



lpm_counter 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

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



UG-MF9104-2.3

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

Chapter 1. About this Megafunction

Device Family Support	1-1
Introduction	1-1
Features	1-2
General Description	1-2
Common Applications	1-2
Resource Utilization and Performance	1-3

Chapter 2. Getting Started

Software and System Requirements	2-1
MegaWizard Customization	2-1
MegaWizard Page Descriptions	2-1
Inferring Megafunctions from HDL Code	2-6
Instantiating Megafunctions in HDL Code or Schematic Designs	2-7
Identifying a Megafunction after Compilation	2-7
Simulation	2-7
Quartus II Software Simulation	2-7
EDA Simulation	2-8
SignalTap II Embedded Logic Analyzer	2-8
Design Example: 9-Bit Up Counter	2-9
Design Files	2-9
Example	2-9
Generate a 9-Bit Up-Direction Counter	2-9
Implement the 9-Bit Up-Direction Counter	2-15
Functional Results—Simulate the Counter Design in Quartus II	2-17
Functional Results—Simulate the Counter in ModelSim-Altera	2-18
Conclusion	2-19

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	<ul style="list-style-type: none">• Updated Device Family Support on page 1–1• Updated Resource Utilization and Performance on page 1–3• Updated MegaWizard Customization on page 2–1• Updated MegaWizard Page Descriptions on page 2–1• Updated Design Example: 9-Bit Up Counter on page 2–9	Updated for the Quartus® II version 7.1 software release.
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	Revised for Quartus II 6.0 software release.	
September 2006 v1.0	Initial release	

Referenced Documents

This section lists the documents referenced in this user guide.

- [Volume 1 of the Quartus II Handbook](#)
- [Recommended HDL Coding Styles](#) chapter in volume 1 of the *Quartus II Handbook*
- [Design Debugging Using the SignalTap II Embedded Logic Analyzer](#) chapter in volume 3 of the *Quartus II Handbook*

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Altera literature services	Email	literature@altera.com
Non-technical support (General) (Software Licensing)	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: f_{MAX} , lqdesigns directory, d: drive, chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75: High-Speed Board Design</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: <i>t_{PIA}</i> , <i>n</i> + 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."

Visual Cue	Meaning
Courier type	Signal and port names are shown in lowercase Courier type. Examples: <code>data1</code> , <code>tdi</code> , <code>input</code> . Active-low signals are denoted by suffix <code>n</code> , e.g., <code>resetn</code> . Anything that must be typed exactly as it appears is shown in Courier type. For example: <code>c:\qdesigns\tutorial\chiptrip.gdf</code> . Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword <code>SUBDESIGN</code>), as well as logic function names (e.g., <code>TRI</code>) 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.
■ ● ●	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
 CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work.
 WARNING	A warning calls attention to a condition or possible situation that can cause injury to the user.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

Device Family Support

The `lpm_counter` megafunction supports the following target Altera® device families:

- Arria™ GX
- Stratix® III
- Stratix II
- Stratix
- Stratix II GX
- Stratix GX
- Cyclone® III
- Cyclone II
- Cyclone
- HardCopy® II
- HardCopy Stratix
- MAX® II
- MAX 7000AE
- MAX 7000B
- MAX 7000S
- MAX 2000A
- APEX™ II
- APEX 20KC
- APEX 20KE
- APEX 20K
- ACEX 1K®
- FLEX® 10KE
- FLEX 10KA
- FLEX 10K®
- 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 setting parameters.

Features

The `lpm_counter` megafunction implements a basic counter and offers many additional features, which include:

- Optimized for Arria GX, Stratix III, Stratix II, Stratix II GX, Stratix, Stratix GX, Cyclone III, Cyclone II, and Cyclone FPGAs
- Generates up, down, and up/down counters
- Supports counts from 2 to 256 bits wide
- Supports optional synchronous and asynchronous load capability
- Supports optional clock enable and asynchronous and synchronous controls
- Supports optional count enable control
- Supports user-defined maximum count modulus
- Provides optional counter decode output and active high when the counter reaches the specified count value (available only for the lowest 16 bits)
- Provides optional carry-in and carry-out ports

General Description

The `lpm_counter` megafunction is provided by the Quartus® II software MegaWizard® Plug-In Manager. The `lpm_counter` megafunction is a binary counter that creates up counters, down counters, and up/down counters with outputs up to 256 bits wide, and provides optional synchronous or asynchronous clear, set, and load ports. Altera recommends using the `lpm_counter` megafunction instead of other binary counters in Altera devices because it is optimized for performance and resource utilization in Altera architectures.

Common Applications

Counters are common and basic building-block functions that are useful in state machines, basic control logic, address increment to memory blocks, and timers. The `lpm_counter` megafunction lets you specify the counter's modulus (plain binary or customized modulus), preset the counter, set clock enable and count enable ports, and specify carry-in and carry-out ports. The `lpm_counter` megafunction meets most counter requirements.

Resource Utilization and Performance

The `lpm_counter` megafunction is implemented using logic elements (LEs) in FPGA devices. The `lpm_counter` megafunction uses the carry chains feature of Altera's FPGA devices to achieve high performance. [Table 1–1](#) shows the resource utilization and performance of the `lpm_counter` megafunction as used in different devices.

<i>Table 1–1. lpm_counter_Megafunction Resource Usage</i>				
Device Family	Bit Size	Resource Use <i>(1)</i>		Performance (MHz) <i>(2)</i>
		Register	LUT	
Stratix II	8	8	8	500.00
	64	64	64	242.07
Stratix III	8	8	8	772.80
	64	64	64	315.86
Cyclone III	8	8	8	525.21
	64	64	64	190.01

Notes to [Table 1–1](#):

- (1) You can get the resource usage information from the MegaWizard Plug-In Manager. The information in this table is valid and accurate in the Quartus II version 7.1.
- (2) The performance shown in [Table 1–1](#) is based on compilation reports from the Quartus II software version 7.1, using the default settings. If you use other versions of the Quartus II software, you might see minor differences in performance.

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.

You can use the MegaWizard Plug-In Manager to specify the `lpm_counter` megafunction features and parameters for each counter in your design. The MegaWizard Plug-In Manager also provides a wizard that allows you to set parameter values and select optional ports.

