Floating-Point IP Cores User Guide

Subscribe

Send Feedback

UG-01058
2016.12.09

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

now part of Intel

Contents

About Floating-Point IP Cores 1

2016.12.09

UG-01058 Subscribe Send Feedback

The Altera® floating-point IP cores enable you to perform floating-point arithmetic in FPGAs through optimized parameterizable functions for Altera device architectures.

You can customize the IP cores by configuring various parameters to accommodate your needs.

Related Information

Floating-Point IP Cores User Guide Document Archives on page 19-1
Provides a list of user guides for previous versions of the Floating-Point IP Cores.

List of Floating-Point IP Cores

This table lists the Floating-Point IP cores.

Table 1-1: List of IP Cores

IP Core Name	Function Overview
Operator Functions	
ALTFP_ADD_SUB	Adder/Subtractor
ALTFP_DIV	Divider
ALTFP_MULT	Multiplier
ALTFP_SQRT	Square Root
Algebraic and Trancendental Functions	
ALTFP_EXP	Exponential
ALTFP_INV	Inverse
ALTFP_INV_SQRT	Inverse Square Root
ALTFP_LOG	Natural Logarithm
Trigonometric Functions	
ALTFP_ATAN	Arctangent
ALTFP_SINCOS	Trigonometric Sine/Cosine
Other Functions	

IP Core Name	Function Overview
ALTFP_ABS	Absolute value
ALTFP_COMPARE	Comparator
ALTFP_CONVERT	Converter
ALTERA_FP_ACC_CUSTOM	An Application Specific Accumulator
ALTERA_FP_FUNCTIONS	A Collection of Floating-Point Functions. This IP core replaces all other Floating-Point IP cores listed in this table for Arria 10 devices.
Complex Functions	
ALTFP_MATRIX_INV	Matrix Inverse
ALTFP_MATRIX_MULT	Matrix Multiplier

Related Information

Introduction to Altera IP Cores
Provides general information about Altera IP cores

Installing and Licensing IP Cores

The Quartus® Prime software installation includes the Altera FPGA IP library. This library provides useful IP core functions for your production use without the need for an additional license. Some MegaCore® IP functions in the library require that you purchase a separate license for production use. The OpenCore® feature allows evaluation of any Altera FPGA IP core in simulation and compilation in the Quartus Prime software. Upon satisfaction with functionality and performance, visit the Self Service Licensing Center to obtain a license number for any Altera FPGA product.

The Quartus Prime software installs IP cores in the following locations by default:

Figure 1-1: IP Core Installation Path

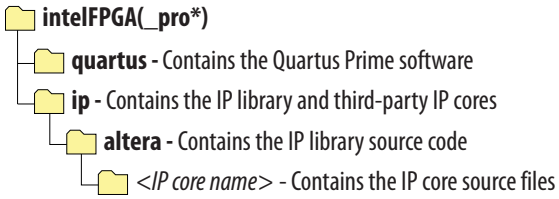


Table 1-2: IP Core Installation Locations

Location	Software	Platform
`<drive>:\intelFPGA_pro\quartus\ip\altera`	Quartus Prime Pro Edition	Windows

Location	Software	Platform
`<drive>:\intelFPGA\quartus\ip\altera`	Quartus Prime Standard Edition	Windows
`<home directory>:/intelFPGA_pro/quartus/ip/altera`	Quartus Prime Pro Edition	Linux
`<home directory>:/intelFPGA/quartus/ip/altera`	Quartus Prime Standard Edition	Linux

Design Flow

Use the IP Catalog and parameter editor to define and instantiate complex IP cores. Using the GUI ensures that you set all IP core ports and parameters properly.

If you are an expert user, and choose to configure the IP core directly through parameterized instantiation in your design, refer to the port and parameter details. The details of these ports and parameters are hidden in the parameter editor.

IP Catalog and Parameter Editor

The IP Catalog displays the IP cores available for your project. Use the following features of the IP Catalog to locate and customize an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, to open the IP core's installation folder, and for links to IP documentation.
- Click **Search for Partner IP** to access partner IP information on the web.

The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Quartus Prime IP file (`.ip`) for an IP variation in Quartus Prime Pro Edition projects.

The parameter editor generates a top-level Quartus IP file (`.qip`) for an IP variation in Quartus Prime Standard Edition projects. These files represent the IP variation in the project, and store parameterization information.

Figure 1-2: IP Parameter Editor (Quartus Prime Pro Edition)

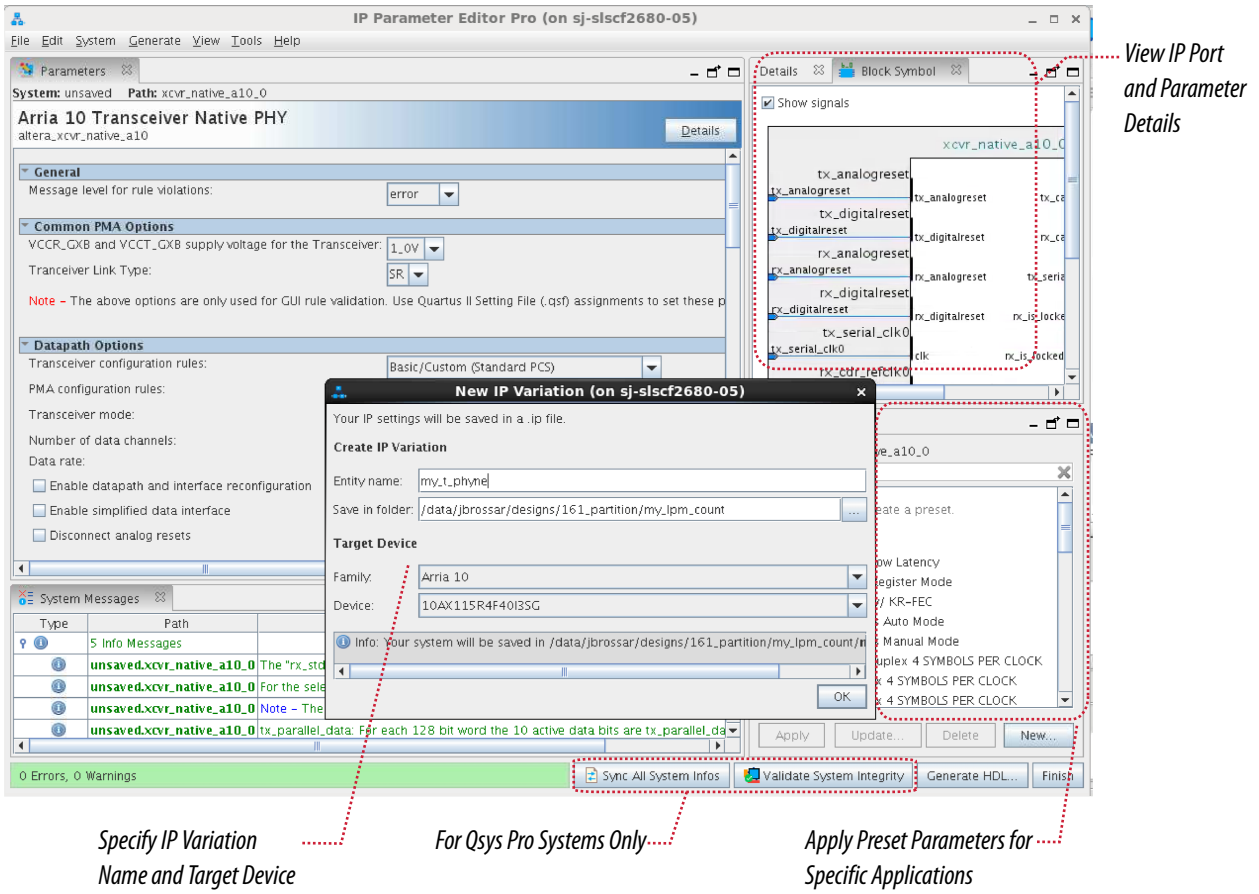
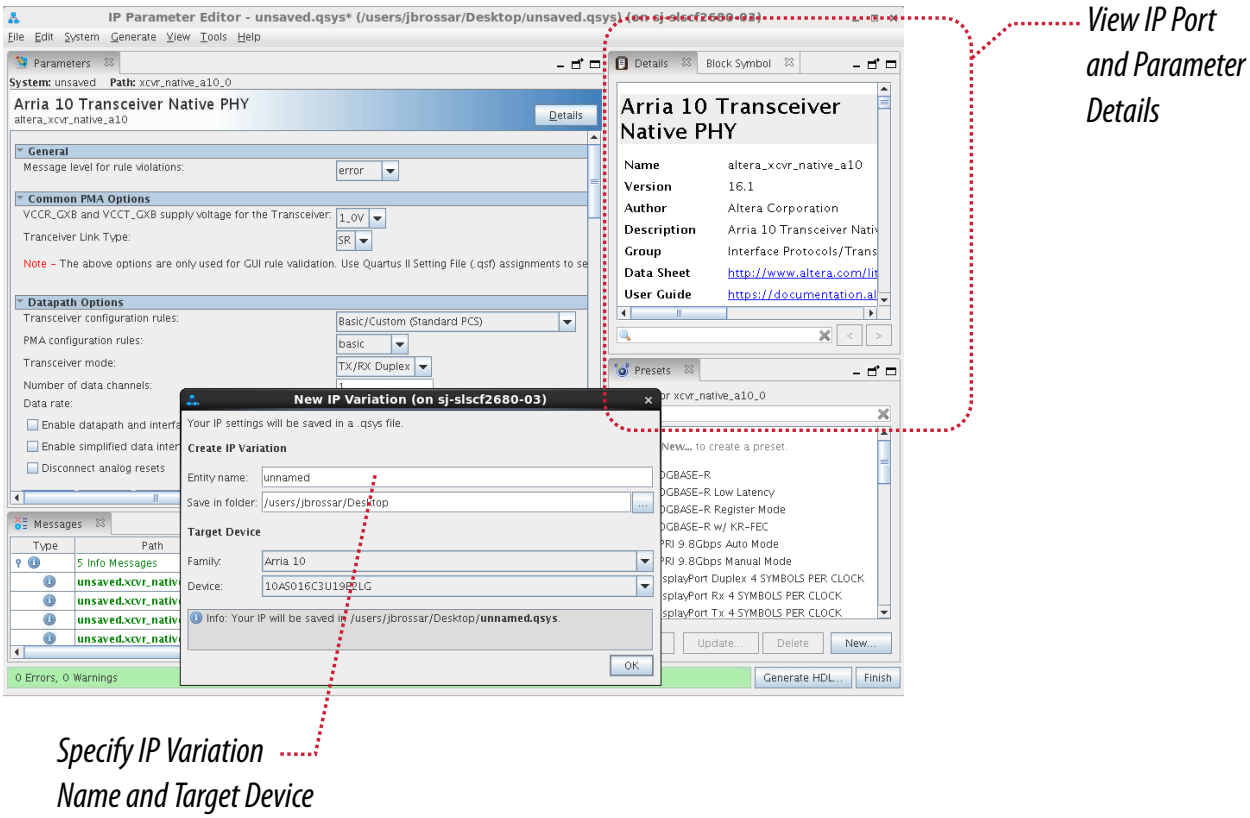


Figure 1-3: IP Parameter Editor (Quartus Prime Standard Edition)



The Parameter Editor

The parameter editor helps you to configure IP core ports, parameters, and output file generation options. The basic parameter editor controls include the following:

- Use the **Presets** window to apply preset parameter values for specific applications (for select cores).
- Use the **Details** window to view port and parameter descriptions, and click links to documentation.
- Click **Generate** > **Generate Testbench System** to generate a testbench system (for select cores).
- Click **Generate** > **Generate Example Design** to generate an example design (for select cores).
- Click **Validate System Integrity** to validate a system's generic components against companion files. (Qsys Pro systems only)
- Click **Sync All System Infos** to validate a system's generic components against companion files. (Qsys Pro systems only)

The IP Catalog is also available in Qsys and Qsys Pro (**View** > **IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus Prime IP Catalog. Refer to *Creating a System with Qsys Pro* or *Creating a System with Qsys* for information on use of IP in Qsys and Qsys Pro, respectively.

Related Information

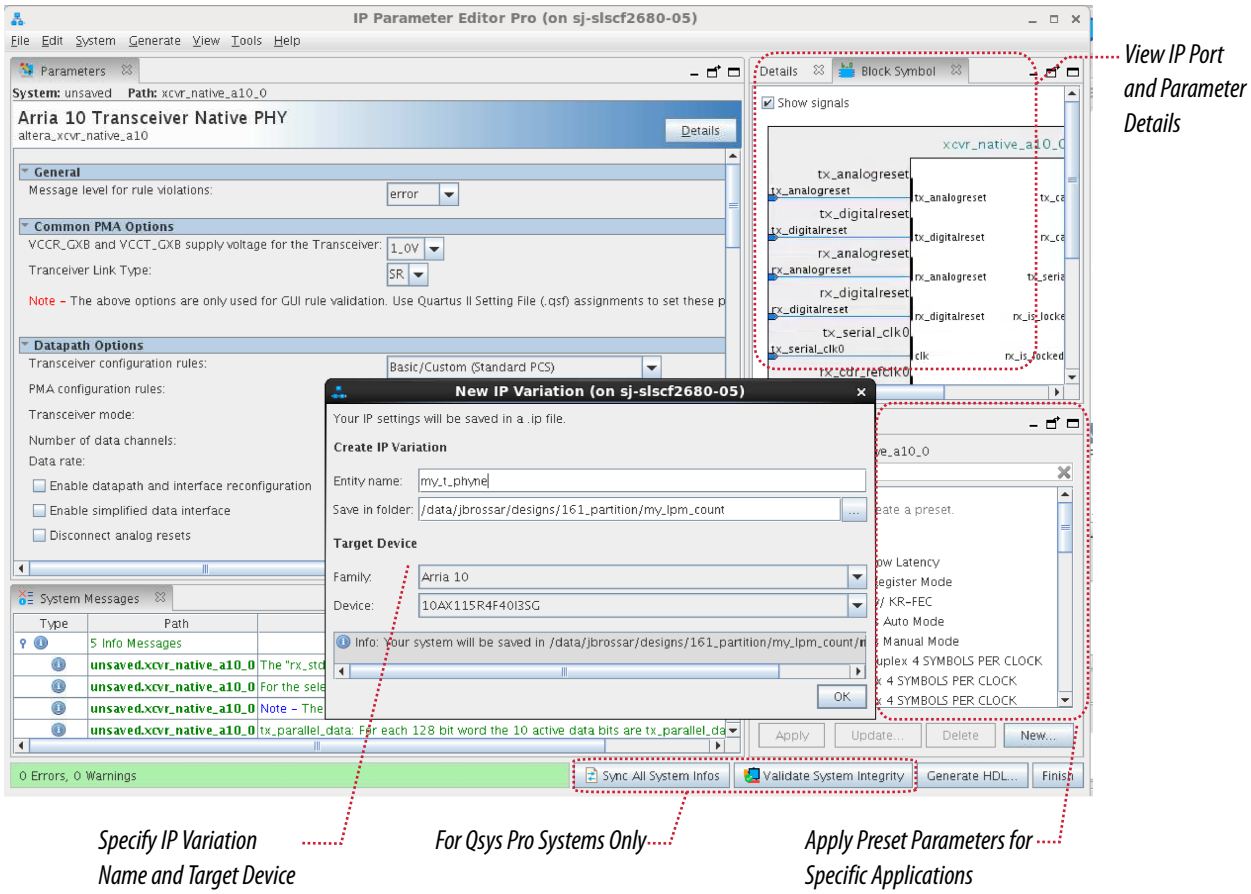
- Creating a System with Qsys Pro
- Creating a System with Qsys

Generating IP Cores (Quartus Prime Pro Edition)

Configure a custom IP variation in the parameter editor.

Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to define a custom variation of the selected IP core. The parameter editor generates the IP variation and adds the corresponding `.ip` file to your project automatically.

Figure 1-4: IP Parameter Editor (Quartus Prime Pro Edition)



Follow these steps to locate, instantiate, and customize an IP variation in the parameter editor:

1. Click **Tools** > **IP Catalog**. To display details about device support, installation location, versions, and links to documentation, right-click any IP component name in the IP Catalog.
2. To locate a specific type of component, type some or all of the component's name in the IP Catalog search box. For example, type `memory` to locate memory IP components, or `axi` to locate IP components with AXI in the IP name. Apply filters to the IP Catalog display from the right-click menu.
3. To launch the parameter editor, double-click any component. Specify a top-level name for your custom IP variation. The parameter editor saves the IP variation settings in a file named `<your_ip>.ip`. Click **OK**. Do not include spaces in IP variation names or paths.
4. Set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in IP parameters:

- Optionally select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
- Specify parameters defining the IP core functionality, port configurations, and device-specific features.
- Specify options for processing the IP core files in other EDA tools.

Note: Refer to your IP core user guide for information about specific IP core parameters.

5. Click **Generate HDL**. The **Generation** dialog box appears.
6. Specify output file generation options, and then click **Generate**. The synthesis and/or simulation files generate according to your specifications.
7. To generate a simulation testbench, click **Generate** > **Generate Testbench System**. Specify testbench generation options, and then click **Generate**.
8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate** > **Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project.
10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

Note: Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

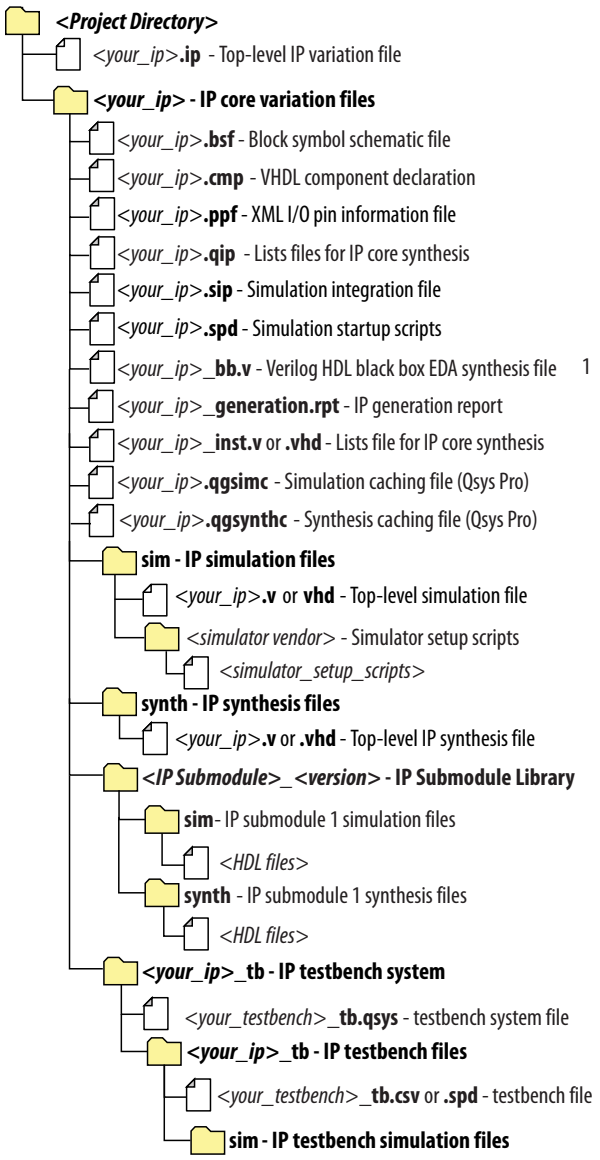
Related Information

- **IP User Guide Documentation**
- **Altera FPGA IP Release Notes**

IP Core Generation Output (Quartus Prime Pro Edition)

The Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Qsys system.

Figure 1-5: Individual IP Core Generation Output (Quartus Prime Pro Edition)



1. If supported and enabled for your IP core variation.

Table 1-3: Files Generated for IP Cores

File Name	Description
`<my_ip>.ip`	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Qsys Pro system, the parameter editor also generates a `.qsys` file.
`<my_ip>.cmp`	The VHDL Component Declaration (`.cmp`) file is a text file that contains local generic and port definitions that you use in VHDL design files.

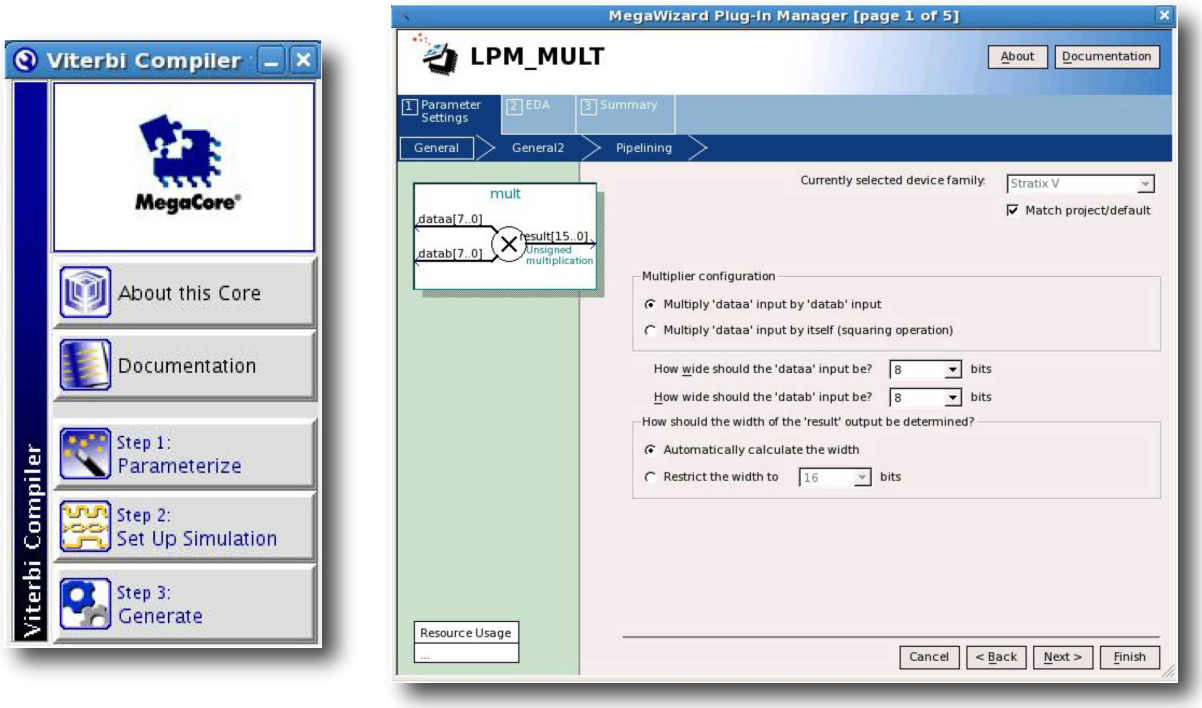
File Name	Description
`<my_ip>_generation.rpt`	IP or Qsys generation log file. A summary of the messages during IP generation.
`<my_ip>.qgsimc` (Qsys Pro systems only)	Simulation caching file that compares the `.qsys` and `.ip` files with the current parameterization of the Qsys Pro system and IP core. This comparison determines if Qsys Pro can skip regeneration of the HDL.
`<my_ip>.qgsynth` (Qsys Pro systems only)	Synthesis caching file that compares the `.qsys` and `.ip` files with the current parameterization of the Qsys Pro system and IP core. This comparison determines if Qsys Pro can skip regeneration of the HDL.
`<my_ip>.qip`	Contains all information to integrate and compile the IP component.
`<my_ip>.csv`	Contains information about the upgrade status of the IP component.
`<my_ip>.bsf`	A symbol representation of the IP variation for use in Block Diagram Files (`.bdf`).
`<my_ip>.spd`	Required input file for `ip-make-simscript` to generate simulation scripts for supported simulators. The `.spd` file contains a list of files you generate for simulation, along with information about memories that you initialize.
`<my_ip>.ppf`	The Pin Planner File (`.ppf`) stores the port and node assignments for IP components you create for use with the Pin Planner.
`<my_ip>_bb.v`	Use the Verilog blackbox (`_bb.v`) file as an empty module declaration for use as a blackbox.
`<my_ip>.sip`	Contains information you require for NativeLink simulation of IP components. Add the `.sip` file to your Quartus Prime Standard Edition project to enable NativeLink for supported devices. The Quartus Prime Pro Edition software does not support NativeLink simulation.
`<my_ip>_inst.v` or `_inst.vhd`	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
`<my_ip>.regmap`	If the IP contains register information, the Quartus Prime software generates the `.regmap` file. The `.regmap` file describes the register map information of master and slave interfaces. This file complements the `..sopcinfo` file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
`<my_ip>.svd`	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Qsys Pro system. During synthesis, the Quartus Prime software stores the `.svd` files for slave interface visible to the System Console masters in the `.sof` file in the debug session. System Console reads this section, which Qsys Pro queries for register map information. For system slaves, Qsys Pro accesses the registers by name.

File Name	Description
`<my_ip>.v` `<my_ip>.vhd`	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
`mentor/`	Contains a ModelSim® script `msim_setup.tcl` to set up and run a simulation.
`aldec/`	Contains a Riviera-PRO script `rivierapro_setup.tcl` to setup and run a simulation.
`/synopsys/vcs` `/synopsys/vcsmx`	Contains a shell script `vcs_setup.sh` to set up and run a VCS® simulation. Contains a shell script `vcsmx_setup.sh` and `synopsys_sim.setup` file to set up and run a VCS MX® simulation.
`/cadence`	Contains a shell script `ncsim_setup.sh` and other setup files to set up and run an NCSIM simulation.
`/submodules`	Contains HDL files for the IP core submodule.
`<IP submodule>/`	For each generated IP submodule directory Qsys Pro generates `/synth` and `/sim` sub-directories.

Generating IP Cores (Quartus Prime Standard Edition)

This topic describes parameterizing and generating an IP variation using a legacy parameter editor in the Quartus Prime Standard Edition software.

Figure 1-6: Legacy Parameter Editors



Note: The legacy parameter editor generates a different output file structure than the Quartus Prime Pro Edition software.

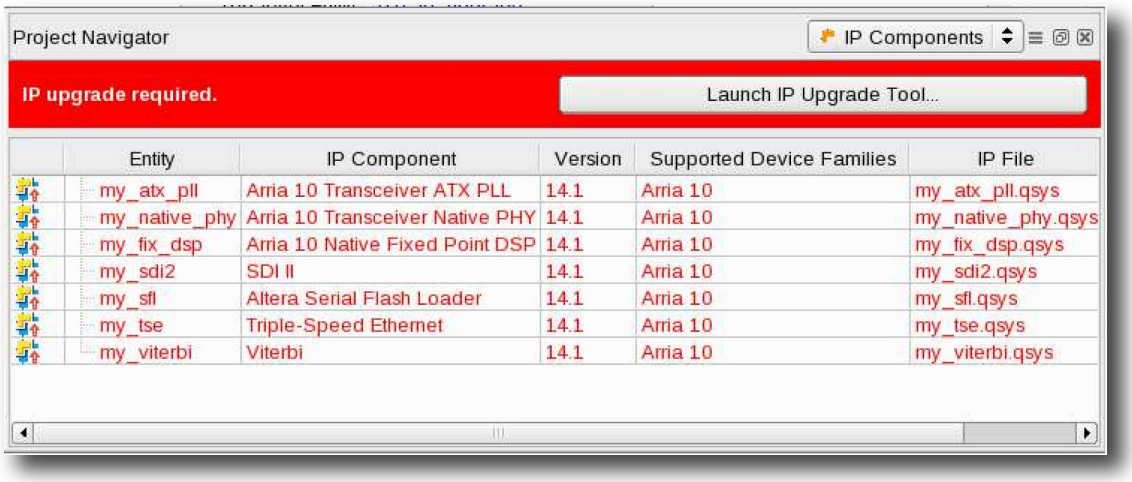
1. In the IP Catalog (**Tools** > **IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name and output HDL file type for your IP variation. This name identifies the IP core variation files in your project. Click **OK**. Do not include spaces in IP variation names or paths.
3. Specify the parameters and options for your IP variation in the parameter editor. Refer to your IP core user guide for information about specific IP core parameters.
4. Click **Finish** or **Generate** (depending on the parameter editor version). The parameter editor generates the files for your IP variation according to your specifications. Click **Exit** if prompted when generation is complete. The parameter editor adds the top-level `.qip` file to the current project automatically.

   Note: For devices released prior to Arria® 10 devices, the generated `.qip` and `.sip` files must be added to your project to represent IP and Qsys systems. To manually add an IP variation generated with legacy parameter editor to a project, click **Project** > **Add/Remove Files in Project** and add the IP variation `.qip` file.

Upgrading IP Cores

Any IP variations that you generate from a previous version or different edition of the Quartus Prime software, may require upgrade before compilation in the current software edition or version.
The Project Navigator displays a banner indicating the IP upgrade status. Click **Launch IP Upgrade Tool** or **Project** > **Upgrade IP Components** to upgrade outdated IP cores.

Figure 1-7: IP Upgrade Alert in Project Navigator



Icons in the **Upgrade IP Components** dialog box indicate when IP upgrade is required, optional, or unsupported for an IP variation in the project. Upgrade IP variations that require upgrade before compilation in the current version of the Quartus Prime software.

