lpm_compare Megafunction

User Guide

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com

Document Version: 2.2
Software Version: 7.0
Document Date: March 2007

UG-MF83004-2.2

Contents

About this User Guide

Revision History

The table below displays the revision history for this User Guide.

Date/Version	Changes Made	Summary of Changes
March 2007 v2.2	Added Cyclone® III to list of supported devices.	Updated for Quartus® II version 7.0 by adding support for Cyclone III device.
December 2006 v2.1	Minor addition of Stratix III information.	
May 2006 v2.0	Revised for the Quartus II 6.0 software release.	
September 2004 v1.0	Initial release.	

How to Contact Altera

For the most up-to-date information about Altera® products, go to the Altera world-wide web site 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 below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/	altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 *(1)* (7:00 a.m. to 5:00 p.m. Pacific Time)
Product literature	www.altera.com	www.altera.com
Altera literature services	lit_req@altera.com *(1)*	lit_req@altera.com *(1)*
Non-technical customer service	(800) 767-3753 (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
FTP site	ftp.altera.com	ftp.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 below.

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.
■ ● •	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.

Chapter 1. About this MegaCore Function

Device Family Support

The lpm_compare megafunction supports the following target Altera device families:

- 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
- ACEX 1K®
- APEX™ II
- APEX 20KC
- APEX 20KE
- FLEX 10K®
- FLEX® 10KA
- FLEX 10KE
- FLEX 6000

Introduction

As design complexities increase, use of vendor-specific 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_compare megafunction implements a basic comparator and offers many features, including:

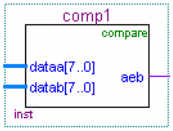
- Parameterizable input data widths
- Parameterizable output types
- Ability to specify a constant value for the `datab[]` input to optimize implementation
- Support for both signed and unsigned data representation formats
- Support for pipelining with parameterized output latency
- Active high asynchronous clear and clock enable control inputs

General Description

The lpm_compare megafunction is one of the arithmetic megafunctions provided in the Quartus® II software MegaWizard® Plug-In Manager. The basic function of a comparator is to compare the value of two sets of data to determine the relationship. In its simplest form, you can use an **exclusive-OR** gate to determine whether two bits of data are equal.

The lpm_compare megafunction lets you implement a comparator, called a "relational operator," in AHDL, VHDL, and Verilog HDL. It is used to compare nodes, groups, and numbers. Figure 1–1 shows a basic lpm_compare megafunction testing for equality with an input data width of eight bits and a signed number representation.

Figure 1–1. Basic lpm_compare Symbol



Common Applications

Comparator applications include parity and magnitude detectors, state machines, basic control logic, and any others which involve determining data relationships.

For details about using a comparator in parity detection applications, refer to the white paper *Using Parity to Detect Memory Errors in Stratix Devices*.

Resource Utilization & Performance

The lpm_compare megafunction is implemented in adaptive look-up tables (ALUTs) of the Stratix II device and in logic elements (LEs) of the Stratix, Stratix GX, HardCopy Stratix, Cyclone II, and Cyclone devices. Figure 1–2 shows how the Quartus II software illustrates the placement of lpm_compare megafunction in the Floorplan Editor.

Figure 1–2. Floorplan Editor View of lpm_compare Implementation

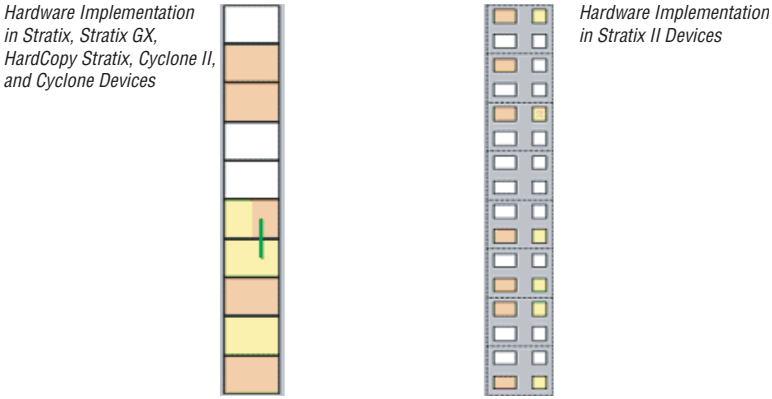


Table 1–1 summarizes the resource usage for an lpm_compare function used to implement an 8-bit unsigned comparator.

Table 1–1. lpm_compare Megafunction Resource Usage

Device	Optimization	Width	Logic Usage
Stratix II	Speed	8	7 ALUTs
	Balanced	8	7 ALUTs
	Area	8	7 ALUTs
Stratix, Stratix GX	Speed	8	7 LEs
Cyclone II, Cyclone	Balanced	8	7 LEs
HardCopy Stratix	Area	8	7 LEs

Chapter 2. Getting Started

Software and System Requirements

The instructions in this section require the following software:

- For operating system support information, refer to:

http://www.altera.com/support/software/os_support/oss-index.html

- Quartus® II software beginning with version 6.1

MegaWizard Plug-In Manager Customization

You can use the MegaWizard® Plug-In Manager to set the lpm_compare megafunction features for each comparator in the design.