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 the command prompt:

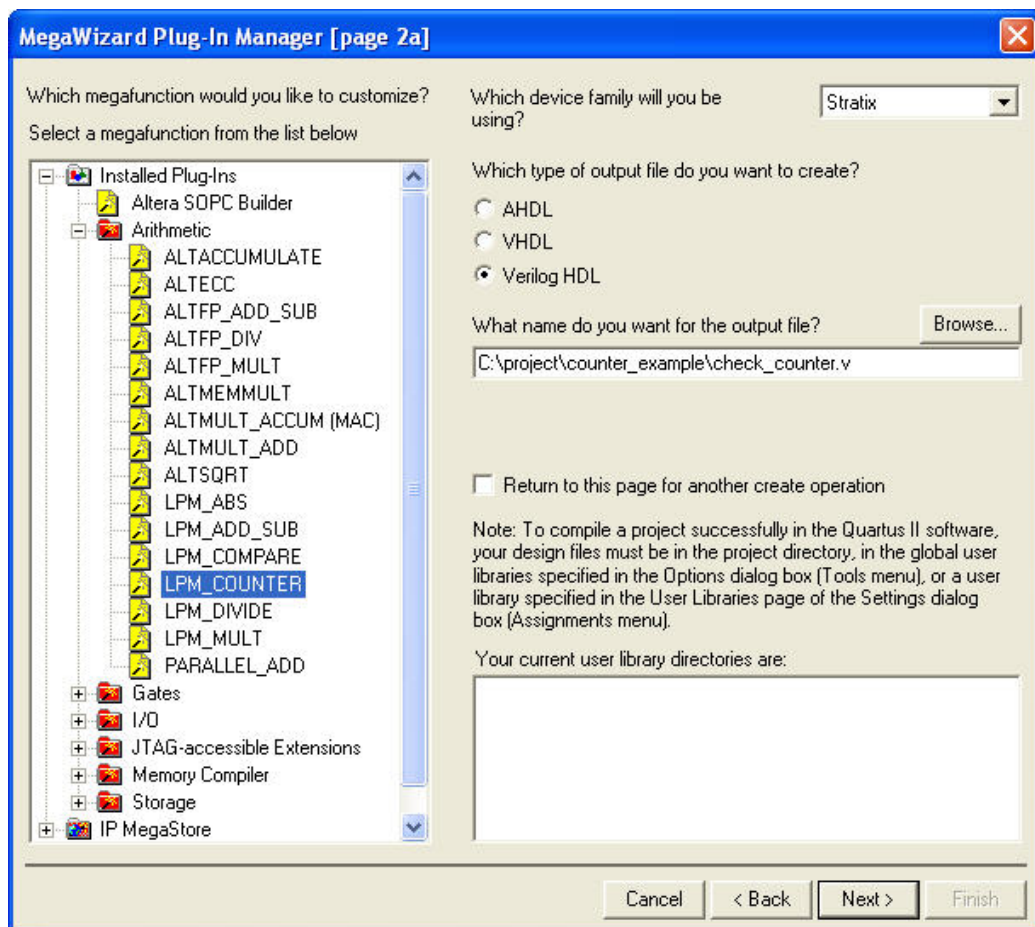
```
qmegawiz ↵
```

MegaWizard Page Descriptions

This section provides descriptions for the options available on the individual pages of the `lpm_counter` megafunction wizard.

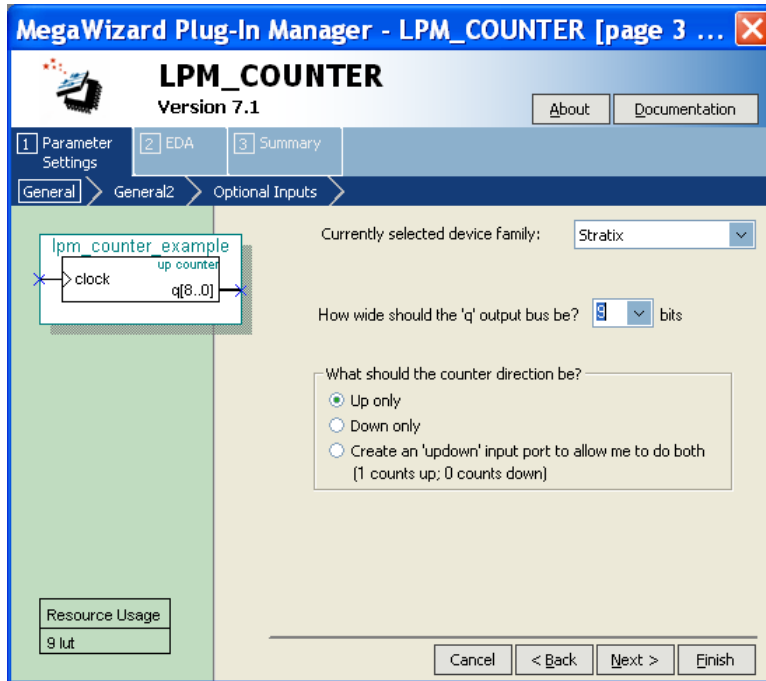
On page 2a, select the `lpm_counter` megafunction from the Arithmetic category, select the device you intend to use, the type of output file you want to create (Verilog, VHDL, or AHDL), and what you want to name the output file (Figure 2–1).

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



On page 3 of the `lpm_counter` wizard, specify the width and direction of the counter ([Figure 2-2](#)).

Figure 2-2. MegaWizard Plug-In Manager - LPM_COUNTER [page 3 of 7]



[Tables 2-1, 2-2, and 2-3](#) show the features and settings of the `lpm_counter` megafunction. Use these tables with the hardware descriptions for the features to determine the appropriate settings.

Table 2-1. `lpm_counter` Page 3 Options

Function	Description
How wide should the 'q' output bus be?	Specify the width of the <code>q []</code> counter output port. See the port description in Table 3-1 .
What should the counter direction be?	Specify the counter direction. See the parameter description in Table 3-3 .

On page 4 of the `lpm_counter` wizard, specify the counter modulus and other optional ports (Figure 2–3).