Note: Upgrading IP cores may append a unique identifier to the original IP core entity name(s), without similarly modifying the IP instance name. There is no requirement to update these entity references in any supporting Quartus Prime file, such as the Quartus Prime Settings File (`.qsf`), Synopsys Design Constraints File (`.sdc`), or SignalTap File (`.stp`), if these files contain instance names. The Quartus Prime software reads only the instance name and ignores the entity name in paths that specify both names. Use only instance names in assignments.

Table 1-4: IP Core Upgrade Status

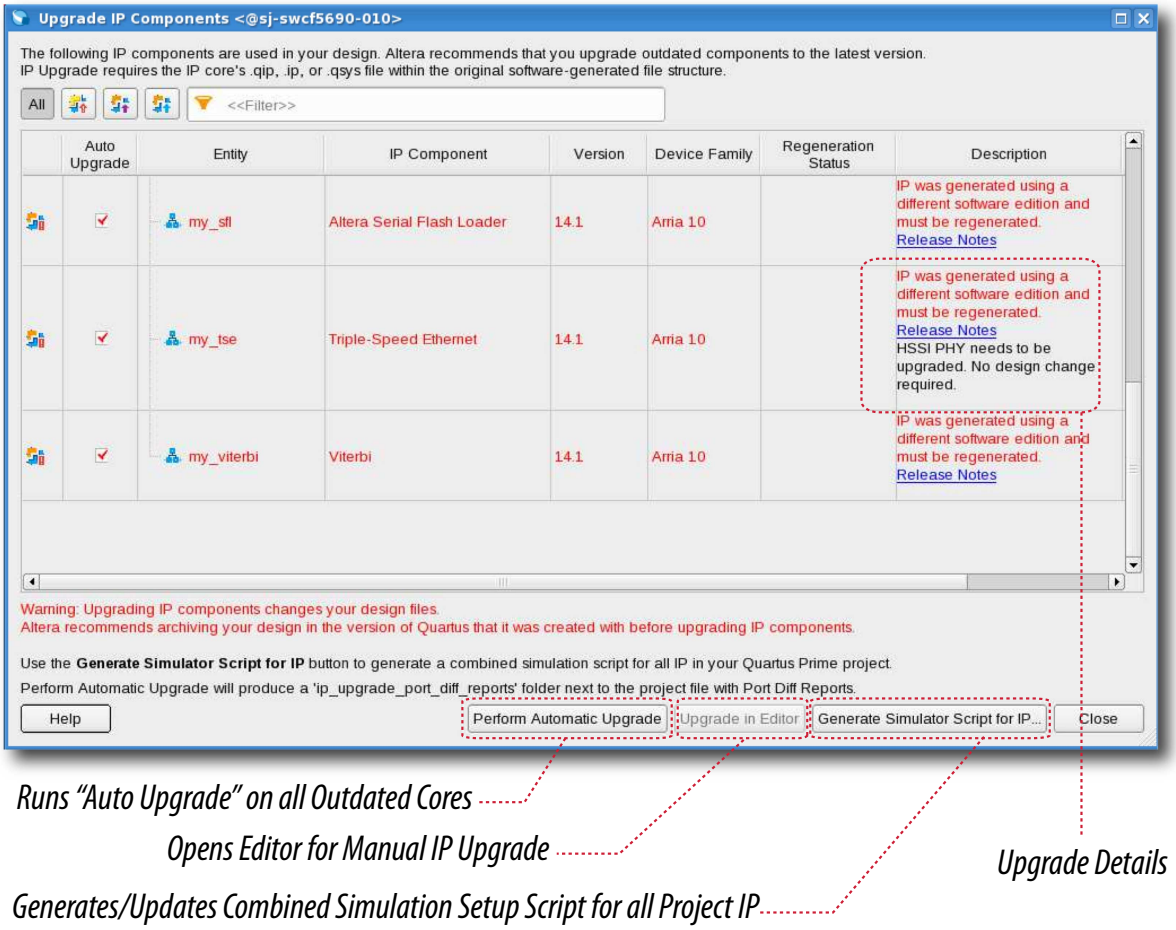
IP Core Status	Description
IP Upgraded 	Indicates that your IP variation uses the latest version of the IP core.

IP Core Status	Description
IP Upgrade Optional	Indicates that upgrade is optional for this IP variation in the current version of the Quartus Prime software. Optionally, upgrade this IP variation to take advantage of the latest development of this IP core. Retain previous IP core characteristics by declining to upgrade. Refer to the Description for details about IP core version differences. If you do not upgrade the IP, the IP variation synthesis and simulation files remain unchanged, and you cannot modify parameters until upgrading.
IP Upgrade Required	Indicates that you must upgrade the IP variation before compiling in the current version of the Quartus Prime software. Refer to the Description for details about IP core version differences.
IP Upgrade Unsupported	Indicates that Quartus Prime software does not support upgrade of the IP variation due to incompatibility in the current software version. The Quartus Prime software prompts you to replace the unsupported IP core with equivalent IP core from the IP Catalog. Refer to the Description for details about IP core version differences and links to Release Notes.
IP End of Life	Indicates that Altera designates the IP core as end-of-life status. You may or may not be able to edit the IP core in the parameter editor. Support for this IP core discontinues in future releases of the Quartus Prime software.
IP Upgrade Mismatch Warning	Provides warning of non-critical IP core differences in migrating IP to another device family.

Follow these steps to upgrade IP cores:

1. In the latest version of the Quartus Prime software, open the Quartus Prime project containing an outdated IP core variation. The **Upgrade IP Components** dialog box automatically displays the status of IP cores in your project, along with instructions for upgrading each core. To access this dialog box manually, click **Project** > **Upgrade IP Components**.
2. To upgrade one or more IP cores that support automatic upgrade, ensure that you turn on the **Auto Upgrade** option for the IP core(s), and click **Perform Automatic Upgrade**. The **Status** and **Version** columns update when upgrade is complete. Example designs provided with any Altera FPGA IP core regenerate automatically whenever you upgrade an IP core.
3. To manually upgrade an individual IP core, select the IP core and click **Upgrade in Editor** (or simply double-click the IP core name). The parameter editor opens, allowing you to adjust parameters and regenerate the latest version of the IP core.

Figure 1-8: Upgrading IP Cores



Note: IP cores older than Quartus Prime software version 12.0 do not support upgrade. Altera verifies that the current version of the Quartus Prime software compiles the previous two versions of each IP core. The *Altera FPGA IP Core Release Notes* reports any verification exceptions for Altera IP cores. Altera does not verify compilation for IP cores older than the previous two releases.

Related Information

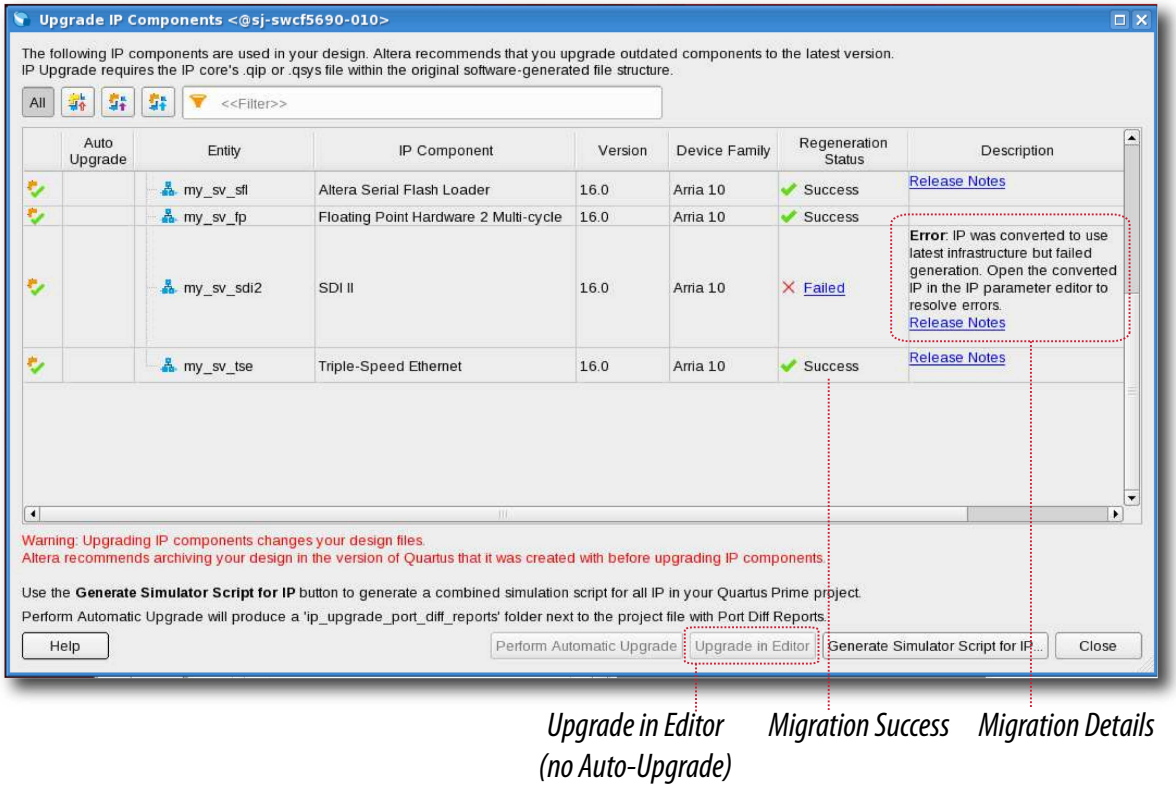
Altera FPGA IP Core Release Notes

Migrating IP Cores to a Different Device

Migrate an IP variation when you want to target a different (often newer) device. Most Altera FPGA IP cores support automatic migration. Some IP cores require manual IP regeneration for migration. A few IP cores do not support device migration, requiring you to replace them in the project. The **Upgrade IP Components** dialog box identifies the migration support level for each IP core in the design.

1. To display the IP cores that require migration, click **Project** > **Upgrade IP Components**. The **Description** field provides migration instructions and version differences.
2. To migrate one or more IP cores that support automatic upgrade, ensure that the **Auto Upgrade** option is turned on for the IP core(s), and click **Perform Automatic Upgrade**. The **Status** and **Version** columns update when upgrade is complete.
3. To migrate an IP core that does not support automatic upgrade, double-click the IP core name, and click **OK**. The parameter editor appears. If the parameter editor specifies a **Currently selected device family**, turn off **Match project/default**, and then select the new target device family.
4. Click **Generate HDL**, and confirm the **Synthesis** and **Simulation** file options. Verilog HDL is the default output file format. If you specify VHDL as the output format, select **VHDL** to retain the original output format.
5. Click **Finish** to complete migration of the IP core. Click **OK** if the software prompts you to overwrite IP core files. The **Device Family** column displays the new target device name when migration is complete.
6. To ensure correctness, review the latest parameters in the parameter editor or generated HDL.

Figure 1-9: IP Core Device Migration



Note: IP migration may change ports, parameters, or functionality of the IP variation. These changes may require you to modify your design or to re-parameterize your IP variant. During migration, the IP variation's HDL generates into a library that is different from the original output location of the IP core. Update any assignments that reference outdated locations. If a symbol in a supporting Block Design File schematic represents your upgraded IP core, replace the symbol

with the newly generated <*my_ip*>.bsf. Migration of some IP cores requires installed support for the original and migration device families.

Related Information

Altera FPGA IP Release Notes

Floating-Point IP Cores General Features

All Altera floating-point IP cores offer the following features:

- Support for floating-point formats.
- Input support for not-a-number (NaN), infinity, zero, and normal numbers.
- Optional asynchronous input ports including asynchronous clear (aclr) and clock enable (clk_en).
- Support for round-to-nearest-even rounding mode.
- Compute results of any mathematical operations according to the IEEE-754 standard compliance with a maximum of 1 unit in the last place (u.l.p.) error. This assumption is applied to all floating-point IP cores excluding complex matrix multiplication and inverse operations (for example, ALTFP_MATRIX_MULTI and ALFP_MATRIX_INV), where a slight increase in errors is observed due to the accumulation of errors during the mathematical operation.

Altera floating-point IP cores do not support denormal number inputs. If the input is a denormal value, the IP core forces the value to zero and treats the value as a zero before going through any operation.

Related Information

FFT MegaCore Function User Guide

Altera also offers the single-precision floating-point option in the FFT MegaCore.

IEEE-754 Standard for Floating-Point Arithmetic

The floating-point IP cores implement the following representations in the IEEE-754 standard:

- Floating-point numbers
- Special values (zero, infinity, denormal numbers, and NaN bit combinations)
- Single-precision, double-precision, and single-extended precision formats for floating-point numbers

Floating-Point Formats

All floating-point formats have binary patterns. In Figure 1–1, S represents a sign bit, E represents an exponent field, and M is the mantissa (part of a logarithm, or fraction) field.

For a normal floating-point number, a leading 1 is always implied, for example, binary 1.0011 or decimal 1.1875 is stored as 0011 in the mantissa field. This format saves the mantissa field from using an extra bit to represent the leading 1. However, the leading bit for a denormal number can be either 0 or 1. For zero, infinity, and NaN, the mantissa field does not have an implied leading 1 nor any explicit leading bit.

Figure 1-10: IEEE-754 Floating-Point Format

This figure shows a floating-point format.

Single-Precision Format

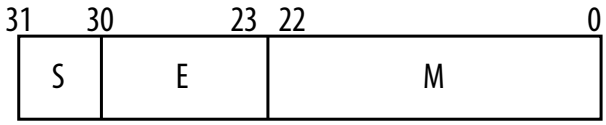
The single-precision format contains the following binary patterns:

- The MSB holds the sign bit.
- The next 8 bits hold the exponent bits.
- 23 LSBs hold the mantissa.

The total width of a floating-point number in the single-precision format is 32 bits. The bias for the single-precision format is 127.

Figure 1-11: Single-Precision Representation

This figure shows a single-precision representation.



Double-Precision Format

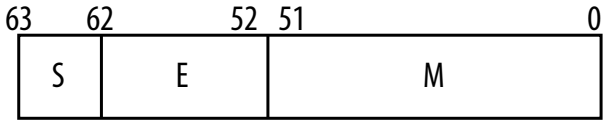
The double-precision format contains the following binary patterns:

- The MSB holds the sign bit.
- The next 11 bits hold the exponent bits.
- 52 LSBs hold the mantissa.

The total width of a floating-point number in the double-precision format is 64 bits. The bias for the double-precision format is 1023.

Figure 1-12: Double-Precision Representation

This figure shows a double-precision representation.



Single-Extended Precision Format

The single-extended precision format contains the following binary patterns:

- The MSB holds the sign bit.
- The exponent and mantissa fields do not have fixed widths.
- The minimum exponent field width is 11 bits and must be 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 widths of the sign bit, exponent field, and mantissa field must be a minimum of 43 bits and a maximum of 64 bits. The bias for the single-extended precision format is unspecified in the IEEE-754 standard. In these IP cores, a bias of $2^{(WIDTH\_EXP-1)}-1$ is assumed for the single-extended precision format.

Special Case Numbers

The following table lists the special case numbers defined by the IEEE-754 standard and the data bit representations.

Table 1-5: Special Case Numbers in IEEE-754 Representation

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

The IEEE-754 standard defines four types of rounding modes, which are:

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

Altera floating-point IP cores support only the most commonly used rounding mode, which is the round-to-nearest-even mode (`TO_NEAREST`). With round-to-nearest-even, the IP core rounds the result to the nearest floating-point number. If the result is exactly halfway between two floating-point numbers, the IP core rounds the result so that the LSB becomes a zero, which is even.

Non-IEEE-754 Standard Format

Only the ALTFP_CONVERT and ALTERA_FP_FUNCTIONS (when the convert function is selected) support the fixed point format.

The fixed-point data type is similar to the conventional integer data type, except that the fixed-point data carries a predetermined number of fractional bits. If the width of the fraction is 0, the data becomes a normal signed integer.

The notation for fixed-point format numbers in this user guide is Qm.f, where Q designates that the number is in Q format notation, m is the number of bits used to indicate the integer portion of the number, and f is the number of bits used to indicate the fractional portion of the number.

For example, Q4.12 describes a number with 4 integer bits and 12 fractional bits in a 16-bit word.

The following figures show the difference between the signed-integer format and the fixed-point format for a 32-bit number.

Figure 1-13: Signed-Integer Format

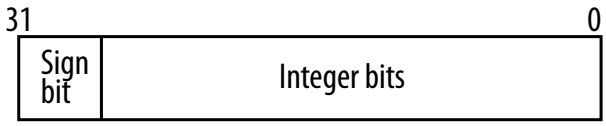
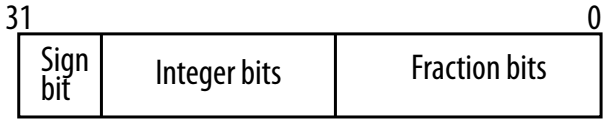


Figure 1-14: Fixed-Point Format



Floating-Points IP Cores Output Latency

The IP cores measure the output latency in clock cycles and is different for each IP core. In some IP cores, the precision modes determine the number of clock cycles between the input and output result. When you select a mode, the options for latency are fixed for that mode.

For specific details about latency options, refer to the Output *Latency* section of your selected IP core in this user guide.

Floating-Point IP Cores Design Example Files

The design examples for each IP core in this user guide use the IP Catalog and parameter editor to define custom IP variations.

Simulate the designs in the ModelSim®-Altera software to generate a waveform display of the device behavior. You must be familiar with the ModelSim-Altera software before trying out the design examples.

Table 1-6: Design Files for Floating-Point IP Cores

Floating-Point IP Cores	Design Files
ALTFP_ADD_SUB	• `altfp_add_sub_DesignExample.zip` (Quartus II design files) • `altfp_add_sub_ex_msim.zip` (ModelSim-Altera files)

Floating-Point IP Cores	Design Files
ALTFP_DIV	• altfp_div_DesignExample.zip (Quartus II design files) • altfp_div_ex_msim.zip (ModelSim-Altera files)
ALTFP_MULT	• altfp_mult_DesignExample.zip (Quartus II design files) • altfp_mult_ex_msim.zip (ModelSim-Altera files)
ALTFP_SQRT	• altfp_sqrt_DesignExample.zip (Quartus II design files) • altfp_sqrt_ex_msim.zip (ModelSim-Altera files)
ALTFP_EXP	• altfp_exp_DesignExample.zip (Quartus II design files) • altfp_exp_ex_msim.zip (ModelSim-Altera files)
ALTFP_INV	• altfp_inv_DesignExample.zip (Quartus II design files) • altfp_inv_ex_msim.zip (ModelSim-Altera files)
ALTFP_INV_SQRT	• altfp_inv_sqrt_DesignExample.zip (Quartus II design files) • altfp_inv_sqrt_ex_msim.zip (ModelSim-Altera files)
ALTFP_LOG	• altfp_log_DesignExample.zip (Quartus II design files) • altfp_log_ex_msim.zip (ModelSim-Altera files)
ALTFP_ATAN	Not Available
ALTFP_SINCOS	Not Available
ALTFP_ABS	• altfp_mult_abs_DesignExample.zip (Quartus II design files) • altfp_mult_abs_ex_msim.zip (ModelSim-Altera files)
ALTFP_COMPARE	• altfp_compare_DesignExample.zip (Quartus II design files) • altfp_compare_ex_msim.zip (ModelSim-Altera files)
ALTFP_CONVERT	• altfp_convert_DesignExample.zip (Quartus II design files) • altfp_convert_float2int_msim.zip (ModelSim-Altera files)
ALTERA_FP_ACC_CUSTOM	Not Available

Floating-Point IP Cores	Design Files
ALTERA_FP_FUNCTIONS	Not Available
ALTERA_FP_MATRIX_INV	• `altfp_matrix_inv_DesignExample.zip (Quartus II design files)` • `altfp_matrix_inv_ex_msim.zip (ModelSim-Altera files)`
ALTERA_FP_MATRIX_MULT	Not Available

Related Information

- **ALTERA_FP_MATRIX_INV Design Example: Matrix Inverse of Single-Precision Format Numbers** on page 2-6
- **ALTFP_ADD_SUB Design Example: Addition of Double-Precision Format Numbers** on page 5-3
- **ALTFP_DIV Design Example: Division of Single-Precision** on page 6-4
- **ALTFP_MULT Design Example: Multiplication of Double-Precision Format Numbers** on page 7-3
- **ALTFP_SQRT Design Example: Square Root of Single-Precision Format Numbers** on page 8-3
- **ALTFP_EXP Design Example: Exponential of Single-Precision Format Numbers** on page 9-2
- **ALTFP_INV Design Example: Inverse of Single-Precision Format Numbers** on page 10-2
This design example uses the ALTFP_INV IP core to compute the inverse of single-precision format numbers. This example uses the parameter editor in the Quartus II software.
- **ALTFP_INV_SQRT Design Example: Inverse Square Root of Single-Precision Format Numbers** on page 11-2
- **ALTFP_LOG Design Example: Natural Logarithm of Single-Precision Format Numbers** on page 12-2
- **ALTFP_ABS Design Example: Absolute Value of Multiplication Results** on page 15-2
- **ALTFP_COMPARE Design Example: Comparison of Single-Precision Format Numbers** on page 16-2
- **ALTFP_CONVERT Design Example: Convert Double-Precision Floating-Point Format Numbers** on page 17-6
- **Floating-Point IP Cores Design Examples**
Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
Provides information about installation, usage, and troubleshooting

VHDL Component Declaration

The VHDL component declaration is located in the `<Quartus Prime installation directory>\libraries\vhdl\altera_mf\altera_mf_components.vhd`

VHDL LIBRARY-USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf_altera_mf_components.all;
```

ALTERA_FP_MATRIX_INV IP Core 2

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core enables you to perform matrix inversion operation using a combination of Cholesky decomposition, triangular matrix inversion, and matrix multiplication.

ALTERA_FP_MATRIX_INV Features

The ALTERA_FP_MATRIX_INV IP core offers the following features:

- Inversion of a matrix.
- Support for floating-point format in single precision.
- Support for VHDL and Verilog HDL languages.
- Support for matrix sizes up to are 4 × 4, 6 × 6, 8 × 8, 16 ×16, 32 × 32, and 64 × 64.
- Use of control signal, `load`.
- Use of handshaking signals: `busy`, `outvalid`, and `done`.

ALTERA_FP_MATRIX_INV Output Latency

The ALTERA_FP_MATRIX_INV IP core does not have a fixed output latency. Instead, it uses handshaking signals to interface with external circuitry.

ALTERA_FP_MATRIX_INV Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTERA_FP_MATRIX_INV IP core. The information was derived using the Quartus II software version 10.0

Table 2-1: ALTERA_FP_MATRIX_INV Resource Utilization and Performance for the Stratix IV Device Family

Precision	Matrix Size	Blocks	Logic usage					Latency	Throughput (kb/s)	Giga Floating-Point Operations per Second (GFLOPS)	f_{MAX} (MHz)
			Adaptive Logic Modules (ALMs)	DSP Usage (18 x 18 DSPs)	M9K	M144K	Memory (Bits)				
Single	4× 4	2	21159	222	139	—	19919	Pending	Pending	Pending	221
	6 × 6	2	59827	574	90	—	15759	Pending	Pending	Pending	170
	8 × 8	2	5,538	63	49	—	53,736	2,501	3,987	15.26	332
	16 × 16	4	8,865	95	80	—	138,051	11,057	855	30.93	329
	32 × 32	8	15,655	159	193	—	699,164	52,625	165	55.12	290
	64 × 64	16	29,940	287	386	22	4,770,369	281,505	25	83.16	218

ALTERA_FP_MATRIX_INV Functional Description

A matrix inversion function is composed of the following components:

- Cholesky decomposition function.

  The Cholesky decomposition function generates a lower triangular matrix.
- Triangular matrix inversion function.

  The triangular matrix inversion process then generates the inverse of the lower triangular using backward substitution.
- Matrix multiplication function.

  The matrix multiplier multiplies the transpose of the inverse triangular matrix with the inverse triangular matrix.

In linear algebra, the Cholesky decomposition states that every positive definite matrix A is decomposed as A = L×LT

where, L is a lower triangular matrix, and LT denotes the transpose of L.

The property of invertible matrices states that (X×Y)-1 = X-1×Y-1 and the property of transpose states that (XT)-1 = (X-1)T. Combining these two properties, the following equation represents a derivation of a matrix inversion using the Cholesky decomposition method:

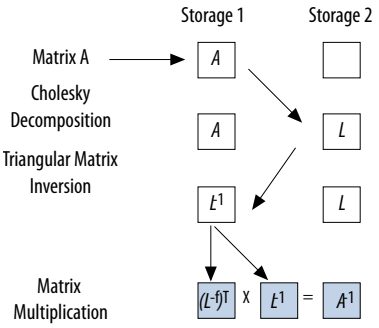
A-1 = (L×LT)-1

$= (LT)\text{-}1 \times L\text{-}1$

$= (L\text{-}1)T \times L\text{-}1$

where a Cholesky decomposition function is needed to obtain L, a triangular matrix inversion is needed to obtain L-1, and a matrix multiplication is needed for (L-1)T × L-1.

Figure 2-1: Matrix Inversion Flow Diagram



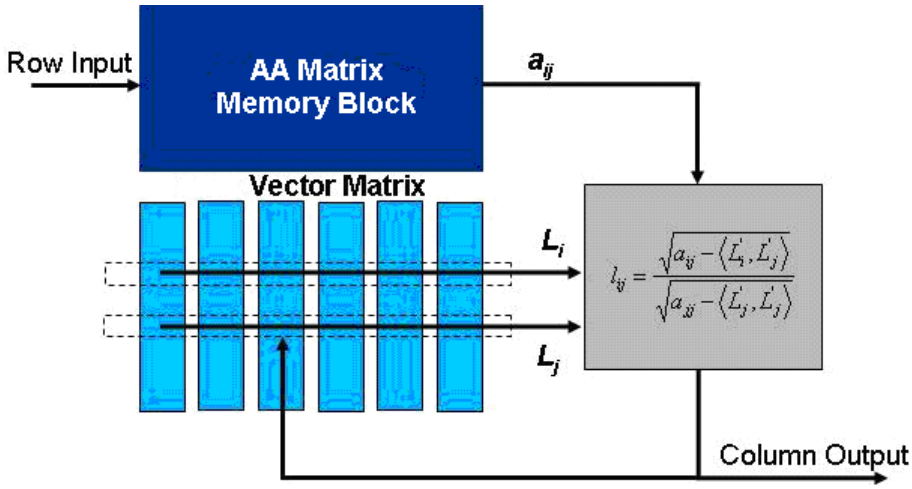
Cholesky Decomposition Function

The functions consists of two memory and two processing blocks. One of the memory blocks is the input matrix memory block and is loaded with the input matrix in a row order, one element at a time. However, during processing, this block is read in a column order, one element at a time when required.

The other memory block is the processing matrix block which consists of multiple column memories to enable an entire row to be read at once. During the loading of the input memory, the FPC datapath preprocesses the input elements to generate the first column of the resulting triangular matrix. The top element of the first column, l00, is the square root of the input matrix value `a00`. The rest of the first column, `li0` is the input value `ai0` divided by l00. This preprocessing step introduces latency into the load, during which the `INIT_BUSY` signal is asserted. The `CALCULATE` signal initiates and starts processing after the `INIT_BUSY` signal is deasserted.

This figure shows the top-level architecture of the Cholesky decomposition function, where the monolithic input memory and the column-wise processing memory, also known as the vector matrix, are shown. The gray block is the FPC datapath section.

Figure 2-2: Cholesky Decomposition Function Top-level Diagram



Although the Cholesky decomposition algorithm only operates on the lower triangular matrix, the core requires the entire matrix to be loaded, during which the processing or vector memory is initialized.

The FPC datapath is split into two sections. The first section, also known as the vector section, takes the inner product of two vectors and subtracts it from the input matrix element, a_{ij}. The second section, also known as the root section, calculates square roots and performs division by the square root. The first element is loaded into both inputs of the root section and the outcome is its own square root. The first element continues to stay latched in the left input field of the root section while all the other elements of the first column are loaded into the right input field. The resulting output is the value of the respective column element divided by the value of the first element of the Cholesky decomposition matrix.

During processing, two rows from the processing matrix are loaded. For the first element in each new column, both rows have the same index; hence contain the same values. The first row is latched into the input register of the vector section. For the rest of the column, the row index is increased, and a new a_{ij} element and triangular matrix vector, L_j is loaded. The first result out of the vector section is latched onto the left register of the root section. All results from the column, including the first result, are loaded into the right register of the root section. The root section generates the square root of the first vector result, while for the other results coming from the vector section, the number is divided by the square root of the first result.

All calculated values are written to another memory block for further processing. The first column values are output singly during preprocessing, while the values of other columns are burst out during processing.

There are only minor differences between the architectures for real and complex matrices. For the complex matrix, both the input and processing memory blocks contain complex values. Similarly, all values going into the vector section are complex numbers. The complex conjugate of the latched register is obtained by simply inverting the sign bit. As for the root section, the structure is simplified by the nature of the positive definite matrix. The diagonal value, which is the first value at the top of each column in the decomposition, is always a real number so that the result from the inverse square root calculation is always a real number. The complex multiplier in the root section is therefore a real scalar, so only two real multipliers are required.

Triangular Matrix Inversion

The triangular matrix, L, obtained from the Cholesky decomposition function is computed using the triangular matrix inversion algorithm to get its inversion. The following MatLab pseudo code shows how the inversion is carried out:

```
for j = n:-1:1,
X(j,j) = 1/L(j, j);
for k = j+1:n
for i = j+1:n
X(k, j) = X(k, j) + X(k, i)*L(i, j);
end;
end;
for k = j+1:n
X(k, j) = -X(j, j)*X(k, j);
end;
```

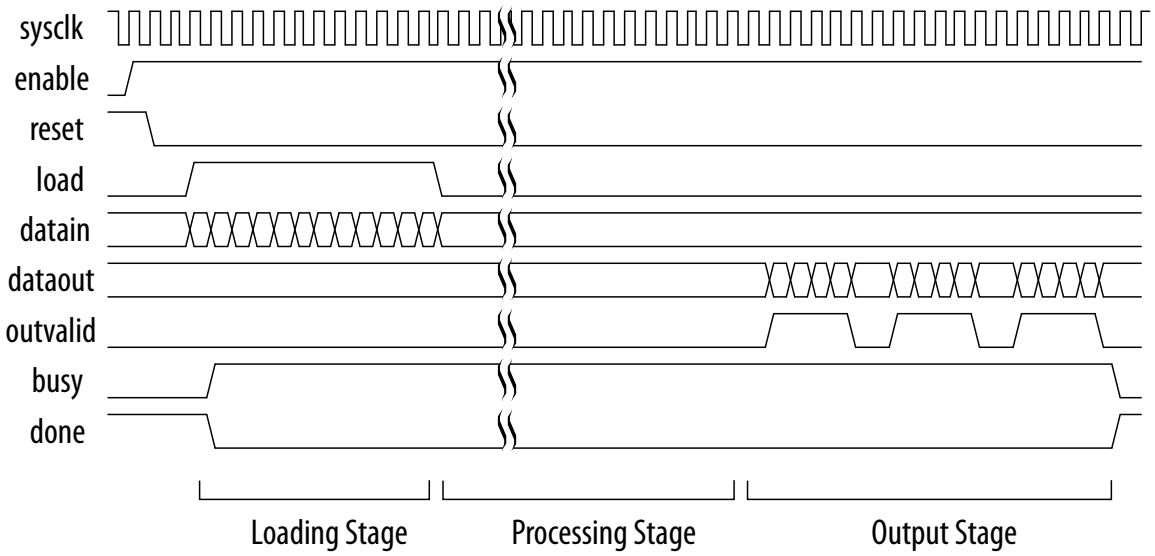
The pseudo code is converted into an RTL file. The result, L-1 is stored in the input matrix storage in the Cholesky decomposition function.

Matrix Multiplication

The final stage of the matrix inversion process involves multiplying the transpose of the inverse triangular matrix with the inverse triangular matrix using the Altera Floating-Point Matrix Multiplier. The original version of the matrix multiplier is modified for this purpose. As there are memory blocks already available for the storage of the input matrices in the Cholesky decomposition function, the memory blocks in the matrix multiplier are redundant and can be removed. Data is instead fed directly from the results stored at the end stage of the triangular matrix inversion algorithm.

Matrix Inversion Operation

Figure 2-3: Matrix Inversion Timing Diagram



The following sequence describes the matrix inversion operation:

1. The operation begins when the `enable` signal is asserted and the `reset` signal is deasserted.
2. The `load` signal is asserted to load data from the `loaddata[]` port for the input matrix. As long as the `load` signal is high, data is loaded continuously for the input matrix.
3. The `busy` signal is asserted and the done signal is deasserted for a few clock cycles after the `datain[]` signal is asserted.
4. The `outvalid` signal is asserted multiple times to signify the availability of valid data on the `dataout[]` port. The number of times this signal is asserted equals the number of rows found in the output matrix.
5. The `busy` and `done` signals are asserted when the last row of the output matrix has been burst out. This assertion signifies the end of the matrix inversion operation on the first set of data.

ALTERA_FP_MATRIX_INV Design Example: Matrix Inverse of Single-Precision Format Numbers

This design example uses the ALTERA_FP_MATRIX_INV IP core to show the matrix inversion operation. The input matrix applied is an 8 × 8 matrix with a block size of 2. This example uses the parameter editor GUI to define the core.

Related Information

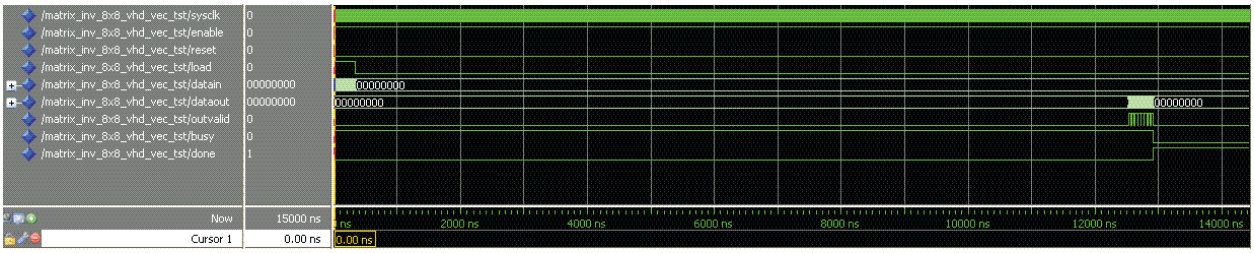
- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTERA_FP_MATRIX_INV Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

Figure 2-4: ALTERA_FP_MATRIX_INV ModelSim Simulation Waveform (Input Data)

This figure shows the expected simulation results in the ModelSim-Altera software.



This design example implements a floating-point matrix inversion to calculate the inverse value of matrices in single-precision formats. The optional input ports (`enable` and `reset`) are enabled.

Table 2-2: Summary of Input Values and Corresponding Outputs

This table lists the inputs and corresponding outputs obtained from the simulation waveform. The number of clock cycles obtained for each stage is based on the particular matrix size and parameter settings used in this design example.

Time	Event
0 ns – 10 ns	Start sequence: • The `reset` signal deasserts. • The `enable` signal asserts.
19.86 ns – 340 ns	Matrix input data load: • The `load` signal asserts and remains high for 80 clock cycles. • As long as the `load` signal is high, data for the input matrix is loaded row by row. • Input data is burst in regularly, one at every clock cycle. • The `load` signal deasserts at 340 ns. The deassertion of the `load` signal signifies the completion of the data load operation for the matrix.
27.5 ns	Processing stage: • The `busy` signal asserts while the `done` signal deasserts. • The assertion of the `busy` signal and the deassertion of the `done` signal indicate that the matrix inversion core is processing the input data. • There are about 2500 clock cycles between the beginning of the processing stage and the first available output value.

Time	Event
12527.5 – 12922.5 ns	Output stage: • The `outvalid` signal asserts in intervals of 8 clock cycles. These series of assertions signify the availability of valid data for the output matrix on the `outdata[]` port. • The output is an 8 x 8 matrix. Data is burst out regularly, row by row. • At 12922.5 ns, the `busy` signal is asserted and the `done` signal is deasserted. • The assertion of the `busy` signal and the deassertion of the `done` signal indicate that the final output is written and a new matrix can be processed.

Sample Matrix Data

This section shows the random test data assigned to the input matrices and the results obtained from the matrix inversion operation.

The following two sets of results are computed:

- PC-based results—these are results obtained from running the simulation in Matlab.
- FPGA-based results—these are results obtained from running the simulation in ModelSim.

This table lists the input and output data values presented in IEEE-754 Floating-point format.

Table 2-3: Input and Output Data

Matrix	Data
Input Matrix	40c89c6c 40b16187 40e21dfb 40847306 40c00d1d 40bbf0c4 40be4fc1 40953a30 40b16187 41244acb 410e61b9 40defe3a 40f8e982 40eff916 410e0ff4 41121d78 40e21dfb 410e61b9 41217d87 40d7f5f4 40fd78fa 410618c0 41060327 40ff4517 40847306 40defe3a 40d7f5f4 40b10427 40b6be88 40bbff4a 40d12685 40ca69f9 40c00d1d 40f8e982 40fd78fa 40b6be88 41146829 40ee188a 40fa2d80 40cf065c 40bbf0c4 40eff916 410618c0 40bbff4a 40ee188a 40ecbddf 40e3aa3a 40d60773 40be4fc1 410e0ff4 41060327 40d12685 40fa2d80 40e3aa3a 4111ed09 40ecd83c 40953a30 41121d78 40ff4517 40ca69f9 40cf065c 40d60773 40ecd83c 410847da

Matrix	Data
PC-based Output Matrix	42148e03 42f5794f 421b33f4 430e0587 41ff0d66 c2f579a3 c2df1c28 c2f945bc 42f5794f 43d60be5 430944db 43f2dd63 42da2dd0 c3d1dd59 c3bff960 c3d98c47 421b33f4 430944db 424b067c 43204d17 421907da c3107054 c2fc035b c30d24b3 430e0587 43f2dd63 43204d17 440cc66b 43002bbb c3f4e779 c3dcd667 c3f7e3f3 41ff0d66 42da2dd0 421907da 43002bbb 41f5048b c2e44480 c2c91e6d c2df60c9 c2f579a3 c3d1dd59 c3107054 c3f4e779 c2e44480 43d89b61 43c003b9 43d685d3 c2df1c28 c3bff960 c2fc035b c3dcd667 c2c91e6d 43c003b9 43ae19b0 43c37f99 c2f945bc c3d98c47 c30d24b3 c3f7e3f3 c2df60c9 43d685d3 43c37f99 43ddb1bc
FPGA-based Output Matrix	42148d06 42f5773e 421b32c4 430e0484 41ff0bb7 c2f577f4 c2df1a71 c2f943b1 42f5773e 43d609cf 430943a0 43f2db4a 42da2c09 c3d1db95 c3bff79e c3d98a34 421b32c4 430943a0 424b0515 43204be2 421906da c3106f53 c2fc014f c30d237c 430e0484 43f2db4a 43204be2 440cc563 43002adf c3f4e5c0 c3dcd4a7 c3f7e1df 41ff0bb7 42da2c09 421906da 43002adf 41f50322 c2e44314 c2c91cf5 c2df5f08 c2f577f4 c3d1db95 c3106f53 c3f4e5c0 c2e44314 43d899f3 43c00242 43d68414 c2df1a71 c3bff79e c2fc014f c3dcd4a7 c2c91cf5 43c00242 43ae1837 43c37dda c2f943b1 c3d98a34 c30d237c c3f7e1df c2df5f08 43d68414 43c37dda 43ddafad

The difference between each result element of the PC-based and FPGA-based output matrices are as shown:

Result differences (in decimal)

253 529 304 259 431 431 439 523

529 534 315 537 455 452 450 531

304 315 359 309 256 257 524 311

259 537 309 264 220 441 448 532

431 455 256 220 361 364 376 449

431 452 257 441 364 366 375 447

439 450 524 448 376 375 377 447

523 531 311 532 449 447 447 527

The difference between the two output matrices are due to the following reasons:

- Method of processing—Matlab uses sequential processing while Modelsim uses parallel processing.
- Method of conversion—Matlab first computes in double-precision format, and then only converts the result into single-precision format. During this conversion, some units in the last place (ulp) are expected to be lost.

ALTERA_FP_MATRIX_INV Signals

Figure 2-5: ALTERA_FP_MATRIX_INV Signals

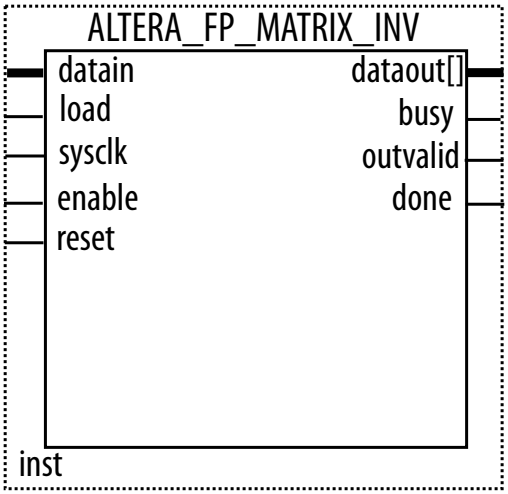


Table 2-4: ALTERA_FP_MATRIX_INV Input Signals

Port Name	Required	Description
`sysclk`	Yes	The clock input to the ALTERA_FP_MATRIX_INV IP core. This is the main system clock. All operations occur on the rising edge.
`enable`	No	Optional port. Allow calculation to take place when asserted. When deasserted, no operation will take place and the outputs are unchanged.
`reset`	No	Optional port. The core resets asynchronously when the `reset` signal is asserted.
`load`	Yes	When asserted, loads the `LOADDATA` bus into the memory.
`loaddata`	Yes	Single-precision 32-bit matrix input value. Matrices load row by row.

Table 2-5: ALTERA_FP_MATRIX_INV Output Signals

Port Name	Required	Description
`ready`	Yes	When asserted, the core preprocesses the input data. The calculate signal cannot be asserted until the `ready` signal is low.
`outdata`	Yes	Single-precision 32-bit matrix result value. The matrix result value is written out row by row.

Port Name	Required	Description
`outvalid`	Yes	When asserted, a valid output data is available. An entire row of the result matrix is written out as a burst. There is a gap between row outputs, which will depend on the parameters.
`done`	Yes	When asserted, the last output has been written. A new matrix multiply can be started with `calculate`. `done` will follow `ready` by some fixed amount, depending on the parameters.

ALTERA_FP_MATRIX_INV Parameters

Table 2-6: ALTERA_FP_MATRIX_INV Parameters

Port Name	Type	Required	Description
`BLOCKS`	Integer	No	The number of memory blocks for the double-buffered storage of matrix multiplication. The allowable range is from `2` to `16`.
`DIMENSION`	Integer	Yes	The number of rows in the matrix. As the matrix is square, this is also the number of columns in the matrix. The supported dimensions are 4 x 4, 6 x 6, 8 x 8, 16 x 16, 32 x 32, and 64 x 64. The maximum supported input dimension is 64 × 64. This parameter also acts as the VECTORSIZE when calling the ALTERA_FP_MATRIX_MULT IP core internally.
`WIDTH_EXP`	Integer	Yes	Specifies the precision of the exponent. The bias of the exponent is always set to `2`(WIDTH_EXP-1) `-1` (that is, `127` for single-precision format). `WIDTH_EXP` must be `8` for single-precision format and must be less than `WIDTH_MAN`. The available value for `WIDTH_EXP` is `8`.
`WIDTH_MAN`	Integer	Yes	Specifies the precision of the mantissa. `WIDTH_MAN` must be 23 when `WIDTH_EXP` is 8. Otherwise, `WIDTH_MAN` must be a minimum of 31. `WIDTH_MAN` must be greater than `WIDTH_EXP`. The sum of `WIDTH_EXP` and `WIDTH_MAN` must be less than 64. Current available value for `WIDTH_MAN` is only 23 for single precision.

ALTERA_FP_MATRIX_MULT IP Core 3

2016.12.09

UG-01058

This IP core performs floating-point multiplication between two matrices.

ALTERA_FP_MATRIX_MULT Features

The ALTERA_FP_MATRIX_MULT IP core offers the following features:

- Multiplication of two matrices.
- Support for floating-point formats in single and double precisions.
- Support for configurable performance and resource usage.
- Avalon streaming interfaces and full QSys compliance.

ALTERA_FP_MATRIX_MULT Output Latency

The ALTERA_FP_MATRIX_MULT IP core does not have a fixed output latency. Instead, the IP core uses Avalon streaming interfaces and the `c_valid` signal on the output interface to indicate when output data is available.

ALTERA_FP_MATRIX_MULT Resource Utilization and Performance

These tables list the resource utilization and performance information for the ALTERA_FP_MATRIX_MULT IP core. The information was derived using the Quartus II software version 14.1.

Table 3-1: ALTERA_FP_MATRIX_MULT Resource Utilization and Performance for the Arria 10 and Stratix V Devices

Family	Data Format	Matrix A Size	Matrix B Size	Vector Size	Memory Blocks	ALMs	M20ks	DSP Blocks	FMax (MHz)	Latency (cycles) [1]
Arria 10 (10AX066H2F34I2LP)	Single	8x8	8x8	8	4	979	12	8	409	131
		16x16	16x16	8	4	1052	12	8	408	595
		32x32	32x32	16	8	1579	25	16	373	2155
		64x64	64x64	32	16	2677	49	32	379	8339
Stratix V (5SGXEA7K2F40C2)	Single	8x8	8x8	8	4	2637	14	8	404	125
		16x16	16x16	8	4	2868	15	8	367	588
		32x32	32x32	16	8	5427	27	16	356	2146
		64x64	64x64	32	16	10311	51	32	348	8328

ALTERA_FP_MATRIX_MULT Functional Description

The matrix multiplier in the ALTERA_FP_MATRIX_MULT IP core multiplies matrix A and matrix B to generate the output matrix C.

The following figure shows the equation:

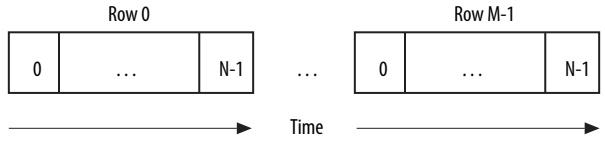
Figure 3-1: ALTERA_FP_MATRIX_MULT Equation

$$C = A \cdot B$$

The matrix A and B can be loaded when the ready signal on their respective interfaces are asserted. When the input matrices are loaded, the core will start computing the output. Valid signal on the output interface will be asserted to indicate valid output data. The input data may be loaded at any time the ready signal is asserted even when the previously loaded data is still being computed.

Figure 3-2: Matrix Serialization Format

An input matrix with M rows and N columns must be input as shown in this figure, where the Row 0 and Column 0 element is first and Row M-1 and Column N-1 element is last. The result matrix will be output in the same format.



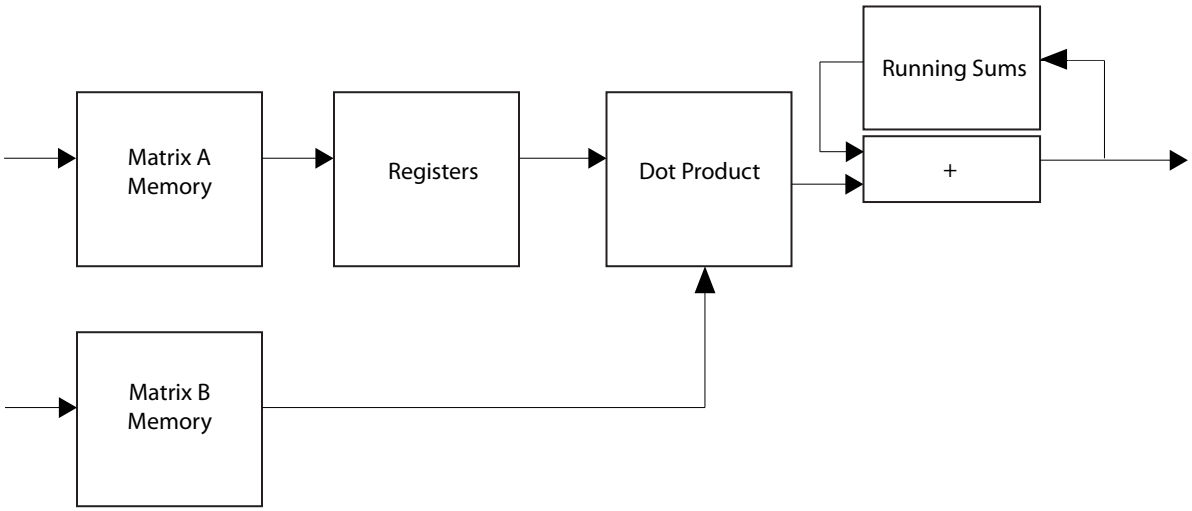
[1] Latency is the time take to compute a dot product and does not include the time taken to load the input matrices

The ALTERA_FP_MATRIX_MULT IP core consists of the following components:

- Memory blocks for the matrix A storage
- Memory blocks for the matrix B storage
- Dot product
- Accumulator

Figure 3-3: Top-Level View of the ALTERA_FP_MATRIX_MULT IP Core

This figure shows the top-level view of the ALTERA_FP_MATRIX_MULT IP core.



The following lists the key features of the architecture:

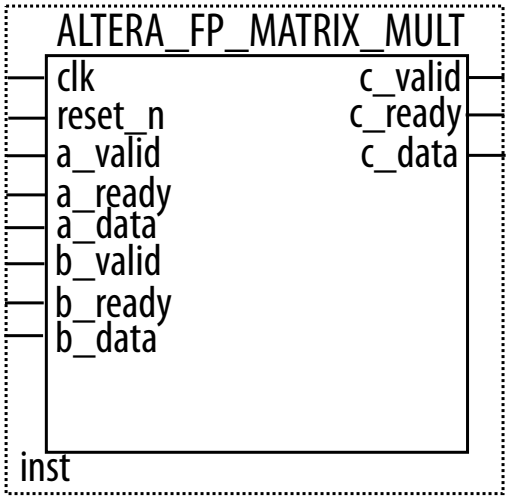
- Matrix A and B storage are double buffered to allow processing to happen in parallel with data loading.
- Where the number of columns of A (`A_COLUMNS`) and rows of B (same as `A_COLUMNS`) are greater than the size of the dot product (`VECTOR_SIZE`), the rows of A and columns of B are divided into sub rows and sub columns respectively, each containing `VECTOR_SIZE` elements. In this case, `A_COLUMNS/VECTOR_SIZE` iterations are needed to compute a full dot product corresponding to a single output element.
- Matrix B memory has sufficient bandwidth so that all the data needed for the dot product can be loaded at once.
- Matrix A memory is allocated with less bandwidth. The bandwidth of the matrix A is a parameter (`NUM_BLOCKS`) that you can control. A sub row of matrix A is loaded into local registers over a number of cycles before an iteration of the dot product. Once a sub row of Matrix A has been loaded into local registers, all partial dot products involving that sub row are computed before another sub row is loaded.
- For Arria 10 devices, where hardened single precision floating-point DSP blocks exist, those will be used for single precision floating point arithmetic.

The matrix multiply architecture is not optimized for sparse matrices and constant matrices.

ALTERA_FP_MATRIX_MULT Signals

Figure 3-4: ALTERA_FP_MATRIX_MULT Signals

This figure shows the signals for the ALTERA_FP_MATRIX_MULT IP core.



These tables list the signals for the ALTERA_FP_MATRIX_MULT IP core.

Table 3-2: ALTERA_FP_MATRIX_MULT Input Signals

Port Name	Required	Description
`clk`	Yes	The clock input port for the IP core.
`reset_n`	No	Asynchronous active low reset port.
`a_data`	Yes	Matrix A input data.
`a_valid`	Yes	Matrix A Avalon streaming valid signal. When this signal is asserted, data on `a_data` is valid.
`b_data`	Yes	Matrix B input data.
`b_valid`	Yes	Matrix B Avalon streaming valid signal. When this signal is asserted, data on `b_data` is valid.
`c_ready`	Yes	Matrix C Avalon streaming ready signal. Ready latency is `0`.

Table 3-3: ALTERA_FP_MATRIX_MULT Output Signals

Port Name	Required	Description
`a_ready`	Yes	Matrix A Avalon streaming ready signal. Ready latency is `0`.
`b_ready`	Yes	Matrix B Avalon streaming ready signal. Ready latency is `0`.
`c_data`	Yes	Matrix C input data.
`c_valid`	Yes	Matrix C Avalon streaming valid signal. When this signal is asserted, data on `c_data` is valid.

ALTERA_FP_MATRIX_MULT Parameters

This table lists the parameters for the ALTERA_FP_MATRIX_MULT IP core.

Table 3-4: ALTERA_FP_MATRIX_MULT IP Core Parameters

Parameter	Value	Description
Format	Single (32 bit) or Double (64 bit)	The format of the input data.
Rows in Matrix A	2-256	Number of rows in matrix A.
Columns in Matrix A	8-256 Integer multiples of vector size. (Integer multiples of Memory Blocks.)	Number of columns in matrix A. This is also the number of rows in matrix B.
Rows in Matrix B	8-256	Number of rows in matrix B.
Columns of matrix B	2-256	Number of columns in matrix B.
Vector Size	Allowed values are 8,16, 32, 64, 96, and 128.	The size of the dot product which can be computed in parallel. Where the number of columns of matrix A and rows of matrix B are greater than Vector Size a number of iterations are required to compute a full dot product. Vector Size also controls the matrix B memory configuration. Increasing the "Vector Size" increases the matrix B memory bandwidth and the number of memory blocks used.
Memory Blocks	The Vector Size must be an integer multiple of Memory Blocks. The number of memory blocks must be smaller than the vector size. The number of memory blocks must be greater than or equals to the ratio of vector size divided by the number of columns of matrix B.	Controls the memory configuration of the matrix A storage. Increasing this number increases the memory bandwidth and the number of memory blocks used.

ALTERA_FP_ACC_CUSTOM IP Core 4

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs floating-point accumulation and allows you to restrict the range of inputs and maximum accumulated value to save resources. The core uses device latency models to generate RTL to meet a target FMax at the cost of latency.

ALTERA_FP_ACC_CUSTOM Features

The ALTERA_FP_ACC_CUSTOM IP core offers the following features:

- Supports frequency driven cores.
- Supports VHDL RTL generation.
- Supports customization of the required range of the input and output values.

ALTERA_FP_ACC_CUSTOM Output Latency

The amount of latency is driven by the target frequency and the selected device family. You must set the desired frequency and the target device before generating the IP core. The IP core reports the latency when you set the parameters and when you generate the IP core. Then, use the reported latency to incorporate the IP core into your design.

ALTERA_FP_ACC_CUSTOM Resource Utilization and Performance

Table 4-1: ALTERA_FP_ACC_CUSTOM Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTERA_FP_ACC_CUSTOM IP core. The information was derived using the Quartus II software version 13.1.

Device Family	Input Data		Accumulator Size		Target Frequency (MHz)	Latency	ALMs	DSP Blocks	Logic Registers		M10K	M20K	f_{MAX}
	Floating Point Format	MaxMSBX	MSBA	LSBA					Primary	Secondary			
Arria V (5AGXFB3H4F40C5)	Double	24	40	-52	270	15	866	0	1,166	106	0	--	265
Cyclone V (5CGXFC7D6F31C7)	Double	24	40	-52	230	15	830	0	1,102	32	0	--	198
Stratix V (5SGXEA7K2F40C2)	Double	24	40	-52	400	15	968	0	1,655	27	--	0	426
Arria V (5AGXFB3H4F40C5)	Single	12	20	-26	270	12	337	0	588	52	0	--	309
Cyclone V (5CGXFC7D6F31C7)	Single	12	20	-26	230	12	383	0	494	28	0	--	225
Stratix V (5SGXEA7K2F40C2)	Single	12	20	-26	400	13	475	0	903	20	--	0	450

Related Information

Fitter Resources Reports

Provides information about Quartus II resource utilization

ALTERA_FP_ACC_CUSTOM Signals

Figure 4-1: ALTERA_FP_ACC_CUSTOM

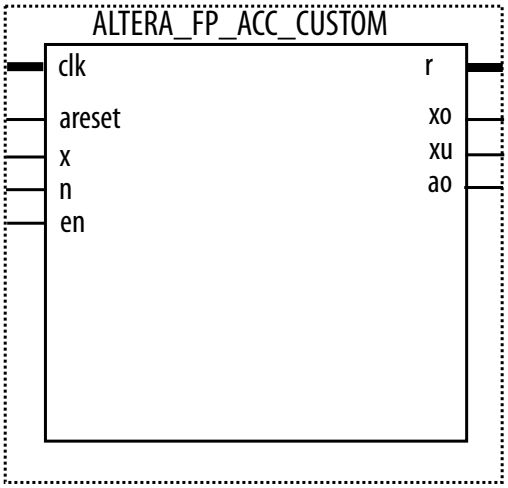


Table 4-2: ALTERA_FP_ACC_CUSTOM Input Ports

Port Name	Required	Description
`clk`	Yes	All input signals, otherwise explicitly stated, must be synchronous to this clock
`areset`	Yes	Asynchronous active-high reset. Deassert this signal synchronously to the input clock to avoid metastability issues.
`en`	No	Global enable signal. This port is optional.
`x`	Yes	Data input port.
`n`	Yes	Boolean port which signals the beginning of a new data set to be accumulated. This should go high together with the first element in the new data set and should go low the next cycle. The data sets may be of variable length and a new data set may be started at any time. The accumulation result for an input will be available after the reported latency.

Table 4-3: ALTERA_FP_ACC_CUSTOM Output Ports

Port Name	Required	Description
`r`	Yes	The running value of the accumulation.

Port Name	Required	Description
xo	Yes	The overflow flag for port x. The signal goes high when the exponent of the input x is larger than maxMSBX. The signal remains high for the entire data set. This flag invalidates port r. You should consider increasing maxMSBX. This flag also indicate infinity and NaN.
xu	Yes	The underflow flag for port x. The signal goes high when the exponent of the input x is smaller than LSBA. The signal remains high for the entire data set. This flag does not invalidate port r. You should consider lowering LSBA.
ao	Yes	The overflow flag for Accumulator. The signal goes high when the exponent of the accumulated value is larger than MSBA. The signal remains high for the entire data set. This flag invalidates port r. You should consider increasing MSBA.

ALTERA_FP_ACC_CUSTOM Parameters

Table 4-4: ALTERA_FP_ACC_CUSTOM Parameters

Category	Parameter	Values	Description
Input Data	Floating point format	**single**, **double**	Choose the floating point format of the input data values. The output data values of the accumulator is in the same format. The default is **single**.
	maxMSBX	—	The maximum weight of the MSB of an input. For example, when adding probabilities in the 0 to 1 range set this weight to ceil($\log_2(1)$)=0. The xo output signal goes high when the MSB of an input value has a weight larger than maxMSBX. The result of the accumulation is then invalid. If you are unsure about the range of the inputs, then set the **maxMSBX** parameter to MSBA, at the possible expense of increased resource usage. The default value is **12**.

Category	Parameter	Values	Description
Accumulator Size	MSBA	—	The weight of the MSB of the accumulator. For example, in a financial simulation, if the value of a stock cannot exceed 100,000 dollars, use a value of ceil($\log_2$(100000))=17. In a circuit simulation where the circuit adds numbers in the 0 to 1 range, for one year, at 400 MHz, use a value of ceil($\log_2$(365 x 60 x 60 x 24 x 400 x 10^6))=54. The ao output signal goes high when the MSB of the accumulated value has a weight larger than MSBA. The result of the accumulation is then invalid. Altera recommends adding a few guard bits to avoid possible accumulator overflow. A few guard bits have little impact on the accumulator size. The default value is **20**.
	LSBA	—	The weight of the LSB of the accumulator and the accuracy of the accumulator. Because an N term accumulation can invalidate the $\log_2$(N) LSBs of the accumulator, you must consider the length of the accumulation and the range of the inputs when setting this parameter. For example, if a 2^{-30} accuracy is required over an accumulation of 1024 numbers, then set the LSBA to: (-30 - $\log_2$(1024)) = -40. Any input $2^e \times 1.F$, where F is the mantissa and e is less than the LSBA will be shifted out of the accumulator. The au output signal goes high to indicate this situation. The default value is **-26**.