Search for lpm_compare in the Quartus II Help for a listing of the parameters to use when instantiating the megafunction without using the MegaWizard Plug-In Manager.

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

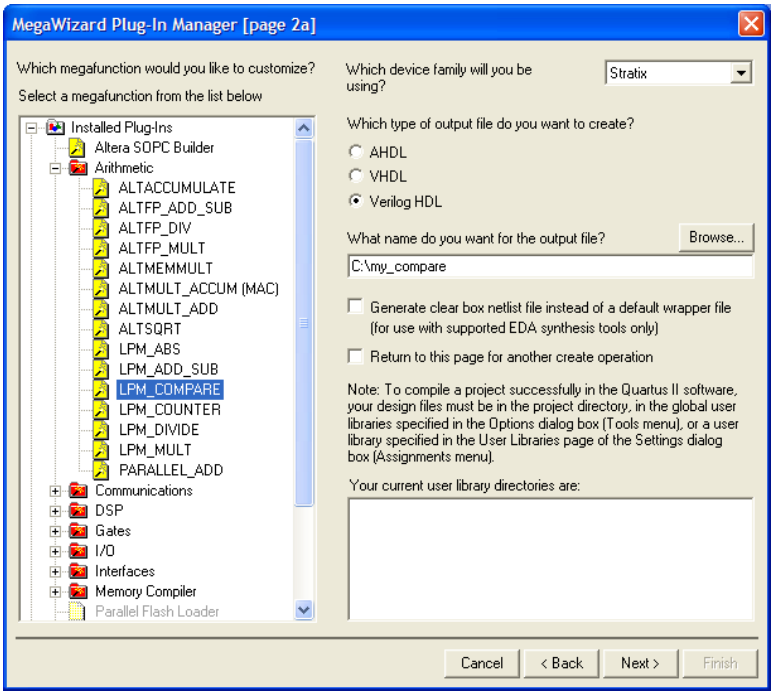
- On the Tools menu, click **MegaWizard Plug-In Manager**.
- When working in the Block Editor, click **MegaWizard Plug-In Manager** in the **Symbol** dialog box.
- Start the stand-alone version of the MegaWizard Plug-In Manager by typing the following command at the command prompt:
  `qmegawiz↵`

Using the MegaWizard Plug-In Manager

This section provides an in-depth description of each page in the lpm_compare megafunction. Tables 2–1, 2–2, and 2–3 show the features and settings for the lpm_compare megafunction. You can use these tables to determine appropriate settings for your comparator designs.

On Page 2a of the lpm_compare MegaWizard Plug-In Manager, select the LPM_COMPARE 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 - LPM_COMPARE [Page 2a]



On Page 3 of the lpm_compare MegaWizard Plug-In Manager specify the input data widths and which outputs you want (Figure 2–2).

Figure 2–2. MegaWizard Plug-In Manager—LPM_COMPARE [Page 3 of 7]

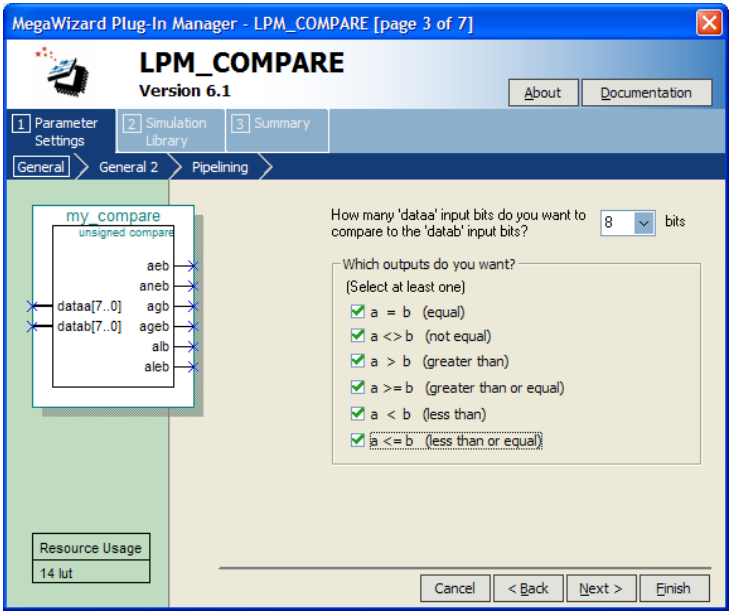


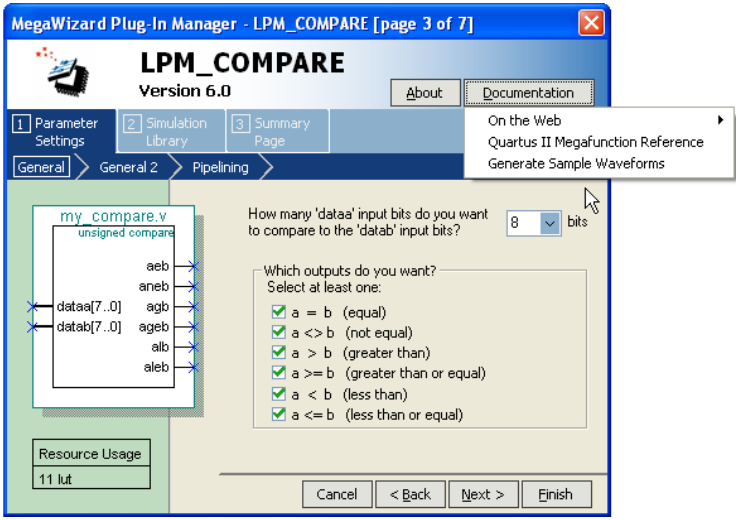
Table 2–1 shows the options available on Page 3 of the lpm_compare MegaWizard Plug-In Manager.