Figure 2–3. MegaWizard Plug-In Manager - LPM_COUNTER [page 4 of 7]

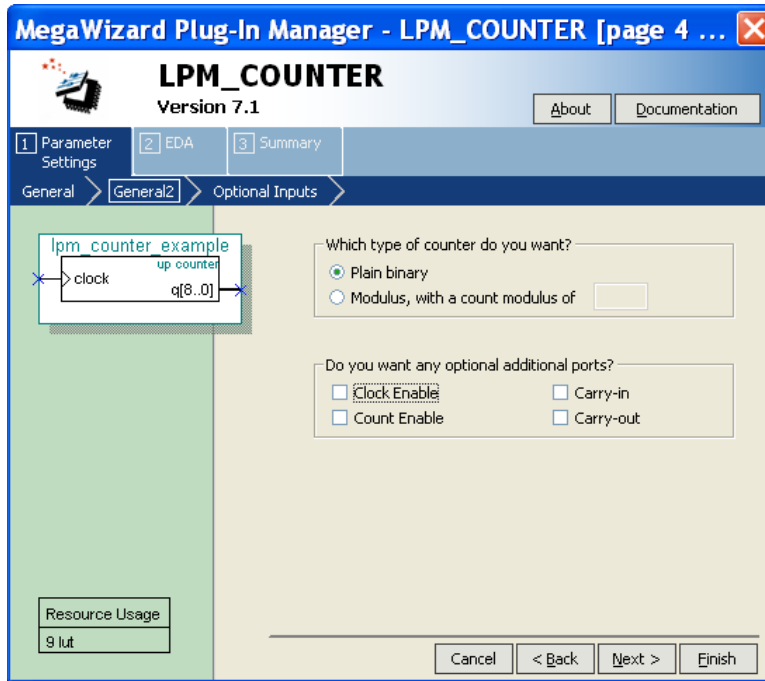


Table 2–2 shows the options available on Page 4 of the `lpm_counter` wizard.

Table 2–2. <code>lpm_counter</code> Wizard Page 4 Options	
Function	Description
Which type of counter do you want?	Specify whether the counter is a binary counter or a counter of user-specified modulus.
Do you want any optional additional ports?	Specify optional Clock Enable, Count Enable, Carry-in, and Carry-out ports. See the port descriptions in Table 3–1.

On page 5 of the `lpm_counter` wizard, specify the optional synchronous and asynchronous control inputs (Figure 2–4).

Figure 2–4. MegaWizard Plug-In Manager - LPM_COUNTER [page 5 of 7]

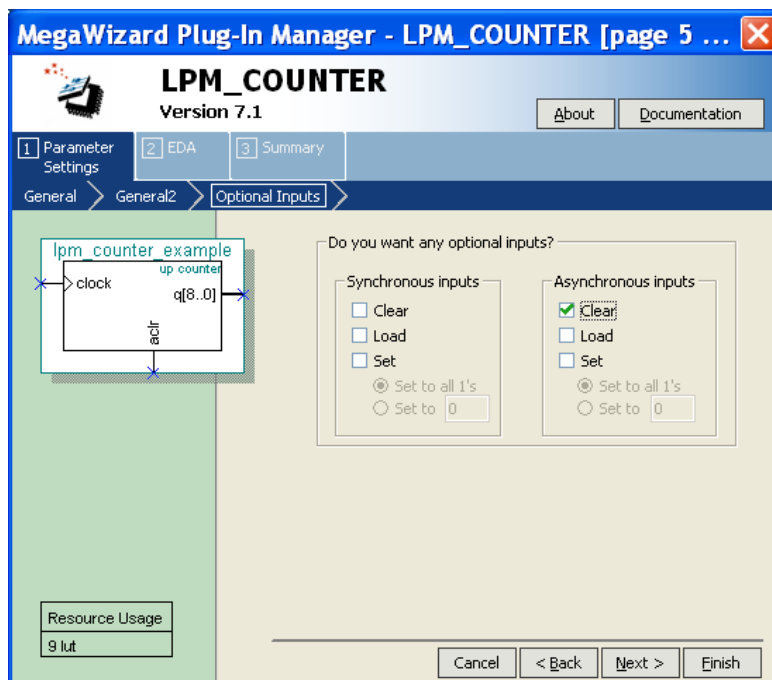


Table 2–3 shows the options available on page 5 of the lpm_counter wizard.

Table 2–3. lpm_counter MegaWizard Plug-In Manager Page 5 Options	
Function	Description
Do you want any optional inputs? (For Synchronous inputs)	Set optional synchronous Clear, Load, and Set inputs. You can specify the Set value. See the port descriptions in Table 3–1 .
Do you want any optional inputs? (For Asynchronous inputs)	Specify optional asynchronous Clear, Load, and Set inputs. You can specify the Set value. See the port descriptions in Table 3–1 .

Inferring Megafunctions from HDL Code

Synthesis tools, including the Quartus II integrated synthesis, recognize certain types of HDL code and automatically infer the appropriate megafunction when a megafunction provides 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/or 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.

To infer counter functions, synthesis tools look for a set of registers that feed through a plus-one adder (for example, $x = x + 1$), a minus-one adder (for example, $x = x - 1$), or both, and then convert the registers and logic to an `lpm_counter` megafunction. If a design also has logic that implements control signals that feed the function's registers or control the functionality of the counter, the synthesis tool recognizes them as well. For example, the Quartus II software recognizes the following signals:

- Asynchronous clear
- Asynchronous set (only to all logic value 1s)
- Asynchronous load
- Count enable
- Synchronous clear
- Synchronous set (only to all logic value 1s)
- Synchronous load
- Clock enable
- Up/down

Instantiating Megafunctions in HDL Code or Schematic Designs

When you use the MegaWizard Plug-In Manager to customize and parameterize a megafunction, it creates a set of output files that allow you to instantiate the customized function in your design. Depending on the language you choose in the MegaWizard Plug-In Manager, the MegaWizard instantiates the megafunction with the correct parameter values and generates a megafunction variation file (wrapper file) in Verilog HDL (.v), VHDL (.vhd), or AHDL (.tdf), along with other supporting files.