Required Performance	Target frequency	Any positive integer value.	Choose the frequency in MHz at which this core is expected to run. This together with the target device family will determine the amount of pipelining in the core. The default value is **200** MHz.
Optional	Generate an enable port	—	Choose if the accumulator should have an enable signal. This parameter is disabled by default.

Category	Parameter	Values	Description
Report	—	—	Reports the latency of the device, which is the number of cycles it takes for an accumulation to propagate through the block from input to output.

ALTFP_ADD_SUB IP Core 5

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core allows you to perform floating-point addition or subtraction between two inputs dynamically.

ALTFP_ADD_SUB Features

The ALTFP_ADD_SUB IP core offers the following features:

- Dynamically configurable adder and subtracter functions.
- Optional exception handling output ports such as `zero`, `overflow`, `underflow`, and `nan`.
- Optimization of speed and area.

ALTFP_ADD_SUB Output Latency

The output latency options for the ALTFP_ADD_SUB IP core are the same for all three precision formats —single, double, and single-extended. The options available are 7, 8, 9, 10, 11, 12, 13, and 14 clock cycles.

ALTFP_ADD_SUB Truth Table

Table 5-1: Truth Table for Addition/Subtraction Operations

DATAA[]	DATAB[]	SIGN BIT	RESULT[]	Overflow	Underflow	Zero	NaN
Normal	Normal	0	Zero	0	0	1	0
Normal	Normal	0/1	Normal	0	0	0	0
Normal	Normal	0/1	Denormal	0	1	1	0
Normal	Normal	0/1	Infinity	1	0	0	0
Normal	Denormal	0/1	Normal	0	0	0	0
Normal	Zero	0/1	Normal	0	0	0	0
Normal	Infinity	0/1	Infinity	1	0	0	0
Normal	NaN	X	NaN	0	0	0	1

DATAA[]	DATAB[]	SIGN BIT	RESULT[]	Overflow	Underflow	Zero	NaN
Denormal	Normal	0/1	Normal	0	0	0	0
Denormal	Denormal	0/1	Normal	0	0	0	0
Denormal	Zero	0/1	Zero	0	0	1	0
Denormal	Infinity	0/1	Infinity	1	0	0	0
Denormal	NaN	X	NaN	0	0	0	1
Zero	Normal	0/1	Normal	0	0	0	0
Zero	Denormal	0/1	Zero	0	0	1	0
Zero	Zero	0/1	Zero	0	0	1	0
Zero	Infinity	0/1	Infinity	1	0	0	0
Zero	NaN	X	NaN	0	0	0	1
Infinity	Normal	0/1	Infinity	1	0	0	0
Infinity	Denormal	0/1	Infinity	1	0	0	0
Infinity	Zero	0/1	Infinity	1	0	0	0
Infinity	Infinity	0/1	Infinity	1	0	0	0
Infinity	NaN	X	NaN	0	0	0	1
NaN	Normal	X	NaN	0	0	0	1
NaN	Denormal	X	NaN	0	0	0	1
NaN	Zero	X	NaN	0	0	0	1
NaN	Infinity	X	NaN	0	0	0	1
NaN	NaN	X	NaN	0	0	0	1

ALTFP_ADD_SUB Resource Utilization and Performance

The following lists the resource utilization and performance information for the ALTFP_ADD_SUB IP core. The information was derived using the Quartus II software version 10.0.

Table 5-2: ALTFP_ADD_SUB Resource Utilization and Performance for the Stratix Series of Devices

Device Family	Precision	Optimization	Output latency	Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	f_{MAX} (MHz)
Stratix IV	single	speed	7	594	376	385	228
			14	674	686	498	495
		area	7	576	345	375	227
			14	596	603	421	484
	double	speed	7	1,198	687	824	187
			14	997	1,607	1,080	398
		area	7	1,106	630	762	189
			14	904	1,518	1,013	265

ALTFP_ADD_SUB Design Example: Addition of Double-Precision Format Numbers

This design example uses the ALTFP_ADD_SUB IP core to perform the addition of double-precision format numbers using the parameter editor in the Quartus II software.

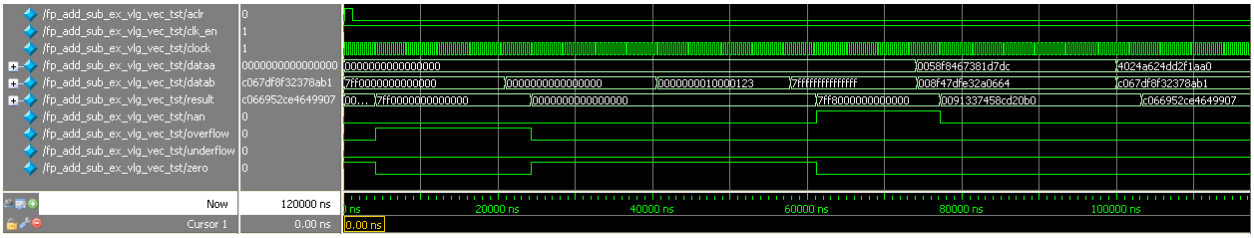
Related Information

- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_ADD_SUM Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

Figure 5-1: ALTFP_ADD_SUB Simulation Waveform



This design example implements a floating-point adder for the addition of double-precision format numbers. All the optional input ports (`clk_en` and `aclr`) and optional output ports (`overflow`, `underflow`, `zero`, and `nan`) are enabled.

In this example, the output latency of the multiplier is set to 7 clock cycles. Every addition result appears at the `result[]` port 7 clock cycles after the input values are captured on the `dataa[]` and `datab[]` ports.

The following lists the inputs and corresponding outputs obtained from the simulation waveform.

Table 5-3: Summary of Input Values and Corresponding Outputs

Time	Event
0 ns, start-up	`dataa[]` value: 0000 0000 0000 0000h `datab[]` value: 7FF0 0000 0000 0000h Output value: All values seen on the output port before the 7th clock cycle are merely due to the behavior of the system during startup and should be disregarded.
4250 ns	Output value: 7FF0 0000 0000 0000h Exception handling ports: `overflow` asserts The addition of zero at the input port `dataa[]`, and infinity value at the input port `datab[]` results in infinity value.
40,511 ns	`dataa[]` value: 0000 0000 0000 0000h `datab[]` value: 0000 0000 1000 0123h The is the addition of a zero and a denormal value.
43,750 ns	Output value: 0000 0000 0000 0000h Exception handling ports: `zero` remains asserted. Denormal inputs are not supported and are forced to zero before addition takes place.This results in a zero.

ALTFP_ADD_SUB Signals

Figure 5-2: ALTFP_ADD_SUB

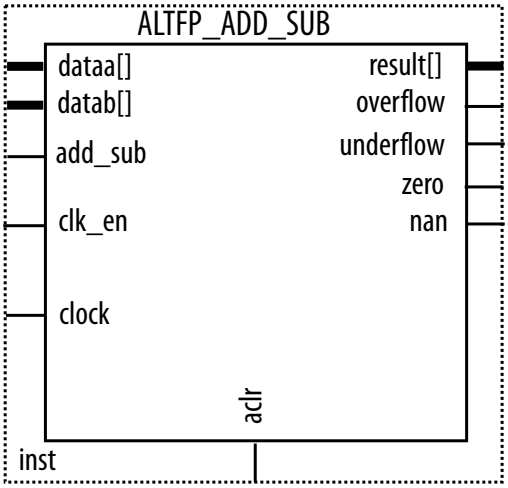


Table 5-4: ALTFP_ADD_SUB Input Ports

Port Name	Required	Description
`aclr`	No	Asynchronous clear input for floating-point adder or subtractor. The source is asynchronously reset when the `aclr` signal is asserted high.
`add_sub`	No	Optional input port to enable dynamic switching between the adder and subtractor functions. The `add_sub` port must be used when the `DIRECTION` parameter is set to `VARIABLE`. When the `add_sub` port is high, `result[] = dataa[] + datab[]`, otherwise, `result[] = dataa[] - datab[]`.
`clk_en`	No	Clock enable to the floating-point adder or subtractor. This port allows addition or subtraction to occur when asserted high. When asserted low, no operations occur and the outputs are unchanged.
`clock`	Yes	Clock input to the IP core.
`dataa[]`	Yes	Data input to the floating-point adder or subtractor. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa bits. The size of this port is the total width of the sign bit, the exponent bits, and the mantissa bits.
`datab[]`	Yes	Data input to the floating-point adder or subtractor. This port is configured in the same way as `dataa[]`.

Table 5-5: ALTFP_ADD_SUB Output Ports

Port Name	Required	Description
`nan`	Yes	NaN exception output. Asserted when an illegal addition or subtraction occurs, such as infinity minus infinity. When an invalid addition or subtraction occurs, a NaN value is output to the `result[]` port. Any adding or subtracting involving NaN values also produces a NaN value.

Port Name	Required	Description
`overflow`	Yes	Overflow exception port. Asserted when the result of the addition or subtraction, after rounding, exceeds or reaches infinity. Infinity is defined as a number in which the exponent exceeds $2^{WIDTH\_EXP}-1$.
`result[]`	Yes	Floating-point output result. Like the input values, the MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits, and mantissa bits.
`underflow`	Yes	Underflow port for the adder or subtractor. Asserted when the result of the addition or subtraction, after rounding, the value is zero and the inputs are not equal. The `underflow` port is also asserted when the result is a denormalized number.
`zero`	No	Zero port for the adder or subtractor. Asserted when the `result[]` port is zero.

ALTFP_ADD_SUB Parameters

Table 5-6: ALTFP_ADD_SUB Parameters

Parameter Name	Type	Required	Description
`DIRECTION`	String	Yes	Specifies addition or subtraction operations. Values are `ADD`, `SUB`, or `VARIABLE`. If this parameter is not specified, the default is `ADD`. When the value is `VARIABLE`, the `add_sub` port determines whether the operation is addition or subtraction. The `add_sub` port must be connected if the `DIRECTION` parameter is set to `VARIABLE`. If the value is `ADD` or `SUB`, the `add_sub` port is ignored.
`PIPELINE`	Integer	No	Specifies the latency in clock cycles used in the ALTFP_ADD_SUB IP core. The `PIPELINE` parameter supports values of `7` through `14`. If this parameter is not specified, the default value is `11`. In general, a higher pipeline value produces better f_{MAX} performance.
`ROUNDING`	String	Yes	Specifies the rounding mode. The default value is `TO_NEAREST`. Other rounding modes are currently not supported.
`OPTIMIZE`	String	No	Defines the design preference, whether the design is optimized for speed (faster f_{MAX}), or optimized for area (lower resource count). Values are `SPEED` and `AREA`. If this parameter is not specified, the default is `SPEED`.

Parameter Name	Type	Required	Description
WIDTH_EXP	Integer	No	Specifies the precision of the exponent. The bias of the exponent is always set to 2 (WIDTH_EXP-1) -1 (that is, 127 for single-precision format and 1023 for double-precision format). The WIDTH_EXP parameter must be 8 for the single-precision mode and 11 for the double-precision mode, or a minimum of 11 for the single-extended precision mode. The WIDTH_EXP parameter must be less than the WIDTH_MAN parameter. The sum of WIDTH_EXP and the WIDTH_MAN parameters must be less than 64. If this parameter is not specified, the default is 8.
WIDTH_MAN	Integer	No	Specifies the precision of the mantissa. The WIDTH_MAN parameter must be 23 (to comply with the IEEE-754 standard for the single-precision mode) when the WIDTH_EXP parameter is 8. Otherwise, the WIDTH_MAN parameter must have a value that is greater than or equal to 31. The WIDTH_MAN parameter must be greater than the WIDTH_EXP parameter. The sum of the WIDTH_EXP and WIDTH_MAN parameters must be less than 64. If this parameter is not specified, the default is 23.

ALTFP_DIV IP Core 6

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs floating-point division operation.

ALTFP_DIV Features

The ALTFP_DIV IP core offers the following features:

- Division functions.
- Optional exception handling output ports such as `zero`, `division_by_zero`, `overflow`, `underflow`, and nan.
- Optimization of speed and area.
- Low latency option.

ALTFP_DIV Output Latency

The output latency options for the ALTFP_DIV IP core differs depending on the precision selected, the width of the mantissa, or both. You have the choice of selecting the smaller figures of clock cycles delay in your design if the low latency option is desired.

Table 6-1: Latency Options for Each Operation

Precision	Mantissa Width	Latency (in clock cycles)
Single	23	6, 14, 33
Double	52	10, 24, 61

Precision	Mantissa Width	Latency (in clock cycles)
Single Extended	31 – 32	8, 18, 41
	33 – 34	8, 18, 43
	35 – 36	8, 18, 45
	37 – 38	8, 18, 47
	39 – 40	8, 18, 49
	41	10, 24, 41
	42	10, 24, 51
	43 – 44	10, 24, 53
	45 – 46	10, 24, 55
	47 – 48	10, 24, 57
	49 – 50	10, 24, 59
	51 – 52	10, 24, 61

ALTFP_DIV Truth Table

Table 6-2: Truth Table for Division Operations

DATAA[]	DATAB[]	SIGN BIT	RESULT[]	Overflow	Underflow	Zero	Division-by-zero	NaN
Normal	Normal	0/1	Normal	0	0	0	0	0
Normal	Normal	0/1	Denormal	0	0	1	0	0
Normal	Normal	0/1	Infinity	1	0	0	0	0
Normal	Normal	0/1	Zero	0	1	1	0	0
Normal	Denormal	0/1	Infinity	0	0	0	1	0
Normal	Zero	0/1	Infinity	0	0	0	1	0
Normal	Infinity	0/1	Zero	0	0	1	0	0
Normal	NaN	X	NaN	0	0	0	0	1
Denormal	Normal	0/1	Zero	0	0	1	0	0
Denormal	Denormal	0/1	NaN	0	0	0	0	1
Denormal	Zero	0/1	NaN	0	0	0	0	1

DATAA[]	DATAB[]	SIGN BIT	RESULT[]	Overflow	Underflow	Zero	Division-by-zero	NaN
Denormal	Infinity	0/1	Zero	0	0	1	0	0
Denormal	NaN	X	NaN	0	0	0	0	1
Zero	Normal	0/1	Zero	0	0	1	0	0
Zero	Denormal	0/1	NaN	0	0	0	0	1
Zero	Zero	0/1	NaN	0	0	0	0	1
Zero	Infinity	0/1	Zero	0	0	1	0	0
Zero	NaN	X	NaN	0	0	0	0	1
Infinity	Normal	0/1	Infinity	0	0	0	0	0
Infinity	Denormal	0/1	Infinity	0	0	0	0	0
Infinity	Zero	0/1	Infinity	0	0	0	0	0
Infinity	Infinity	0/1	NaN	0	0	0	0	1
Infinity	NaN	X	NaN	0	0	0	0	1
NaN	Normal	X	NaN	0	0	0	0	1
NaN	Denormal	X	NaN	0	0	0	1	1
NaN	Zero	X	NaN	0	0	0	1	1
NaN	Infinity	X	NaN	0	0	0	0	1
NaN	NaN	X	NaN	0	0	0	0	1

ALTFP_DIV Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_DIV IP core. The information was derived using the Quartus II software version 10.0.

Table 6-3: ALTFP_DIV Resource Utilization and Performance for Stratix IV Devices

Device family	Precision	Optimization	Output latency	Logic Usage				f_{MAX}(MHz)
				Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-bit DSP	
Stratix IV	Single	Speed	33	3,593	3,351	2,500	—	313
		Area	33	1,646	2,074	1,441	—	308
	Double	Speed	61	13,867	13,143	10,196	—	292
		Area	61	5,125	7,360	4,842	—	267
Low Latency Option								
Stratix IV	Single	—	6	207	304	212	16	154
		—	14	253	638	385	16	358
	Double	—	10	714	1,077	779	44	151
		—	24	765	2,488	1,397	44	238

ALTFP_DIV Design Example: Division of Single-Precision

This design example uses the ALTFP_DIV IP core to implement a floating-point divider for the division of single-precision format numbers with low latency. This example uses the parameter editor to define the core.

Related Information

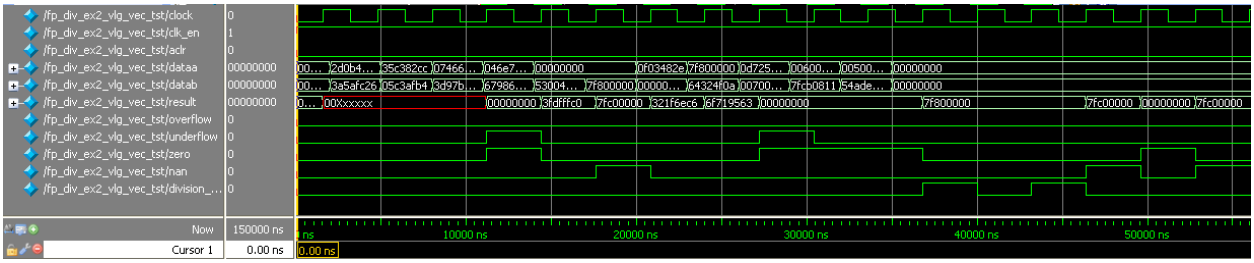
- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_DIV Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

Figure 6-1: ALTFP_DIV Simulation Waveform

This figure shows the expected simulation results in the ModelSim-Altera software.



This design example implements a floating-point divider for the division of single-precision numbers with a low latency option. The output latency is 6, hence every division generates the output result 6 clock cycles later.

Table 6-4: Summary of Input Values and Corresponding Outputs

This table lists the inputs and corresponding outputs obtained from the simulation in the waveform.

Time	Event
0 ns, start-up	`dataa[]` value: 0000 0000h `datab[]` value: 0000 0000h Output value: The undefined value is seen on the `result[]` port, which is ignored. All values seen on the output port before the 6th clock cycle are merely due to the behavior of the system during start-up and should be disregarded.
17600 ns	Output value: 7FC0 0000h Exception handling ports: nan asserts The division of zeros result in a NaN.
2000 ns	`dataa[]` value: 2D0B 496Ah `datab[]` value: 3A5A FC26h Both inputs hold normal values.
20800 ns	Output result: 321F 6EC6h Exception output ports: nan deasserts The division of two normal value results in a normal value.
11000 ns	`dataa[]` value: 046E 78BCh `datab[]` value: 6798 698Bh Both inputs hold normal values.

Time	Event
27200 ns	Output value: 0h Exception handling ports: `underflow` and `zero` asserts The division of the two normal values results in a denormal value. As denormal values are not supported, the result is zero and the `underflow` port asserts. The `zero` port is also asserted to indicate that the result is zero.
2600 ns	`dataa[]` value: 0D72 54A8h `datab[]` value: 0070 0000h The input port `dataa[]` holds a normal value while the input port `datab[]` holds a denormal value.
36800 ns	Output value: 7F80 0000h Exception handling ports: `division_by_zero` asserts Denormal numbers are forced-zero values, therefore, attempts to divide a normal value with a zero result in an infinity value.

ALTFP_DIV Signals

Figure 6-2: ALTFP_DIV Signals

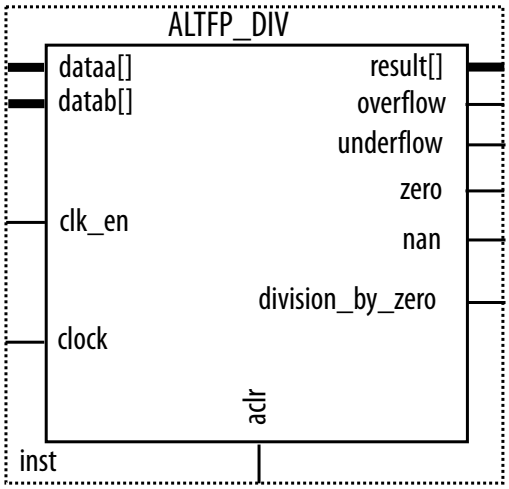


Table 6-5: ALTFP_DIV Input Signals

Port Name	Required	Description
`aclr`	No	Asynchronous clear input for the floating-point divider. The source is asynchronously reset when the `aclr` signal is asserted high.

Port Name	Required	Description
`clock`	Yes	Clock input to the IP core.
`clk_en`	No	Clock enable to the floating-point divider. This port enables division. This signal is active high. When this signal is low, no division takes place and the outputs remain the same.
`dataa[]`	Yes	Numerator data input. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits and mantissa bits.
`datab[]`	Yes	Denominator data input.The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits and mantissa bits.

Table 6-6: ALTFP_DIV Output Signals

Port Name	Required	Description
`result[]`	Yes	Divider output port. The division result (after rounding). As with the input values, the MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits, and mantissa bits.
`overflow`	No	Overflow port for the divider. Asserted when the result of the division (after rounding) exceeds or reaches infinity. Infinity is defined as a number in which the exponent exceeds 2WIDTH_EXP–1.
`underflow`	No	Underflow port for the divider. Asserted when the result of the division (after rounding) is zero even though neither of the inputs to the divider is zero, or when the result is a denormalized number.
`zero`	No	Zero port for the divider. Asserted when the value of `result[]` is zero.
`division_by_zero`	No	Division-by-zero output port for the divider. Asserted when the value of `datab[] is a zero.`
`nan`	No	NaN port. Asserted when an invalid division occurs, such as infinity dividing infinity or zero dividing zero. A NaN value appears as output at the `result[]` port. Any division of a NaN value causes the `nan` output port to be asserted.

ALTFP_DIV Parameters

Table 6-7: ALTFP_DIV Parameters

Parameter Name	Type	Required	Description
`WIDTH_EXP`	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is `8`. The bias of the exponent is always set to (2 ^ (`WIDTH_EXP` - 1)) - 1, that is, 127 for single precision and 1023 for double precision. The value of `WIDTH_EXP` must be `8` for single precision, `11` for double precision, and a minimum of 11 for single extended precision. The value of `WIDTH_EXP` must be less than the value of `WIDTH_MAN`, and the sum of `WIDTH_EXP` and `WIDTH_MAN` must be less than `64`.
`WIDTH_MAN`	Integer	Yes	Specifies the precision of the mantissa. If this parameter is not specified, the default is `23`. When `WIDTH_EXP` is `8` and the floating-point format is the single-precision format, the `WIDTH_MAN` value must be `23`. Otherwise, the value of `WIDTH_MAN` must be a minimum of `31`. The value of `WIDTH_MAN` must be greater than the value of `WIDTH_EXP`, and the sum of `WIDTH_EXP` and `WIDTH_MAN` must be less than `64`.
`ROUNDING`	String	Yes	Specifies the rounding mode. The default value is `TO_NEAREST`. The floating-point divider does not support other rounding modes.
`OPTIMIZE`	String	No	Specifies whether to optimize for area or for speed. Values are `AREA` and `SPEED`. A value of `AREA` optimizes the design using less total logic utilization or resources. A value of `SPEED` optimizes the design for better performance. If this parameter is not specified, the default value is `SPEED`.
`PIPELINE`	Integer	No	Specifies the number of clock cycles needed to produce the result. For the single-precision format, the latency options are `33`, `14` or `6`. For the double-precision format, the latency options are `61`, `24` or `10`. For the single-extended precision format, the value ranges from a minimum of `41` to a maximum of `61`. For the low-latency option, the latency is determined from the mantissa width. For a mantissa width of `31` to `40` bits, the value is `8` or `18`. For a mantissa width of `41` bits or more, the value is `10` or `24`.

ALTFP_MULT IP Core 7

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs floating-point multiplication operation.

ALTFP_MULT IP Core Features

The ALTFP_MULT IP core offers the following features:

- Multiplication functions.
- Optional exception handling output ports such as `zero`, `overflow`, `underflow`, and `nan`.
- Optional dedicated multiplier circuitries in Cyclone and Stratix series.

ALTFP_MULT Output Latency

The output latency options for the ALTFP_MULT IP core are similar for all precisions.

Table 7-1: Latency Options for Each Precision Format

Precision	Mantissa Width	Latency (in clock cycles)
Single	23	5, 6, 10,11
Double	52	5, 6, 10,11
Single-Extended	31–52	5, 6, 10,11

ALTFP_MULT Truth Table

Table 7-2: Truth Table for Multiplier Operations

DATAA[]	DATAB[]	RESULT[]	Overflow	Underflow	Zero	NaN
Normal	Normal	Normal	0	0	0	0
Normal	Normal	Denormal	0	1	1	0
Normal	Normal	Infinity	1	0	0	0

DATAA[]	DATAB[]	RESULT[]	Overflow	Underflow	Zero	NaN
Normal	Normal	Zero	0	1	1	0
Normal	Denormal	Zero	0	0	1	0
Normal	Zero	Zero	0	0	1	0
Normal	Infinity	Infinity	1	0	0	0
Normal	NaN	NaN	0	0	0	1
Denormal	Normal	Zero	0	0	1	0
Denormal	Denormal	Zero	0	0	1	0
Denormal	Zero	Zero	0	0	1	0
Denormal	Infinity	NaN	0	0	0	1
Denormal	NaN	NaN	0	0	0	1
Zero	Normal	Zero	0	0	1	0
Zero	Denormal	Zero	0	0	1	0
Zero	Zero	Zero	0	0	1	0
Zero	Infinity	NaN	0	0	0	1
Zero	NaN	NaN	0	0	0	1
Infinity	Normal	Infinity	1	0	0	0
Infinity	Denormal	NaN	0	0	0	1
Infinity	Zero	NaN	0	0	0	1
Infinity	Infinity	Infinity	1	0	0	0
Infinity	NaN	NaN	0	0	0	1
NaN	Normal	NaN	0	0	0	1
NaN	Denormal	NaN	0	0	0	1
NaN	Zero	NaN	0	0	0	1
NaN	Infinity	NaN	0	0	0	1
NaN	NaN	NaN	0	0	0	1

ALTFP_MULT Resource Utilization and Performance

The following tables list the resource utilization and performance information for the ALTFP_MULT IP core. The information was derived using the Quartus II software version 10.0.

Table 7-3: ALTFP_MULT Resource Utilization and Performance for Stratix IV Devices with Dedicated Multiplier Circuitry

Device Family	Precision	Output latency	Logic usage				f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-bit DSP	
Stratix IV	Single	5	138	148	100	4	274
		11	185	301	190	4	445
	Double	5	306	367	272	10	255
		11	419	523	348	10	395

ALTFP_MULT Design Example: Multiplication of Double-Precision Format Numbers

This design example uses the ALTFP_MULT IP core to compute the multiplication results of two double-precision format numbers. This example uses the parameter editor GUI to define the core.

Related Information

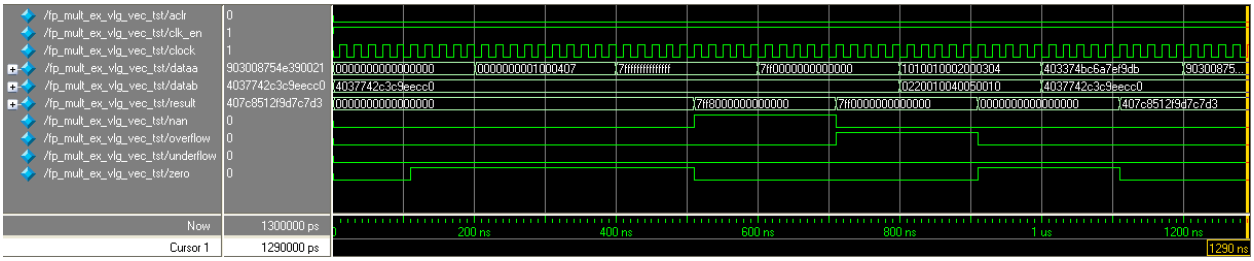
- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

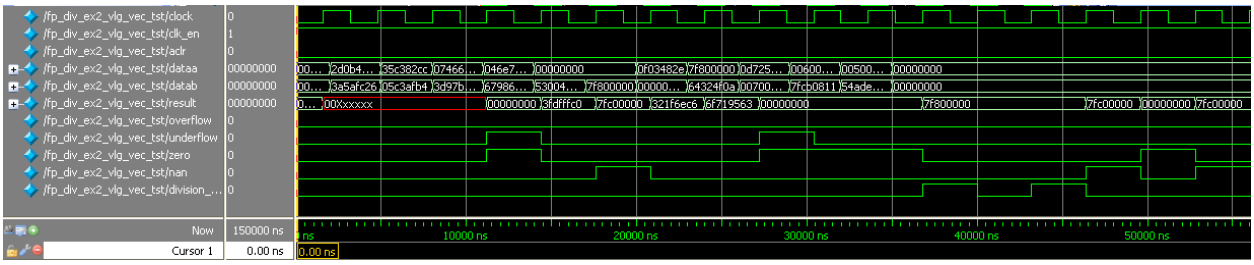
ALTFP_MULT Design Example: Understanding the Simulation Waveform

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

Figure 7-1: ALTFP_MULT Simulation Waveform

This figure shows the expected simulation results in the ModelSim-Altera software.





This design example implements a floating-point multiplier for the multiplication of double-precision format numbers. All the optional input ports (`clk_en` and `aclr`) and output ports (`overflow`, `underflow`, `zero`, and `nan`) are enabled.

In this example, the latency is set to 6 clock cycles. Therefore, every multiplication result appears at the result port 6 clock cycles later.

Table 7-4: Summary of Input Values and Corresponding Outputs

This table lists the inputs and corresponding outputs obtained from the simulation in the waveform.

Time	Event
0 ns, start-up	`dataa[]` value: 0000 0000 0000 0000h `datab[]` value: 4037 742C 3C9E ECC0h Output value: All values seen on the output port before the 6th clock cycle are merely due to the behavior of the system during start-up and should be disregarded.
110 ns	Output value: 0000 0000 0000 0000h Exception handling ports: `zero` asserts The multiplication of zero at the input port `dataa[]`, and a non-zero value at the input port `datab[]` results in a zero.
600 ns	`dataa[]` value: 7FF0 0000 0000 0000h `datab[]` value: 4037 742C 3C9E ECC0h This is the multiplication of an infinity value and a normal value.
710 ns	Output value: 7FF0 0000 0000 0000h Exception handling ports: `overflow` asserts The multiplication of an infinity value and a normal value results in infinity. All multiplications with an infinity value results in infinity except when infinity is multiplied with a zero.

Parameters