Table 2–1. lpm_compare MegaWizard Plug-In Manager Page 3

Function	Description
How many 'dataa' input bits do you want to compare to the 'datab' input bits?	Specify the width of the `dataa[]` input to be compared to the `datab[]` input.
Which outputs do you want? Select at least one:	Select one or more outputs from the compare operation.

Starting on Page 3, you can generate a sample simulation waveform or launch the Quartus II Help for the lpm_compare megafunction (Figure 2–3). To generate a sample waveform, click the **Documentation** button and select **Generate Sample Waveforms**. To launch the help, click the **Documentation** button and select **Quartus II Megafunction Reference**.

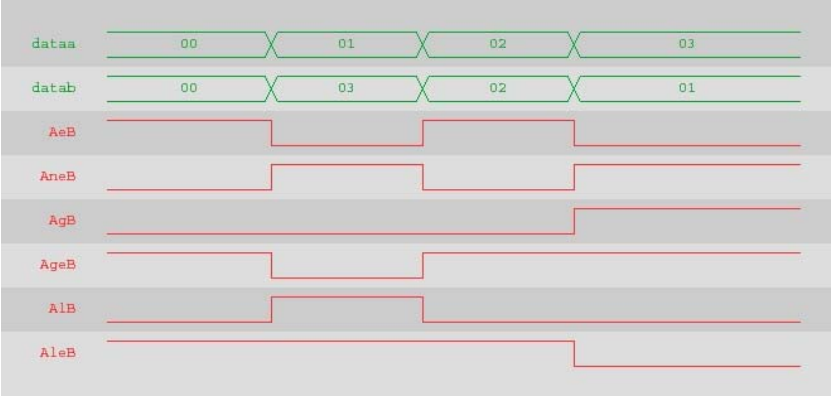
Figure 2–3. Generating Sample Waveforms for the lpm_compare Megafunction [Page 3 of 7]



The option to Generate A Sample Simulation Waveform and to launch the Quartus II Help for the lpm_compare megafunction is only available from the Quartus II software version 4.0 and later.

The sample waveform illustrates the behavior of the lpm_compare megafunction for a chosen set of parameters in the lpm_compare design module (Figure 2–4). This option generates a sample waveform file in HTML format in the specified lpm_compare design directory. The HTML file contains descriptions showing the comparator operation.

Figure 2–4. Sample Waveforms for the lpm_compare Megafunction *Note (1)*



Note to Figure 2–4:

(1) The `datab[]` input of this lpm_compare instantiation is a constant value of 1.

On Page 4 of the lpm_compare MegaWizard Plug-In Manager, specify whether the `datab[]` input bus value is variable or constant and set the sign representation of the inputs (Figure 2–5).

Figure 2–5. MegaWizard Plug-In Manager - LPM_COMPARE [Page 4 of 7]

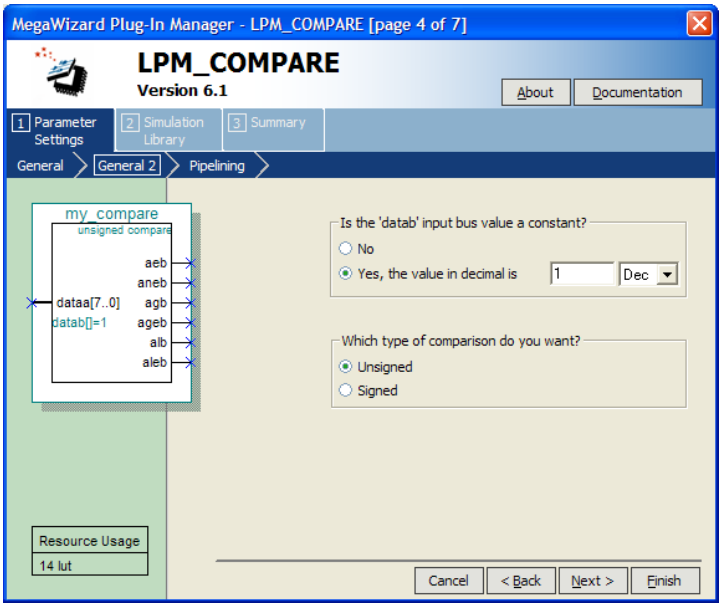


Table 2–2 shows the options available on Page 4 of the lpm_compare MegaWizard Plug-In Manager.

Table 2–2. lpm_compare MegaWizard Plug-In Manager Page 4

Function	Description
Is the 'datab' input bus value a constant?	Select **Yes** to set the width of the `datab[]` port to a constant value.
Which type of comparison do you want?	Select the sign representation format of the inputs: **Unsigned** or **Signed**. The default is **Unsigned**.