The MegaWizard Plug-In Manager provides options to create the following files:

- A sample instantiation template for the language of the variation file (**_inst.v**, **_inst.vhd**, or **_inst.tdf**)
- Component Declaration File (**.cmp**) that can be used in VHDL Design Files
- ADHL Include File (**.inc**) that can be used in Text Design Files (**.tdf**)
- Quartus II Block Symbol File (**.bsf**) that can be used in schematic designs
- Verilog HDL module declaration file that can be used when instantiating the megafunction as a black box in a third-party synthesis tool (**_bb.v**)



For more information about the MegaWizard-generated files, refer to the Quartus II Help or to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

Identifying a Megafunction after Compilation

The Quartus II software performs analysis and elaboration to build the structure of your design during compilation. 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 **Browse** 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 Software Simulation

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 delays 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 Registers: Presynthesis, Design Entry, or Pin Filters in the Node Finder. These three filters find the top-level ports of megafunctions.

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, 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 and 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.



For more information about these attributes, refer to 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, 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: 9-Bit Up Counter

Counters are common and basic building-block functions that are useful in state machines, basic control logic, address increment to memory blocks, and timers. This section presents a design example that uses the `lpm_counter` megafunction to generate a parameterized 9-bit up-direction counter.

This example uses the MegaWizard Plug-In Manager in the Quartus II software. As you go through the example, each page of the wizard is described in detail. When you are finished with this example, you can incorporate it into your overall project.

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_counter Megafunction User Guide* to download the design files.

Example

In this example, you perform the following activities:

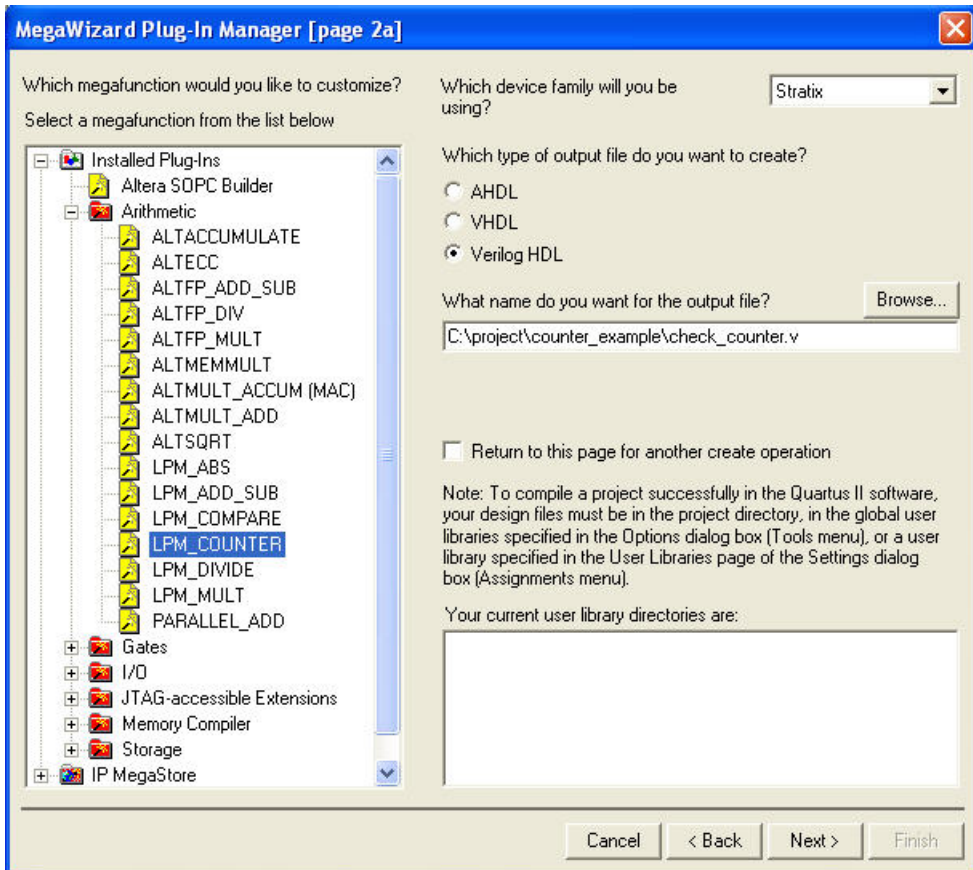
- Generate a parameterized 9-bit up-direction counter using the `lpm_counter` megafunction in the MegaWizard Plug-In Manager
- Implement the counter in the device by assigning the EP1S10F780C5 device to the project and compiling the project
- Simulate the up-direction counter design

Generate a 9-Bit Up-Direction Counter

1. Open the `lpm_counter_DesignExample.zip` project and extract `check_counter.qar`. In the Quartus II software, open `check_counter.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, `counter_example.bdf`.
3. Double-click on a blank area in the block design (`.bdf`) file.
4. In the **Symbol** dialog box, click **MegaWizard Plug-In Manager**.

5. On page 1 of the MegaWizard Plug-In Manager, in the **What action do you want to perform?** section, select **Create a new custom megafunction variation**.
6. Click **Next**. Page 2a appears (Figure 2–5).

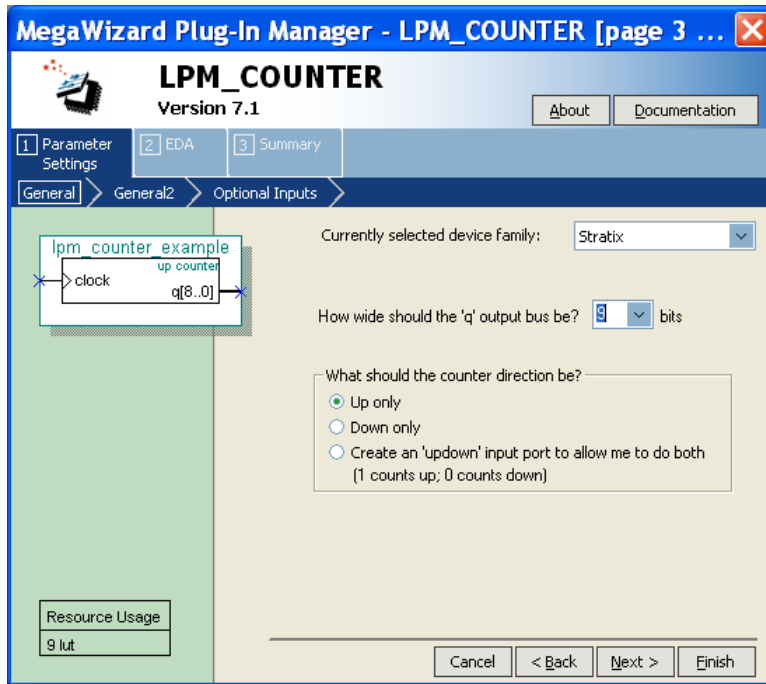
Figure 2–5. MegaWizard Plug In Manager [page 2a]



7. On page 2a, expand the **Arithmetic** folder and select **LPM_COUNTER**.
8. For **Which device family will you be using?**, select **Stratix**.
9. Under **Which type of output file do you want to create?** select the **Verilog HDL** option.

