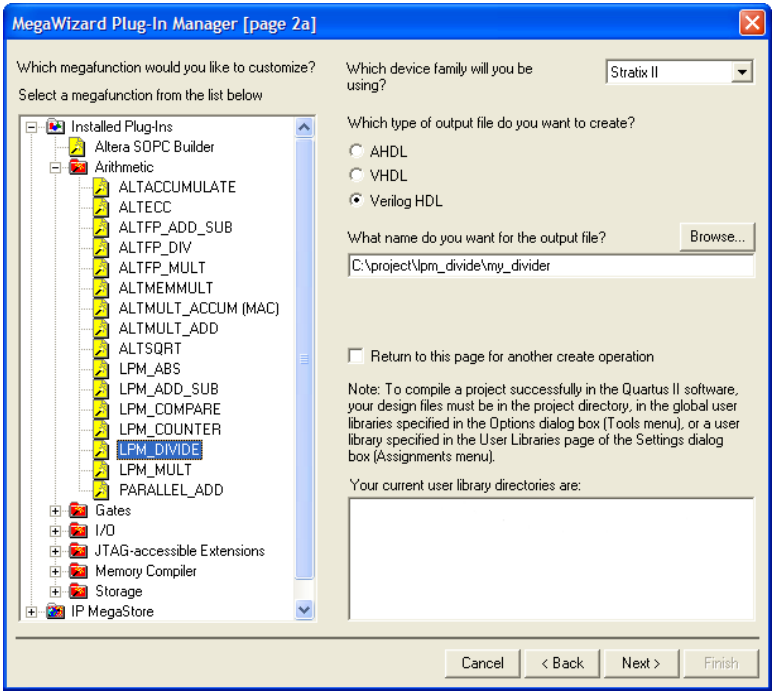
- On the Tools menu, click **MegaWizard Plug-In Manager**.
- When working in the Block Editor, from the Edit menu, click **Insert Symbol as Block**, or right-click in the Block Editor, point to Insert, and click **Symbol as Block**. In the **Symbol** dialog box, click **MegaWizard Plug-In Manager**.
- Start the stand-alone version of the MegaWizard Plug-In Manager by typing the following command at a command prompt:
  `qmegawiz↵`

MegaWizard Page Descriptions

This section provides an in-depth description of each page in the lpm_divide wizard. Tables 2–1 and 2–2 show the features or settings for the lpm_divide megafunction. You can use these tables to determine appropriate settings for your divider designs.

On page 2a, select the lpm_divide megafunction from the Arithmetic category, select the device you intend to use, the type of output file you want to create (Verilog, VHDl, or AHDL), and what you want to name the output file. You also have the option to enable the generation of a clear box netlist for this megafunction (Figure 2–1).

Figure 2–1. MegaWizard Plug-In Manager [page 2a]



On page 3 of the lpm_divide wizard, you control the sign representation for the inputs and specify the input data widths (Figure 2–2).

Figure 2–2. MegaWizard Plug-In Manager - LPM_DIVIDE [page 3 of 6]

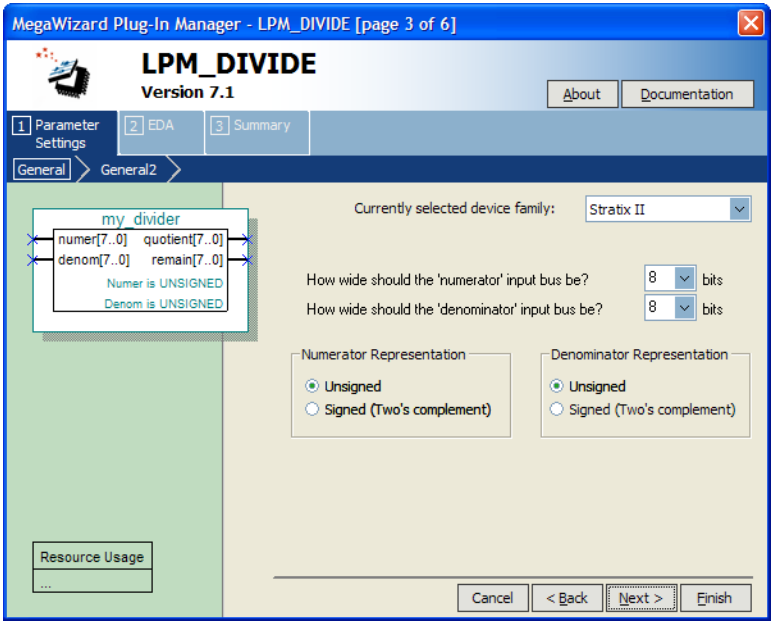


Table 2–1 shows the features available on page 3 of the lpm_divide wizard.

Table 2–1. lpm_divide Wizard Page 3 Options

Function	Description
How wide should the 'numerator' input bus be?	Specify the width of the `numer[]` input.
How wide should the 'denominator' input bus be?	Specify the width of the `denom[]` input.
Numerator Representation	Select the sign representation of the numerator: **Unsigned** or **Signed**.
Denominator Representation	Select the sign representation of the denominator: **Unsigned** or **Signed**.

