



# Floating Point Converter (ALTFP\_CONVERT)

---

## Megafunction User Guide



101 Innovation Drive  
San Jose, CA 95134  
[www.altera.com](http://www.altera.com)

Software Version: 8.0  
Document Version: 1.0  
Document Date: May 2008

Copyright © 2008 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-01031-1.0

<b>About this User Guide</b> .....	<b>v</b>
Revision History .....	v
Referenced Documents .....	v
How to Contact Altera .....	v
Typographic Conventions .....	vi
<b>Chapter 1. About this Megafunction</b>	
Device Family Support .....	1-1
Introduction .....	1-1
Features .....	1-1
Integer-to-Float Conversion .....	1-2
Float-to-Integer Conversion .....	1-2
Float-to-Float Conversion .....	1-2
General Description .....	1-2
Single-Precision Floating-Point Format .....	1-3
Double-Precision Floating-Point Format .....	1-3
Single-Extended-Precision Floating-Point Format .....	1-4
Special Case Numbers .....	1-4
Rounding .....	1-4
Exception Handling .....	1-5
Common Applications .....	1-5
Resource Usage and Performance .....	1-6
<b>Chapter 2. Getting Started</b>	
Software and System Requirements .....	2-1
MegaWizard Plug-In Manager Customization .....	2-1
MegaWizard Plug-In Manager Page Descriptions .....	2-2
Instantiating Megafunctions in HDL Code or Schematic Designs .....	2-9
Generating a Netlist for EDA Tool Use .....	2-9
Using the Port and Parameter Definitions .....	2-10
Identifying a Megafunction after Compilation .....	2-10
Simulation .....	2-10
Quartus II Software Simulation .....	2-10
EDA Tool Simulation .....	2-11
Design Example 1: Converting 32-Bit Signed Integers to Single-Precision Floating-Point Numbers .....	2-11
Design Files .....	2-11
Conversion Procedure .....	2-11
Functional Simulation in the ModelSim-Altera Simulator .....	2-15
Understanding the Simulation Results .....	2-17
Design Example 2: Converting Double-Precision Floating-Point Numbers to 64-Bit Signed	

## Contents

---

Integers .....	2-18
Design Files .....	2-18
Conversion Procedure .....	2-18
Functional Simulation in the ModelSim-Altera Simulator .....	2-22
Understanding the Simulation Results .....	2-23
Conclusion .....	2-25
<b>Chapter 3. Specifications</b>	
Ports and Parameters .....	3-1



# About this User Guide

**Revision History** The following table shows the revision history for this user guide.

Date and Document Version	Changes Made	Summary of Changes
May 2008 v1.0	Initial release.	—

## Referenced Documents

This user guide references the following documents:

- *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*
- *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*
- *Simulation* section in volume 3 of the *Quartus II Handbook*
- *Synthesis* section in volume 1 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	<a href="http://www.altera.com/support">www.altera.com/support</a>
Technical training	Website	<a href="http://www.altera.com/training">www.altera.com/training</a>
	Email	<a href="mailto:custrain@altera.com">custrain@altera.com</a>
Product literature	Website	<a href="http://www.altera.com/literature">www.altera.com/literature</a>
Non-technical support (General) (Software Licensing)	Email	<a href="mailto:nacomp@altera.com">nacomp@altera.com</a>
	Email	<a href="mailto:authorization@altera.com">authorization@altera.com</a>

**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
<b>Bold Type with Initial Capital Letters</b>	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: <b>Save As</b> dialog box.
<b>bold type</b>	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: <b>\qdesigns</b> directory, <b>d:</b> drive, <b>chiptrip.gdf</b> 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<sub>PIA</sub></i> , <i>n + 1</i> .  Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <file name>, <project name>.pdf 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.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work.
	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 about a particular topic.

## Device Family Support

The Floating Point Converter (ALTFP\_CONVERT) megafunction supports the following target Altera® device families:

- Arria™ GX
- Cyclone® III
- Cyclone II
- Cyclone
- HardCopy® II
- HardCopy Stratix®
- Stratix III
- Stratix II
- Stratix II GX
- Stratix
- Stratix GX

## Introduction

As design complexities increase, the 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 size of the megafunction by setting parameters.

## Features

The ALTFP\_CONVERT megafunction implements the following conversion functions:

- Integer-to-float conversion
- Float-to-integer conversion
- Float-to-float conversion

The ALTFP\_CONVERT megafunction supports the following features:

- Support for floating-point formats in single, double, and single-extended precision
- Support for signed integers
- Support for input of normal numbers, infinity, zero, and not-a-number (NaN)

- Optional exception-handling output ports for overflow, underflow, and NaN
- Optional input ports, including the asynchronous-clear and clock-enable ports

### Integer-to-Float Conversion

This operation converts integer bits to the IEEE-754 floating-point representation. Conversions of signed integers to floating-point numbers in single, double, or single-extended precision are supported in this mode.

### Float-to-Integer Conversion

This operation converts IEEE-754 floating-point representations to the integer-bit format. Conversions of floating-point numbers—in single, double, or single-extended precision—to integers are supported in this mode.

### Float-to-Float Conversion

This operation converts between IEEE-754 floating-point representations. Conversions between floating-point numbers in single, double, or single-extended precision are supported in this mode.

## General Description

The ALTFP\_CONVERT megafunction follows the IEEE-754 standard for floating-point conversion, which defines the following features:

- The formats for representing floating-point numbers
- The representations of special values—zero, infinity, denormal numbers, and bit combinations that do not represent a number (NaN)

The IEEE-754 standard also defines four formats for floating-point numbers: single precision, double precision, single-extended precision, and double-extended precision. The most commonly used floating-point formats are single precision and double precision. The ALTFP\_CONVERT megafunction supports only three formats: single precision, double precision, and single-extended precision.

All floating-point formats are implemented as shown in [Figure 1–1](#). In this figure:

- *S* represents a sign bit
- *E* represents an exponent field
- *M* represents the mantissa (part of a logarithm or fraction) field

**Figure 1–1. IEEE-754 Floating Point Format**



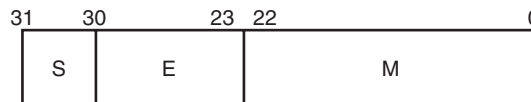
For a normal floating-point number, the leading 1 is always implied. For example, the binary value, 1.0011, of decimal value, 1.1875, is stored as 0011 in the mantissa field. With the implied 1, the mantissa field does not have to use an extra bit to represent the leading 1.

However, for a denormal number, the leading bit can be 0 or 1. For zero, infinity, and NaN, there is no implied leading 1 or any leading bit for the mantissa field.

### Single-Precision Floating-Point Format

For a single-precision floating-point number, the most significant bit (MSB) is a sign bit, followed by eight intermediate bits to represent an exponent, and 23 least significant bits (LSBs) to represent the mantissa. As a result, the total width for a single-precision floating-point number is 32 bits. The bias for the representation is 127. Refer to [Figure 1–2](#).

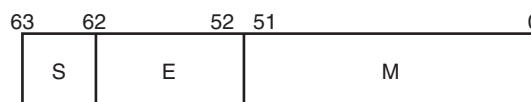
**Figure 1–2. Single-Precision Representation**



### Double-Precision Floating-Point Format

For a double-precision floating-point number, the MSB is a sign bit, followed by 11 intermediate bits to represent an exponent, and 52 LSBs to represent the mantissa. As a result, the total width for a double-precision floating-point number is 64 bits. The bias for the representation is 1023. Refer to [Figure 1–3](#).

**Figure 1–3. Double-Precision Representation**



## Single-Extended-Precision Floating-Point Format

For a single-extended-precision floating-point number, the MSB is a sign bit. However, the exponent and mantissa fields do not have fixed widths. The width of the exponent field must be a minimum of 11 bits and less than the width of the mantissa field. The width of the mantissa field must be a minimum of 31 bits. The sum of the width of the sign bit, exponent field, and mantissa field must be a minimum of 43 bits and a maximum of 64 bits. The bias of a floating-point number in single-extended precision is unspecified in the IEEE-754 standard. This megafunction assumes a bias of  $(2^{(\text{WIDTH\_EXP}-1)}-1)$  for the representation.