Table 7-5: ALTFP_MULT Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH_EXP	Integer	No	Specifies the value of the exponent. If this parameter is not specified, the default is 8. The bias of the exponent is always $2^{(WIDTH\_EXP - 1)}-1$ (that is, 127 for the single-precision format and 1023 for the double-precision format). WIDTH_EXP must be 8 for the single-precision format or a minimum of 11 for the double-precision format and the single-extended precision format. WIDTH_EXP must less than WIDTH_MAN. The sum of WIDTH_EXP and WIDTH_MAN must be less than 64.
WIDTH_MAN	Integer	No	Specifies the value of the mantissa. If this parameter is not specified, the default is 23. When WIDTH_EXP is 8 and the floating-point format is single-precision, the WIDTH_MAN value must be 23; otherwise, the value of WIDTH_MAN must be a minimum of 31. The WIDTH_MAN value must always be greater than the WIDTH_EXP value. The sum of WIDTH_EXP and WIDTH_MAN must be less than 64.
DEDICATED_MULTIPLIER_ CIRCUITRY	String	No	Specifies whether to use dedicated multiplier circuitry. Values are AUTO, YES, or NO. If this parameter is not specified, the default is AUTO. If a device does not have dedicated multiplier circuitry, the DEDICATED_MULTIPLIER_CIRCUITRY parameter has no effect and defaults to NO.
PIPELINE	Integer	No	Specifies the number of clock cycles needed to produce the multiplied result. Values are 5, 6, 10, and 11. If this parameter is not specified, the default is 5.

ALTFP_MULT Signals

Table 7-6: ALTFP_MULT IP Core Input Signals

Port Name	Required	Description
clock	Yes	Clock input to the IP core.
clk_en	No	Clock enable. Allows multiplication to take place when asserted high. When signal is asserted low, no multiplication occurs and the outputs remain unchanged.

Port Name	Required	Description
`aclr`	No	Synchronous clear. Source is asynchronously reset when asserted high.
`dataa[]`	Yes	Floating-point input data input to the multiplier. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of sign bit, exponent bits, and mantissa bits.
`datab[]`	Yes	Floating-point input data to the multiplier. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of sign bit, exponent bits, and mantissa bits.

Table 7-7: ALTFP_MULT IP Core Output Signals

Port Name	Required	Description
`result[]`	Yes	Output port for the multiplier. The floating-point result after rounding. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa.
`overflow`	No	Overflow port for the multiplier. Asserted when the result of the multiplication, after rounding, exceeds or reaches infinity. Infinity is defined as a number in which the exponent exceeds $2^{\texttt{WIDTH\_EXP}}\texttt{-1}$.
`underflow`	No	Underflow port for the multiplier. Asserted when the result of the multiplication (after rounding) is `0` while none of the inputs to the multiplication is `0`, or asserted when the result is a denormalized number.
`zero`	No	Zero port for the multiplier. Asserted when the value of `result[]` is `0`.
`nan`	No	NaN port for the multiplier. This port is asserted when an invalid multiplication occurs, such as the multiplication of infinity and zero. In this case, a NaN value is the output generated at the `result[]` port. The multiplication of any value and NaN produces NaN.

ALTFP_SQRT 8

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs square root calculation based on the input provided.You can use the ports and parameters available to customize the ALTFP_SQRT IP core according to your application.

ALTFP_SQRT Features

The ALTFP_SQRT IP core offers the following features:

- Square root functions.
- Optional exception handling output ports such as `zero`, `overflow`, and `nan`.

Output Latency

The output latency options for the ALTFP_SQRT megafunction differs depending on the precision selected, the width of the mantissa, or both.

Table 8-1: Latency Options for Each Precision Format

Precision	Mantissa Width	Latency (in clock cycles)
Single	23	16, 28
Double	52	30, 57

Precision	Mantissa Width	Latency (in clock cycles)
Single-extended	31	20, 36
	32	20, 37
	33	21, 38
	34	21, 39
	35	22, 40
	36	22, 41
	37	23, 42
	38	23, 43
	39	24, 44
	40	24, 45
	41	25, 46
	42	25, 47
	43	26, 48
	44	26, 49
	45	27, 50
	46	27, 51
	47	28, 52
	48	28, 53
	49	29, 54
	50	29, 55
	51	30, 56

ALTFP_SQRT Truth Table

Truth Table for Square Root Operations

DATA[]	SIGN BIT	RESULT[]	NaN	Overflow	Zero
Normal	0	Normal	0	0	0
Denormal	0/1	Zero	0	0	1
Positive Infinity	0	Infinity	0	1	0
Negative Infinity	1	All 1's	1	0	0
Positive NaN	0	All 1's	1	0	0
Negative NaN	1	All 1's	1	0	0

DATA[]	SIGN BIT	RESULT[]	NaN	Overflow	Zero
Zero	0/1	Zero	0	0	1
Normal	1	All 1's	1	0	0

ALTFP_SQRT Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_SQRT IP core. The information was derived using the Quartus II software version 10.0.

Table 8-2: ALTFP_SQRT Resource Utilization and Performance for Stratix IV Devices

Device Family	Precision	Output latency	Logic usage			f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Login Registers (DLRs)	Adaptive Logic Modules (ALMs)	
Stratix IV	Single	28	502	932	528	472
	Double	57	2,177	3,725	2,202	366

ALTFP_SQRT Design Example: Square Root of Single-Precision Format Numbers

This design example uses the ALTFP_SQRT IP core to compute the square root of single-precision format numbers. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

Related Information

- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_SQRT Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

These figures show the expected simulation results in the ModelSim-Altera software.

Figure 8-1: ALTFP_SQRT ModelSim Simulation Waveform (Input Data)

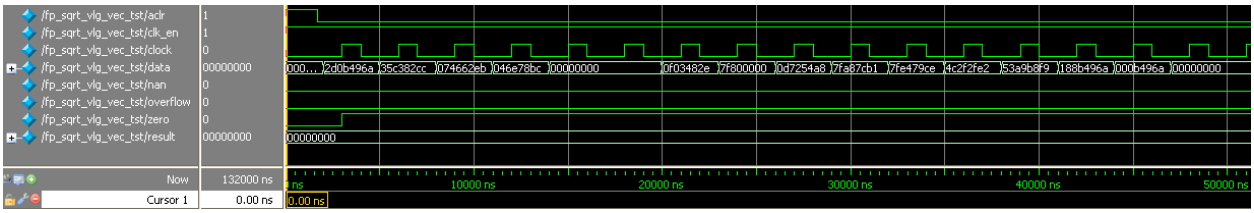
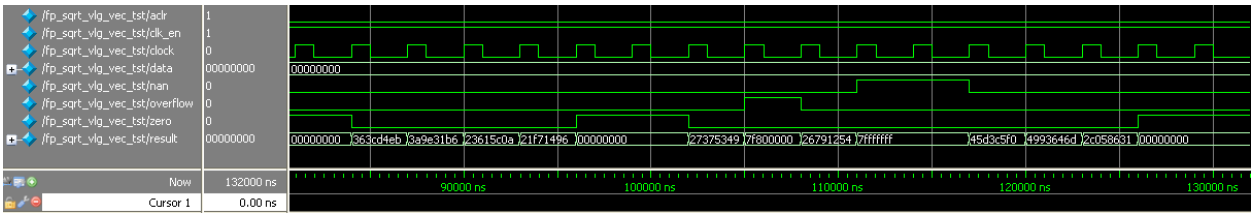


Figure 8-2: ALTFP_SQRT ModelSim Simulation Waveform (Output Data)



This design example implements a floating-point square root function for single-precision format numbers with all the exception output ports instantiated. The output ports include `overflow`, `zero`, and `nan`.

The output latency is 28 clock cycles. Every square root computation generates the output result 28 clock cycles later.

Table 8-3: Summary of Input Values and Corresponding Outputs

This table lists the inputs and corresponding outputs obtained from the simulation in the waveforms.

Time	Event
0 ns, start-up	Output value: All values seen on the output port before the 28th clock cycle are merely due to the behavior of the system during start-up and should be disregarded.
2 000 ns	`data[]` value: 2D0B 496Ah The data input is a normal number.
84 000 ns	Output value: 363C D4EBh The square root computation of a normal input results in a normal output.
14 000 ns	`data[]` value: 0000 0000h
96 000 ns	Output value: 0000 0000h Exception handling ports: `zero` asserts The square root computation of zero results in a zero.

Time	Event
23 000 ns	`data[]` value: 7F80 0000h The input is infinity.
105 000 ns	Output value: 7F80 0000h Exception handling ports: `overflow` asserts

ALTFP_SQRT Signals

Figure 8-3: ALTFP_SQRT Signals

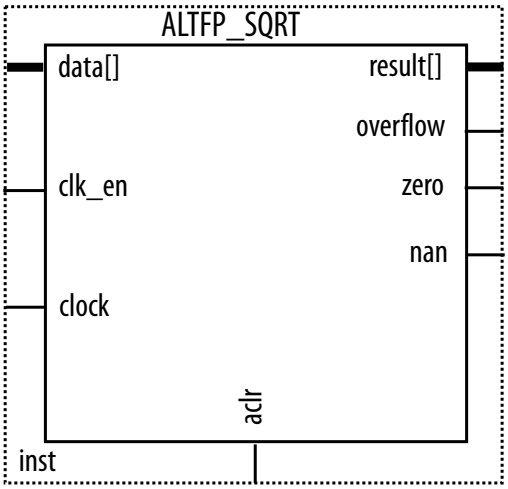


Table 8-4: ALTFP_SQRT IP Core Input Signals

Port Name	Required	Description
`clock`	Yes	Clock input to the IP core.
`clk_en`	No	Clock enable that allows square root operations when the port is asserted high. When the port is asserted low, no operation occurs and the outputs remain unchanged.
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high, the function is asynchronously reset.
	Yes	Floating-point input data. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of sign bit, exponent bits, and mantissa bits.

Table 8-5: ALTFP_SQRT IP Core Output Signals

Port Name	Required	Description
`result[]`	Yes	Square root output port for the floating-point result. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits, and mantissa bits.
`overflow`	Yes	Overflow port. Asserted when the result of the square root (after rounding) exceeds or reaches infinity. Infinity is defined as a number in which the exponent exceeds 2^{WIDTH_EXP} `-1`.
`zero`	Yes	Zero port. Asserted when the value of the `result[]` port is `0`.
`nan`	Yes	NaN port. Asserted when an invalid square root occurs, such as negative numbers or NaN inputs.

ALTFP_SQRT Parameters

Table 8-6: ALTFP_SQRT Parameters

Parameter Name	Type	Required	Description
`WIDTH_EXP`	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is `8`. The bias of the exponent is always set to `2` (`WIDTH_EXP` `-1`) `-1`, that is, `127` for the single-precision format and `1023` for the double-precision format. The value of the `WIDTH_EXP` parameter must be `8` for the single-precision format, `11` for the double-precision format, and a minimum of `11` for the single-extended precision format. The value of the `WIDTH_EXP` parameter must be less than the value of the `WIDTH_MAN` parameter, and the sum of the `WIDTH_EXP` and `WIDTH_MAN` parameters must be less than `64`.
`WIDTH_MAN`	Integer	Yes	Specifies the value of the mantissa. If this parameter is not specified, the default is `23`. When the `WIDTH_EXP` parameter is `8` and the floating-point format is single-precision, the `WIDTH_MAN` parameter value must be `23`. Otherwise, the value of the `WIDTH_MAN` parameter must be a minimum of `31`. The value of the `WIDTH_MAN` parameter must be greater than the value of the `WIDTH_EXP` parameter. The sum of the `WIDTH_EXP` and `WIDTH_MAN` parameters must be less than `64`.
`ROUNDING`	String	Yes	Specifies the rounding mode. The default value is `TO_NEAREST`. Other rounding modes are not supported.

Parameter Name	Type	Required	Description
`PIPELINE`	Integer	Yes	Specifies the number of clock cycles for the square root results of the `result[]` port. Values are `WIDTH_MAN` + `5` and (`(WIDTH_MAN` + `5/2)+2`) as specified by truncating the radix point.

ALTFP_EXP IP Core 9

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs exponential calculation based on the input provided.

ALTFP_EXP Features

The ALTFP_EXP IP core offers the following features:

- Exponential value of a given input.
- Optional exception handling output ports such as `zero`, `overflow`, `underflow`, and `nan`.

Output Latency

The output latency options for the ALTFP_EXP megafunction differs depending on the precision selected, the width of the mantissa, or both.

Precision	Mantissa Width	Latency (in clock cycles)
Single	23	17
Double	52	25
Single-extended	31 – 38	22
	39 – 52	25

ALTFP_EXP Truth Table

Table 9-1: Truth Table for Exponential Operations

DATAA[]	Calculation	RESULT[]	NaN	Overflow	Underflow	Zero
Normal	edata	Normal	0	0	0	0
Normal	edata	Infinity	0	1	0	0

DATAA[]	Calculation	RESULT[]	NaN	Overflow	Underflow	Zero
Normal (numbers of small magnitude)	edata	1	0	0	1	0
Normal (negative numbers of large magnitude)	edata	0	0	0	1	0
Denormal	e0	1	0	0	0	0
Zero	e0	1	0	0	0	0
Infinity (+)	e+	Infinity	0	0	0	0
Infinity (-)	e-	0	0	0	0	1
NaN	—	NaN	1	0	0	0

ALTFP_EXP Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_EXP IP core. The information was derived using the Quartus II software version 10.0.

Table 9-2: ALTFP_EXP Resource Utilization and Performance for Stratix IV Devices

Device Family	Precision	Output Latency	Logic usage				f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-bit DSP	
Stratix IV	Single	17	631	521	448	19	284
	Double	25	4,104	2,007	2,939	46	279

ALTFP_EXP Design Example: Exponential of Single-Precision Format Numbers

This design example uses the ALTFP_EXP IP core to compute the exponential value of single-precision format numbers. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

Related Information

- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores

(2) Any denormal input is treated as a zero before going through the exponential process.

- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_EXP Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

These figures show the expected simulation results in the ModelSim-Altera software.

Figure 9-1: ALTFP_EXP ModelSim Simulation Waveform (Input Data)

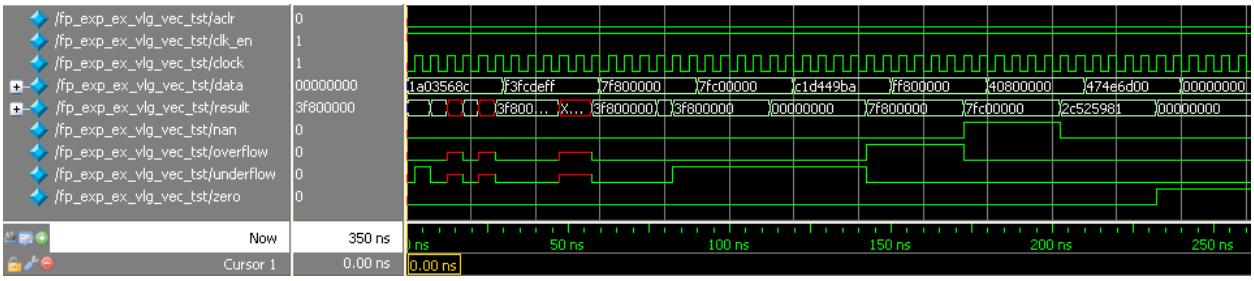
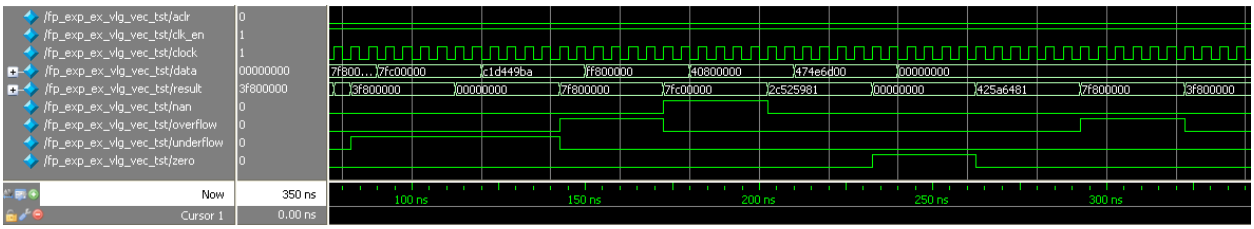


Figure 9-2: ALTFP_EXP ModelSim Simulation Waveform (Output Data)



This design example implements a floating-point exponential for the single-precision format numbers. The optional input ports (`clk_en` and `aclr`) and all four exception handling output ports (`nan`, `overflow`, `underflow`, and `zero`) are enabled.

For single-precision format numbers, the latency is fixed at 17 clock cycles. Therefore, every exponential operation outputs the results 17 clock cycles later.

Table 9-3: Summary of Input Values and Corresponding Outputs

This table lists the inputs and corresponding outputs obtained from the simulation in the waveforms.

Time	Event
0 ns, start-up	`data[]` value: 1A03 568Ch Output value: An undefined value is seen on the `result[]` port, which is ignored. All values seen on the output port before the 17th clock cycle are merely due to the behavior of the system during start-up and should be disregarded.

Time	Event
82.5 ns	Output value: 3F80 0000h As the input value of 1A03568Ch is a very small number, it is seen as a value that is approaching zero, and the result approaches 1 (which is represented by 3F800000). Exponential operations carried out on numbers of very small magnitudes result in a 1 and assert the `underflow` flag. Exception handling ports: `underflow` asserts
30 ns	`data[]` value: F3FC DEFFh This is a normal negative value of a very large magnitude.
112.5 ns	Output value: 0000 0000h The outcome of exponential operations on negative numbers of very large magnitudes approaches zero. Exception handling ports: `underflow` remains asserted
60 ns	`data[]` value: 7F80 0000h This is a positive infinite value.
142.5 ns	Output value: 7F80 0000h The operation on positive infinite values results in infinity. Exception handling ports: `underflow` deasserts, `overflow` asserts
90 ns	`data[]` value: 7FC0 0000h This is a NaN.
172.5 ns	Output value: 7FC0 0000h The exponential of a NaN results in a NaN. Exception handling ports: `nan` asserts
120 ns	`data[]` value: C1D4 49BAh This is a normal value.
202.5 ns	Output value: 2C52 5981h The result is a normal value. Exception handling ports: `nan` deasserts

ALTFP_EXP Signals

Figure 9-3: ALTFP_EXP Signals

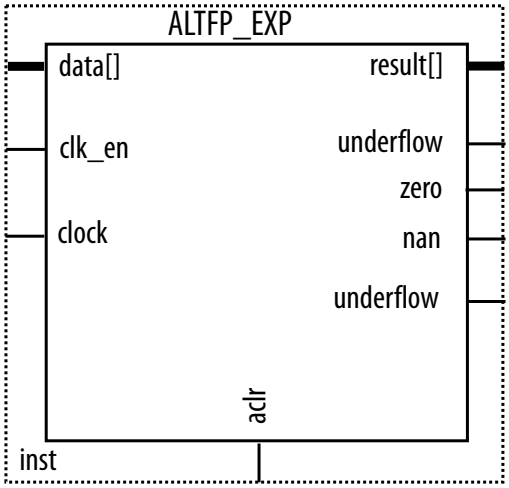


Table 9-4: ALTFP_EXP IP Core Input Signals

Port Name	Required	Description
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high the function is asynchronously reset.
`clk_en`	No	Clock enable. When the `clk_en` port is asserted high, an exponential value operation takes place. When this signal is asserted low, no operation occurs and the outputs remain unchanged.
`clock`	Yes	Clock input to the IP core.
`data[]`	Yes	Floating-point input data. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of the sign bit, exponent bits, and mantissa bits.

Table 9-5: ALTFP_EXP IP Core Output Signals

Port Name	Required	Description
`result[]`	Yes	The floating-point exponential result of the value at `data[]`. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits, and mantissa bits.
`overflow`	No	Overflow exception output. Asserted when the result of the operation (after rounding) is infinite.
`underflow`	No	Underflow exception output. Asserted when the result of the exponential approaches `1` (from numbers of very small magnitude), or when the result approaches `0` (from negative numbers of very large magnitudes).
`zero`	No	Zero exception output. Asserted when the value in the `result[]` port is zero.

Port Name	Required	Description
nan	No	NaN exception output. Asserted when an invalid operation occurs. Any operation involving NaN also asserts the nan port.

ALTFP_EXP Parameters

Table 9-6: ALTFP_EXP IP Core Parameters

Parameter Name	Type	Required	Description
WIDTH_EXP	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is 8. The bias of the exponent is always set to $2^{(WIDTH\_EXP\ -1)}\ -1$, that is, 127 for the single-precision format and 1023 for the double-precision format. The value of the WIDTH_EXP parameter must be 8 for the single-precision format, 11 for the double-precision format, and a minimum of 11 for the single-extended precision format. The value of the WIDTH_EXP parameter must be less than the value of the WIDTH_MAN parameter, and the sum of the WIDTH_EXP and WIDTH_MAN parameters must be less than 64.
WIDTH_MAN	Integer	Yes	Specifies the value of the mantissa. If this parameter is not specified, the default is 23. When the WIDTH_EXP parameter is 8 and the floating-point format is single-precision, the WIDTH_MAN parameter value must be 23. Otherwise, the value of the WIDTH_MAN parameter must be a minimum of 31. The value of the WIDTH_MAN parameter must be greater than the value of the WIDTH_EXP parameter. The sum of the WIDTH_EXP and WIDTH_MAN parameters must be less than 64.
PIPELINE	Integer	Yes	Specifies the amount of latency, expressed in clock cycles, used in the ALTFP_EXP IP core. Acceptable pipeline values are 17, 22, and 25 cycles of latency. Create the ALTFP_EXP IP core with the MegaWizard Plug-In Manager to calculate the value for this parameter.
ROUNDING	String	Yes	Specifies the rounding mode. The default value is TO_NEAREST. Other rounding modes are not supported.

ALTFP_INV IP Core 10

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs the function of 1/a where a is the given input.

ALTFP_INV Features

The ALTFP_INV IP core offers the following features:

- Inverse value of a given input.
- Optional exception handling output ports such as `zero`, `division_by_zero`, `underflow`, and `nan`.

Output Latency

The output latency options for the ALTFP_INV megafunction differs depending on the precision selected, the width of the mantissa, or both.

Precision	Mantissa Width	Latency (in clock cycles)
Single	23	20
Double	52	27
Single Extended	31 – 39	20
	40 – 52	27

ALTFP_INV Truth Table

Table 10-1: Truth Table for Inverse Operations

DATA[]	SIGN BIT	RESULT[]	Underflow	Zero	Division_by_zero	NaN
Normal	0/1	Normal	0	0	0	0

DATA[]	SIGN BIT	RESULT[]	Underflow	Zero	Division_by_zero	NaN
Normal	0/1	Denormal	1	1	0	0
Normal	0/1	Infinity	0	0	0	0
Normal	0/1	Zero	1	1	0	0
Denormal	0/1	Infinity	0	0	1	0
Zero	0/1	Infinity	0	0	1	0
Infinity	0/1	Zero	0	1	0	0
NaN	X	NaN	0	0	0	1

ALTFP_INV Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_INV IP core. The information was derived using the Quartus II software version 10.0.

Table 10-2: ALTFP_INV Resource Utilization and Performance for Stratix IV Devices

Device Family	Precision	Output Latency	Logic usage				f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-Bit DSP	
Stratix IV	Single	20	401	616	373	16	412
	Double	27	939	1,386	912	48	203

ALTFP_INV Design Example: Inverse of Single-Precision Format Numbers

This design example uses the ALTFP_INV IP core to compute the inverse of single-precision format numbers. This example uses the parameter editor in the Quartus II software.

Related Information

- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

(3) Any calculated or computed **denormal** output is replaced by a zero and asserts the zero and underflow flags.
(4) Any denormal input is treated as a zero before going through the inverse process.

ALTFP_INV Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

These figures show the expected simulation results in the ModelSim-Altera software.

Figure 10-1: ALTFP_INV ModelSim Simulation Waveform (Input Data)

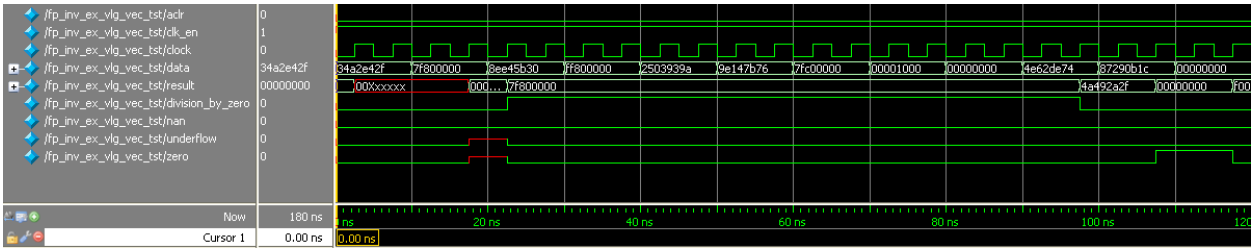
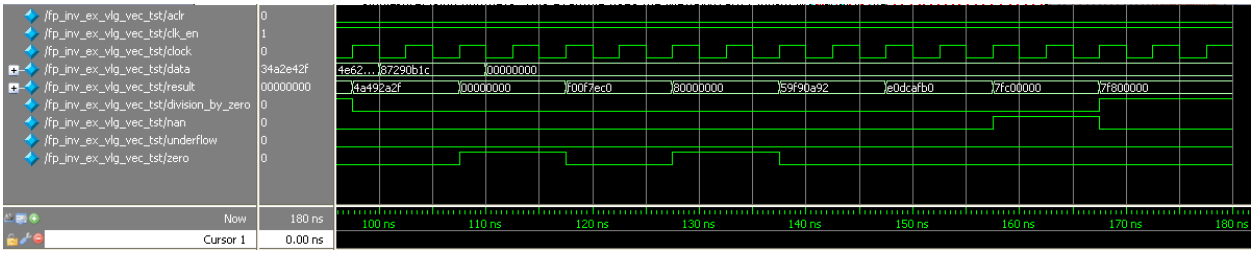


Figure 10-2: ALTFP_INV ModelSim Simulation Waveform (Output Data)



This design example implements a floating-point inverse for single-precision format numbers. The optional input ports (`clk_en` and `aclr`) and all four exception handling output ports (`division_by_zero`, `nan`, `zero`, and `underflow`) are enabled.

The latency is fixed at 20 clock cycles; therefore, every inverse operation outputs results 20 clock cycles later.

This table lists the inputs and corresponding outputs obtained from the simulation in the waveforms.

Table 10-3: Summary of Input Values and Corresponding Outputs

Time	Event
0 ns, start-up	`data[]` value: 34A2 E42Fh Output value: An undefined value is seen on the `result[]` port, which is ignored. All values seen on the output port before the 20th clock cycle are merely due to the behavior of the system during start-up and should be disregarded.

Time	Event
97.5 ns	Output value: 4A49 2A2Fh Exception handling ports: `division_by_zero` deasserts The inverse of a normal number results in a normal value.
10 ns	`data[]` value: 7F80 0000h This is an infinity value.
107.5 ns	Output value: 0000 0000h Exception handling ports: `zero` asserts The inverse of an infinity value produces a zero.
60 ns	`data[]` value: 7FC0 0000h This is a NaN.
157.5 ns	Output value: 7FC0 0000h Exception handling ports: `nan` asserts The inverse of a NaN results in a NaN
70 ns	`data[]` value: 0000 1000h This is a denormal number.
167.5 ns	Output value: 7F80 0000h Exception handling ports: `nan` deasserts, `division_by_zero` asserts Denormal numbers are forced-zero values, therefore, the inverse of a zero results in infinity.

Ports

Table 10-4: ALTFP_INV Megafunction Input Ports

Port Name	Required	Description
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high, the function is asynchronously cleared.
`clk_en`	No	Clock enable. When the `clk_en` port is asserted high, an inversion value operation takes place. When signal is asserted low, no operation occurs and the outputs remain unchanged.
`clock`	Yes	Clock input to the megafunction.

Port Name	Required	Description
data[]	Yes	Floating-point input data. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of the sign bit, exponent bits, and mantissa bits.

Table 10-5: ALTFP_INV Megafunction Output Ports

Port Name	Required	Description
result[]	Yes	The floating-point inverse result of the value at the data[]input port. The MSB is the sign, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits, and mantissa bits.
underflow	No	Underflow exception output. Asserted when the result of the inversion (after rounding) is a denormalized number.
zero	No	Zero exception output. Asserted when the value at the result[] port is a zero.
division_by_zero	No	Division-by-zero exception output. Asserted when the denominator input is a zero.
nan	No	NaN exception output. Asserted when an invalid inversion occurs, such as the inversion of NaN. In this case, a NaN value is output to the result[] port. Any operation involving NaN also asserts the nan port.

Parameters

Table 10-6: ALTFP_INV Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH_EXP	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is 8. The bias of the exponent is always set to $2^{(WIDTH\_EXP\ -1)}\ -1$, that is, 127 for the single-precision format and 1023 for the double-precision format. The value of the WIDTH_EXP parameter must be 8 for the single-precision format, 11 for the double-precision format, and a minimum of 11 for the single-extended precision format. The value of the WIDTH_EXP parameter must be less than the value of the WIDTH_MAN parameter, and the sum of the WIDTH_EXP and WIDTH_MAN parameters must be less than 64.

Parameter Name	Type	Required	Description