On page 4 of the lpm_divide wizard, you specify the output latency for pipelining, output a positive remainder, enable asynchronous clear and clock enable inputs, and control the optimization technique for the divider function (Figure 2–3).

Figure 2–3. MegaWizard Plug-In Manager - LPM_DIVIDE [page 4 of 6]

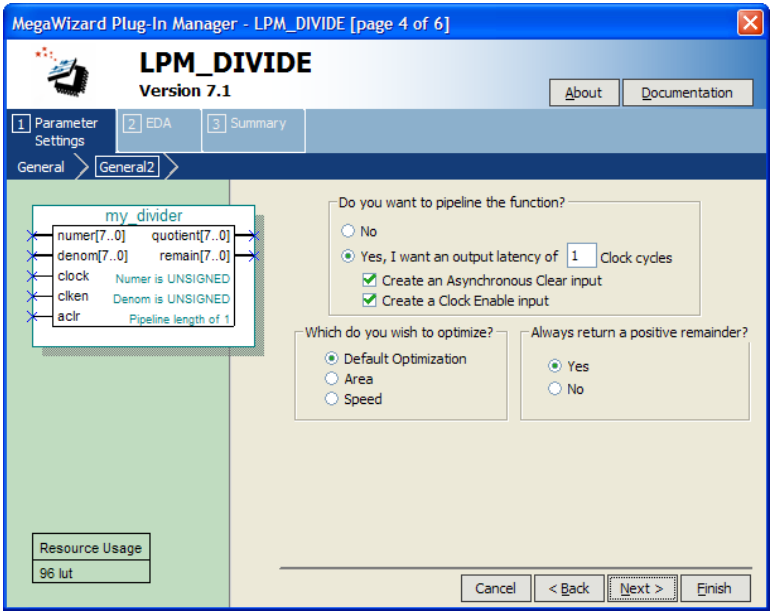


Table 2–2 shows the options available on page 4 of the lpm_divide wizard.

Table 2–2. lpm_divide Wizard Page 4 Options

Function	Description
Do you want to pipeline the function?	Creates a clock port to provide pipelined operation for the lpm_divide megafunction. You must manually specify the amount of latency desired.
Create an Asynchronous Clear input	Creates an active, high, asynchronous clear for pipelined usage.
Create a Clock Enable input	Creates an active, high, clock enable for pipelined usage.
Which do you wish to optimize?	Select the optimization technique for the divider function: **Area** or **Speed**.
Always return a positive remainder?	Select **Yes** to output a positive remainder only.

Inferring Megafunctions from HDL Code

Synthesis tools, including Quartus II integrated synthesis, recognize certain types of HDL code and automatically infer the appropriate megafunction when a megafunction will provide optimal results. The Quartus II software uses the Altera® megafunction code when compiling your design—even if you did not specifically instantiate the megafunction. The Quartus II software infers megafunctions because they are optimized for Altera devices, so the area and performance may be better than generic HDL code. Additionally, you must use megafunctions to access certain Altera architecture-specific features—such as memory, DSP blocks, and shift registers—that generally provide improved performance compared with basic logic elements.

Refer to volume 1 of the *Quartus II Handbook* for specific information about your particular megafunction.

Instantiating Megafunctions in HDL Code

When you use the MegaWizard Plug-In Manager to set up and parameterize a megafunction, it creates either a VHDL or Verilog HDL wrapper file that instantiates the megafunction (a black-box methodology). For some megafunctions, you can optionally generate a netlist for area and timing estimation in EDA synthesis tools such as Synplify and Precision RTL Synthesis. These methodologies for instantiating the MegaWizard-generated files are described in volume 1 of the *Quartus II Handbook*:

- *Recommended HDL Coding Styles* chapter
- *Quartus II Integrated Synthesis* chapter
- *Synplicity Synplify and Synplify Pro Support* chapter
- *Mentor Graphics Precision RTL Synthesis Support* chapter

Identifying a Megafunction after Compilation

During compilation with the Quartus II software, analysis and elaboration is performed to build the structure of your design. To locate your megafunction in the Project Navigator window, expand the compilation hierarchy and find the megafunction by its name.

To search for node names within the megafunction (using the Node Finder), in the **Look in** box, click the browse button and select the megafunction in the **Hierarchy** box.

Simulation

The Quartus II Simulation tool provides an easy-to-use, integrated solution for performing simulations. The following sections describe the simulation options.

Quartus II Simulation

The Quartus II Simulator is a powerful tool for testing and debugging the logical operation and internal timing of Altera megafunctions instantiated in your design.

With the Quartus II Simulator, you can perform two types of simulations: functional and timing. A functional simulation in the Quartus II program enables you to verify the logical operation of your design without taking into consideration the timing delay in the FPGA. This simulation is performed using only RTL code. When performing a functional simulation, add only signals that exist before synthesis. You can find these signals with the Registers: Presynthesis, Design Entry, or Pin Filter in the Node Finder.