## Special Case Numbers

Table 1–1 shows the special case numbers defined by the IEEE-754 standard and their respective data bit representations.

Meaning	Sign Field	Exponent Field	Mantissa Field
Zero	Don't care	All 0's	All 0's
Positive Denormalized	0	All 0's	Non-zero
Negative Denormalized	1	All 0's	Non-zero
Positive Infinity	0	All 1's	All 0's
Negative Infinity	1	All 1's	All 0's
Not-a-Number (NaN)	Don't care	All 1's	Non-zero

## Rounding

In the IEEE-754 standard, there are four types of rounding modes:

- Round-to-nearest-even
- Round-toward-zero
- Round-toward-positive-infinity
- Round-toward-negative-infinity

The most commonly used rounding mode is round-to-nearest-even. This converter uses only the round-to-nearest-even mode. In this mode, the result is rounded to the nearest floating-point number. If the result is exactly halfway between two floating-point numbers, it is rounded so that the LSB becomes zero, which is even.

## Exception Handling

Three optional exception signals are provided—`overflow`, `underflow`, and `nan`. [Table 1–2](#) shows the supported exception ports and their respective operations.

<i>Table 1–2. Operations and Exception Ports Supported</i>	
<b>Operation</b>	<b>Exception Ports Supported</b>
Integer-to-Float	Exception ports not supported
Float-to-Integer	<code>overflow</code> , <code>underflow</code> , <code>nan</code>
Float-to-Float	<code>overflow</code> , <code>underflow</code> , <code>nan</code>

## Common Applications

The advantage of using floating-point numbers is that they can represent a much larger range of values. In a fixed-point number representation, the radix point is always at the same location. Although the fixed radix point simplifies numeric operations and conserves memory, it limits the magnitude and precision of the number representation. In situations that require a large range of numbers or high resolution, a relocatable radix point is desirable. In the floating-point format, very large or very small numbers can be represented.

## Resource Usage and Performance

Table 1–3 shows the resource usage of the ALTFP\_CONVERT megafunction for Stratix II devices.

**Table 1–3. Resource Usage of ALTFP\_CONVERT Megafunction for Stratix II Devices *Note (1)***

Operation	Format	Pipeline	Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Logic Usage	f <sub>MAX</sub>
Integer-to-Float	32-bit integer to single precision	6	192	238	288	396
	32-bit integer to double precision	6	167	116	204	399
	64-bit integer to single precision	6	391	371	541	328
	64-bit integer to double precision	6	413	461	617	314
Float-to-Integer	Single precision to 32-bit integer	6	261	257	338	342
	Single precision to 64-bit integer	6	418	363	482	226
	Double precision to 32-bit integer	6	416	387	518	315
	Double precision to 64-bit integer	6	546	483	686	225
Float-to-Float	Single precision to double precision	2	45	71	78	840
	Double precision to single precision	3	104	139	166	440

**Note to Table 1–3:**

- (1) You can obtain the performance information of the ALTFP\_CONVERT megafunction by compiling your design in the Quartus II software. The information in this table is valid and accurate in the Quartus II software version 8.0.

Table 1–4 shows the resource usage of the ALTFP\_CONVERT megafunction for Stratix III devices.

**Table 1–4. Resource Usage of ALTFP\_CONVERT Megafunction for Stratix III Devices** *Note (1)*

Operation	Format	Pipeline	Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Logic Usage	f <sub>MAX</sub>
Integer-to-Float	32-bit integer to single precision	6	192	238	281	514
	32-bit integer to double precision	6	182	140	223	484
	64-bit integer to single precision	6	390	371	531	425
	64-bit integer to double precision	6	413	461	606	419
Float-to-Integer	Single precision to 32-bit integer	6	261	257	327	443
	Single precision to 64-bit integer	6	418	363	478	322
	Double precision to 32-bit integer	6	414	386	518	400
	Double precision to 64-bit integer	6	545	482	677	316
Float-to-Float	Single precision to double precision	2	45	71	78	1,032
	Double precision to single precision	3	104	139	163	600

**Note to Table 1–4:**

- (1) You can obtain the performance information of the ALTFP\_CONVERT megafunction by compiling your design in the Quartus II software. The information in this table is valid and accurate in the Quartus II software version 8.0.



### Software and System Requirements

The instructions in this section require the following software:

- Quartus® II software version 8.0 or later
- For operating system support information, refer to: [www.altera.com/support/software/os\\_support/oss-index.html](http://www.altera.com/support/software/os_support/oss-index.html)

### MegaWizard Plug-In Manager Customization

The MegaWizard® Plug-In Manager creates or modifies design files that contain custom megafunction variations, which can then be instantiated in a design file. The MegaWizard Plug-In Manager provides a wizard that allows you to specify options for the Floating Point Converter (ALTFP\_CONVERT) megafunction features in your 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, on 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** window, 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 ←`

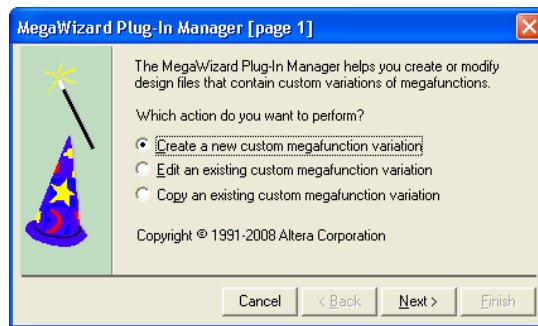
## MegaWizard Plug-In Manager Page Descriptions

This section provides descriptions of the options available on the individual pages of the ALTFP\_CONVERT wizard. Click **Next** on each page to move to the next page.

On page 1 of the MegaWizard Plug-In Manager, you can select **Create a new custom megafunction variation**, **Edit an existing custom megafunction variation**, or **Copy an existing custom megafunction variation** (Figure 2-1).