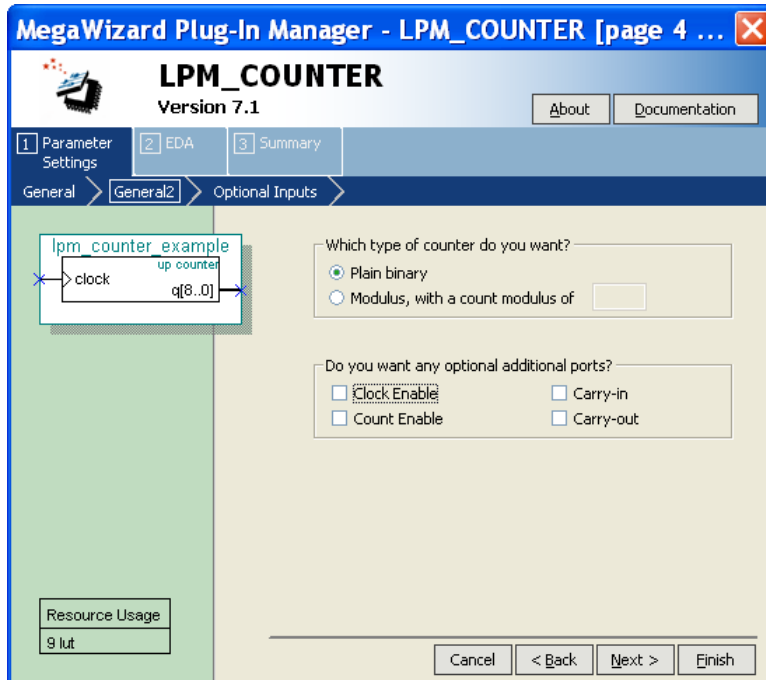
10. Name the output file `check_counter`.
11. Click **Next**. Page 3 displays (Figure 2–6).

Figure 2–6. MegaWizard Plug-In Manager - LPM_COUNTER [page 3 of 7]

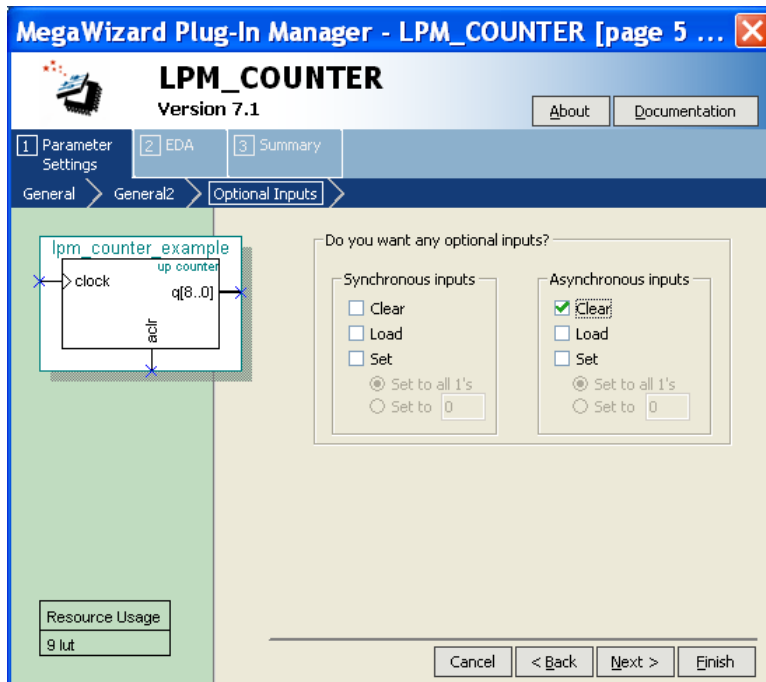


12. For **How wide should the 'q' output bus be?**, select 9 bits.
13. Under **What should the counter direction be?** select **Up only**.
14. Click **Next**. Page 4 appears (Figure 2–7).

Figure 2–7. MegaWizard Plug-In Manager - LPM_COUNTER [page 4 of 7]

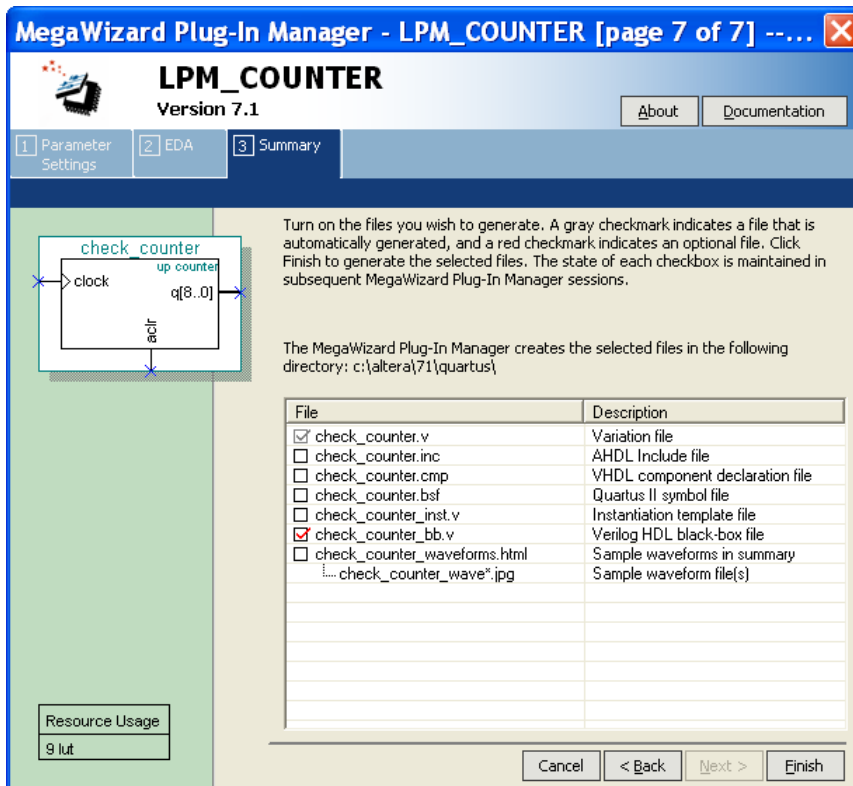


15. Under **Which type of counter do you want?**, select **Plain binary**.
16. Click **Next**. Page 5 appears (Figure 2–8).