In contrast, timing simulation in the Quartus II software verifies the operation of your design with annotated timing information. This simulation is performed using the post place-and-route netlist. When performing a timing simulation, you add only signals that exist after place-and-route. These signals are found with the post-compilation filter of the Node Finder. During synthesis and place-and-route, the names of RTL signals change. Therefore, it might be difficult to find signals from your megafunction instantiation in the post-compilation filter. To preserve the names of your signals during the synthesis and place-and-route stages, use the synthesis attributes `keep` or `preserve`. These are Verilog and VHDL synthesis attributes that direct Analysis & Synthesis to keep a particular wire, register, or node intact. Use these synthesis attributes to keep a combinational logic node so you can observe the node during simulation. More information on these attributes is available in volume 1 of the *Quartus II Handbook*.

EDA Simulation

Depending on your simulation tool, refer to the appropriate chapter in volume 3 of the *Quartus II Handbook*. The *Quartus II Handbook* shows you how to perform functional and gate-level timing simulations that include the megafunctions, with details about the files that are needed and the directories in which those files are located.

SignalTap II Embedded Logic Analyzer

The SignalTap® II embedded logic analyzer provides a non-intrusive method of debugging all of the Altera megafunctions within your design. With the SignalTap II embedded logic analyzer, you can capture and analyze data samples for the top-level ports of the Altera megafunctions in your design while your system is running at full speed.

To monitor signals from your Altera megafunctions, first configure the SignalTap II embedded logic analyzer in the Quartus II software, and then include the analyzer as part of your Quartus II project. The Quartus II software then embeds the analyzer with your design in the selected device seamlessly.

For more information about using the SignalTap II embedded logic analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Design Example: 8-Bit Divider

In some applications, dividers may be necessary to perform mathematical computation in hardware. A divider performs basic mathematical functions that are often found or implemented in digital signal processing (DSP) applications.

This section presents a design example that uses the lpm_divide megafunction to generate a basic divider. This example uses the MegaWizard Plug-In Manager in the Quartus II software to customize the megafunction. As you go through the wizard, each page is described in detail. When you are finished with this example, you will incorporate it into the overall design.

Design Files

The design files are available in the User Guide section on the Literature page of the Altera website.

Select the links below the *lpm_divide Megafunction User Guide* to download the design files.

Example

In this example, you perform the following activities:

- Generate an 8-bit divider design module.
- Implement the divider design module in a device by assigning the Stratix® II EP2S15F484C3 device and compile the project.
- Simulate the customized divider design module.

Generate an 8-Bit lpm_divide Megafunction

1. Open the **lpm_divide_DesignExample.zip** project and extract **divide.qar**. In the Quartus II software, open **divide.qar** and restore the archive file into your working directory.

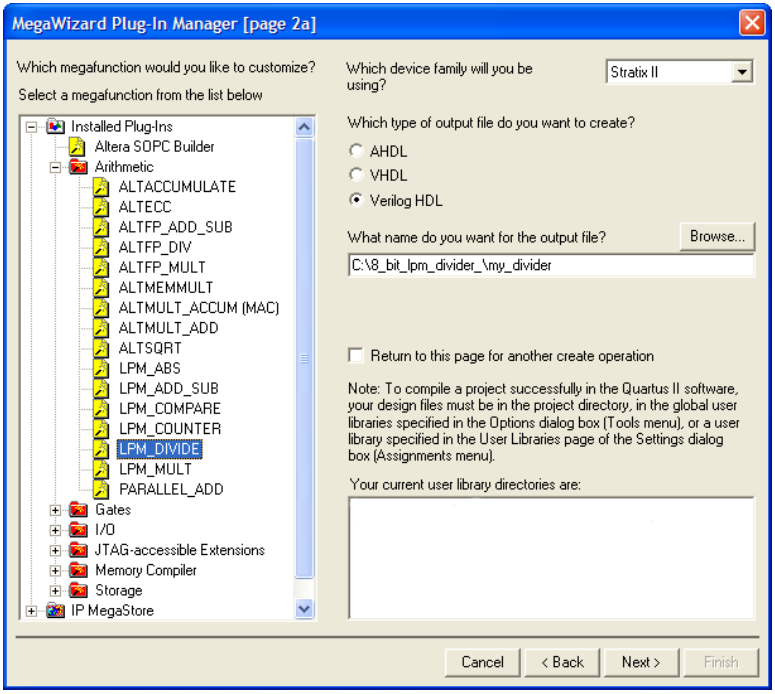
2. In the Quartus II software, browse to your working directory and open the block design file, **divider.bdf**.

3. Double-click on a blank area in the block design (**.bdf**) file and then click on the **MegaWizard Plug-In Manager** button in the **Symbol** dialog box or on the Tools menu, click **MegaWizard Plug-In Manager**.