On Page 5 of the lpm_compare MegaWizard Plug-In Manager, you specify whether or not you want to pipeline the function, and if so, the number of output latency Clock cycles you want. You can also choose to create an asynchronous Clear input, a Clock Enable input, or both (Figure 2–6).

Figure 2–6. MegaWizard Plug-In Manager - LPM_COMPARE [Page 5 of 7]

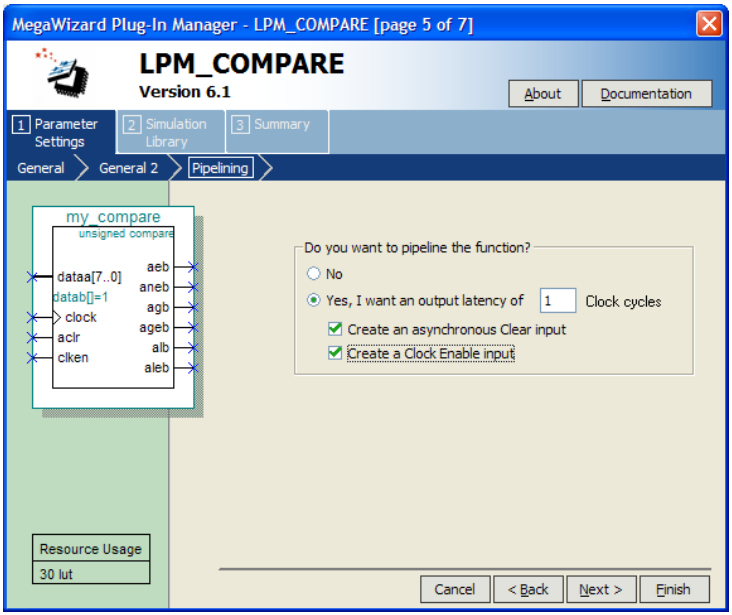


Table 2–3 shows the options available on the Page 5 of the lpm_compare MegaWizard Plug-In Manager.

Table 2–3. lpm_compare MegaWizard Plug-In Manager Page 5

Function	Description
Do you want to pipeline the function?	Creates a clock port to register and provide a pipelined operation for the lpm_compare function.
Create an asynchronous Clear input	Enables an active high asynchronous clear control signal for registered usage.
Create a Clock Enable input	Enables an active high clock enable control signal for registered usage.

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 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 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 generate a fully synthesizable netlist for improved results with EDA synthesis tools, such as Synplify and Precision RTL Synthesis (a clear-box methodology). Both clear-box and black-box methodologies are described in volume 1 of the *Quartus II Handbook*.

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 **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 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. With the registers, you can find pre-synthesis, design entry, or pin filters in the Node Finder. The top-level ports of megafunctions are found using these three filters.

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 & 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 about these attributes is available in volume 1 of the *Quartus II Handbook*.

EDA Simulation

Depending on the simulation tool you are using, refer to the appropriate chapter in volume 3 of the *Quartus II Handbook*. The *Quartus II Handbook* chapters show 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 where 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 volume 3 in the *Quartus II Handbook*.

Design Example: 8-Bit Comparator

Comparators are common basic building block functions useful in determining data relationships. This section presents a design example that uses the lpm_compare megafunction to generate a basic, 8-bit comparator. This example uses the MegaWizard Plug-In Manager in the Quartus II software to customize this megafunction. As you go through the wizard, each page is described in detail. When you are finished with this example, you can incorporate it into the overall design.

Design Files

The example design files are available in the Quartus II Projects section of the Design Examples page of the Altera web site.

Example

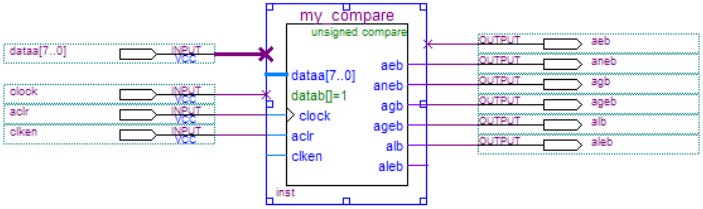
This example shows how to instantiate an lpm_compare megafunction using the MegaWizard Plug-In Manager. In this case, an lpm_compare megafunction is instantiated with all the possible output combinations enabled to show the various available results in simulation. You can change the parameters as needed for your design.

In this example, you will perform the following activities:

- Generate an 8-bit comparator design module.
- Implement the comparator design module in an Altera architecture by assigning the Stratix® II EP2S15F484C3 device and compiling the project.
- Simulate the customized comparator design module.

Figure 2–7 shows the lpm_compare megafunction design.