Figure 2–8. MegaWizard Plug-In Manager - LPM_COUNTER [page 5 of 7]

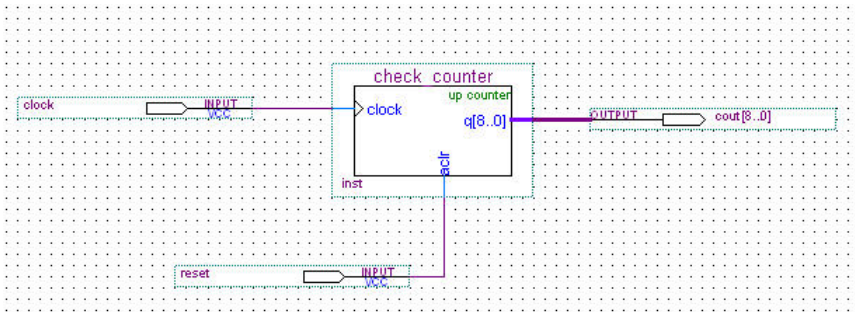
17. Under **Asynchronous inputs**, turn on **Clear**.
18. Click **Finish**. Page 7 appears (Figure 2–9).

Figure 2–9. MegaWizard Plug-In Manager - LPM_COUNTER [page 7 of 7]



19. Turn on the **check_counter_bb.v** (Verilog HDL black box file).
20. Click **Finish**.
21. In the Symbol dialog box, expand **Project** and select **check_counter**. Click **OK**.
22. Move the pointer to place the **check_counter** symbol in between the input/output ports in the **counter_example.bdf** file. Click to place the symbol. Adjust the symbol as necessary so the input/output ports connect.

You have now completed the design file, as shown in [Figure 2–10](#).

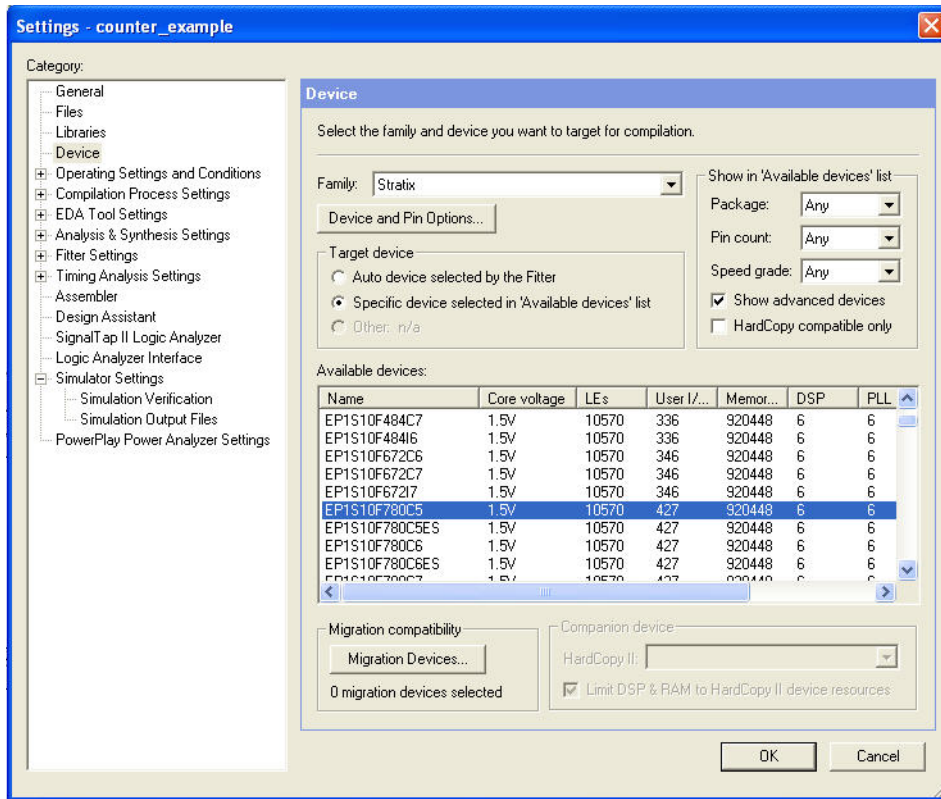
Figure 2–10. *lpm_counter* Design

23. On the File menu, select **Save**.

Implement the 9-Bit Up-Direction Counter

1. To open the **Settings** dialog box, on the Assignments menu, click **Settings**.
2. Click the **Devices** category. Ensure that **Stratix** is selected under the **Family** field, as shown in [Figure 2–11](#).