4. On page 1 of the MegaWizard Plug-In Manager, for the **What action do you want to perform?** section., click **Create a new custom megafunction variation**.

5. Click **Next**. Page 2a appears (Figure 2–4).

Figure 2–4. MegaWizard Plug-In Manager - LPM_DIVIDE [page 2a]



6. On page 2a, expand the **Arithmetic** folder and select lpm_divide.

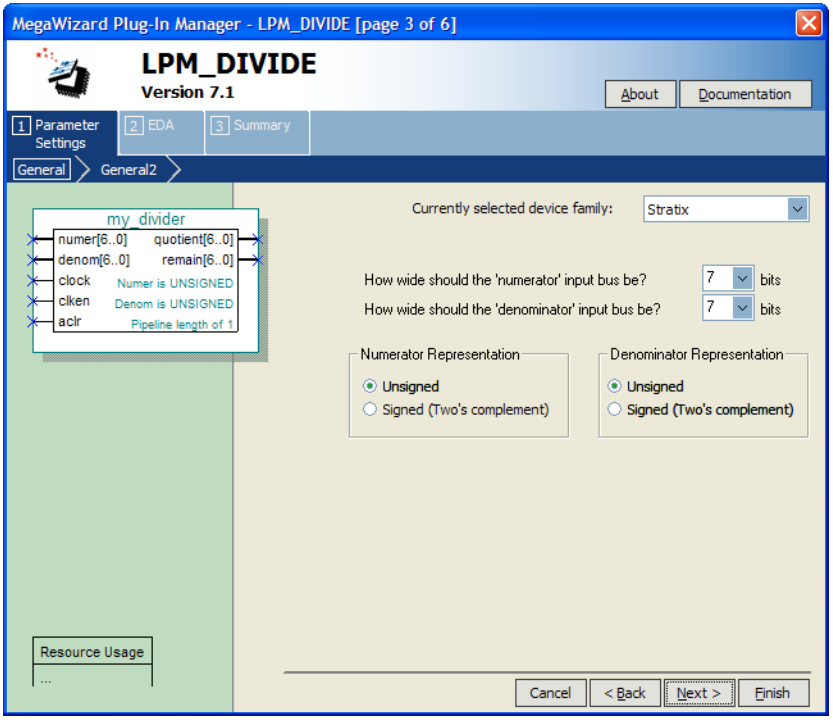
7. For **Which device family will you be using?**, select **Stratix II**.

8. For **Which type of output file do you want to create?**, select the Verilog HDL option.

9. Set the output file name as <*project directory*>**\my_divider**.

10. Click **Next**. Page 3 appears (Figure 2–5).

Figure 2–5. MegaWizard Plug-In Manager - LPM_DIVIDE [page 3 of 6]



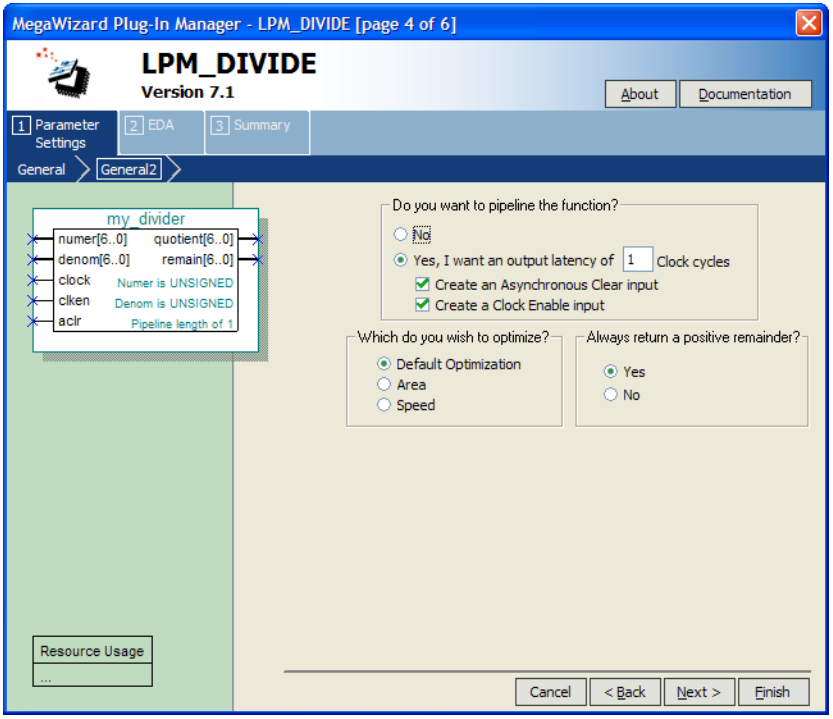
11. On page 3, in the **How wide should the 'numerator' input bus be?** and **How wide should the 'denominator' input bus be?** sections, select **8 bits.**