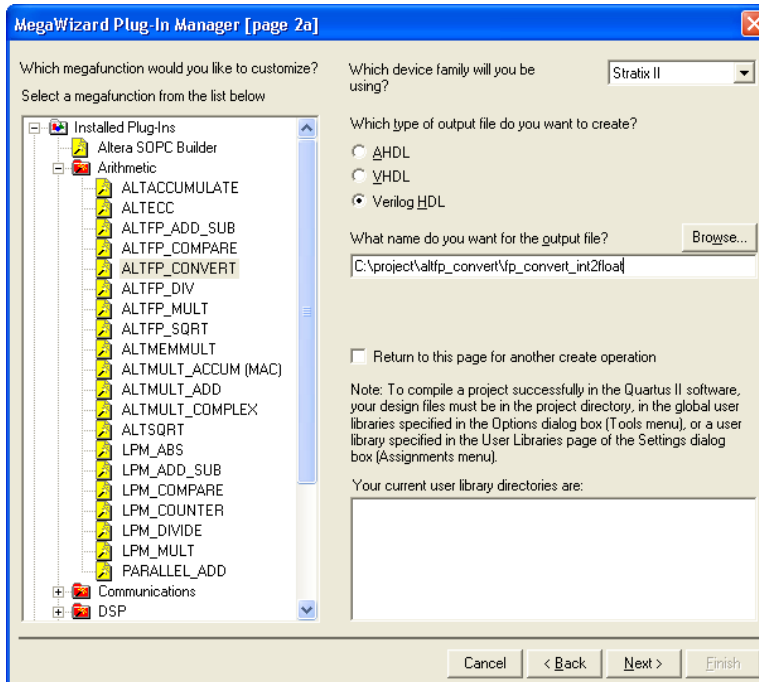
---

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



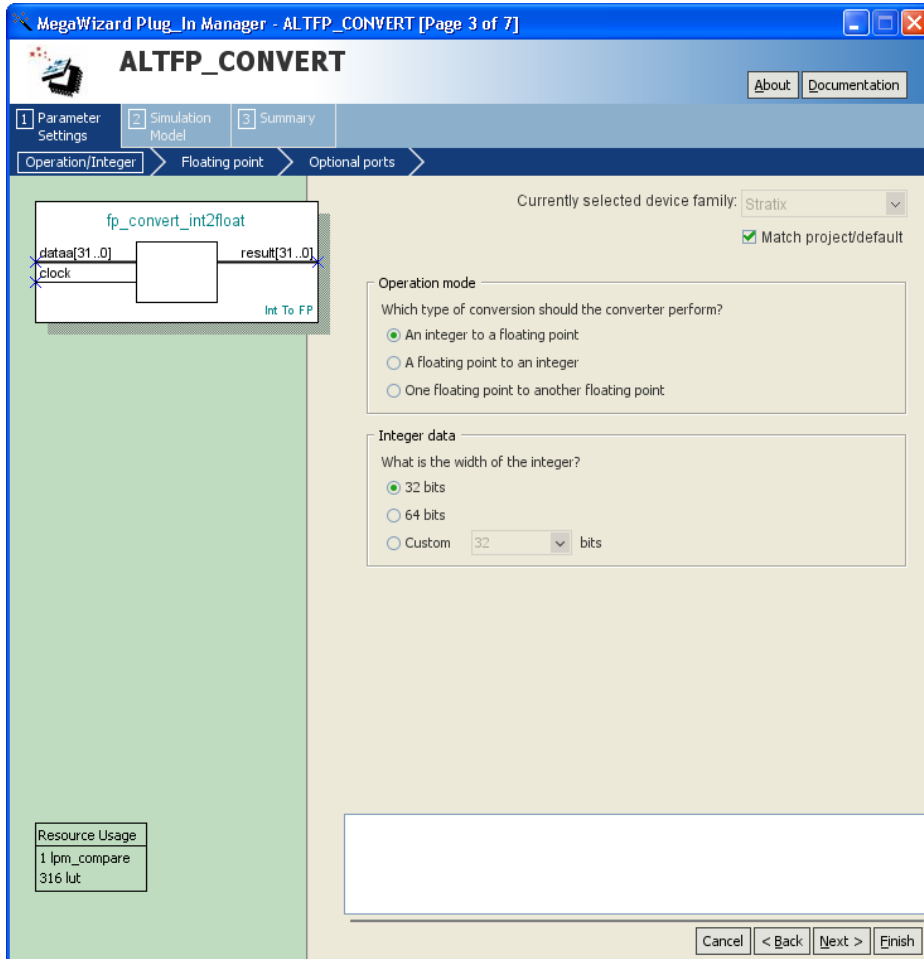
On page 2a of the MegaWizard Plug-In Manager, specify the megafunction, the device family to use, the type of output file to create, and the name of the output file (Figure 2-2). Choose AHDL (.tdf), VHDL (.vhd), or Verilog HDL (.v) as the output file type.

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



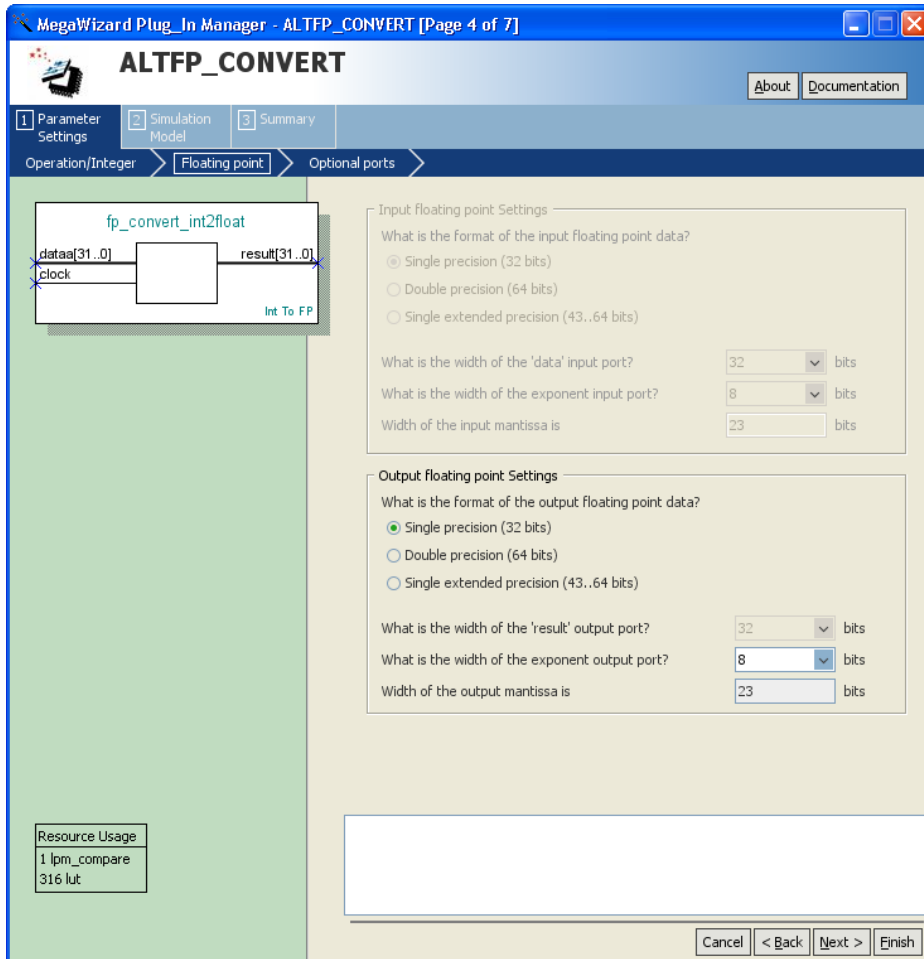
On page 3 of the ALTFP\_CONVERT MegaWizard Plug-In Manager, select the operation mode of the converter and, if integers are involved, the width of the integer (Figure 2-3).

Figure 2-3. ALTFP\_CONVERT MegaWizard Plug-In Manager [page 3 of 7]



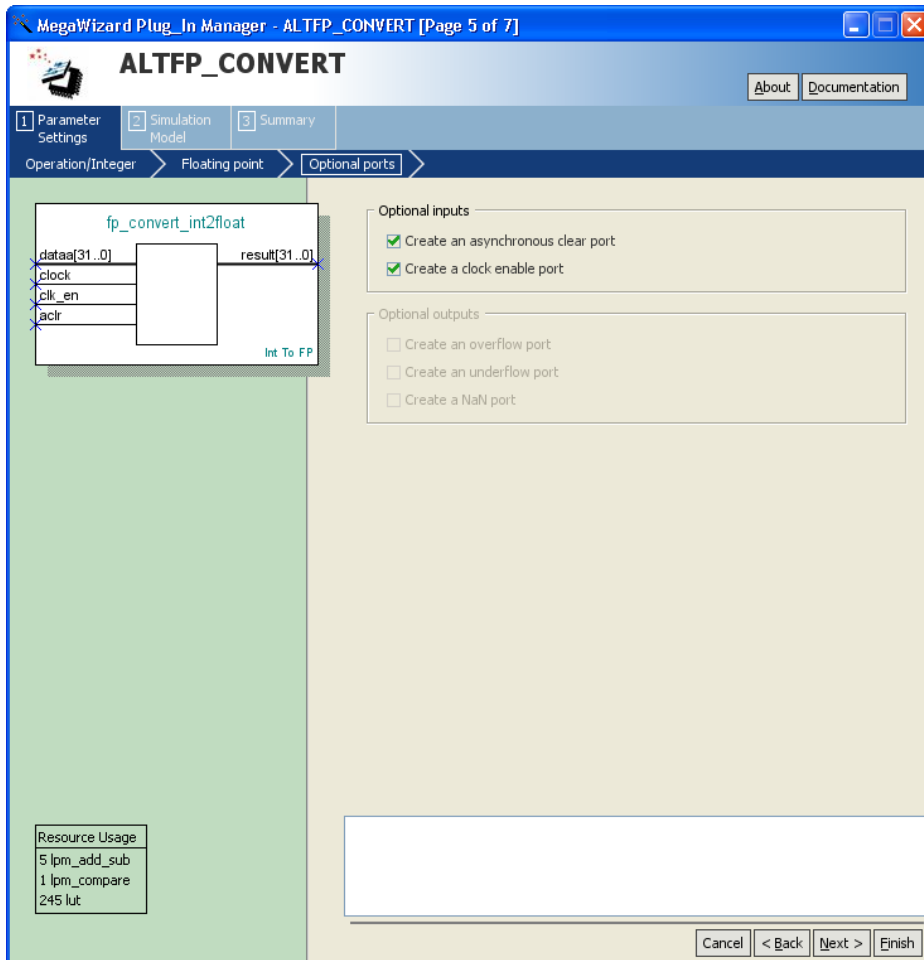
On page 4 of the ALTFP\_CONVERT MegaWizard Plug-In Manager, select the format of the input or output floating-point data, the width of the data input port or result output port, and the width of the input or output mantissa. However, if neither the input nor output data is a floating-point number, the corresponding Settings section is disabled (Figure 2–4).