Figure 2–11. Settings - Device



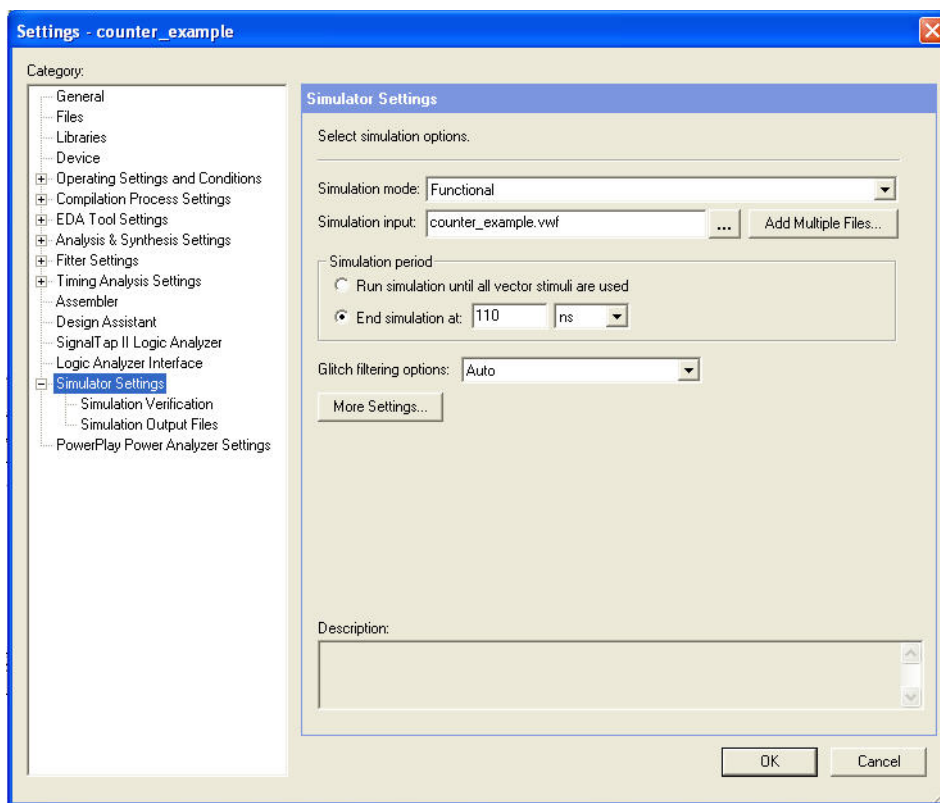
- Under **Target device**, select **EP1S10F780C5** in the **Available Devices:** list.
- Click **OK**.
- On the Processing menu, click **Start Compilation**, or on the toolbar, click the compilation button to compile the design.
- When the **Full Compilation was successful** message box appears, click **OK**.

Functional Results—Simulate the Counter Design in Quartus II

Simulate the design to verify the results. Set up the Quartus II Simulator by performing the following steps:

1. On the Processing menu, select **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**.

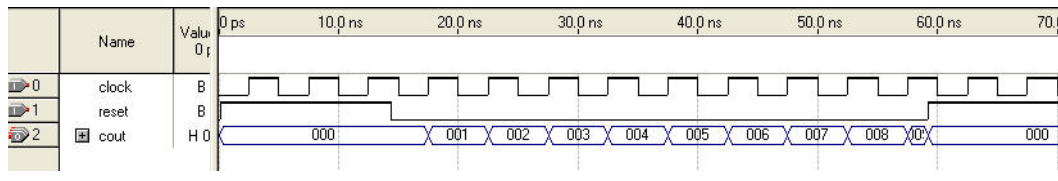
Figure 2–12. Settings - Simulator



4. In the **Category** list, select **Simulator Settings**.
5. In the **Simulation mode** list, select **Functional**.

6. Type `counter_example.vwf` in the **Simulation input** box, or click **Browse** to select the file in the project folder.
7. Turn on **End simulation at**. Type 110 and select **ns** (nanoseconds).
8. Click **OK**.
9. On the Processing menu, click **Start Simulation**, or on the toolbar, click the simulation button.
10. When the **Simulator was successful** message box appears, click **OK**.
11. In the Simulation Report window, view the simulation output waveforms and verify the results. Figure 2–13 shows the expected simulation results.

Figure 2–13. Simulation Waveforms



Functional Results—Simulate the Counter 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 <http://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:

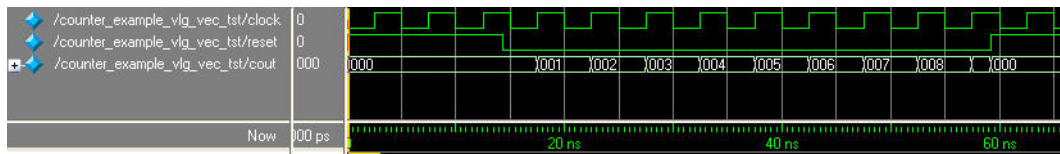
1. Unzip the **lpm_counter_ex_msim.zip** file to any working directory on your PC.
2. Start **ModelSim-Altera**.
3. On the File menu, click **Change Directory**.

4. Select the folder in which you unzipped the files. Click **OK**.
5. On the Tools menu, select **Execute Macro**.
6. Select the **counter_example.do** file and click **Open**. This is a script file for ModelSim that automates all 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 2–14 shows the expected simulation results in ModelSim.

Figure 2–14. 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. These megafunctions automate the coding process and save you valuable design time. You should use these functions during design implementation so you can consistently meet your design goals.

Ports and Parameters

Figure 3–1 below shows the ports and parameters for the `lpm_counter` megafunction. Table 3–1 shows the input ports, Table 3–2 shows the output ports, and Table 3–3 shows the `lpm_counter` megafunction parameters.



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

The 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 MegaWizard Plug-In Manager interface users.

Figure 3–1. `lpm_counter` Port and Parameter Description Symbol

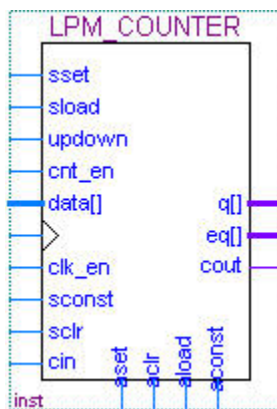


Table 3–1. `lpm_counter` Megafunction Input Ports (Part 1 of 2)

Name	Required	Description	Comment
<code>data[]</code>	No	Parallel data input to the counter	Input port LPM_WIDTH wide. Uses <code>aload</code> or <code>sload</code> .
<code>clock</code>	Yes	Positive-edge-triggered clock.	—
<code>clk_en</code>	No	Clock enable input	Enables all synchronous activities. If omitted, the default is 1 (enabled).