Figure 2–7. lpm_compare Megafunction Design



Generate an 8-Bit Comparator

1. Open the project file **compare.qpf**.

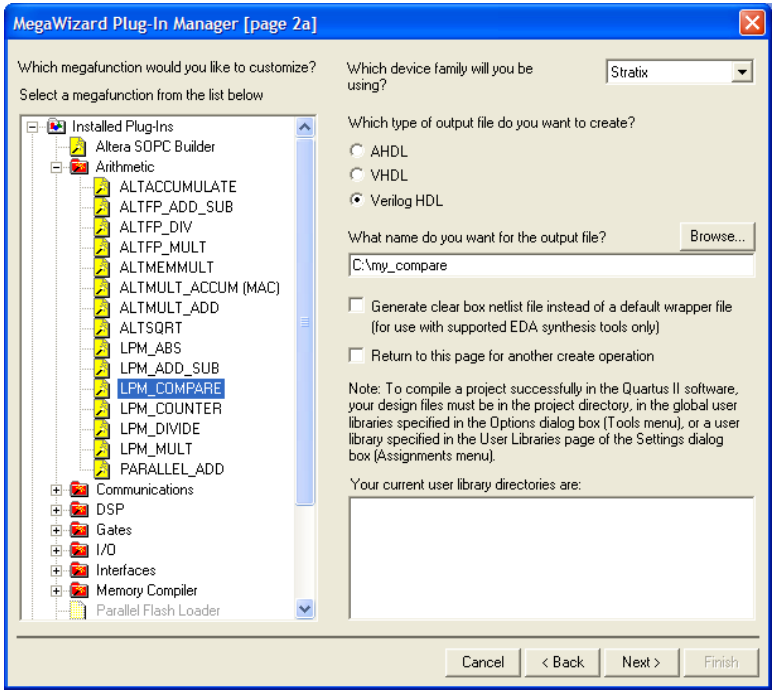
2. Open the top-level file **compare.bdf**. This file is an incomplete file that you will complete in the course of this example.

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

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

5. Click **Next**. Page 2a appears (Figure 2–8).

Figure 2–8. MegaWizard Plug-In Manager - LPM_COMPARE [Page 2a]



6. On Page 2a, expand the **Arithmetic** folder and select the LPM_COMPARE megafunction.

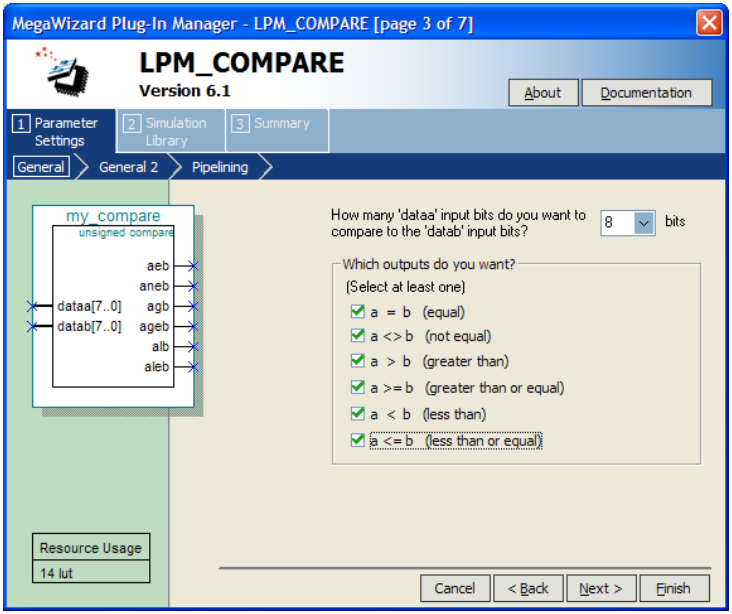
7. For the **Which device family will you be using?** option, select **Stratix II**.

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

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

10. Click **Next**. Page 3 appears (Figure 2–9).

Figure 2–9. MegaWizard Plug-In Manager - LPM_COMPARE [Page 3 of 7]

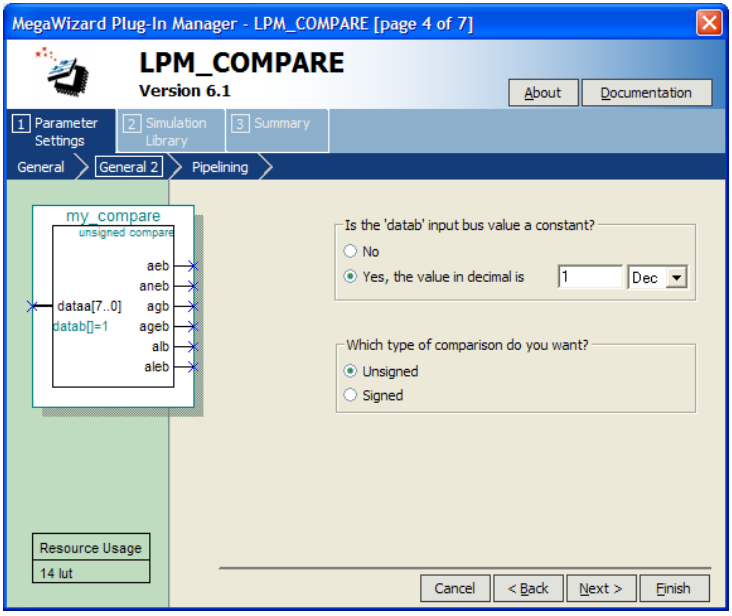


11. On Page 3 of the lpm_compare MegaWizard Plug-In Manager, select **8 bits** for **How many 'dataa' input bits do you want to compare to the 'datab' input bits?**.