Figure 2–4. ALTFP\_CONVERT MegaWizard Plug-In Manager [page 4 of 7]



On page 5 of the ALTFP\_CONVERT MegaWizard Plug-In Manager, you can create the asynchronous-clear port and clock-enable port for input. Both ports are optional. The option to create output ports is applicable only for float-to-integer and float-to-float conversions; it is disabled for integer-to-float conversion (Figure 2-5).

Figure 2-5. ALTFP\_CONVERT MegaWizard Plug-In Manager [page 5 of 7]







## 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 allows you to instantiate the customized function in your design. Depending on the language you choose in the MegaWizard Plug-In Manager, the wizard 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 wizard-generated files, refer to Quartus II Help or to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

### Generating a Netlist for EDA Tool Use

If you use a third-party EDA synthesis tool, you can instantiate the megafunction variation file as a black box for synthesis. Use the VHDL component declaration or Verilog HDL module declaration black box file to define the function in your synthesis tool, and then include the megafunction variation file in your Quartus II project.

If you enable the option to generate a synthesis area and timing estimation netlist in the MegaWizard Plug-In Manager, the wizard generates an additional netlist file (`_syn.v`). The netlist file is a representation of the customized logic used in the Quartus II software. The file provides the connectivity of the architectural elements in the megafunction but may not represent true functionality. This information enables certain third-party synthesis tools to better report area and timing estimates. In addition, synthesis tools can use the timing information to focus timing-driven optimizations and improve the quality of results.



For more information about using megafunctions in your third-party synthesis tool, refer to the appropriate chapter in the *Synthesis* section in volume 1 of the *Quartus II Handbook*.

### Using the Port and Parameter Definitions

Instead of using the MegaWizard Plug-In Manager, you can instantiate the megafunction directly in your Verilog HDL, VHDL, or AHDL code by calling the megafunction and setting its parameters as you would any other module, component, or subdesign.



Altera strongly recommends that you use the MegaWizard Plug-In Manager for complex megafunctions. The MegaWizard Plug-In Manager ensures that you set all megafunction parameters properly.

For a list of the megafunction ports and parameters, refer to [Chapter 3, Specifications](#).

### Identifying a Megafunction after Compilation

During compilation with the Quartus II software, analysis and elaboration are 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), click **Browse** in the **Look in** box and select the megafunction in the **Hierarchy** box.

### Simulation

The Quartus II Simulator 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 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 your RTL code. When performing a functional simulation, add only signals that exist before synthesis. You can find these signals with the Registers: 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, the 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 may 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 HDL 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 the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*.

### EDA Tool Simulation

The *Quartus II Handbook* chapters describe 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 the files are located.



Depending on which simulation tool you are using, refer to the appropriate chapter in the *Simulation* section in volume 3 of the *Quartus II Handbook*.

## Design Example 1: Converting 32-Bit Signed Integers to Single-Precision Floating-Point Numbers

This design example uses the `ALTFP_CONVERT` megafunction to convert 32-bit signed integers to single-precision floating-point numbers. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

### Design Files

The example design files are available in the User Guides section on the Literature page of the Altera website ([www.altera.com](http://www.altera.com)).

### Conversion Procedure

In this example, the following tasks are performed:

1. “Generate Integer-to-Float Converter” on page 2–12
2. “Implement Integer-to-Float Converter” on page 2–13

### Generate Integer-to-Float Converter

To create the integer-to-float converter, perform the following steps:

1. Open the `alftp_convert_DesignExample1.zip` file and extract `fp_convert_int2float.qar`.
2. In the Quartus II software, open `fp_convert_int2float.qar` and restore the archive file into your working directory.
3. On the Tools menu, click **MegaWizard Plug-In Manager**. Page 1 of the MegaWizard Plug-In Manager appears.
4. Select **Create a new custom megafunction variation**.
5. Click **Next**. Page 2a of the MegaWizard Plug-In Manager appears.
6. In the MegaWizard Plug-In Manager pages, select or verify the configuration settings shown in [Table 2–1](#). Click **Next** to advance from one page to the next.

**Table 2–1. Design Example 1: Configuration Settings (Part 1 of 2)**

MegaWizard Plug-In Manager Page	MegaWizard Plug-In Manager Configuration Setting	Value
2a	Select a megafunction	Under Arithmetic, select <code>ALFTP_CONVERT</code>
	Which device family will you be using?	Stratix II
	Which type of output file do you want to create?	Verilog HDL
	What name do you want for the output file?	<code>fp_convert_int2float</code>
3	Currently selected family	Stratix II
	Match project/default	Selected
	What type of conversion should the converter perform?	An integer to a floating point
	What is the width of the integer?	32 bits
4 (Under Output Floating-Point Settings)	What is the format of the output floating point data?	Single precision (32 bits)
	What is the width of the result output port?	32 bits
	What is the width of the exponent output port?	8 bits
	Width of the output mantissa is	23 bits
5	Create an asynchronous clear port	Selected
	Create a clock enable port	Selected
6	Generate netlist	Not selected

*Table 2–1. Design Example 1: Configuration Settings (Part 2 of 2)*

MegaWizard Plug-In Manager Page	MegaWizard Plug-In Manager Configuration Setting	Value
7	Variation file	Selected
	Instantiation template file	Not selected
	Verilog HDL black-box file	Selected
	AHDL Include file	Not selected
	VHDL component declaration file	Not selected
	Quartus II symbol file	Not selected
	Quartus II IP Advisor file	Not selected

- Click **Finish**.



If the Quartus II IP Files dialog box appears with an option to add the Quartus II IP Advisor file (**.qip**) to your project, click **Yes**.

The ALTFP\_CONVERT module is now built.