WIDTH_MAN	Integer	Yes	Specifies the value of the mantissa. If this parameter is not specified, the default is `23`. When the `WIDTH_EXP` parameter is `8` and the floating-point format is single-precision, the `WIDTH_MAN` parameter value must be `23`. Otherwise, the value of the `WIDTH_MAN` parameter must be a minimum of `31`. The value of the `WIDTH_MAN` parameter must be greater than the value of the `WIDTH_EXP` parameter. The sum of the `WIDTH_EXP` and `WIDTH_MAN` parameters must be less than `64`.
PIPELINE	Integer	Yes	Specifies the amount of latency in clock cycles used in the ALTFP_INV megafunction. Create the ALTFP_INV megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.
ROUNDING	String	No	Specifies the rounding mode. The default value is `TO_NEAREST`. Other rounding modes are not supported.

ALTFP_INV_SQRT IP Core 11

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs inverse square root value of a given input.

ALTFP_INV_SQRT Features

The ALTFP_INV_SQRT IP core offers the following features:

- Inverse square root value of a given input.
- Optional exception handling output ports such as `zero`, `division_by_zero`, and `nan`.

Output Latency

The output latency options for the ALTFP_INV_SQRT megafunction differs depending on the precision selected, the width of the mantissa, or both.

Table 11-1: Latency Options for Each Precision Format

Precision	Mantissa Width	Latency (in clock cycles)
Single	23	26
Double	52	36
Single-Extended	31– 39	26
	40 – 52	36

ALTFP_INV_SQRT Truth Table

Table 11-2: Truth Table for Inverse Square Root Operations

DATA[]	SIGN BIT	RESULT[]	Zero	Division_by_zero	NaN
Normal	0	Normal	0	0	0
Normal	1	NaN	0	0	1
Denormal	0/1	Infinity	0	1	0
Zero	0/1	Infinity	0	1	0
Infinity	0/1	Zero	1	0	0
NaN	X	NaN	0	0	1

ALTFP_INV_SQRT Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_INV_SQRT IP core. The information was derived using the Quartus II software version 10.0.

Table 11-3: ALTFP_INV_SQRT Resource Utilization and Performance forStratix IV Devices

Device Family	Precision	Output Latency	Logic usage				f_{MAX} (MHz)
			Adaptive Look-up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-Bit DSP	
Stratix IV	Single	26	502	658	430	22	413
	Double	36	1,324	1,855	1,209	78	209

ALTFP_INV_SQRT Design Example: Inverse Square Root of Single-Precision Format Numbers

This design example uses the ALTFP_INV_SQRT IP core to compute the inverse square root of single-precision format numbers. This example uses the parameter editor GUI to define the core.

Related Information

- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

(5) Any denormal input is treated as a zero before going through the inverse process.

ALTFP_INV_SQRT Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

These figures show the expected simulation results in the ModelSim-Altera software.

Figure 11-1: ALTFP_INV_SQRT ModelSim Simulation Waveform (Input Data)

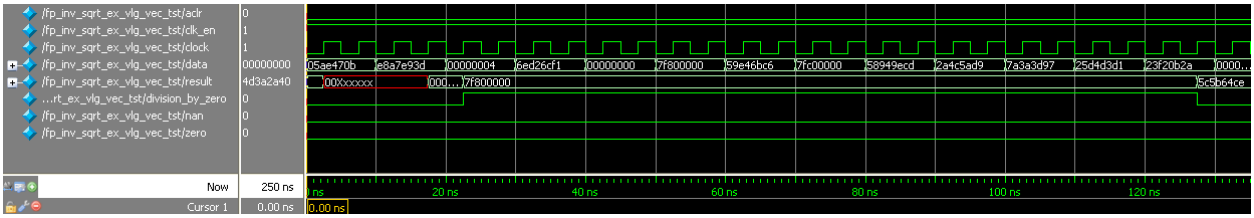
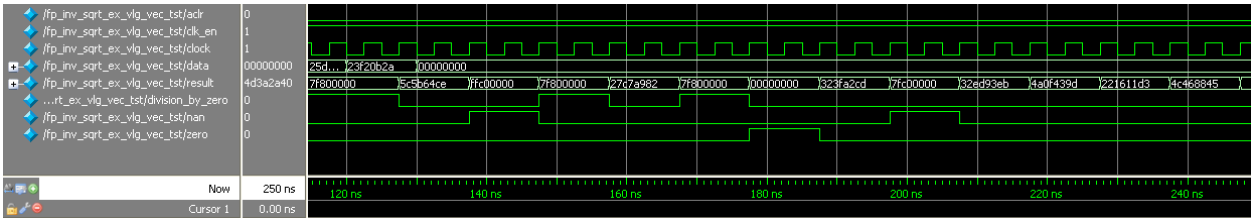


Figure 11-2: ALTFP_INV_SQRT ModelSim Simulation Waveform (Output Data)



This design example implements a floating-point inverse square root for single-precision format numbers. The optional input ports (`clk_en` and `aclr`) and all three exception handling output ports (`division_by_zero`, `nan`, and `zero`) are enabled.

The latency is fixed at 26 clock cycles. Therefore, every inverse square root operation outputs the results 26 clock cycles later.

This table lists the inputs and corresponding outputs obtained from the simulation in the waveforms.

Table 11-4: Summary of Input Values and Corresponding Outputs

Time	Event
0 ns, start-up	`data[]` value: 05AE 470Bh Output value: An undefined value is seen on the `result[]` port, which can be ignored. All values seen on the output port before the 26th clock cycle are merely due to the behavior of the system during start-up and should be disregarded.
127.5 ns	Output value: 5C5B 64CEh The inverse square root of a normal number results in a normal value.

Time	Event
10 ns	`data[]` value: E8A7 E93Dh This is a negative normal value.
137.5 ns	Output value: FFC0 0000h Exception handling ports: `nan` asserts The inverse square root of a negative value produces a NaN.
20 ns	`data[]` value: 0000 0004h The is a denormal value.
147.5 ns	Output value: 7F80 0000h Denormal numbers are forced-zero values, therefore the inverse square root of zero results in infinity. Exception handling ports: `nan` deasserts, `division_by_zero` asserts
50 ns	`data[]` value: 7F80 0000h This is an infinity value.
177.5 ns	Output value: 0000 0000h The inverse square root of an infinity value produces a zero. Exception handling ports: `zero` asserts

Ports

Figure 11-3: ALTFP_INV_SQRT Signals

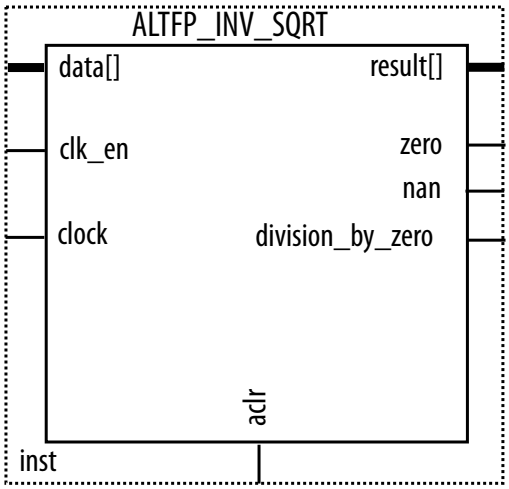


Table 11-5: ALTFP_INV_SQRT IP Core Input Signals

Port Name	Required	Description
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high, the function is asynchronously cleared.
`clk_en`	No	Clock enable. When the `clk_en` port is asserted high, an inversion value operation takes place. When signal is asserted low, no operation occurs and the outputs remain unchanged.
`clock`	Yes	Clock input to the IP core.
`data[]`	Yes	Floating-point input data. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of the sign bit, exponent bits, and mantissa bits.

Table 11-6: ALTFP_INV_SQRT IP Core Output Signals

Port Name	Required	Description
`result[]`	Yes	The floating-point inverse result of the value at the `data[]` input port. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. The size of this port is the total width of the sign bit, exponent bits, and mantissa bits.
`zero`	No	Zero exception output. Asserted when the value at the `result[]` port is a zero.
`division_by_zero`	No	Division-by-zero exception output. Asserted when the denominator input is a zero.
`nan`	No	NaN exception output. Asserted when an invalid inversion of square root occurs, such as the square root of a negative number. In this case, a NaN value is output to the `result[]` output port. Any operation involving a NaN will also produce a NaN.

Parameters

Table 11-7: ALTFP_INV_SQRT Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH_EXP	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is 8. The bias of the exponent is always set to $2^{(WIDTH\_EXP - 1)} - 1$, that is, 127 for the single-precision format and 1023 for the double-precision format. The value of the WIDTH_EXP parameter must be 8 for the single-precision format, 11 for the double-precision format, and a minimum of 11 for the single-extended precision format. The value of the WIDTH_EXP parameter must be less than the value of the WIDTH_MAN parameter, and the sum of the WIDTH_EXP and WIDTH_MAN parameters must be less than 64.
WIDTH_MAN	Integer	Yes	Specifies the value of the mantissa. If this parameter is not specified, the default is 23. When the WIDTH_EXP parameter is 8 and the floating-point format is single-precision, the WIDTH_MAN parameter value must be 23. Otherwise, the value of the WIDTH_MAN parameter must be a minimum of 31. The value of the WIDTH_MAN parameter must be greater than the value of the WIDTH_EXP parameter. The sum of the WIDTH_EXP and WIDTH_MAN parameters must be less than 64.
PIPELINE	Integer	Yes	Specifies the amount of latency, expressed in clock cycles, used in the ALTFP_INV_SQRT megafunction. Create the ALTFP_INV_SQRT megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.
ROUNDING	String	No	Specifies the rounding mode. The default value is TO_NEAREST. Other rounding modes are not supported.

ALTFP_LOG 12

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core performs natural logarithm function. You can use the ports and parameters available to customize the ALTFP_LOG IP core according to your application.

ALTFP_LOG Features

The ALTFP_LOG IP core offers the following features:

- Natural logarithm functions.
- Optional exception handling output ports such as `zero` and `nan`.

Output Latency

The output latency options for the ALTFP_LOG megafunction differs depending on the precision selected, the width of the mantissa, or both.

Table 12-1: Latency Options for Each Precision Format

Precision	Mantissa Width	Latency (in clock cycles)
Single	23	21
Double	52	34
Single Extended	31–36	25
	37–42	28
	43–48	31
	49–52	34

ALTFP_LOG Truth Table

This table lists the truth table for the natural logarithm operation.

Table 12-2: Truth Table for Natural Logarithm Operations

DATA[]	SIGN BIT	RESULT[]	Zero	NaN
Normal	0	Normal	0	0
Normal	1	NaN (6)	0	1
1 (7)	0	Zero	1	0
Denormal (8)	0	Negative Infinity	0	0
Zero (9)	0/1	Negative Infinity	0	0
Infinity	0	Positive Infinity	1	0
NaN	X	NaN	0	1

ALTFP_LOG Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_LOG IP core. The information was derived using the Quartus II software version 10.0.

Table 12-3: ALTFP_LOG Resource Utilization and Performance for Stratix IV Devices

Device Family	Precision	Output Latency	Logic usage				f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-Bit DSP	
Stratix IV	Single	21	1,950	1,864	1,378	8	385
	Double	34	5,451	6,031	4,151	64	211

ALTFP_LOG Design Example: Natural Logarithm of Single-Precision Format Numbers

This design example uses the ALTFP_LOG IP core to compute the natural logarithm of single-precision format numbers. This example uses the parameter editor GUI to define the core.

(6) The natural logarithm of a negative value is invalid. Therefore, the output produced is a NaN.
(7) The "1" in this case is equivalent to In 1.
(8) The value of positive denormalized numbers is a value that approximates zero, and the output produced is a negative infinity number.
(9) The zero in this case represents zero special case of the IEEE standard. It is not equivalent to In 0, but instead approximates to it.

Related Information

- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_LOG Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

These figures show the expected simulation results in the ModelSim-Altera software.

Figure 12-1: ALTFP_LOG ModelSim Simulation Waveform (Input Data)

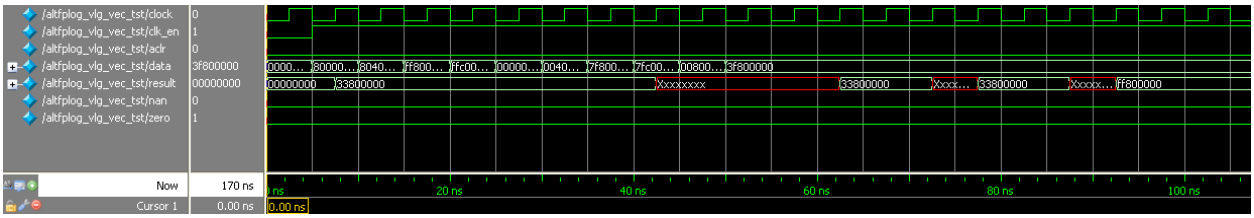
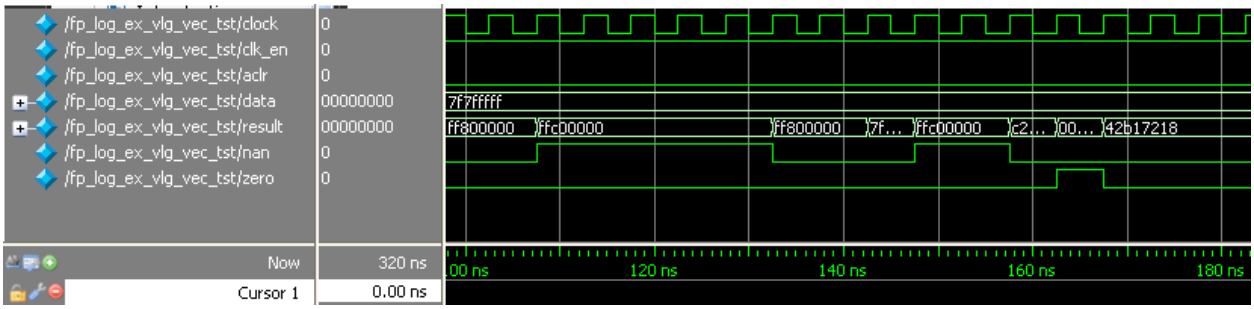


Figure 12-2: ALTFP_LOG ModelSim Simulation Waveform (Output Data)



This design example includes the input of special cases to show the exception handling of the IP core, such as the smallest valid input and the input value of "1".

In this example, the output delay is set to 21 clock cycles. Therefore, the result is only shown at the output port after the 21st clock cycle at 102.5 ns.

Table 12-4: Summary of Input Values and Corresponding Outputs

This table lists the inputs and corresponding outputs obtained from the simulation in the waveforms.

Time	Event
0 ns, start-up	`data[]` value: 0000 0000h Output value: An undefined value is seen on the `result[]` port, which is ignored. All values seen on the output port before the 21st clock cycle are merely due to the behavior of the system during start-up and should be disregarded.
102.5 ns	Output value: FF80 0000h The natural logarithm of zero is negative infinity.
5 ns	`data[]` value: 8000 0000h This is a negative number.
107.5 ns	Output value: FFC0 0000h Exception handling ports: `nan` asserts The natural logarithm of a negative value is invalid. Therefore, the output produced is a NaN.
30 ns	`data[]` value: 0040 0000h The is a denormal value.
132.5 ns	Output value: FF80 0000h As denormal numbers are not supported, the input is forced to zero before going through the logarithm function. The natural logarithm of zero is negative infinity.
45 ns	`data[]` value: 0080 0000h This is the smallest valid input. All the input bits are 0 except the LSB of the exponent field.
147.5 ns	Output value: C2AE AC50h
60 ns	`data[]` value: 3F80 0000h The input value 3F80 0000h is equivalent to the actual value, $1.0 \times 20 = 1$.
152.5 ns	Output value: 0000 0000h Exception handling ports: `zero` asserts Since In 1 results in zero, it produces an output of zero.

Signals

Figure 12-3: ALTFP_LOG Signals

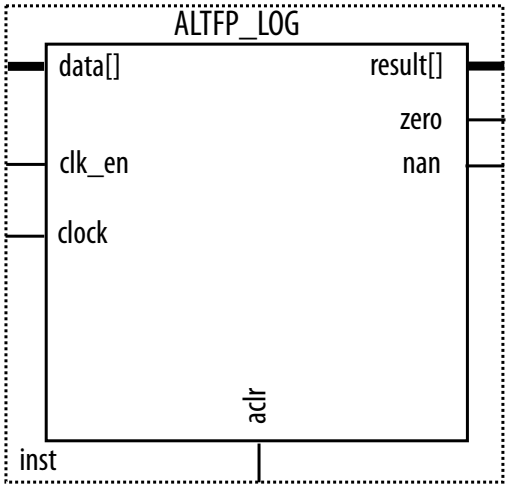


Table 12-5: ALTFP_LOG IP Core Input Signals

Port Name	Required	Description
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high, the function is asynchronously cleared.
`clk_en`	No	Clock enable. When the `clk_en` port is asserted high, a natural logarithm operation takes place. When signal is asserted low, no operation occurs and the outputs remain unchanged. Deasserting `clk_en` halts operation until it is asserted again. Assert the `clk_en` signal for the number of clock cycles equivalent to the required output latency (`PIPELINE` parameter value) for the results to be shown at the output.
`clock`	Yes	Clock input to the IP core.
`data[]`	Yes	Floating-point input data. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of the sign bit, exponent bits, and mantissa bits. For single precision, the width is fixed to 32 bits. For double precision, the width is fixed to 64 bits. For single extended precision, you can choose a width in the range from 43 to 64 bits.

Table 12-6: ALTFP_LOG IP Core Output Signals

Port Name	Required	Description
`result[]`	Yes	The natural logarithm of the value on input data. The natural logarithm of the `data[]` input port, shown in floating-point format. The widths of the `result[]` output port and `data[]` input port are the same.

Port Name	Required	Description
`zero`	No	Zero exception output. Asserted when the exponent and mantissa of the output port are zero. This occurs when the actual input value is `1` because *ln* `1 = 0`.
`nan`	No	NaN exception output. Asserted when the exponent and mantissa of the output port are all 1's and non-zero, respectively. This occurs when the input is a negative number or NaN.

Parameters

Table 12-7: ALTFP_LOG Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH_EXP	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is `8`. The bias of the exponent is always set to `2 (WIDTH_EXP -1) -1`, that is, `127` for the single-precision format and `1023` for the double-precision format. The value of the `WIDTH_EXP` parameter must be `8` for the single-precision format, `11` for the double-precision format, and a minimum of `11` for the single-extended precision format. The value of the `WIDTH_EXP` parameter must be less than the value of the `WIDTH_MAN` parameter, and the sum of the `WIDTH_EXP` and `WIDTH_MAN` parameters must be less than `64`.
WIDTH_MAN	Integer	Yes	Specifies the precision of the mantissa. If this parameter is not specified, the default is `23`. The value of `WIDTH_MAN` must be `23` for the single-precision format, and `52` for the double-precision format. For the single-extended precision format, the valid value ranges from `31` to `52`. The value of `WIDTH_MAN` must be greater than the value of `WIDTH_EXP`, and the sum of `WIDTH_EXP` and `WIDTH_MAN` must be less than `64`.
PIPELINE	Integer	Yes	Specifies the amount of latency in clock cycles used in the ALTFP_LOG megafunction. Create the ALTFP_LOG megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.

ALTFP_ATAN IP Core 13

2016.12.09

UG-01058

This IP core performs arctangent calculation. You can use the ports and parameters available to customize the ALTFP_ATAN IP core according to your application.

Output Latency

The output latency option for the ALTFP_ATAN megafunction have a fixed latency level for single-precision format.

Table 13-1: Latency Option

Trigonometric Function	Precision	Mantissa Width	Latency (in clock cycles)
Arctangent	Single	23	34

ALTFP_ATAN Features

The ALTFP_ATAN IP core offers the following features:

- Arctangent value of a given angle, θ in unit radian.
- Support for single-precision floating point format.
- Support for optional input ports such as asynchronous clear (`aclr`) and clock enable (`clk_en`) ports.

ALTFP_ATAN Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_ATAN IP core. The information was derived using the Quartus II software version 11.0.

Table 13-2: ALTFP_ATAN Resource Utilization and Performance

Device Family	Function	Precision	Output Latency	Logic usage				f_{MAX} (MHz)
				Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-Bit DSP	
Stratix V	ArcTangent	Single	36	2,454	1,010	1,303	27	255.49

Ports

Table 13-3: ALTFP_ATAN Megafunction Input Ports

Port Name	Required	Description
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high, the function is asynchronously cleared.
`clk_en`	No	Clock enable. When the `clk_en` port is asserted high, division takes place. When the signal is deasserted, no operation occurs and the outputs remain unchanged.
`clock`	Yes	Clock input to the megafunction.
`data[]`	Yes	Floating-point input data. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of the sign bit, exponent bits, and mantissa bits.

Port Name	Required	Description
`result[]`	Yes	The result of the trigonometric function in floating-point format. The widths of the `result[]` output port and `data[]` input port are the same.

ALTFP_ATAN Parameters

Table 13-4: ALTFP_ATAN Parameters

Parameter Name	Type	Required	Description
`WIDTH_EXP`	Integer	Yes	Specifies the precision of the exponent. The bias of the exponent is always set to `2(WIDTH_EXP-1) -1` (that is, `127` for single-precision format). The value of `WIDTH_EXP` must be `8` for single-precision format. The default value for `WIDTH_EXP` is `8`.
`WIDTH_MAN`	Integer	Yes	Specifies the precision of the mantissa. The value of `WIDTH_MAN` must be `23` when `WIDTH_EXP` is `8`. The default value for `WIDTH_MAN` is `23`.
`PIPELINE`	Integer	Yes	The number of pipeline is fixed for the mantissa width and some internal parameter. For the correct settings, refer to Table 12–1 on page 12–2.
`ROUNDING`	Integer	No	Specifies the rounding mode. The default value is `TO_NEAREST`. Other rounding modes are not supported.

ALTFP_SINCOS IP Core 14

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core perform trigonometric Sine/Cosine functions. You can use the ports and parameters available to customize the ALTFP_SINCOS IP core according to your application.

ALTFP_SINCOS Features

The ALTFP_SINCOS IP core offers the following features:

- Implements sine and cosine calculations.
- Support for single-precision floating point format.
- Support for optional input ports such as asynchronous clear (`aclr`) and clock enable (`clk_en`) ports.

Output Latency

The output latency options for the ALTFP_SINCOS megafunction have a fixed latency level for sine and cosine functions.

Trigonometric Function	Precision	Mantissa Width	Latency (in clock cycles)
Sine	Single	23	36
Cosine	Single	23	36

Related Information

ALTFP_SINCOS Parameters on page 14-3

ALTFP_SINCOS Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_SINCOS IP core. The information was derived using the Quartus II software version 10.1.

Table 14-1: ALTFP_SINCOS Resource Utilization and Performance

Device Family	Function	Precision	Output Latency	Logic usage				f_{MAX} (MHz)
				Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	18-Bit DSP	
Stratix IV	Sine	Single	36	2,859	2,190	1,830	16	292.96
	Cosine	Single	35	2,753	2,041	1,745	16	258.26

ALTFP_SINCOS Signals

Figure 14-1: ALTFP_SINCOS Signals

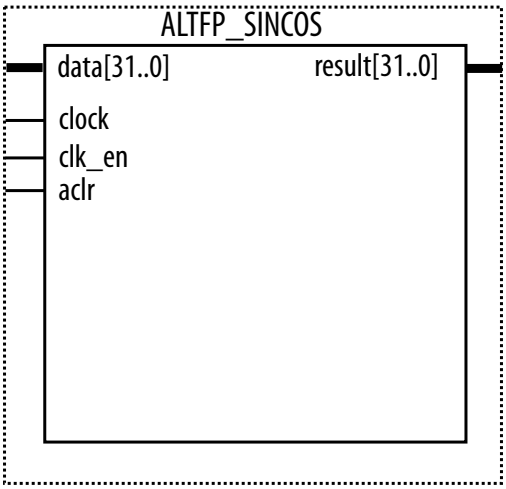


Table 14-2: ALTFP_SINCOS IP Core Input Signals

Port Name	Required	Description
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high, the function is asynchronously cleared.
`clk_en`	No	Clock enable. When the `clk_en` port is asserted high, sine or cosine operation takes place. When the signal is asserted low, no operation occurs and the outputs remain unchanged.
`clock`	Yes	Clock input to the megafunction.
`data[]`	Yes	Floating-point input data. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of the sign bit, exponent bits, and mantissa bits.

Table 14-3: ALTFP_SINCOS IP Core Output Signals

Port Name	Required	Description
`result[]`	Yes	The trigonemetric of the `data[]` input port in floating-point format. The widths of the `result[]` output port and `data[]` input port are the same.

ALTFP_SINCOS Parameters

Table 14-4: ALTFP_SINCOS IP Core Parameters

Parameter Name	Type	Required	Description
WIDTH_EXP	Integer	Yes	Specifies the precision of the exponent. The bias of the exponent is always set to $2^{(WIDTH\_EXP-1)} -1$ (that is, `127` for single-precision format). The value of `WIDTH_EXP` must be `8` for single-precision format and must be less than `WIDTH_MAN`. The available value for `WIDTH_EXP` is `8`.
WIDTH_MAN	Integer	Yes	Specifies the precision of the mantissa. The value of `WIDTH_MAN` must be `23` when `WIDTH_EXP` is `8`. Otherwise, `WIDTH_MAN` must be a minimum of `31`. The value of `WIDTH_MAN` must be greater than `WIDTH_EXP`. The available value for `WIDTH_MAN` is `23`.
PIPELINE	Integer	Yes	The number of pipeline is fixed for the mantissa width and some internal parameter. For the correct settings, refer to Output Latency.

Related Information

Output Latency on page 14-1

ALTFP_ABS IP Core 15

2016.12.09

UG-01058

Subscribe Send Feedback

This IP core performs absolute value calculation for the given input.

ALTFP_ABS Features

The ALTFP_ABS IP core offers the following features:

- Absolute value of a given input.
- Optional exception handling output ports such as `zero`, `division_by_zero`, `overflow`, `underflow`, and nan.
- Carry-through exception ports from other floating-point modules that act as inputs to the ALTFP_ABS IP core.

ALTFP_ABS Output Latency

The output latency options for the ALTFP_ABS IP core are the same for all three precision formats—single, double, and single-extended. The options available are zero without pipeline, and 1 clock cycle.

ALTFP_ABS Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_ABS IP core. The information was derived using the Quartus II software version 10.0.

Table 15-1: ALTFP_ABS Resource Utilization and Performance for the Stratix III Device Family

Precision	Output Latency	Logic usage				f_{MAX} (MHz)
		Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	18-Bit DSP	Memory	
Single	0	0	0	0	0	The f_{MAX} of this IP core depends on the speed of the selected device
	1	0	36	0	0	
Double	0	0	0	0	0	
	1	0	68	0	0	

ALTFP_ABS Design Example: Absolute Value of Multiplication Results

This design example uses the ALTFP_ABS IP core to compute the absolute value of the multiplication result of single-precision format numbers. This example incorporates the ALTFP_MULT IP core and uses the parameter editor in the Quartus II software.

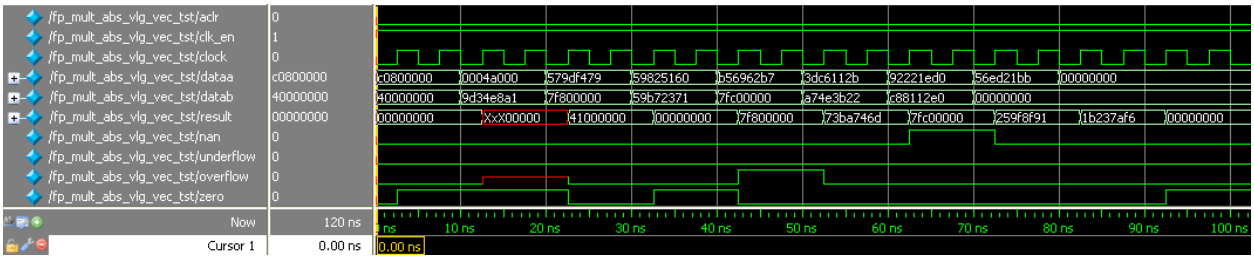
Related Information

- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_ABS Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

Figure 15-1: ALTFP_ABS Simulation Waveform



This design example produces a floating-point absolute value function for the multiplication results of single-precision format numbers. All the optional input ports (`clk_en` and `aclr`) and optional output ports (`overflow`, `underflow`, `zero`, `division_by_zero`, and `nan`) are enabled.

In this example, the latency of the multiplier is set to five clock cycles, while none is being set for the absolute value function. Thus, the absolute value result only appears at the `result[]` port five cycles after the input values are captured on the input ports.