12. Under **Which outputs do you want?**, select all available outputs.

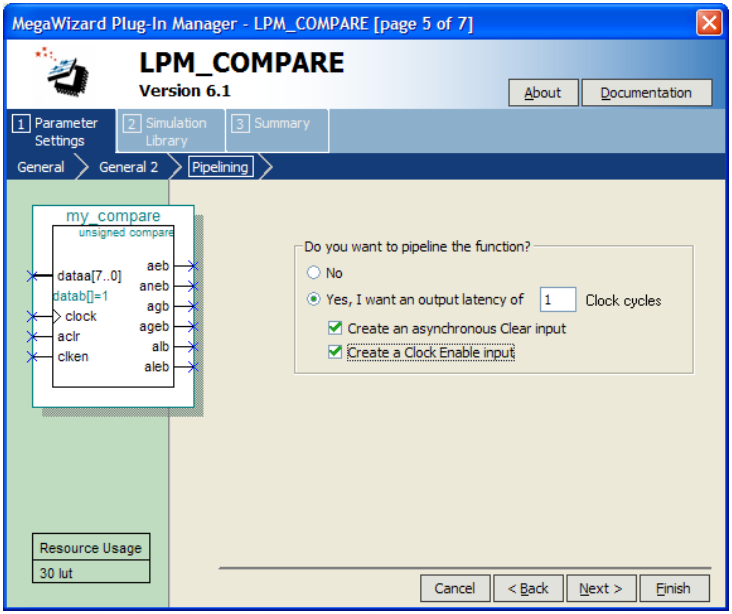
13. Click **Next**. Page 4 appears (Figure 2–10).

Figure 2–10. MegaWizard Plug-In Manager - LPM_COMPARE [Page 4 of 7]



14. On Page 4, under the **Is the 'datab' input bus value a constant?** section, select **Yes, the value in decimal is**.
15. In the **Yes, the value in decimal is** box, type `1`.
16. Under the **Which type of comparison do you want**? section, select **Unsigned**.
17. Click **Next**. Page 5 appears (Figure 2–11).

Figure 2–11. MegaWizard Plug-In Manager - LPM_COMPARE [Page 5 of 7]

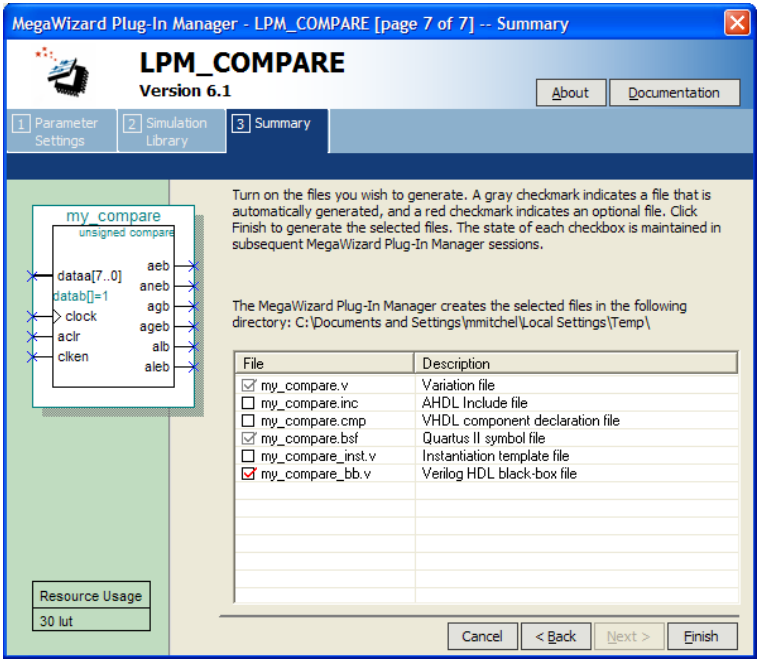


18. On Page 5, under the **Do you want to pipeline the function**? section, select **Yes, I want an output latency of,** and type `1`.

19. Select **Create an Asynchronous Clear input** and **Create a Clock Enable input**.

20. Click **Finish**. Page 7 appears (Figure 2–12).

Figure 2–12. MegaWizard Plug-In Manager - LPM_COMPARE [Page 7 of 7]--Summary



21. Ensure that the option to generate the Quartus II software Block Symbol File (**.bsf**) is selected.

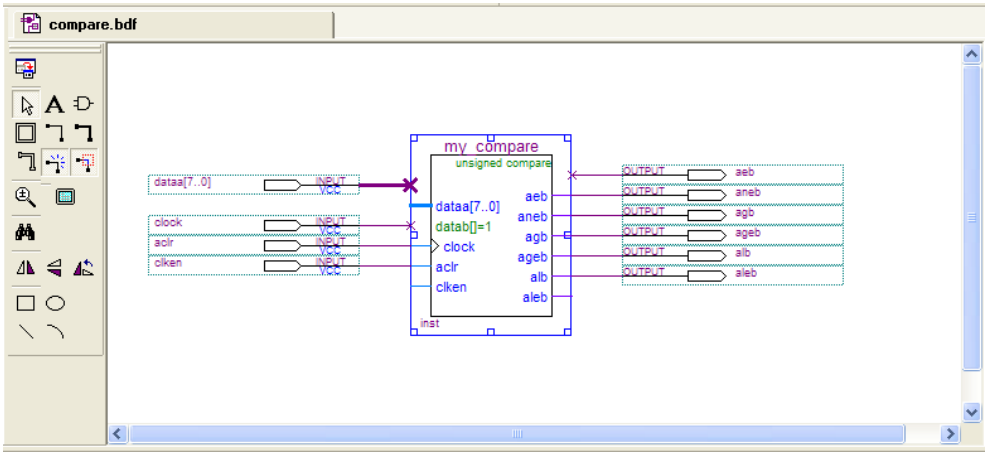
22. Click **Finish**. The lpm_compare megafunction is built.