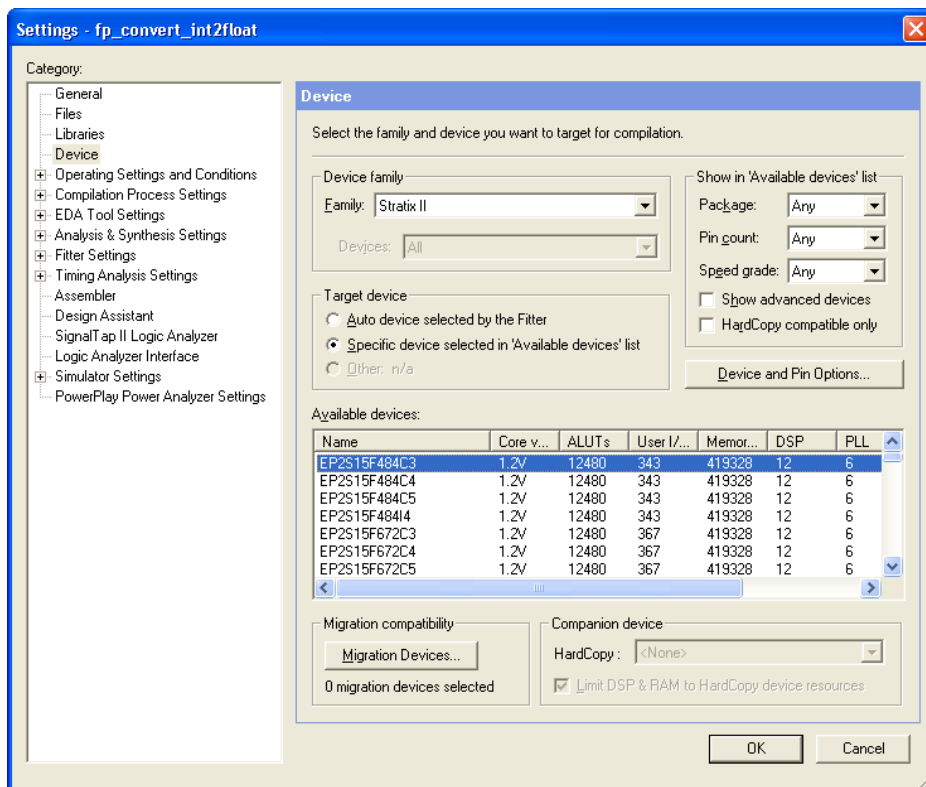
### *Implement Integer-to-Float Converter*

Next, perform the following steps to assign the EP2S15F484C3 device to the project and compile the project:

- On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
- Under Category, select **Device**.
- In the Family list, select **Stratix II**.
- Under Target Device, select **Specific device selected in 'Available devices' list**.
- In the Available devices list, select **EP2S15F484C3**.

**Figure 2–8** shows the **Device Settings** dialog box after these selections are made.

Figure 2–8. Design Example 1: Device Settings Dialog Box



6. Click **OK**.
7. To compile the design, on the Processing menu, click **Start Compilation**. Alternatively, you can click the Start Compilation button on the toolbar.
8. When the **Full Compilation was successful** message box appears, click **OK**. The design is now assigned to the EP2S15F484C3 device and is compiled.

## Functional Simulation in the ModelSim-Altera Simulator

Simulate the design in the ModelSim®-Altera software to generate a waveform display of the device behavior.

You should be familiar with the ModelSim-Altera software before trying out the design example. If you are unfamiliar with the ModelSim-Altera software, refer to the support page for software products on the Altera website ([www.altera.com](http://www.altera.com)). On the support page, there are links to such topics as installation, usage, and troubleshooting.

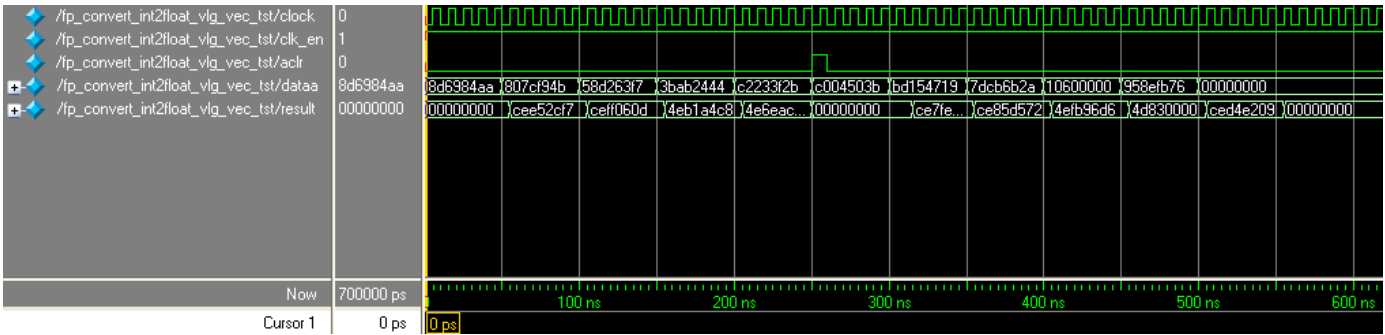
Set up and simulate the design in the ModelSim-Altera software by performing the following steps:

1. Unzip the **alftp\_convert\_int2float\_msim.zip** file to any working directory on your PC.
2. Start the ModelSim-Altera software.
3. On the File menu, click **Change Directory**.
4. Select the folder in which you unzipped the files.
5. Click **OK**.
6. On the Tools menu, click **Execute Macro**.
7. Select the **fp\_convert\_int2float.do** file and click **Open**. This is a script file for the ModelSim-Altera software to automate all the necessary settings for the simulation.
8. Verify the results shown in the Wave window.

You can rearrange signals, remove signals, add signals, and change the radix by modifying the script in the **fp\_convert\_int2float.do** file.

Figure 2-9 shows the expected simulation results in the ModelSim-Altera software. For more information about the simulation results, refer to “Understanding the Simulation Results” on page 2-17.

Figure 2–9. Design Example 1: Simulation Waveforms in the ModelSim-Altera Software



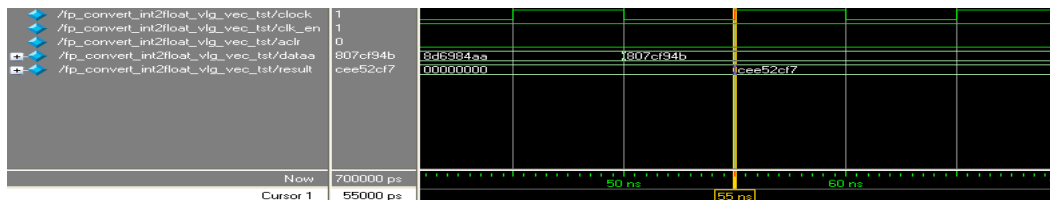
## Understanding the Simulation Results

Design Example 1 implements an integer-to-float converter for converting 32-bit integers to single-precision numbers. The only output port involved is `result`, as exception handling is not supported for this operation.

The latency for the integer-to-float conversion is 6 clock cycles. Therefore, each conversion generates the output result 6 clock cycles after receiving the input value.

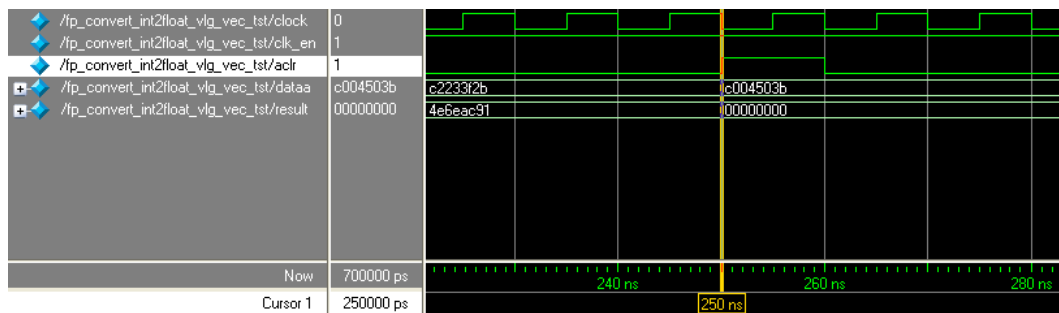
During power-up at 0 ns, the `result` port displays 0 regardless of what the input value is. The `result` port generates the output from the conversion of the first input value 6 clock cycles after power-up. The output result of this conversion can be seen at 55 ns, as shown in Figure 2-10.