The `dataa[]` and `datab[]` values in the simulation waveform above portray the two input values that are being fed to the multiplier. The value in the `result[]` port depicts the multiplication result that has gone through the absolute value operation.

This table lists the inputs and corresponding outputs obtained from the simulation.

Table 15-2: Summary of Input Values and Corresponding Outputs

Time	Event
0 ns, start-up	`dataa[]` value: C080 0000h `datab[]` value: 4000 0000h Output value: All values seen on the output port before the 5th clock cycle are merely due to the behavior of the system during start-up and should be disregarded.
22.5 ns	Output value: 4100 0000h The multiplication of a negative number with a positive number results in a negative number. The absolute value of the result is reflected on the `result[]` port.
20 ns	`dataa[]` value: 579D F479h `datab[]` value: 7F80 0000h The value of `dataa[]` is normal while the value of `datab[]` is infinity.
42.5 ns	Output value: 7F80 0000h Exception handling ports: `overflow` asserts The multiplication of a normal value with infinity results in infinity and sets the `overflow` port in the multiplier. The absolute value of the output is infinity and the `overflow` port is also set as this assertion of the port is being carried through from the corresponding `overflow` port in the multiplier.

ALTFP_ABS Signals

Figure 15-2: ALTFP_ABS Signals

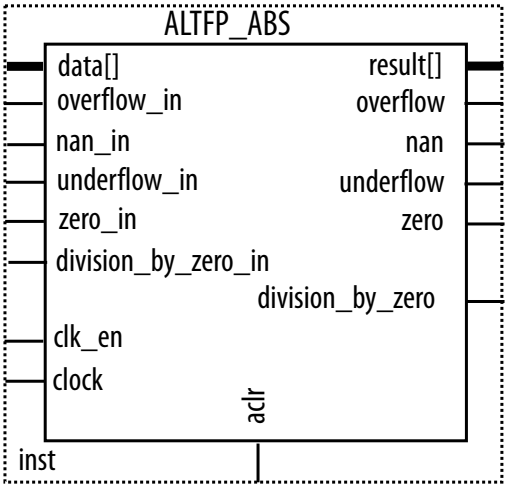


Table 15-3: ALTFP_ABS Input Signals

Port Name	Required	Description
`aclr`	No	Asynchronous clear. When the `aclr` port is asserted high, the function is asynchronously cleared.
`clk_en`	No	Clock enable. When the `clk_en` port is asserted high, an absolute value operation takes place. When the signal is asserted low, no operation occurs and the outputs remain unchanged.
`clock`	Yes	Clock input to the IP core.
`data[]`	Yes	Floating-point input data. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of sign bit, exponent bits, and mantissa bits.
`zero_in`	No	Zero exception input. Carry-through exception input port from other floating-point modules.
`nan_in`	No	NaN exception input. Carry-through exception input port from other floating-point modules.
`overflow_in`	No	Overflow exception input. Carry-through exception input port from other floating-point modules.
`underflow_in`	No	Underflow exception input. Carry-through exception input port from other floating-point modules.
`division_by_zero_in`	No	Division-by-zero exception input. Carry-through exception input port from other floating-point modules.

Table 15-4: ALTFP_ABS Output Signals

Port Name	Required	Description
`result[]`	Yes	The absolute value result of the input data. The size of this port corresponds to the size of the input `data[]` port.
`zero`	No	Zero exception output carried from the input. Asserted if the corresponding carry-through port from the input is asserted.
`nan`	No	NaN output carried from the input. Asserted if the corresponding carry-through port from the input is asserted.
`overflow`	No	Overflow exception output carried from the input. Asserted if the corresponding carry-through port from the input is asserted.
`underflow`	No	Underflow exception output carried from the input. Asserted if the corresponding carry-through port from the input is asserted.
`division_by_zero`	No	Division-by-zero exception output carried from the input. Asserted if the corresponding carry-through port from the input is asserted.

ALTFP_ABS Parameters

Table 15-5: ALTFP_ABS Parameters

Port Name	Type	Required	Description
`WIDTH_EXP`	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is `8`. The bias of the exponent is always set to `2` (WIDTH_EXP - 1) - `1`, that is, `127` for the single-precision format and `1023` for the double-precision format. The value of `WIDTH_EXP` must be `8` for the single-precision format, `11` for the double-precision format, and a minimum of `11` for the single-extended precision format. The value of `WIDTH_EXP` must be less than the value of `WIDTH_MAN`, and the sum of `WIDTH_EXP` and `WIDTH_MAN` must be less than `64`.

Port Name	Type	Required	Description
WIDTH_MAN	Integer	Yes	Specifies the precision of the mantissa. If this parameter is not specified, the default is 23. When WIDTH_EXP is 8 and the floating-point format is single-precision, the WIDTH_MAN value must be 23. Otherwise, the value of WIDTH_MAN must be a minimum of 31. The value of WIDTH_MAN must be greater than the value of WIDTH_EXP, and the sum of WIDTH_EXP and WIDTH_MAN must be less than 64.
PIPELINE	Integer	Yes	Specifies the amount of latency, expressed in clock cycles, used in the ALTFP_ABS IP core. Create the ALTFP_ABS IP core with the parameter editor to calculate the value for this parameter.

ALTFP_COMPARE IP Core 16

2016.12.09

UG-01058

Subscribe Send Feedback

ALTFP_COMPARE Features

The ALTFP_COMPARE IP core offers the following features:

- Comparison functions between two inputs.
- Seven status output ports:
  - `aeb` (input A is equal to input B).
  - `aneb` (input A is not equal to input B).
  - `agb` (input A is greater than input B).
  - `ageb` (input A is greater than or equal to input B).
  - `alb` (input A is less than input B).
  - `aleb` (input A is less than or equal to input B).
  - `unordered` (used as an output to flag if one or both input ports are NaN).

ALTFP_COMPARE Output Latency

The output latency options for the ALTFP_COMPARE IP core are the same for all three precision formats —single, double, and single-extended. The options available are 1, 2, and 3 clock cycles.

ALTFP_COMPARE Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_COMPARE IP core. The information was derived using the Quartus II software version 10.0.

Table 16-1: ALTFP_COMPARE Resource Utilization and Performance for Stratix IV Devices

Device Family	Precision	Output Latency	Logic Usage			f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Look-Up Modules (ALMs)	
Stratix IV	single	3	68	33	47	794
	double	3	121	47	87	680

ALTFP_COMPARE Design Example: Comparison of Single-Precision Format Numbers

This design example uses the ALTFP_COMPARE IP core to implement the comparison of single-precision format numbers using the parameter editor in the Quartus II software.

Related Information

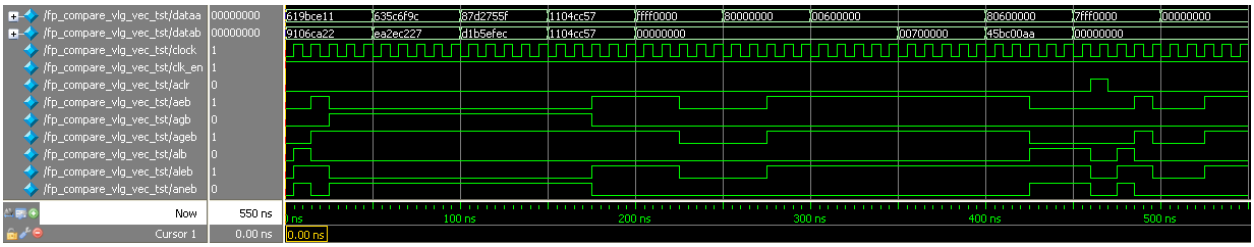
- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_COMPARE Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

This figure shows the expected simulation results in the ModelSim-Altera software.

Figure 16-1: ALTFP_COMPARE Simulation Waveform



This design example implements a floating-point comparator for single-precision numbers. Both optional input ports (`clk_en` and `aclr`) and all seven output ports (`ageb`, `aeb`, `agb`, `aneb`, `alb`, `aleb`, and `unordered`) are enabled.

The chosen output latency is 3. Therefore, the comparison operation generates the output result 3 clock cycles later.

This table lists the inputs and corresponding outputs obtained from the simulation in the waveform.

Table 16-2: Summary of Input Values and Corresponding Outputs

Time	Event
0 ns, start-up	`dataa[]` value: 619B CE11h `datab[]` value: 9106 CA22h Output value: An undefined value is seen on the `result[]` port, which is ignored. All values seen on the output port before the 3rd clock cycle are merely due to the behavior of the system during start-up and should be disregarded.
25 ns	Output ports: `ageb`, `aneb`, and `agb` assert
350 ns	`dataa[]` value: 0060 0000h `datab[]` value: 0070 0000h Both input values are denormal numbers.
375 ns	Output ports: `aeb`, `ageb`, and `aleb` assert Denormal inputs are not supported and are forced to zero before comparison takes place, which results in the `dataa[]` value being equal to `datab[]`.
460 ns	The `aclr` signal is set for 1 clock cycle.
495.5 ns	The comparisons of subsequent data inputs are performed 3 clock cycles after the `aclr` signal deasserts.

ALTFP_COMPARE Signals

Figure 16-2: ALTFP_COMPARE Signals

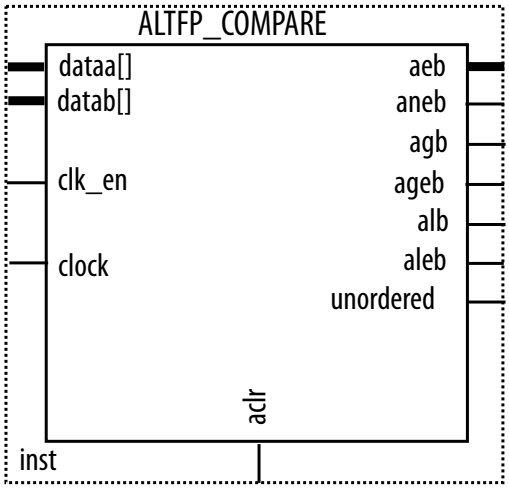


Table 16-3: ALTFP_COMPARE Input Signals

Port Name	Required	Description
`aclr`	No	Asynchronous clear. The source is asynchronously reset when asserted high.
`clk_en`	No	Clock enable. When this port is asserted high, a compare operation takes place. When signal is asserted low, no operation occurs and the outputs remain unchanged.
`clock`	Yes	Clock input to the IP core.
`dataa[]`	Yes	Data input. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of sign bit, exponent bits, and mantissa bits.
`datab[]`	Yes	Data input. The MSB is the sign bit, the next MSBs are the exponent, and the LSBs are the mantissa. This input port size is the total width of sign bit, exponent bits, and mantissa bits.

Table 16-4: ALTFP_COMPARE Output Signals

Port Name	Required	Description
`aeb`	Yes	Output port for the comparator. Asserted if the value of the `dataa[]` port equals the value of the `datab[]` port.
`agb`	Yes	Output port for the comparator. Asserted if the value of the `dataa[]` port is greater than the value of the `datab[]` port.
`ageb`	Yes	Output port for the comparator. Asserted if the value of the `dataa[]` port is greater than or equal to the value of the `datab[]` port.
`alb`	Yes	Output port for the comparator. Asserted if the value of the `dataa[]` port is less than the value of the `datab[]` port.
`aleb`	Yes	Output port for the comparator. Asserted if the value of the `dataa[]` port is less than or equal to the value of the `datab[]` port.
`aneb`	Yes	Output port for the comparator. Asserted if the value of the `dataa[]` port is not equal to the value of the `datab[]` port.
`unordered`	Yes	Output port for the comparator. Asserted when either the `dataa[]` port and the `datab[]` port is set to NaN, or if both the `dataa[]` port and the `datab[]` port are set to NaN.

ALTFP_COMPARE Parameters

Table 16-5: ALTFP_COMPARE Parameters

Port Name	Type	Required	Description
WIDTH_EXP	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is 8. The bias of the exponent is always set to 2 (WIDTH_EXP - 1) - 1, that is, 127 for the single-precision format and 1023 for the double-precision format. The value of WIDTH_EXP must be 8 for the single-precision format, 11 for the double-precision format, and a minimum of 11 for the single-extended precision format. The value of WIDTH_EXP must be less than the value of WIDTH_MAN, and the sum of WIDTH_EXP and WIDTH_MAN must be less than 64.
WIDTH_MAN	Integer	Yes	Specifies the precision of the mantissa. If this parameter is not specified, the default is 23. When WIDTH_EXP is 8 and the floating-point format is single-precision, the WIDTH_MAN value must be 23. Otherwise, the value of WIDTH_MAN must be a minimum of 31. The value of WIDTH_MAN must be greater than the value of WIDTH_EXP, and the sum of WIDTH_EXP and WIDTH_MAN must be less than 64.
PIPELINE	Integer	Yes	Specifies the latency in clock cycles used in the ALTFP_COMPARE IP core. The pipeline values are 1, 2, and 3 latency in clock cycles.

ALTFP_CONVERT IP Core 17

2016.12.09

UG-01058 Subscribe Send Feedback

ALTFP_CONVERT Features

The ALTFP_CONVERT IP core offers the following features:

- Conversion functions for the following formats:
  - Integer-to-Float
  - Float-to-Integer
  - Float-to-Float
  - Fixed-to-Float
  - Float-to-Fixed
- Support for signed and unsigned integer
- Optional exception handling output ports such as `overflow`, `underflow`, and `nan`

Table 17-1: Supported Operations and Exception Ports

Operation	Supported Exception Ports
Integer-to-Float	Not supported
Float-to-Integer	`overflow`, `underflow`, and `nan`
Float-to-Float	`overflow`, `underflow`, and `nan`
Fixed-to-Float	Not supported
Float-to-Fixed	`overflow`, `underflow`, and `nan`

ALTFP_CONVERT Conversion Operations

This table lists the features of each conversion operation.

Table 17-2: ALTFP_CONVERT Conversion Operations

Operation	Features
Integer-to-Float Conversion	• Converts integers to the IEEE-754 standard floating-point representation. • Supports conversions of signed integers to floating-point numbers in single, double, and single-extended precision formats.
Float-to-Integer Conversion	• Converts IEEE-754 standard floating-point representations to the integer-bit format. • Supports conversions of single, double, and single-extended precision formats to signed integers.

Float-to-Float Conversion	• Converts between IEEE-754 standard floating-point representations.
	• Supports conversions of between single double, and single-extended precision formats.
	• This operation offers the following modes:
	  • Single-precision format to single-extended precision format or double-precision format.
	  • Double-precision format to single-precision format or single-extended precision format.
	  • Single-extended precision format to single-precision or double-precision format.

Fixed-to-Float Conversion	• Converts fixed-point format data to the IEEE-754 standard floating-point representation. • Supports conversions of fixed-point format data to floating-point numbers in single, double, and single-extended precision formats.
Float-to-Fixed Conversion	• Converts IEEE-754 standard floating-point representations to the fixed-point format. • Supports conversion of floating-point numbers in single, double, and single-extended precision formats.

ALTFP_CONVERT Output Latency

The output latency options for the all the conversion operations in the ALTFP_CONVERT IP core are fixed, except for the Float-to-Float operation.

Table 17-3: Latency Options for Each Operation

Operation	Conversion From	Latency (in clock cycles)
Integer-to-Float	N/A	6
Float-to-Integer	N/A	6
Float-to-Float	Single-precision format	2
	Double-precision format	3
	Single-extended precision format	3
Fixed-to-Float	N/A	6
Float-to-Fixed	N/A	6

ALTFP_CONVERT Resource Utilization and Performance

This table lists the resource utilization and performance information for the ALTFP_CONVERT IP core. The information was derived using the Quartus II software version 10.0.

Table 17-4: ALTFP_CONVERT Resource Utilization and Performance for Stratix III Devices

Operation	Format	Pipeline	Logic Usage			f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	
Integer to-Float	32-bit integer to single-precision	6	182	238	157	515
	32-bit integer to double-precision	6	150	139	123	510
	64-bit integer to single-precision	6	385	371	296	336
	64-bit integer to single-precision	6	393	461	344	336

Operation	Format	Pipeline	Logic Usage			f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	
Float-to-Integer	Single-precision to 32-bit integer	6	256	255	176	455
	Single-precision to 64-bit integer	6	417	361	257	311
	Double-precision to 32-bit integer	6	406	387	273	409
	Double-precision to 64-bit integer	6	535	480	362	309
Float-to-Float	Single-precision to double-precision	2	44	73	40	868
	Double-precision to single-precision	3	103	140	89	520

Operation	Format	Pipeline	Logic Usage			f_{MAX} (MHz)
			Adaptive Look-Up Tables (ALUTs)	Dedicated Logic Registers (DLRs)	Adaptive Logic Modules (ALMs)	
Fixed-to-Float	16.16 fixed-point to double-precision	6	182	238	155	519
	16.16 fixed-point to double-precision	6	150	139	122	513
	32.32 fixed-point to single-precision	6	384	371	296	334
	32.32 fixed-point to single-precision	6	393	461	336	333
Float-to-Fixed	Single-precision to 16.16 fixed-point	6	319	261	210	438
	Single-precision to 32.32 fixed-point	6	469	367	288	315
	Double-precision to 16.16 fixed-point	6	579	393	402	365
	Double-precision to 32.32 fixed-point	6	695	486	474	306

ALTFP_CONVERT Design Example: Convert Double-Precision Floating-Point Format Numbers

This design example uses the ALTFP_CONVERT IP core to convert double-precision floating-point format numbers to 64-bit integers. This design example uses the parameter editor in the Quartus II software.

Related Information

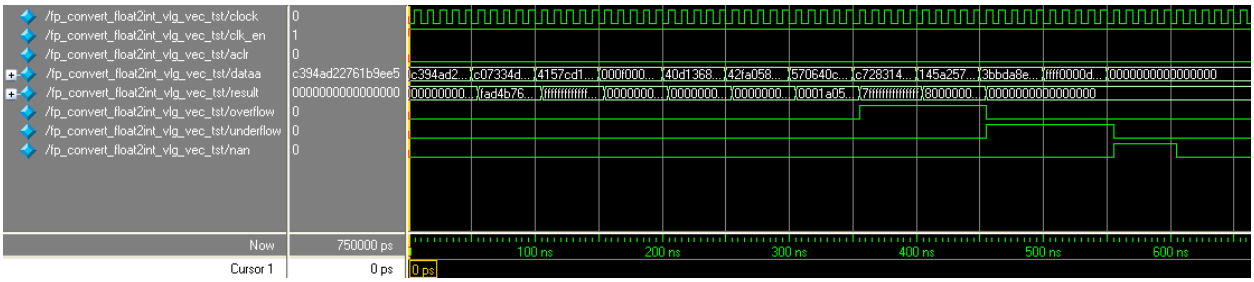
- **Floating-Point IP Cores Design Example Files** on page 1-19
- **Floating-Point IP Cores Design Examples**
  Provides the design example files for the Floating-Point IP cores
- **ModelSim-Altera Software Support**
  Provides information about installation, usage, and troubleshooting

ALTFP_CONVERT Design Example: Understanding the Simulation Results

The simulation waveform in this design example is not shown in its entirety. Run the design example files in the ModelSim-Altera software to see the complete simulation waveforms.

This figure shows the expected simulation results in the ModelSim-Altera software.

Figure 17-1: ALTFP_CONVERT Simulation Waveform



This design example implements a float-to-integer converter for converting double-precision floating-point format 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 six clock cycles. Therefore, each conversion generates the output result six clock cycles after receiving the input value.

This table lists the inputs and corresponding outputs obtained from the simulation in the waveform.

Table 17-5: Summary of Input Values and Corresponding Outputs

Time	Event
0 ns, start-up	`dataa[]` value: C394 AD22 761B 9EE5h Output value: The `result[]` port displays 0 regardless of what the input value is. This value seen on the output port before the 6th clock cycle is merely due to the behavior of the system during start-up and should be disregarded.
55 ns	Output value: FAD4 B762 7918 46C0h
150 ns	`dataa[]` value: 000F 0000 5555 1111h This value is a denormal number.
205 ns	Denormal inputs are not supported and are forced to zero before conversion takes place.
300 ns	`dataa[]` value: 5706 40CF OEC6 1176h
355 ns	Output value: 7FFF FFFF FFFF FFFFh Exception handling ports: `overflow` asserts. 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.
350 ns	`dataa[]` value: C728 3147 8444 1F75h
405 ns	Output value: 8000 0000 0000 0000h Exception handling ports: `overflow` remains asserted. This is a standard value to represent a negative overflow number.
400 ns	`dataa[]` value: 145A 257C 895A B309h
455 ns	Output value: 0000 0000h Exception handling ports: `underflow` asserts. The input value triggers the `underflow` port because the exponent of the input value is less than the exponent bias of 1023.
500 ns	`dataa[]` value: FFFF 0000 DDDD 5555h This value is a NaN.
555 ns	Output value: 0000 0000h Exception handling ports: `nan` asserts.

ALTFP_CONVERT Signals

Figure 17-2: ALTFP_CONVERT Signals

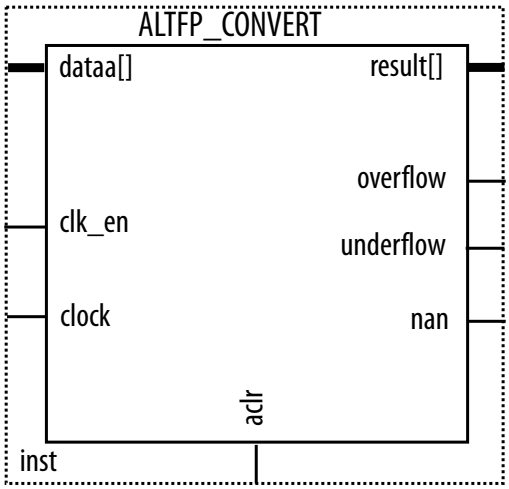


Table 17-6: ALTFP_CONVERT Input Signals

Port Name	Required	Description
`clock`	Yes	The clock input to the ALTFP_CONVERT IP core.
`clk_en`	No	Clock enable that allows conversions to take place when asserted high. When asserted low, no operation occurs and the outputs are unchanged.
`aclr`	No	Asynchronous clear. The source is asynchronously reset when the `aclr` signal is asserted high.
`dataa[]`	Yes	Data input. The size of this input port depends on the `WIDTH_DATA` parameter value. If the operation mode value is `INT2FLOAT` or `FIXED2FLOAT`, the data on the input bus is an integer. If the operation mode value is `FLOAT2INT` or `FLOAT2FIXED`, the input bus is the IEEE floating-point representation. In the single-precision format, the input bus width value is `32`. In the double-precision format, the input bus width value is `64`. In the single-extended precision format, the input bus range is from `43` to `64`. If the operation mode value is `FLOAT2FLOAT`, the input bus value is the IEEE floating-point representation. In the single-precision format, the input bus width value is `32`. In the double-precision format, the input bus width value is `64`. In the single-extended precision format, the input bus range is from `43` to `64`.

Table 17-7: ALTFP_CONVERT Output Signals

Port Name	Required	Description
`result[]`	Yes	Output for the floating-point converter. The size of this output port depends on the `WIDTH_RESULT` parameter value. If the operation mode value is `FLOAT2INT` or `FLOAT2FIXED`, the output bus is an IEEE floating-point representation. If the operation mode is `FLOAT2INT`, the output bus is an integer representation. If the selected precision is the single-precision format, the output bus width value is `32`. If the selected precision is the double-precision format, the output bus width value is `64`. If the selected precision is the single-extended precision format, the input bus range is from `43` to `64`. If the operation mode value is `FLOAT2FLOAT`, the output bus is an IEEE floating-point representation. If the selected precision is the single-precision format, the output bus is in the 64-bit double-precision format. If the selected precision is the double-precision format, the output bus is in the 32-bit single-precision format. If the selected precision is the single-extended precision format, the output bus ranges from `43` to `64`.
`overflow`	No	Optional overflow exception output. This port is available only when the operation mode values are `FLOAT2FIXED`, `FLOAT2INT`, or `FLOAT2FLOAT`. Asserted when the result of the conversion (after rounding), exceeds the maximum width of the `result[]` port, or when the `dataa[]` input is infinity.
`underflow`	No	Optional underflow exception output. This port is available only when the operation mode values are `FLOAT2FIXED`, `FLOAT2INT`, or `FLOAT2FLOAT`. Asserted when the result of the conversion, after rounding, is fractional. In `FLOAT2INT` operations, this port is asserted when the exponent value of the floating-point input is smaller than the exponent bias. In `FLOAT2FLOAT` operations, this port is asserted when the floating-point input has a value smaller than the lowest exponent limit of the target floating-point format.
`nan`	No	Optional NaN exception output. This port is available only when the operation mode values are `FLOAT2INT`, `FLOAT2FLOAT`, or `FLOAT2FIXED`. Asserted when the input port is a NaN representation. If the operation mode value is `FLOAT2INT` or `FLOAT2FIXED`, the `result[]` port is set to zero. If the operation mode value is `FLOAT2FLOAT`, the `result[]` port is set to a NaN representation.

ALTFP_CONVERT Parameters

Table 17-8: ALTFP_CONVERT Parameters

Port Name	Type	Required	Description
WIDTH_EXP_INPUT	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is 8. The bias of the exponent is always set to 2 (WIDTH_EXP - 1) - 1, that is, 127 for the single-precision format and 1023 for the double-precision format. The value of WIDTH_EXP_INPUT must be 8 for the single-precision format, 11 for the double-precision format, and a minimum of 11 for the single-extended precision format. 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 only to the FLOAT2FIXED, FLOAT2INT, and FLOAT2FLOAT operation modes.
WIDTH_MAN_INPUT	Integer	Yes	Specifies the precision of the mantissa. If this parameter is not specified, 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 only to the FLOAT2FIXED, FLOAT2INT, and FLOAT2FLOAT operation modes.
WIDTH_INT	Integer	Yes	Specifies the integer width. If the operation is FIXED2FLOAT or INT2FLOAT, this parameter defines the integer width on the input side. If the operation is FLOAT2INT or FLOAT2FIXED, this parameter defines the result width on the output side. The available settings are 32 bits, 64 bits or n bits. For n bits settings, the range is from 4 bits to 64 bits. If unspecified, the default setting for WIDTH_INT is 32 bits.

Port Name	Type	Required	Description