23. In the Symbol window, click **OK**.

24. Move the pointer to place the **my_compare** symbol in between the input and output ports of the **compare.bdf** file. Click to place the **my_compare** symbol.

    You have now completed the design file (Figure 2–13).

Figure 2–13. lpm_compare Completed Design File



25. On the File menu, select **Save** to save the design.

Implement the 8-Bit Comparator

Next, you will compile the project.

1. On the Processing menu, select **Start Compilation** or click on the compile symbol to compile the design.
2. If you are prompted to **Save changes to compare.bdf?,** click **Yes**.
3. When the **Full Compilation was successful** message box appears, click **OK**.
4. On the Assignment menu, select **Timing Closure Floorplan** to view how the module is implemented in the Stratix II device.

Functional Results—Simulate the 8-Bit Comparator Design in Quartus

Next, simulate the lpm_compare design module and verify the results. Set up the Quartus II simulator by performing the following steps.

1. On the Processing menu, select the **Generate Functional Simulation Netlist** command.

2. When the **Functional Simulation Netlist Generation was successful** message box appears, click **OK**.

3. On the Assignments menu, select **Settings** to open the **Settings** dialog box.

4. From the Category list, select **Simulator Settings**.

5. For Simulation mode, select **Functional**.

6. For Simulation input, set the path to the **compare.vwf** file location.

7. Under Simulation period, select **Run simulation until all vector stimuli have been used**. Leave other options with their defaults.

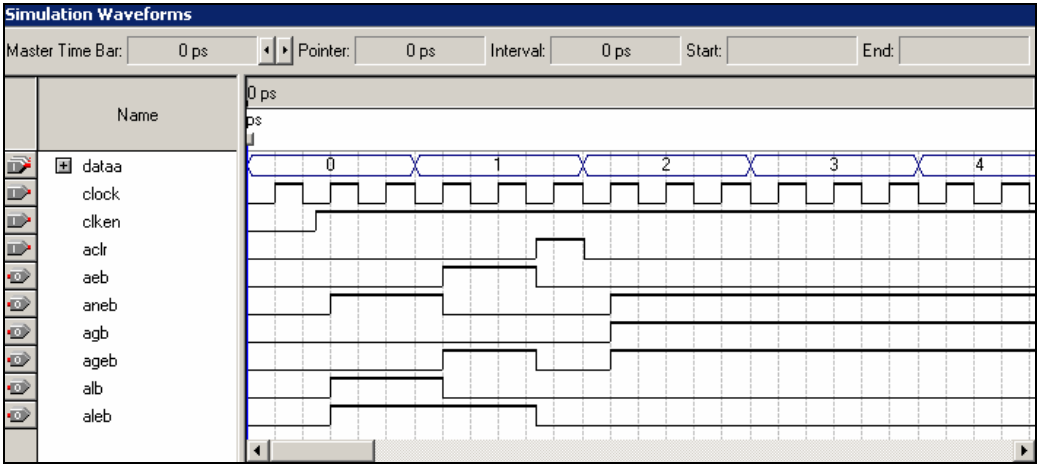
8. Click **OK**.

9. On the Processing menu, select **Start Simulation** or click on the simulation button to simulate the lpm_compare module.

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

11. In the compare Simulation Report window, view the simulation output waveforms and verify the results (Figure 2–14).

Figure 2–14. Compare Simulation Results



Functional Results—Simulate the 8-Bit Comparator in ModelSim-Altera

Simulate the design in ModelSim to compare the results of both simulators. Note that this ModelSim design example is for the ModelSim-Altera (Verilog) version.

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.

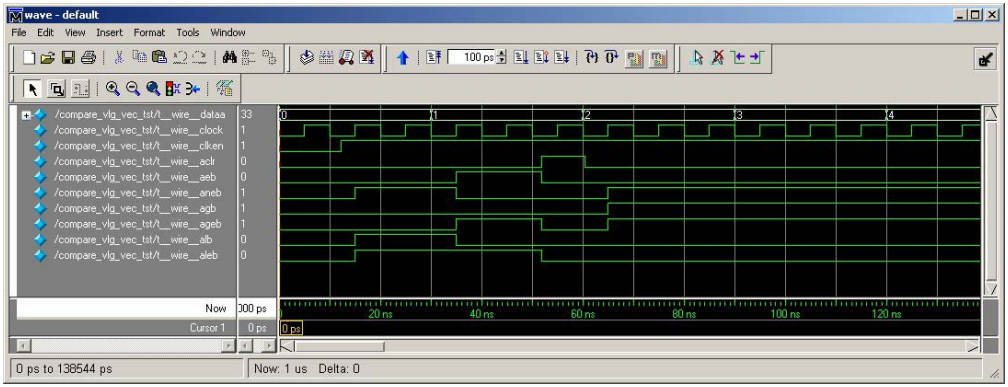
Set up the ModelSim-Altera Simulator by performing the following steps.