**Figure 2-10. Design Example 1: Integer-to-Float Conversion of First Input Value**



At 250 ns, the `aclr` signal is set for one clock period (Figure 2-11). The assertion of the `aclr` signal causes a reset; therefore, the `result` port generates an output value of 0 for as long as 6 clock cycles after the deassertion of `aclr`.

**Figure 2-11. Design Example 1: Assertion of aclr Signal at 250 ns**



## Design Example 2: Converting Double-Precision Floating-Point Numbers to 64-Bit Signed Integers

This design example uses the ALTFP\_CONVERT megafunction to convert double-precision floating-point numbers to 64-bit signed integers. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

### Design Files

The example design files are available in the User Guides section on the Literature page of the Altera website ([www.altera.com](http://www.altera.com)).

### Conversion Procedure

In this example, the following tasks are performed:

1. “Generate Float-to-Integer Converter” on page 2–18
2. “Implement Float-to-Integer Converter” on page 2–20

#### *Generate Float-to-Integer Converter*

To create the float-to-integer converter, perform the following steps:

1. Open **alftp\_convert\_DesignExample2.zip** and extract **fp\_convert\_float2int.qar**.
2. In the Quartus II software, open **fp\_convert\_float2int.qar** and restore the archive file into your working directory.
3. On the Tools menu, click **MegaWizard Plug-In Manager**. Page 1 of the MegaWizard Plug-In Manager appears.
4. Select **Create a new custom megafunction variation**.
5. Click **Next**. Page 2a of the MegaWizard Plug-In Manager appears.

6. In the MegaWizard Plug-In Manager pages, select or verify the configuration settings shown in [Table 2-2](#). Click **Next** to advance from one page to the next.

**Table 2-2. Design Example 2: Configuration Settings**

MegaWizard Plug-In Manager Page	MegaWizard Plug-In Manager Configuration Setting	Value
2a	Select a megafunction	Under Arithmetic, select ALTFP_CONVERT
	Which device family will you be using?	Stratix II
	Which type of output file do you want to create?	Verilog HDL
	What name do you want for the output file?	fp_convert_float2int
3	Currently selected family	Stratix II
	Match project/default	Selected
	What type of conversion should the converter perform?	A floating point to an integer
	What is the width of the integer?	64 bits
4 (Under Input Floating-Point Settings)	What is the format of the input floating point data?	Double precision (64 bits)
	What is the width of the 'data' input port?	64 bits
	What is the width of the exponent input port?	11 bits
	Width of the input mantissa is	52 bits
5	Create an asynchronous clear port	Selected
	Create a clock enable port	Selected
	Create an overflow port	Selected
	Create an underflow port	Selected
	Create a NaN port	Selected
6	Generate netlist	Not selected
7	Variation file	Selected
	Instantiation template file	Not selected
	Verilog HDL black-box file	Selected
	AHDL Include file	Not selected
	VHDL component declaration file	Not selected
	Quartus II symbol file	Not selected
	Quartus II IP Advisor file	Not selected

7. Click **Finish**.



If the Quartus II IP Files dialog box appears with an option to add the Quartus II IP Advisor file (**.qip**) to your project, click **Yes**.

The ALTFP\_CONVERT module is now built.

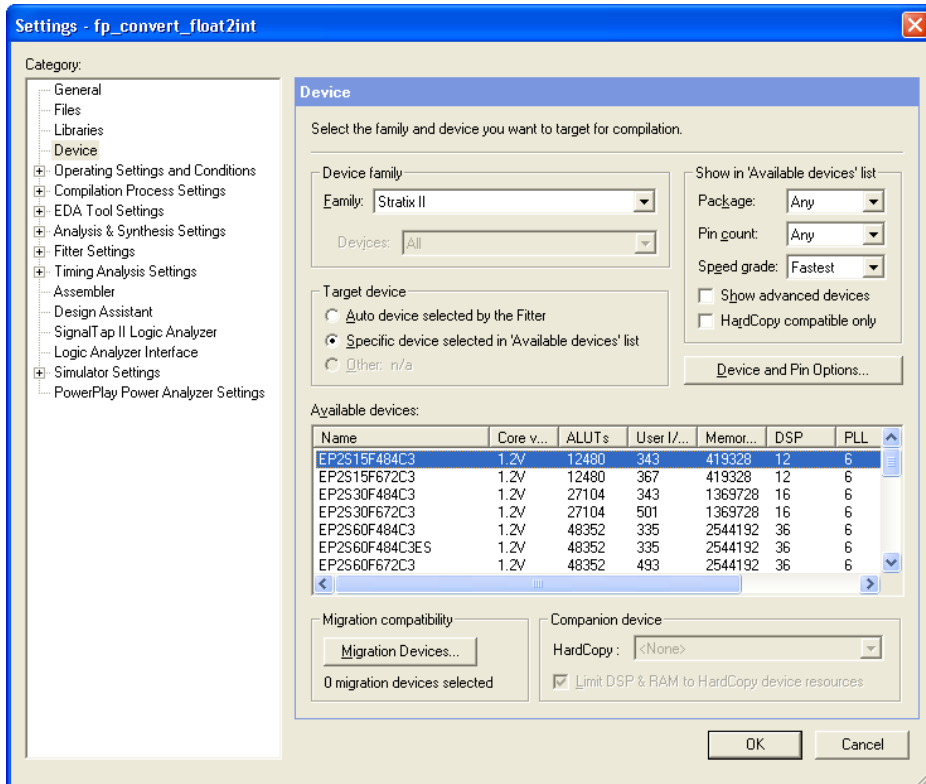
### *Implement Float-to-Integer Converter*

Next, perform the following steps to assign the EP2S15F484C3 device to the project and compile the project:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. Under Category, select **Device**.
3. In the Family list, select **Stratix II**.
4. Under Target Device, select **Specific device selected in 'Available devices' list**.
5. In the Available devices list, select **EP2S15F484C3**.

[Figure 2-12](#) shows the **Device Settings** dialog box after these selections are made.

Figure 2–12. Design Example 2: Device Settings Dialog Box



6. Click **OK**.
7. To compile the design, on the Processing menu, click **Start Compilation**. Alternatively, you can click the Start Compilation button on the toolbar.
8. When the **Full Compilation was successful** message box appears, click **OK**. The design is now assigned to the EP2S15F484C3 device and compiled.

### Functional Simulation in the ModelSim-Altera Simulator

Simulate the design in the ModelSim-Altera software to generate a waveform display of the device behavior.

You should be familiar with the ModelSim-Altera software before trying out the design example. If you are unfamiliar with the ModelSim-Altera software, refer to the support page for software products on the Altera website ([www.altera.com](http://www.altera.com)). On the support page, there are links to such topics as installation, usage, and troubleshooting.

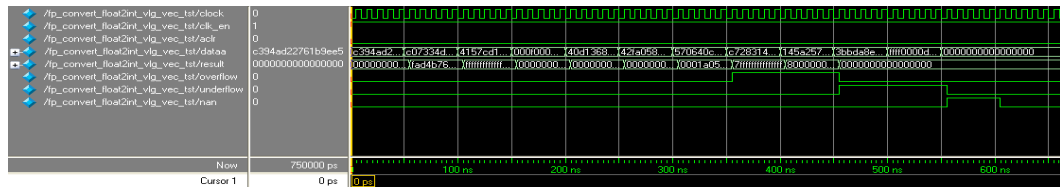
Set up and simulate the design in the ModelSim-Altera software by performing the following steps:

1. Unzip the **alftp\_convert\_float2int\_msim.zip** file to any working directory on your PC.
2. Start the ModelSim-Altera software.
3. On the File menu, click **Change Directory**.
4. Select the folder in which you unzipped the files.
5. Click **OK**.
6. On the Tools menu, click **Execute Macro**.
7. Select the **fp\_convert\_float2int.do** file and click **Open**. This is a script file for the ModelSim-Altera software to automate all the necessary settings for the simulation.
8. Verify the results shown in the Wave window.

You can rearrange signals, remove signals, add signals, and change the radix by modifying the script in **fp\_convert\_float2int.do**.