12. In **Numerator Representation**, select **Unsigned**.

13. In **Denominator Representation**, select **Unsigned**.

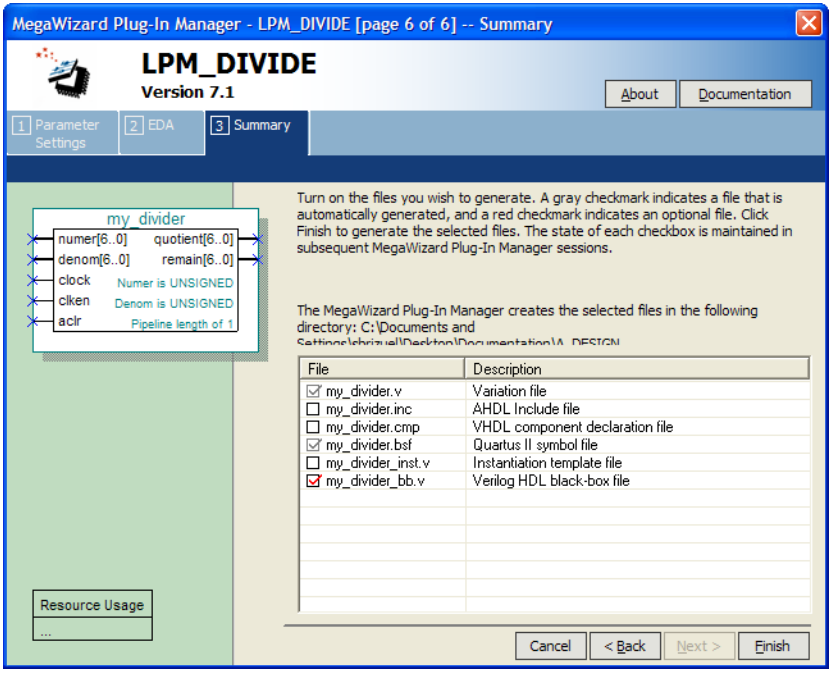
14. Click **Next**. Page 4 appears (Figure 2–6).

Figure 2–6. MegaWizard Plug-In Manager - LPM_DIVIDE [page 4 of 6]



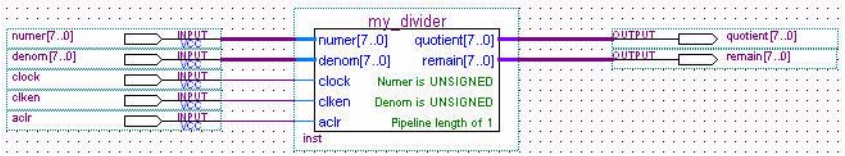
15. On page 4, in **Do you want to pipeline the function?**, select **Yes, I want an output latency of** and type `1` in the **Clock cycles** box.
16. Select **Create an Asynchronous Clear input** and **Create a Clock Enable input**.
17. In **Which do you wish to optimize?**, leave the default value, **Default Optimization**.
18. In **Always return a positive remainder?**, leave the default value, **Yes**.
19. Click **Finish**. Page 6 appears (Figure 2–7).

Figure 2–7. MegaWizard Plug-In Manager - LPM_DIVIDE [page 6 of 6] -- Summary



20. Page 6 of the lpm_divide wizard shows a summary of the megafunction.
21. Click **Finish**. The lpm_divide megafunction is built.
22. In the **Symbol** dialog box, click the "+" icon to expand **Project** and select **check_counter**.
23. Click **OK**.
24. Move the pointer to place the **my_divider** symbol in between the input and output ports of the **divider.bdf** file. Click to place the **my_divider** symbol. Adjust the symbol as necessary so that the input/output ports connect.
25. You have now completed the design file. Figure 2–8 shows the 8-bit lpm_divide megafunction.
26. On the File menu, click **Save** to save the design.