Table 3–1. *lpm_counter* Megafunction Input Ports (Part 2 of 2)

Name	Required	Description	Comment
<code>cnt_en</code>	No	Count enable input.	Disables the count when low (0) without affecting <code>sload</code> , <code>sset</code> , or <code>sclr</code> . If omitted, the default is 1 (enabled).
<code>updown</code>	No	Controls the direction of the count. High (1) = count up. Low (0) = count down.	If omitted, the default is 1 (enabled). If the <code>LPM_DIRECTION</code> parameter is used, the <code>updown</code> port cannot be connected. If <code>LPM_DIRECTION</code> is not used, the <code>updown</code> port is optional.
<code>cin</code>	No	Carry-in to the low-order bit.	When <code>cin</code> and <code>cnt_en</code> is used, both inputs are similar in behavior for “up” and “down” counters. However, when <code>cout</code> is used, the behavior of <code>cin</code> and <code>cout</code> input is combinational for “up” and “down” counters. Thus, when <code>cout</code> and <code>cnt_en</code> is used, the behavior for <code>cin</code> input is not similar to <code>cnt_en</code> in “up” or “down” counters.
<code>aclr</code>	No	Asynchronous clear input.	If omitted, the default is 1 (enabled). If both <code>aset</code> and <code>aclr</code> are used and asserted, <code>aclr</code> overrides <code>aset</code> .
<code>aset</code>	No	Asynchronous set input.	If omitted, the default is 1 (enabled). Sets <code>q[]</code> outputs to all 1s, or to the value specified by <code>LPM_AVALUE</code> . If both <code>aset</code> and <code>aclr</code> are used and asserted, <code>aclr</code> overrides <code>aset</code> .
<code>aload</code>	No	Asynchronous load input. Asynchronously loads the counter with the value on the data input.	If omitted, the default is 1 (enabled). If you use <code>aload</code> , <code>data[]</code> must be connected.
<code>sclr</code>	No	Synchronous clear input. Clears the counter on the next active clock edge.	If omitted, the default is 1 (enabled). If both <code>sset</code> and <code>sclr</code> are used and asserted, <code>sclr</code> overrides <code>sset</code> .
<code>sset</code>	No	Synchronous set input. Sets the counter on the next active clock edge	If omitted, the default is 1 (enabled). Sets <code>q[]</code> outputs to all 1s, or to the <code>LPM_SVALUE</code> value. If both <code>sset</code> and <code>sclr</code> are used and asserted, <code>sclr</code> overrides <code>sset</code> .
<code>sload</code>	No	Synchronous load input. Loads the counter with <code>data[]</code> on the next active clock edge.	If omitted, the default is 1 (enabled). If <code>sload</code> is used, <code>data[]</code> must be connected.

Table 3–2. *lpm_counter* Megafunction Output Ports

Name	Required	Description	Comment
q[]	No	Data output from the counter.	Output port LPM_WIDTH wide. Either q[] or at least one of the eq[15..0] ports must be connected.
eq[15..0] (1)	No	Counter decode output. Active high when the counter reaches the specified count value.	(AHDL only) Either the q[] port or eq[] port must be connected. Up to ceq ports can be used (0 <= c <= 15). Only the 16 lowest count values are decoded. When the count value is c, the eqc output is set high (1). For example, when the count is 0, eq0 = 1. When the count is 1, eq1 = 1. When the count is 15, eq15 = 1. Decoded output for count values of 16 or greater requires external decoding. The eq[15..0] outputs are asynchronous to the q[] output.
cout	No	Carry-out of the MSB.	

Note to Table 3–2:

- (1) The eq[15..0] port is not accessible using the MegaWizard Plug-In Manager. This port can be used only while implementing the counter in AHDL.

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

Name	Type	Required	Comment
LPM_WIDTH	Integer	Yes	The number of bits in the count or the width of the q[] and data[] ports, if they are used
LPM_DIRECTION	String	No	Values are "UP", "DOWN", and "UNUSED". If you use the LPM_DIRECTION parameter, the updown[] port cannot be connected. When the updown[] port is not connected, the default for the LPM_DIRECTION parameter is "UP".
LPM_MODULUS	Integer	No	The maximum count, plus one. Number of unique states in the counter's cycle. If the load value is larger than the LPM_MODULUS parameter, the behavior of the counter is not specified.

Table 3–3. *lpm_counter* Megafunction Parameters (Part 2 of 2)

Name	Type	Required	Comment
LPM_AVALUE	Integer/ String	No	Constant value that is loaded when <code>aset</code> is high. If the value specified is larger than or equal to <code><modulus></code> , the behavior of the counter is an undefined (X) logic level, where <code><modulus></code> is <code>LPM_MODULUS</code> , if present, or $2^{\text{LPM_WIDTH}}$. You should specify this value as a decimal number for AHDL designs.
LPM_SVALUE	Integer/ String	No	Constant value that is loaded on the rising edge of the clock when either <code>sset</code> or <code>sconst</code> is high. Must be used if you use <code>sconst</code> . You should specify this value as a decimal number for AHDL designs.
LPM_HINT	String	No	Lets you specify Altera-specific parameters in VHDL Design Files (<code>.vhd</code>). The default is "UNUSED".
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.
CARRY_CNT_EN	String	No	Altera-specific parameter. Values are "SMART", "ON", "OFF", or "UNUSED". Enables the <code>lpm_counter</code> function to propagate the <code>cnt_en</code> signal through the carry chain. In some cases, the <code>CARRY_CNT_EN</code> parameter setting may have a slight impact on the speed, so you might want to turn it off. The default value is "SMART", which provides the best trade-off between size and speed.
LABWIDE_SCLR	String	No	Altera-specific parameter. Values are "ON", "OFF", or "UNUSED". The default value is "ON". Lets you disable the use of the LAB-wide <code>sclr</code> feature found in ACEX® 1K, APEX™ 20K, APEX II, ARM®-based Excalibur™, FLEX® 6000, FLEX 10K®, and Mercury™ devices. Turning this option off increases the chances of fully using the partially-filled LABs, and might allow higher logic density when <code>SCLR</code> does not apply to a complete LAB. This parameter is available for backward compatibility. You should not use this parameter.
LPM_PORT_UPDOWN	String	No	Specifies the usage of the updown input port. Values are: "PORT_USED" = Port is treated as used. "PORT_UNUSED" = Port is treated as unused. "PORT_CONNECTIVITY" = Port usage is determined by checking the port connectivity. If omitted, the default is "PORT_CONNECTIVITY".