Figure 2–13 shows the expected simulation results in the ModelSim-Altera software. For more information about the simulation results, refer to “Understanding the Simulation Results” on page 2–23.

**Figure 2–13. Design Example 2: Simulation Waveforms in ModelSim-Altera Software**



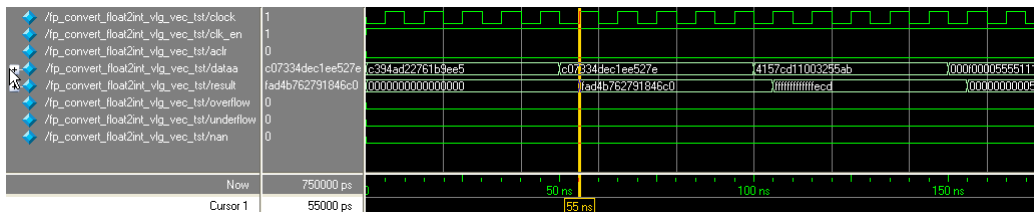
## Understanding the Simulation Results

Design Example 2 implements a float-to-integer converter for converting double-precision floating-point numbers to 64-bit integers. In this operation, the optional exception ports of `overflow`, `underflow`, and `nan` are available apart from the `result` port.

The latency for the float-to-integer operation is 6 clock cycles. Therefore, each conversion generates the output result 6 clock cycles after receiving the input value.

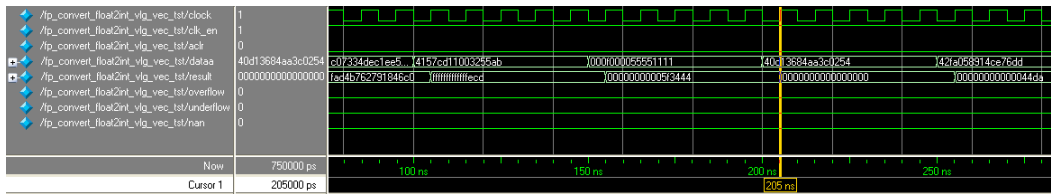
During power-up at 0 ns, the `result` port displays 0 regardless of what the input value is. The `result` port generates output from the conversion of the first input value 6 clock cycles after power-up. The output result of this conversion can be seen at 55 ns, as shown in Figure 2–14.

**Figure 2–14. Design Example 2: Float-to-Integer Conversion of First Input Value**



At 150 ns, a denormal input value, `0x000F000055551111`, is seen on the `dataaa` port because the exponent field for the value is zero and the mantissa field is a non-zero number. As denormal inputs are not supported by the `ALTFP_CONVERT` megafunction, the result is a forced zero at 205 ns, as shown in Figure 2–15.

**Figure 2–15. Design Example 2: Float-to-Integer Conversion of Denormal Numbers Results in Zero**



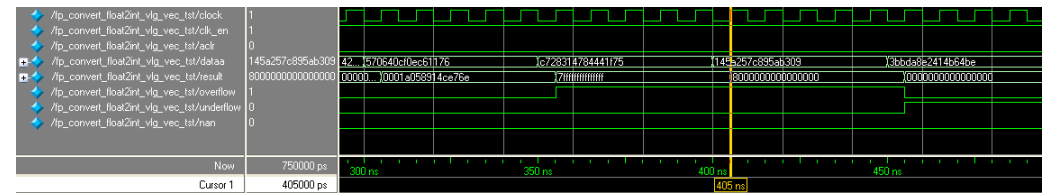
At 300 ns, the input value is 0x570640CF0EC61176. The `overflow` port is asserted 6 clock cycles later at 355 ns, and the value 0x7FFFFFFFFFFFFFFF is seen on the `result` port. The overflow flag is triggered because the width of the resulting integer is more than the maximum width allowed, and the value seen on the `result` port is the standard value used to represent a positive overflow number. Refer to Figure 2–16.

**Figure 2–16. Float-to-Integer Conversion that Results in Positive Overflow**



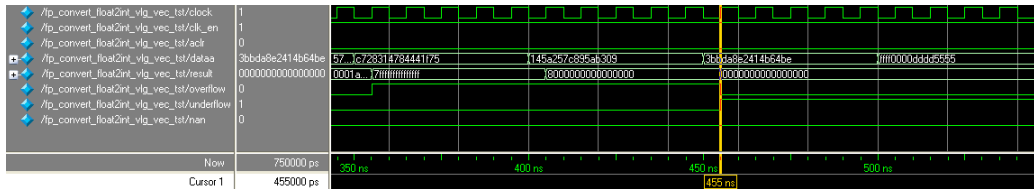
The next input value at 350 ns is 0xC78314784441F75. The overflow flag remains asserted, but the `result` port shows 0x8000000000000000, the standard value used to represent a negative overflow number. Refer to Figure 2–17.

**Figure 2–17. Design Example 2: Float-to-Integer Conversion that Results in Negative Overflow**



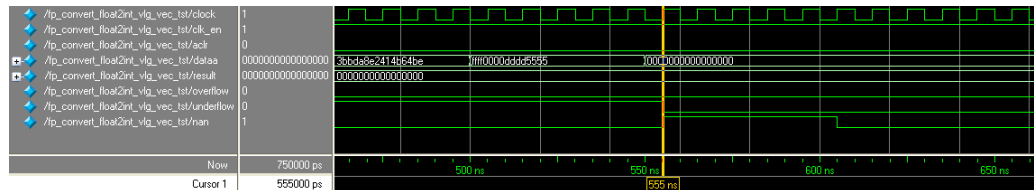
At 400 ns, the input value is 0x145A257C895AB309. This input triggers the assertion of the `underflow` port 6 clock cycles later at 455 ns because the exponent of the value is less than the exponent bias of 1023. The value seen at the `result` port at 455 ns is 0, which represents an underflow number. Refer to [Figure 2-18](#).

**Figure 2-18. Design Example 2: Float-to-Integer Conversion that Results in Underflow**



The input value at 500 ns is a NaN. This value triggers the assertion of the `NaN` flag 6 clock cycles later at 555 ns, and the `result` port is set to all zeroes. Refer to [Figure 2-19](#).

**Figure 2-19. Design Example 2: Float-to-Integer Conversion that Results in NaN**



## Conclusion

The Quartus II software provides parameterizable megafuncions ranging from simple arithmetic units, such as adders and counters, to advanced phase-locked loop (PLL) blocks, divisions, and memory structures. These megafuncions 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 functions during design implementation so you can consistently meet your design goals.



### Ports and Parameters

The Quartus® II software provides the Floating Point Converter (ALTFP\_CONVERT) megafunction to convert integers to floating-point numbers, floating-point numbers to integers and also between floating-point numbers. This chapter describes the ports and parameters of the ALTFP\_CONVERT megafunction. These ports and parameters are available to customize the ALTFP\_CONVERT megafunction according to your application.

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 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 shows the ports for the ALTFP\_CONVERT megafunction.

**Figure 3-1. Input and Output Ports of ALTFP\_CONVERT Megafunction**

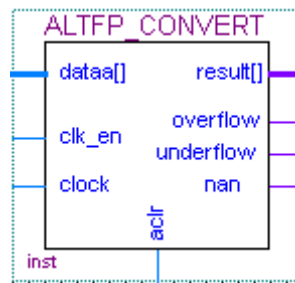


Table 3–1 shows the input ports, Table 3–2 shows the output ports, and Table 3–3 shows the parameters for the ALTFP\_CONVERT megafunction.