Figure 2–8. 8-Bit lpm_divide Megafunction

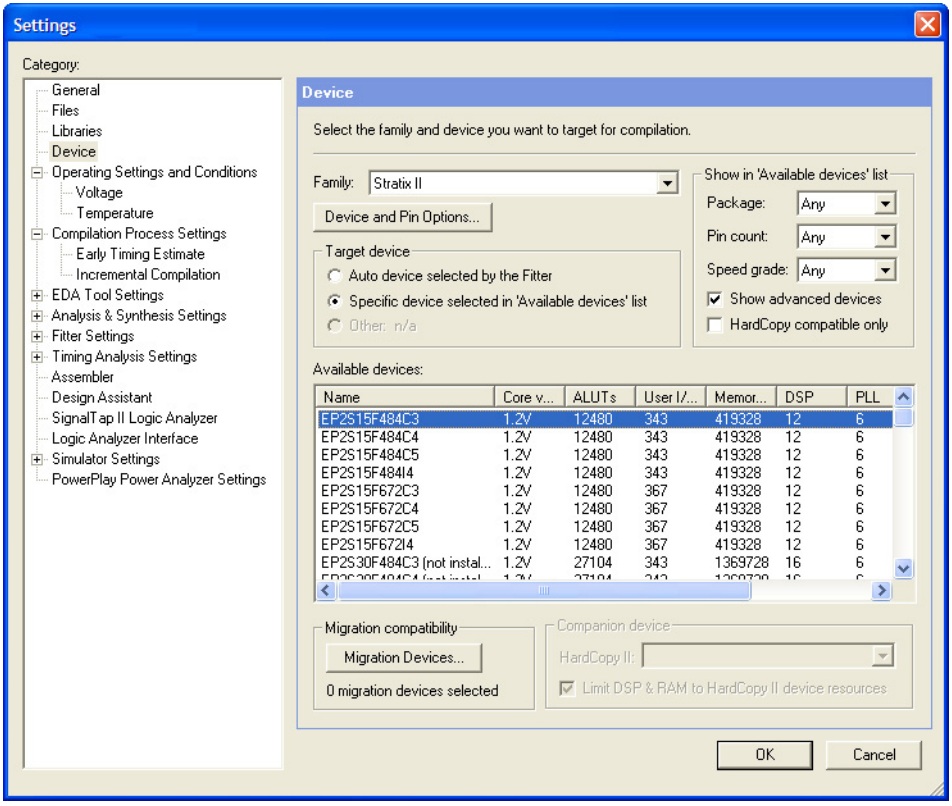


Implement the lpm_divide Megafunction

Next, implement the divider, assign the EP2S15F484C3 device to the project, and compile the project.

1. On the Assignments menu, click **Settings** and then **Files**, and add **divider.bdf** and m**y_divider.v** to the project.
2. In the **Category** list, select **Devices**.
3. In the **Family** list, select **Stratix II**.
4. Under **Target device**, select **Specific device selected in 'Available devices' list**.
5. Under **Available devices**, select **EP2S15F484C3**. Figure 2–9 shows the **Settings** dialog box.

Figure 2–9. Settings Dialog Box



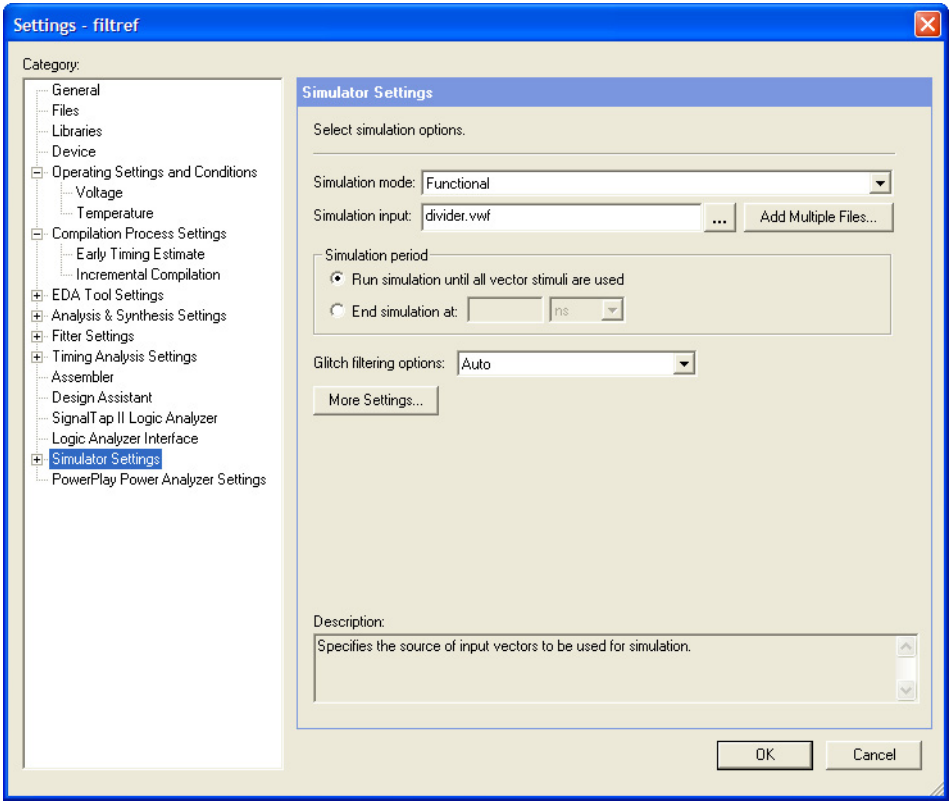
6. Click **OK**.
7. On the Processing menu, click **Start Compilation**, or on the toolbar click the compile button to compile the design.
8. When the **Full Compilation was successful** message box appears, click **OK**.
9. To view how the module is implemented in the Stratix II device, on the Assignments menu, click **Timing Closure Floorplan**.

Functional Results—Simulate the lpm_divide Design in Quartus II

Simulate the lpm_divide design module and verify the results. Set up the Quartus II simulator by performing the following steps:

1. On the Processing menu, click **Generate Functional Simulation Netlist**.
2. When the **Functional Simulation Netlist Generation was successful** message box appears, click **OK**.
3. On the Assignments menu, click **Settings** to open the **Settings** dialog box.
4. In the **Category** list, select **Simulator Settings**.
5. Under **Simulation mode**, select **Functional**, and then select the necessary input vector waveform file (**divider.vwf**). The **Settings** dialog box should look like Figure 2–10.