1. Unzip the **lpm_compare_msim.zip** file to any working directory on your PC.
2. Browse to the folder in which you have unzipped the files and open the **compare.do** file in a text editor.
3. In line 1, replace <*insert_directory_path_here*> with the directory path of the appropriate library files.

   For example, `C:/Modeltech_ae/altera/verilog/stratixii`
4. On the File menu, select **Save**.
5. Start ModelSim-Altera.
6. On the File menu, select **Change Directory**.
7. Select the folder in which you have unzipped your files and click **OK**.
8. On the Tools menu, select **Execute Macro**.
9. Select the **compare.do** file and click **Open**. This is a script file for ModelSim which automates all the necessary settings for the simulation.
10. Verify the results by looking at 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 2–15 shows the expected simulation results in ModelSim.

Figure 2–15. 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. You should use these functions 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_compare megafunction. Table 3–1 shows the input ports, Table 3–2 shows the output ports, and Table 3–3 shows the parameterized megafunction interface of the lpm_compare 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 user of the MegaWizard Plug-In Manager interface.

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_compare Port and Parameter Description Symbol

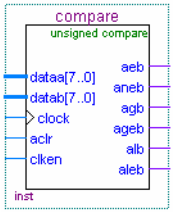


Table 3–1. lpm_compare Megafunction Input Ports

Port Name	Required	Description	Comments
`dataa[]`	Yes	Value to be compared to `datab[]`.	Input port `LPM_WIDTH` wide.
`datab[]`	Yes	Value to be compared to `dataa[]`.	Input port `LPM_WIDTH` wide.
`clock`	No	A clock port to register and provide pipelined usage.	The clock port provides pipelined operation for the lpm_compare function. For `LPM_PIPELINE` values other than 0 (default value), the `clock` port must be connected.
`aclr`	No	An active high asynchronous clear port for registered usage.	The pipeline initializes to an undefined (X) logic level. The `aclr` port can be used at any time to reset the pipeline to all 0s, asynchronously to the clock signal.
`clken`	No	An active high clock enable port for registered usage.	If not used, the default is 1.

Table 3–2. lpm_compare Megafunction Output Ports

Port Name	Required	Description	Comments
`alb`	No	High (1) if `dataa[]` < `datab[]`.	At least one of `alb`, `aeb`, `agb`, `ageb`, `aleb`, or `aneb` must be present.
`aeb`	No	High (1) if `dataa[]` = `datab[]`.	At least one of `alb`, `aeb`, `agb`, `ageb`, `aleb`, or `aneb` must be present.
`agb`	No	High (1) if `dataa[]` > `datab[]`.	At least one of `alb`, `aeb`, `agb`, `ageb`, `aleb`, or `aneb` must be present.
`ageb`	No	High (1) if `dataa[]` ≥ `datab[]`.	At least one of `alb`, `aeb`, `agb`, `ageb`, `aleb`, or `aneb` must be present.
`aleb`	No	High (1) if `dataa[]` ≤ `datab[]`.	At least one of `alb`, `aeb`, `agb`, `ageb`, `aleb`, or `aneb` must be present.
`aneb`	No	High (1) if `dataa[]` ≠ `datab[]`.	At least one of `alb`, `aeb`, `agb`, `ageb`, `aleb`, or `aneb` must be present.

Table 3–3. Parameterized lpm_compare Megafunction Interface

Parameter	Type	Required	Description
LPM_WIDTH	Integer	Yes	Width of the dataa[] and datab[] ports.
LPM_REPRESENTATION	String	No	Sign representation of inputs: "SIGNED", "UNSIGNED", or "UNUSED". The default is "UNSIGNED".
LPM_PIPELINE	Integer	No	Specifies the number of clock cycles of latency associated with the alb, aeb, agb, ageb, aleb, or aneb output. A value of zero (0) indicates that no latency exists and that a purely combinational function will be instantiated. The default value is 0 (non-pipelined).
LPM_HINT	String	No	Lets you specify Altera®-specific parameters in VHDL Design Files (**.vhd**). The default is "UNUSED".
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.
CHAIN_SIZE	Integer	No	Altera-specific parameter. Maximum allowable length of carry chains or cascade chains in ACEX® 1K, APEX™ 20K, APEX II, Excalibur™, FLEX 10K®, and Mercury™ devices. If omitted, the default is 8. This value overrides the value of the Carry Chain Length and Cascade Chain Length logic options. For all other device families, varying this parameter provides different size/speed combinations; smaller values of CHAIN_SIZE generally result in faster and larger comparators, and vice versa. For more information, contact Altera Applications.
ONE_INPUT_IS_CONSTANT	String	No	Altera-specific parameter. Values are "YES", "NO", or "UNUSED". Provides greater optimization if an input is constant. The default is "NO".