Port Name	Required	Description	Comments
aclr	No	Asynchronous-clear port for the floating-point conversion.	The core is asynchronously reset when the <code>aclr</code> signal is asserted high.
clk_en	No	Clock-enable port for the converter.	Allows conversion to take place when asserted high. When deasserted low, no operation takes place and the outputs are unchanged.
clock	Yes	Clock input for the converter.	Clock input for the ALTFP_CONVERT megafunction.
dataa []	Yes	Data input for the converter.	<p>If the operation mode is INT2FLOAT, the input bus is an integer representation.</p> <p>If the operation mode is FLOAT2INT, the input bus is an IEEE floating-point representation. If the precision is single precision, the input bus width is 32.</p> <p>If the precision is double precision, the input bus width is 64.</p> <p>If the precision is single-extended precision, the input bus range is [43..64].</p> <p>If the operation mode is FLOAT2FLOAT, the input bus value is an IEEE floating-point representation.</p> <p>If the precision is single precision, the input bus width is 32.</p> <p>If the precision is double precision, the input bus width is 64.</p> <p>If the precision is single-extended precision, the input bus range is [43..64].</p>

Table 3–2 lists the output ports of the ALTFP\_CONVERT megafunction.

<b>Table 3–2. ALTFP_CONVERT Megafunction Output Ports (Part 1 of 2)</b>			
<b>Port Name</b>	<b>Required</b>	<b>Description</b>	<b>Comments</b>
nan	No	Asserted when the value of the <code>dataa</code> input port is a NaN representation.	Valid only when the operation mode is <code>FLOAT2INT</code> or <code>FLOAT2FLOAT</code> .  If the operation mode is <code>FLOAT2INT</code> , the <code>result</code> port is set to zero.  If the operation mode is <code>FLOAT2FLOAT</code> , the <code>result</code> port is set to a NaN representation.
overflow	No	Asserted when the conversion result, after rounding, exceeds the maximum width of the result, or when the value of the <code>dataa</code> input port is infinity.	Valid only when the operation mode is <code>INT2FLOAT</code> or <code>FLOAT2FLOAT</code> .

**Table 3–2. ALTFP\_CONVERT Megafunction Output Ports (Part 2 of 2)**

Port Name	Required	Description	Comments
result []	Yes	Output for the floating-point conversion.	<p>If the operation mode is INT2FLOAT, the output bus is an IEEE floating-point representation.</p> <p>If the operation mode is FLOAT2INT, the output bus is an integer representation.</p> <p>If the precision is single precision, the output bus width is 32.</p> <p>If the precision is double precision, the output bus width is 64.</p> <p>If the precision is single-extended precision, the output bus range is [43..64].</p> <p>If the operation mode is FLOAT2FLOAT, the output bus is an IEEE floating-point representation.</p> <p>If the precision is single precision, the output bus width is a 64 double-precision representation.</p> <p>If the precision is double precision, the output bus width is a 32 single-precision representation.</p> <p>If the precision is single extended precision, the output bus range is [43..64].</p>
underflow	No	Asserted when the conversion result, after rounding, is fractional.	<p>Valid only when the operation mode is FLOAT2INT or FLOAT2FLOAT.</p> <p>If the operation mode is FLOAT2INT, the underflow flag is asserted when the exponent value of the floating-point input is below the exponent bias.</p> <p>If the operation mode is FLOAT2FLOAT, the underflow flag is asserted when the floating-point input has a value smaller than the lowest exponent limit of the target floating-point format.</p>

Table 3–3 lists the parameters for the ALTFP\_CONVERT megafunction.

<b>Table 3–3. ALTFP_CONVERT Megafunction Parameters (Part 1 of 2)</b>			
<b>Parameter Name</b>	<b>Type</b>	<b>Required</b>	<b>Description</b>
OPERATION	String	Yes	<p>Specifies the operation mode. Values are INT2FLOAT, FLOAT2INT, and FLOAT2FLOAT. If omitted, the default value is INT2FLOAT.</p> <p>When set to INT2FLOAT, the conversion of an integer to an IEEE floating-point representation takes place. This conversion requires six pipeline latency levels.</p> <p>When set to FLOAT2INT, the conversion of an IEEE floating-point to an integer representation takes place. This conversion requires six pipeline latency levels.</p> <p>When set to FLOAT2FLOAT, the conversion between IEEE floating-point representations takes place.</p>
ROUNDING	String	Yes	<p>Specifies the rounding mode. The default value is TO_NEAREST. Other rounding modes are not supported.</p>
WIDTH_EXP_INPUT	Integer	Yes	<p>Specifies the precision of the input exponent. If omitted, the default is 8. The bias of the exponent is always set to <math>(2^{\text{WIDTH\_EXP\_INPUT}} - 1)</math>, which is 127 for single precision and 1023 for double precision. The value of WIDTH_EXP_INPUT must be 8 for single precision, 11 for double precision, and a minimum of 11 for single-extended precision. The value of WIDTH_EXP_INPUT must be less than the value of WIDTH_MAN_INPUT, and the sum of WIDTH_EXP_INPUT and WIDTH_MAN_INPUT must be less than 64. These settings apply to FLOAT2INT and FLOAT2FLOAT operation modes.</p>
WIDTH_MAN_INPUT	Integer	Yes	<p>Specifies the precision of the input mantissa. If omitted, the default is 23. When WIDTH_EXP_INPUT is 8 and the floating-point format is single precision, the WIDTH_MAN_INPUT value must be 23. Otherwise, the value of WIDTH_MAN_INPUT must be a minimum of 31. The value of WIDTH_MAN_INPUT must be greater than the value of WIDTH_EXP_INPUT, and the sum of WIDTH_EXP_INPUT and WIDTH_MAN_INPUT must be less than 64. These settings apply to FLOAT2INT and FLOAT2FLOAT operation modes.</p>

**Table 3–3. ALTFP\_CONVERT Megafunction Parameters (Part 2 of 2)**

Parameter Name	Type	Required	Description
WIDTH_INT	Integer	Yes	Specifies the integer width in 32, 64, or n bits.  If the operation mode is INT2FLOAT, the integer width of the input is predefined. The value range of an n-bit width is 4 to 64 bits.  If the operation mode is FLOAT2INT, the integer width of the output is predefined.
WIDTH_DATA	Integer	Yes	Specifies the input data width.  If the operation mode is INT2FLOAT, the data width is the value of the WIDTH_INT parameter.  If the operation mode is FLOAT2INT or FLOAT2FLOAT, the data width is $(\text{WIDTH\_EXP\_INPUT} + \text{WIDTH\_MAN\_INPUT} + 1)$ .
WIDTH_EXP_OUTPUT	Integer	Yes	Specifies the precision of the output exponent. If omitted, the default is 8. The bias of the exponent is always set to $(2^{\text{WIDTH}-1} - 1)$ , which is 127 for single precision and 1023 for double precision. The value of WIDTH_EXP_OUTPUT must be 8 for single precision, 11 for double precision, and a minimum of 11 for single-extended precision. The value of WIDTH_EXP_OUTPUT must be less than the value of WIDTH_MAN_OUTPUT, and the sum of WIDTH_EXP_OUTPUT and WIDTH_MAN_OUTPUT must be less than 64. These settings apply to FLOAT2INT and FLOAT2FLOAT operation modes.
WIDTH_MAN_OUTPUT	Integer	Yes	Specifies the precision of the output mantissa. If omitted, the default is 23. When WIDTH_EXP_OUTPUT is 8 and the floating-point format is single precision, the WIDTH_MAN_OUTPUT value must be 23. Otherwise, the value of WIDTH_MAN_OUTPUT must be a minimum of 31. The value of WIDTH_MAN_OUTPUT must be greater than the value of WIDTH_EXP_OUTPUT, and the sum of WIDTH_EXP_OUTPUT and WIDTH_MAN_OUTPUT must be less than 64. These settings apply to FLOAT2INT and FLOAT2FLOAT operation modes.
WIDTH_RESULT	Integer	Yes	Specifies the width of the output result. In an INT2FLOAT or FLOAT2FLOAT operation, the result width is $(\text{WIDTH\_EXP\_OUTPUT} + \text{WIDTH\_MAN\_OUTPUT} + 1)$ . In a FLOAT2INT operation, the result width is the value of the WIDTH_INT parameter.