Figure 2–10. Settings Dialog Box



6. Click **OK**.

7. On the Processing menu, click **Start Simulation**, or on the toolbar, click the simulation button to run a simulation.

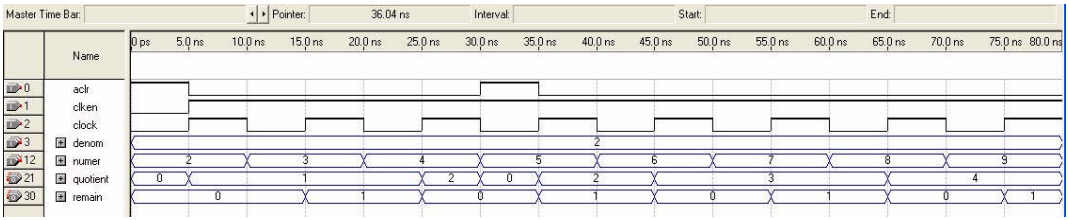
8. When the **Simulator was successful** message box appears, click **OK**.

9. In the Simulation Report window, view the simulation output waveforms and verify the results (see Figure 2–11).

   These output waveforms show the behavior of the lpm_divide megafunction for the chosen set of parameters. The design is an 8-bit unsigned divider that produces an 8-bit output. The output of the

divider has a latency of 1, only returns a positive remainder, and outputs all 1's (15 in decimal) when the asynchronous clear signal is asserted (`aclr` = 1).

Figure 2–11. Divider Simulation Results



Functional Results—Simulate the 8-Bit Multiplier-Adder Design in ModelSim-Altera

Simulate the design in ModelSim to compare the results of both simulators.

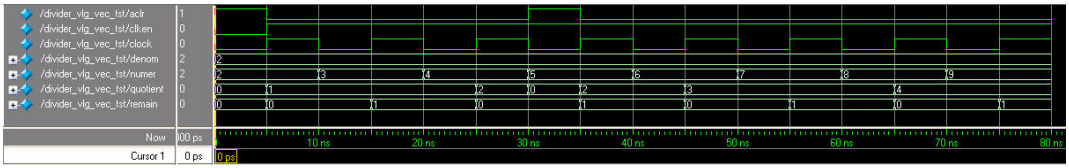
This User Guide assumes that you are familiar with using ModelSim-Altera before trying out the design example. If you are unfamiliar, refer to www.altera.com/support/software/products/modelsim/mod-modelsim.html, which is a support page for ModelSim-Altera. There are various links to topics such as installation, usage, and troubleshooting.

Set up the ModelSim-Altera simulator by performing the following steps:

1. Unzip the **lpm_divide_msim.zip** file to any working directory on your PC.
2. Start **ModelSim-Altera**.
3. On the File menu, click **Change Directory**.
4. Select the folder in which you unzipped the files. Click **OK**.
5. On the Tools menu, click **Execute Macro**.
6. Select the **divider.do** file and click **Open**. This is a script file for ModelSim that automates all the necessary settings for the simulation.
7. Verify the results shown in the **Waveform Viewer** window.

You may need to rearrange signals, remove redundant signals, and change the radix to suit the results in the Quartus II Simulator. Figure 2–12 shows the expected simulation results in ModelSim.

Figure 2–12. ModelSim Simulation Results



Conclusion

The Quartus II software provides parameterizable megafunctions ranging from simple arithmetic units, such as adders and counters, to advanced phase-locked loop (PLL) blocks, multipliers, and memory structures. These megafunctions are performance-optimized for Altera devices and therefore, provide more efficient logic synthesis and device implementation, because they automate the coding process and save valuable design time. Altera recommends using these megafunctions during design implementation so you can consistently meet your design goals.

Chapter 3. Specifications

Ports and Parameters

Figure 3–1 shows the ports and parameters for the lpm_divide megafunction. Table 3–1 shows the input ports, Table 3–2 shows the output ports, and Table 3–3 shows the parameters of the lpm_divide megafunction.

These parameter details are only relevant for users who bypass the MegaWizard® Plug-In Manager interface and use the megafunction as a directly parameterized instantiation in their design. The details of these parameters are hidden from the MegaWizard Plug-In Manager interface users.

Refer to the latest version of the Quartus® II Help for the most current information on the ports and parameters for this megafunction.

Figure 3–1. lpm_divide Port and Parameter Description Symbol

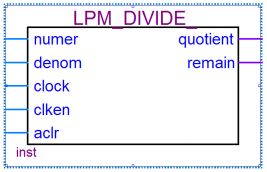


Table 3–1. lpm_divide Megafunction Input Ports

Port Name	Required	Description
numer[]	Yes	Input port LPM_WIDTHN wide.
denom[]	Yes	Input port LPM_WIDTHD wide.
clock	No	Creates a clock port to register and provide pipelined operation for the lpm_divide function. You must connect the clock input if you set LPM_PIPELINE to a value other than '0'.
clken	No	An active high clock enable control signal for pipelined usage.
aclr	No	An active high asynchronous clear control signal for registered usage. The aclr port may be used at any time to reset the pipeline to all '0's asynchronously to the clock input.

Table 3–2. lpm_divide Megafunction Output Ports

Port Name	Required	Description
quotient[]	Yes	Output port LPM_WIDTHN wide. You must use either the quotient[] or the remain[] ports.
remain[]	Yes	Output port LPM_WIDTHD wide. You must use either the quotient[] or the remain[] ports.

Table 3–3. lpm_divide Megafunction Parameters (Part 1 of 2)

Parameter	Type	Required	Description
LPM_WIDTHN	Integer	Yes	Width of the numer[] and quotient[] port.
LPM_WIDTHD	Integer	Yes	Width of the denom[] and remain[] port.
LPM_NREPRESENTATION	String	No	Sign representation of the numerator input: "SIGNED" or "UNSIGNED". The signed representation for all library of parameterized modules (LPM) megafunctions is two's complement.
LPM_DREPRESENTATION	String	No	Sign representation of the denominator input: "SIGNED" or "UNSIGNED". The signed representation for all library of parameterized modules (LPM) megafunctions is two's complement.
LPM_REMAINDERPOSITIVE	String	No	You must use the LPM_HINT parameter to specify the LPM_REMAINDERPOSITIVE parameter in VHDL Design Files. Values are "TRUE" or "FALSE". If this parameter is set to "TRUE", the value of the remain[] port must be greater than or equal to zero. If this parameter is set to "TRUE", the value of the remain[] port is either zero, or the value is the same sign, either positive or negative, as the value of the numer port. See Tables 3–4 and 3–5. To reduce area and improve speed, it is recommended that you set this parameter to "TRUE" in operations where the remainder must be positive or where the remainder is unimportant.
MAXIMIZE_SPEED	Integer	No	Specify a value between 0 and 9. If used, the Quartus II software attempts to optimize a specific instance of the lpm_divide function for speed rather than routability, and overrides the setting of the **Optimization Technique** logic option. If MAXIMIZE_SPEED is unused, the value of the **Optimization Technique** option is used instead. If the setting for MAXIMIZE_SPEED is 6 or higher, the Compiler optimizes lpm_divide megafunctions for higher speed by using carry chains; if the setting is 5 or less, the compiler implements the design without carry chains.

Table 3–3. lpm_divide Megafunction Parameters (Part 2 of 2)

Parameter	Type	Required	Description
LPM_PIPELINE	Integer	No	Specifies the number of clock cycles of latency associated with the quotient[] and remain[] outputs. A value of zero (0) indicates that no latency exists, and that a purely combinational function is instantiated. The default is 0 (non-pipelined). You cannot specify a value for the LPM_PIPELINE parameter that is higher than LPM_WIDTHN.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.
LPM_HINT	String	No	Lets you specify Altera®-specific parameters, for example, LPM_REMAINDERPOSITIVE and MAXIMIZE_SPEED, in VHDL Design Files. The default is "UNUSED".
SKIP_BITS	Integer	No	Allows for more efficient fractional bit division to optimize away logic on the leading bits by providing the number of leading GND to the lpm_divide megafunction. Specify the number of leading GND on the quotient output to this parameter.

Tables 3–4 and 3–5 show the truth table or functionality of the LPM_REMAINDERPOSITIVE parameter.

Table 3–4. lpm_divide Megafunction Truth Table/Functionality: LPM_REMAINDERPOSITIVE = 'TRUE' (Part 1 of 2)

Inputs		Outputs	
numer[]	denom[]	quotient[]	remain[]
3	3	1	0
3	2	1	1
3	1	3	0
3	0	×	×
3	–1	–3	0
3	–2	–1	1
3	–3	–1	0
–3	3	–1	0
–3	2	–2	1
–3	1	–3	0
–3	0	×	×

Table 3–4. lpm_divide Megafunction Truth Table/Functionality: LPM_REMAINDERPOSITIVE = 'TRUE' (Part 2 of 2)

Inputs		Outputs	
numer[]	denom[]	quotient[]	remain[]
–3	–1	3	0
–3	–2	2	1
–3	–3	1	0

Table 3–5. lpm_divide Megafunction Truth Table/Functionality: LPM_REMAINDERPOSITIVE = 'FALSE'

Inputs		Outputs	
numer[]	denom[]	quotient[]	remain[]
3	3	1	0
3	2	1	1
3	1	3	0
3	0	×	×
3	–1	–3	0
3	–2	–1	1
3	–3	–1	0
–3	3	–1	0
–3	2	–1	–1
–3	1	–3	0
–3	0	×	×
–3	–1	3	0
–3	–2	1	–1
–3	–3	1	0