WIDTH_DATA	Integer	Yes	Specifies the input data width. If the operation is INT2FLOAT, the WIDTH_DATA is also WIDTH_INT. If the operation is FIXED2FLOAT, the data width value is WIDTH_INT + fractional width. If the operation is FLOAT2FIXED, FLOAT2INT or FLOAT2FLOAT, the data width value is WIDTH_EXP_INPUT + WIDTH_MAN_INPUT + 1. The available settings are 32 bits, 64 bits or n bits. For n bits settings, the range is from 4 bits to 64 bits. If unspecified, the default setting for WIDTH_DATA is 32 bits.
WIDTH_EXP_OUTPUT	Integer	Yes	Specifies the precision of the exponent. If this parameter is not specified, the default is 8. The bias of the exponent is always set to 2 (WIDTH_EXP - 1) - 1, that is, 127 for the single-precision format and 1023 for the double-precision format. The value of WIDTH_EXP_OUTPUT must be 8 for the single-precision format, 11 for the double-precision format, and a minimum of 11 for the single-extended precision format. 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 only to the FLOAT2FIXED, FLOAT2INT, and FLOAT2FLOAT operation modes.
WIDTH_MAN_OUTPUT	Integer	Yes	Specifies the precision of the mantissa. If this parameter is not specified, 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 only to the FLOAT2FIXED, FLOAT2INT, and FLOAT2FLOAT operation modes.
WIDTH_RESULT	Integer	Yes	Specifies the width of the output result. In an INT2FLOAT, FLOAT2FLOAT, or FIXED2FLOAT operation, the result width is WIDTH_EXP_OUTPUT + WIDTH_MAN_OUTPUT + 1. In a FLOAT2INT operation, the result width is the value of the WIDTH_INT parameter. In a FLOAT2FIXED operation, this parameter is the result width. The available settings are 32 bits, 64 bits or n bits. For n bits settings, the range is from 4 bits to 64 bits.

Port Name	Type	Required	Description
ROUNDING	Integer	Yes	Specifies the rounding mode. The default value is TO_NEAREST. Other modes are not supported.
OPERATION	Integer	Yes	Specifies the operating mode. Values are INT2FLOAT, FLOAT2INT, FLOAT2FLOAT, FLOAT2FIXED, and FIXED2FLOAT. If this parameter is not specified, the default value is INT2FLOAT. When set to INT2FLOAT, the conversion of an integer input to an IEEE floating-point representation output takes place. When set to FLOAT2INT, the conversion of an IEEE floating-point representation input to an integer output takes place. When set to FLOAT2FLOAT, the conversion between IEEE floating-point representations input and output takes place. When set to FIXED2FLOAT, the conversion of a fixed point input to an IEEE floating-point representation output takes place. When set to FLOAT2FIXED, the IEEE floating-point input conversion to fixed point representation output takes place.

ALTERA_FP_FUNCTIONS IP Core 18

2016.12.09

UG-01058 Subscribe Send Feedback

This IP core is only available in Arria 10 devices. It replaces all the functions supported by the existing floating-point IP cores shown in the previous chapters in this document, starting from Quartus II software version 14.0.

Table 18-1: List of Functions Supported by ALTERA_FP_FUNCTIONS

Function	Description
Arithmetic	
Add	Two input addition
Sub	Two input subtraction
Add/Sub	Two input addition and subtraction. The IP core provides both addition and subtraction outputs and an option to generate a `select` signal to dynamically select the desired operation.
Multiply	Two input multiplication
Divide	Two input division
Reciprocal	Performs the function of 1/a where a is the input. **Note:** This function replaces the ALTFP_INV IP core in Arria 10 devices.
Absolute	Generates absolute value of the input
Scalar Product	Performs addition of an arbitrary number if inputs
Multiply-Accumulate	Two input multiplication followed by a single cycle accumulation
Accumulate	Perform single input accumulation in a single cycle
Multiply-Add	Performs two input multiplication followed by addition
Complex-Multiply	Peforms multiplication of two complex value
Roots	
Square Root	Performs square root to the input value

Function	Description
Reciprocal Square Root	Performs the function of 1/√a where a is the input **Note:** This function replaces the ALTFP_INV_SQRT IP core in Arria 10 devices.
Cube Root	Performs cube root to the input value
3D Hypotenuse	Performs the function of Q=√(a^2+b^2+c^2)
Conversions	
Fixed-to-Floating Point	Converts a fixed point input to floating point representation
Floating Point-to-Fixed	Converts a floating-point input to fixed point representation
Floating to Floating Point	Converts a floating-point input to floating-point representation of a different precision
Comparisons	
Minimum	Compares and output the smallest value of two input
Maximum	Compares and output the biggest value of two input
Less Than	Compares and returns true if input a is less than input b
Less Than or Equal	Compares and returns true if input a is less than or equal to input b
Equal	Compares and returns true if input a is equal to input b
Greater Than	Compares and returns true if input a is greater than input b
Greater Than or Equal	Compares and returns true if input a is greater than or equal to input b
Not Equal	Compares and returns true if input a is not equal to input b
Exp/Log/Pow	
Exponent	Performs the function of e^a where a is the input
Exponent base 2	Performs the function of 2^a where a is the input
Exponent base 10	Performs the function of 10^a where a is the input
Log	Performs the function of $\log_e(a)$ where a is the input
Log_2	Performs the function of $\log_2(a)$ where a is the input
Log_{10}	Performs the function of $\log_{10}(a)$ where a is the input
Log(1+x)	Performs the function of $\log_e(1+a)$ where a is the input
Power	
LdExp	Sets the exponential value of a floating-point input
Trigonometry	
Sin	Performs sine function of a single input
Cos	Performs cosine function of a single input

Function	Description
Tan	Performs tangent function of a single input
Arcsin	Performs arc sine function of a single input
Arccos	Performs arc cosine function of a single input
Arctan	Performs arc tangent function of a single input
Arctan2	Performs the function of arctan (a/b) where a and b are the inputs

ALTERA_FP_FUNCTIONS Features

The ALTERA_FP_FUNCTIONS IP core offers the following features:

- Supports both latency and frequency driven cores.
- Supports VHDL code generation.

ALTERA_FP_FUNCTIONS Output Latency

If you require a specific latency, follow these steps:

1. In the ALTERA_FP_FUNCTIONS parameter editor, click the **Basic** tab.
2. Under the Performance category, in the **Goal** option, select **latency**.
3. In the **Target** field, set your desired latency (cycles).
4. Then, click **Check Performance**.

ALTERA_FP_FUNCTIONS Target Frequency

If you require a specific frequency, follow these steps:

1. In the ALTERA_FP_FUNCTIONS parameter editor, click **Basic** tab.
2. Under the Performance category, in the **Goal** option, select **frequency**.
3. In the **Target** field, set your desired frequency (MHz).
4. The IP core reports the latency for the instance that it will generate in the Report category.

   Note: You must verify the frequency by running the TimeQuest Timing Analyzer.

ALTERA_FP_FUNCTIONS Combined Target

If you require a combined target of latency and frequency, follow these steps:

1. In the ALTERA_FP_FUNCTIONS parameter editor, click the **Basic** tab.
2. Under the Performance category, in the **Goal** option, select **Combined**.
3. In the **Target** field, set your desired frequency (MHz).
4. In the **Target** field, set your desired latency (cycles).
5. Then, click **Finish**.

ALTERA_FP_FUNCTIONS Resource Utilization and Performance

These tables list the resource utilization and performance information for the ALTERA_FP_FUNCTIONS IP core. The information was derived using the Quartus II software version 14.1. The frequency target was set to 200 MHz.

Table 18-2: Arithmetic

Family	Function	Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
									Primary	Secondary
Arria V (5AGXFB3H4F40C5)	Abs	Single	0	—	33	0	—	0	0	0
		Double	0	—	65	0	—	0	0	0
	Add	Single	9	233.1	360	0	—	0	507	29
		Double	12	251.95	886	0	—	0	1064	61
	AddSubtract	Single	9	249.31	477	0	0	0	651	63
		Double	12	252.46	1161	0	0	0	1713	91
	Cube Root	Single	9	275.18	132	6	—	2	132	20
		Double	24	185.77	634	17	—	10	1297	58
	Divide	Single	18	249	456	5	—	4	771	100
		Double	35	185.29	1409	39	—	15	3035	138
	Exp base 10	Single	16	212.72	547	3	—	2	675	18
		Double	31	185.77	2194	0	—	10	2626	56
Arria V (5AGXFB3H4F40C5)	Exp base 2	Single	7	236.41	345	0	—	2	214	19
		Double	21	185.84	932	0	—	10	1324	51
	Exp base e	Single	14	217.96	718	0	—	2	597	46
		Double	28	185.87	2134	0	—	10	2398	46
	Reciprocal	Single	12	253.16	210	4	—	3	294	26
		Double	30	185.29	877	9	—	14	1764	105
	Reciprocal Square Root	Single	7	267.52	118	4	—	2	141	14
		Double	20	185.74	539	13	—	9	1210	52
	LDExp	Single	2	367.92	69	0	—	0	85	0
		Double	2	359.32	100	0	—	0	146	0
Arria V (5AGXFB3H4F40C5)	Log base 10	Single	16	250	379	4	--	3	622	65
		Double	34	186.12	1,380	40	--	11	3,025	143

Family	Function	Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
									Primary	Secondary
Arria V (5AGXFB3H4F40C5)	Log(1+x)	Single	21	222.77	766	4	--	3	1,171	82
		Double	43	185.94	2,361	40	--	14	4,702	183
	Log base 2	Single	16	232.29	350	4	--	3	584	64
		Double	37	185.32	1,342	13	--	17	3,156	121
	Log base e	Single	16	248.57	379	4	--	3	616	57
		Double	35	185.84	1,422	40	--	13	3,066	147
	Multiply	Single	5	281.14	156	0	--	1	152	6
		Double	7	186.01	339	0	--	4	549	13
Arria V (5AGXFB3H4F40C5)	Power	Single	45	201.82	1,347	11	--	14	2,410	165
		Double	82	185.43	4,195	20	--	38	8,149	266
	Square Root	Single	8	261.92	119	3	--	2	174	13
		Double	21	185.94	548	8	--	9	1,225	44
	Subtract	Single	9	232.67	363	0	--	0	505	32
		Double	12	257.07	884	0	--	0	1,064	61
Cyclone V (5CGXFC7D6F31C7)	Abs	Single	0	--	33	0	--	0	0	0
		Double	0	--	65	0	--	0	0	0
	Add	Single	12	225.94	403	0	--	0	562	35
		Double	20	208.99	932	0	--	0	1,813	72
	AddSubtract	Single	12	224.67	509	0	--	0	805	65
		Double	20	211.55	1,197	0	--	0	2,647	120
	Cube Root	Single	10	230.47	131	6	--	2	213	11
		Double	34	212.49	890	17	--	10	1,991	54
	Divide	Single	20	232.61	466	5	--	4	991	62
		Double	51	201.01	1,782	41	--	15	4,317	165

Family	Function	Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
									Primary	Secondary
Cyclone V (5CGXFC7D6F31C7)	Exp base 10	Single	20	217.58	552	3	--	2	905	32
		Double	52	212.77	2,317	0	--	10	4,287	122
	Exp base 2	Single	9	211.33	352	0	--	2	314	13
		Double	36	219.3	1,128	0	--	10	2,364	87
	Exp base e	Single	17	207.68	698	0	--	2	860	31
		Double	50	198.85	2,309	0	--	10	4,300	126
	Reciprocal	Single	14	230.95	245	4	--	3	378	26
		Double	44	207.43	1,201	9	--	14	2,694	94
	Reciprocal Square Root	Single	9	233.37	137	4	--	2	223	25
		Double	30	250	782	13	--	9	1,932	46
Cyclone V (5CGXFC7D6F31C7)	LDExp	Single	2	346.02	69	0	--	0	87	1
		Double	3	357.91	104	0	--	0	215	0
	Log base 10	Single	22	203.33	486	4	--	3	1,066	47
		Double	49	196.97	1,888	40	--	11	4,483	153
	Log(1+x)	Single	29	191.5	944	4	--	3	1,844	105
		Double	62	168.27	3,012	40	--	14	6,899	210
	Log base 2	Single	20	202.1	413	4	--	3	918	50
		Double	54	194.21	1,898	13	--	17	4,732	151
Cyclone V (5CGXFC7D6F31C7)	Log base e	Single	22	181.42	482	4	--	3	1,058	45
		Double	50	196.27	1,941	40	--	13	4,611	197
	Multiply	Single	6	268.6	159	0	--	1	223	2
		Double	11	205.17	431	0	--	4	970	18
	Power	Single	62	181.19	1,778	11	--	14	3,562	154
		Double	127	186.53	5,411	22	--	38	12,361	325
	Square Root	Single	8	219.15	126	3	--	2	205	12
		Double	31	250	822	8	--	9	2,056	55
	Subtract	Single	12	232.07	399	0	--	0	566	42
		Double	20	204.25	918	0	--	0	1,839	60

Family	Function	Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
									Primary	Secondary
Stratix V (5SGXEA7K 2F40C2)	Abs	Single	0	--	33	--	0	0	0	0
		Double	0	--	65	--	0	0	0	0
	Add	Single	5	364.83	366	--	0	0	299	19
		Double	7	329.49	834	--	0	0	801	53
	AddSubtract	Single	5	354.74	489	--	0	0	411	29
		Double	7	338.41	1,106	--	0	0	1,039	134
	Cube Root	Single	8	420.17	114	--	5	2	124	11
		Double	20	277.7	520	--	11	10	997	17
	Divide	Single	13	363.5	377	--	3	4	591	71
		Double	23	270.86	1,091	--	20	15	2,274	120
	Exp base 10	Single	11	292.4	486	--	3	2	417	12
		Double	22	271.74	2,033	--	0	10	1,761	48
Stratix V (5SGXEA7K 2F40C2)	Exp base 2	Single	5	387.3	351	--	0	2	160	1
		Double	17	279.56	897	--	0	10	995	27
	Exp base e	Single	8	284.09	653	--	0	2	350	18
		Double	23	268.38	2,043	--	0	10	1,710	44
	Reciprocal	Single	9	279.33	199	--	3	3	211	13
		Double	22	241.31	764	--	9	14	1,391	49
	Reciprocal Square Root	Single	6	420.52	105	--	3	2	129	9
		Double	17	271.37	449	--	8	9	1,009	47
	LDExp	Single	0	--	67	--	0	0	0	0
		Double	0	717.36	99	--	0	0	66	0
	Log base 10	Single	11	359.58	358	--	3	3	443	29
		Double	23	271.96	1,077	--	20	11	2,252	101

Family	Function	Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
									Primary	Secondary
Stratix V (5SGXEA7K2F40C2)	Log(1+x)	Single	15	338.64	748	--	3	3	905	55
		Double	27	280.98	1,911	--	20	13	3,301	122
	Log base 2	Single	11	340.37	304	--	3	3	392	15
		Double	27	258.33	1,053	--	8	16	2,241	110
	Log base e	Single	11	351.86	359	--	3	3	439	35
		Double	23	270.49	1,071	--	20	13	2,210	94
	Multiply	Single	3	399.52	136	--	0	1	72	1
		Double	4	250.75	312	--	0	4	237	5
	Power	Single	31	261.23	1,171	--	8	12	1,492	83
		Double	60	267.81	3,555	--	13	37	5,347	244
	Square Root	Single	6	393.7	112	--	3	2	129	7
		Double	17	274.12	458	--	8	9	1,019	41
	Subtract	Single	5	320.41	360	--	0	0	299	14
		Double	7	338.52	835	--	0	0	801	51
Arria 10 (10AX115H4F34I3SP)	Abs	Single	0	--	33	--	0	0	0	0
		Double	0	--	65	--	0	0	0	0
	Add	Single	4	296.4	49	--	0	1	0	0
		Double	7	296.3	840	--	0	0	779	67
	AddSubtract	Single	5	319.39	483	--	0	0	408	37
		Double	7	289.77	1,106	--	0	0	1,006	156
	Cube Root	Single	10	432.9	126	--	5	2	121	0
		Double	24	282.09	594	--	11	10	1,155	29
	Divide	Single	16	347.34	394	--	3	4	561	66
		Double	30	258.26	1,208	--	20	15	2,175	136
	Exp base 10	Single	14	271.37	502	--	3	2	432	40
		Double	29	242.42	2,185	--	0	10	1,683	90

Family	Function	Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
									Primary	Secondary
Arria 10 (10AX115H4F34I3SP)	Exp base 2	Single	7	317.86	370	--	0	2	124	9
		Double	22	251.45	906	--	0	10	1,172	47
	Exp base e	Single	26	365.36	298	--	3	6	137	11
		Double	28	260.42	2,156	--	0	10	1,724	93
	Reciprocal	Single	12	278.94	225	--	3	3	172	3
		Double	27	260.89	824	--	9	14	1,448	100
	Reciprocal Square Root	Single	8	418.94	117	--	3	2	130	1
		Double	22	243.43	523	--	8	9	950	37
	LDExp	Single	0	--	68	--	0	0	0	0
		Double	0	--	99	--	0	0	66	0
	Log base 10	Single	15	293.69	364	--	3	3	441	42
		Double	28	272.03	1,158	--	20	11	2,095	214
	Log(1+x)	Single	18	301.3	747	--	3	3	882	79
		Double	32	251.95	2,018	--	20	13	3,019	248
Arria 10 (10AX115H4F34I3SP)	Log base 2	Single	14	275.79	316	--	3	3	402	3
		Double	32	271.96	1,173	--	8	16	2,372	132
	Log base e	Single	29	378.07	297	--	3	9	315	6
		Double	29	256.54	1,219	--	20	13	2,338	152
	Multiply	Single	3	288.4	49	--	0	1	0	0
		Double	5	288.35	312	--	0	4	236	26
	Power	Single	40	262.12	1,335	--	8	14	1,523	127
		Double	73	237.7	3,957	--	13	37	5,362	305
	Square Root	Single	8	432.9	124	--	3	2	118	8
		Double	22	249.25	539	--	8	9	1,000	34
	Subtract	Single	4	296.9	49	--	0	1	0	0
		Double	7	296.82	842	--	0	0	783	76

Table 18-3: Trigonometry

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Arria V (5AGXFB3H4F40C5)	Arccos	Single	0	35	217.77	768	9	--	8	1,289	94
		Single	1	39	216.45	819	9	--	9	1,383	92
		Double	0	76	185.7	2,917	27	--	37	6,489	230
		Double	1	83	184.2	3,120	27	--	40	6,899	198
	Arcsin	Single	0	29	215.8	652	9	--	8	1,069	93
		Single	1	34	222.32	747	9	--	9	1,178	80
		Double	0	66	185.32	2,762	29	--	41	6,365	171
		Double	1	72	184.16	2,963	29	--	44	6,696	200
	Arctan	Single	0	27	232.29	603	7	--	6	937	77
		Single	1	31	230.73	664	7	--	7	1,034	89
		Double	0	65	185.7	2,047	23	--	31	4,535	164
		Double	1	71	185.6	2,229	23	--	34	4,854	174

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Arria V (5AGXFB3H4F40C5)	Arctan2	Single	0	43	230.2	1,013	11	--	9	1,719	128
		Single	1	43	230.2	1,013	11	--	9	1,719	128
		Double	0	92	184.2	3,195	44	--	43	6,822	285
		Double	1	92	184.2	3,195	44	--	43	6,822	285
	Cos	Single	0	25	205.3	768	5	--	6	1,563	120
		Single	1	12	242.13	490	0	--	3	475	36
		Double	0	45	184.2	2,879	34	--	33	5,973	244
		Double	1	29	185.87	1,719	0	--	13	2,499	92
Arria V (5AGXFB3H4F40C5)	Sin	Single	0	26	223.51	964	5	--	6	1,439	110
		Single	1	12	240.56	585	0	--	3	563	66
		Double	0	46	184.16	3,019	36	--	33	6,308	249
		Double	1	29	185.77	1,748	0	--	14	2,699	92
	Tan	Single	0	38	221.78	1,368	12	--	12	2,625	163
		Single	1	25	231.43	1,297	4	--	10	1,512	140
		Double	0	68	185.56	5,211	56	--	65	10,670	530
		Double	1	52	184.16	3,874	26	--	43	6,896	238

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Cyclone V (5CGXFC7D6F31C7)	Arccos	Single	0	42	217.2	857	9	--	8	1,701	107
		Single	1	47	196.27	943	9	--	9	1,887	104
		Double	0	113	196.5	3,957	31	--	37	9,739	343
		Double	1	123	210.17	4,218	31	--	40	10,353	333
	Arcsin	Single	0	35	222.62	757	9	--	8	1,464	65
		Single	1	40	215.19	844	9	--	9	1,627	111
		Double	0	101	201.65	3,816	31	--	41	9,709	334
		Double	1	112	197.71	4,046	31	--	44	10,383	260
	Arctan	Single	0	33	227.58	706	7	--	6	1,266	92
		Single	1	38	206.31	787	7	--	7	1,434	90
		Double	0	98	188.79	2,920	24	--	31	7,154	297
		Double	1	109	180.41	3,196	24	--	34	7,875	297

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Cyclone V (5CGXFC7D6F31C7)	Arctan2	Single	0	51	206.14	1,153	11	--	9	2,013	149
		Single	1	51	206.14	1,153	11	--	9	2,013	149
		Double	0	144	191.17	4,589	46	--	43	10,740	417
		Double	1	144	191.17	4,589	46	--	43	10,740	417
	Cos	Single	0	32	174.67	959	5	--	6	2,258	110
		Single	1	15	212.99	517	0	--	3	702	25
		Double	0	75	182.75	3,751	34	--	33	9,177	352
		Double	1	50	212.68	1,985	0	--	13	3,914	169
	Sin	Single	0	33	191.61	1,086	5	--	6	2,394	132
		Single	1	14	207.81	579	0	--	3	783	39
		Double	0	75	196.39	3,787	38	--	33	9,545	284
		Double	1	49	206.53	2,165	0	--	14	4,336	177
	Tan	Single	0	46	185.74	1,655	12	--	12	3,738	200
		Single	1	29	205.47	1,283	4	--	10	2,142	102
		Double	0	112	194.7	7,052	58	--	65	16,793	607
		Double	1	89	197.24	5,327	26	--	43	11,741	376

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Stratix V (5SGXEA7K2F40C2)	Arccos	Single	0	23	291.46	753	--	9	8	801	34
		Single	1	27	288.43	823	--	9	9	891	27
		Double	0	53	247.46	2,380	--	27	37	4,435	145
		Double	1	58	233.15	2,570	--	27	40	4,717	121
	Arcsin	Single	0	20	290.61	598	--	9	8	698	19
		Single	1	23	294.99	678	--	9	9	800	21
		Double	0	47	237.25	2,235	--	27	40	4,407	89
		Double	1	52	240.33	2,411	--	27	43	4,621	134
	Arctan	Single	0	20	293.6	544	--	6	6	646	53
		Single	1	23	290.7	620	--	6	7	715	50
		Double	0	47	241.72	1,837	--	18	30	3,424	145
		Double	1	52	247.34	2,002	--	18	33	3,654	126
	Arctan2	Single	0	31	288.35	890	--	9	9	1,277	71
		Single	1	31	288.35	890	--	9	9	1,277	71
		Double	0	69	239.23	2,983	--	29	42	5,530	212
		Double	1	69	239.23	2,983	--	29	42	5,530	212

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Stratix V (5SGXEA7K2F40C2)	Cos	Single	0	17	267.02	711	--	5	6	890	48
		Single	1	8	364.83	452	--	0	3	368	20
		Double	0	33	242.25	2,529	--	17	31	4,510	187
		Double	1	21	275.48	1,608	--	0	13	1,799	54
	Sin	Single	0	18	309.69	856	--	5	6	917	74
		Single	1	8	317.46	538	--	0	3	382	10
		Double	0	34	257.67	2,714	--	19	31	4,766	229
		Double	1	22	260.89	1,864	--	0	14	1,898	104
	Tan	Single	0	27	272.26	1,312	--	11	12	1,675	117
		Single	1	17	295.68	1,164	--	3	10	1,175	65
		Double	0	52	268.38	4,612	--	30	60	8,152	264
		Double	1	41	260.89	3,886	--	13	43	5,294	193

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Arria 10 (10AX115H4F34I3SP)	Arccos	Single	0	28	270.42	703	--	9	8	656	29
		Single	1	31	261.23	705	--	9	9	624	19
		Double	0	63	257.8	2,626	--	27	37	4,917	241
		Double	1	69	255.62	2,813	--	27	40	5,127	268
	Arcsin	Single	0	25	249.63	665	--	9	8	659	18
		Single	1	28	254.19	673	--	9	9	649	30
		Double	0	57	255.62	2,440	--	29	40	4,750	213
		Double	1	62	251.51	2,596	--	29	43	4,985	190
	Arctan	Single	0	26	271.3	600	--	6	6	578	32
		Single	1	29	274.2	594	--	6	7	583	22
		Double	0	57	254.13	1,866	--	22	30	3,654	171
		Double	1	63	258.33	2,043	--	22	33	3,726	253
	Arctan2	Single	0	40	248.14	1,002	--	9	9	1,258	85
		Single	1	40	248.14	1,002	--	9	9	1,258	85
		Double	0	84	255.1	3,025	--	33	42	5,675	328
		Double	1	84	255.1	3,025	--	33	42	5,675	328

Family	Function	Precision	Scale By Pi	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Arria 10 (10AX115H4F34I3SP)	Cos	Single	0	21	336.93	786	--	5	6	979	154
		Single	1	11	310.37	512	--	0	3	297	22
		Double	0	39	263.92	2,702	--	17	33	3,697	375
		Double	1	29	242.19	1,698	--	0	13	2,030	62
	Sin	Single	0	22	311.33	876	--	5	6	1,003	116
		Single	1	11	279.02	585	--	0	3	330	19
		Double	0	41	265.25	2,791	--	19	33	3,902	334
		Double	1	29	259.61	1,918	--	0	14	1,943	72
	Tan	Single	0	34	265.6	1,359	--	11	12	1,756	155
		Single	1	23	265.89	1,268	--	3	10	1,065	94
		Double	0	64	248.14	5,107	--	30	65	7,578	458
		Double	1	53	251.7	4,002	--	17	43	5,619	343

Table 18-4: FPFXP

Family	Input Precision	Output Width	Output Fraction	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Arria V (5AGXFB3H4F40C5)	Single	32	0	2	277.93	168	0	--	0	75	1
		32	16	2	266.1	169	0	--	0	75	0
		32	32	2	277.93	168	0	--	0	75	1
		64	0	3	226.4	291	0	--	0	172	0
		64	16	3	226.4	291	0	--	0	172	0
		64	32	3	226.4	291	0	--	0	172	0
	Double	32	0	3	332.12	197	0	--	0	115	0
		32	16	3	344.12	197	0	--	0	115	0
		32	32	3	332.12	197	0	--	0	115	0
		64	0	3	256.28	326	0	--	0	205	4
		64	16	3	256.28	326	0	--	0	205	4
		64	32	3	256.28	326	0	--	0	205	4
Cyclone V (5CGXFC7D6F31C7)	Single	32	0	3	245.04	171	0	--	0	110	0
		32	16	3	245.04	171	0	--	0	110	0
		32	32	3	245.04	171	0	--	0	110	0
		64	0	4	190.62	244	0	--	0	269	0
		64	16	4	190.62	244	0	--	0	269	0
		64	32	4	190.62	244	0	--	0	269	0
	Double	32	0	4	291.63	209	0	--	0	160	1
		32	16	4	302.94	209	0	--	0	160	1
		32	32	4	291.63	209	0	--	0	160	1
		64	0	5	207.25	329	0	--	0	347	2
		64	16	5	207.25	329	0	--	0	347	2
		64	32	5	207.25	329	0	--	0	347	2

Family	Input Precision	Output Width	Output Fraction	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Stratix V (5SGXEA7K2F40C2)	Single	32	0	0	717.36	168	--	0	0	38	0
		32	16	0	717.36	168	--	0	0	38	0
		32	32	0	717.36	168	--	0	0	38	0
		64	0	0	717.36	304	--	0	0	70	0
		64	16	0	717.36	304	--	0	0	70	0
		64	32	0	717.36	304	--	0	0	70	0
	Double	32	0	0	717.36	204	--	0	0	38	0
		32	16	0	717.36	204	--	0	0	38	0
		32	32	0	717.36	204	--	0	0	38	0
		64	0	2	456	329	--	0	0	134	1
		64	16	2	456	329	--	0	0	134	1
		64	32	2	456	329	--	0	0	134	1
Arria 10 (10AX115H4F34I3SP)	Single	32	0	0	--	168	--	0	0	38	0
		32	16	0	--	168	--	0	0	38	0
		32	32	0	--	168	--	0	0	38	0
		64	0	0	--	304	--	0	0	70	0
		64	16	0	--	304	--	0	0	70	0
		64	32	0	--	304	--	0	0	70	0
	Double	32	0	0	--	203	--	0	0	38	0
		32	16	0	--	203	--	0	0	38	0
		32	32	0	--	203	--	0	0	38	0
		64	0	2	407.33	328	--	0	0	134	0
		64	16	2	407.33	328	--	0	0	134	0
		64	32	2	407.33	328	--	0	0	134	0

Table 18-5: FXPFP

Family	Input Width	Input Fraction	Output Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Arria V (5AGXFB3H4F40C5)	32	0	Single	6	283.61	154	0	--	0	195	14
	32	0	Double	5	328.19	165	0	--	0	180	17
	32	16	Single	6	283.61	154	0	--	0	195	14
	32	16	Double	5	328.19	165	0	--	0	180	17
	32	32	Single	6	293	152	0	--	0	193	13
	32	32	Double	5	336.59	159	0	--	0	180	16
	64	0	Single	7	282.01	217	0	--	0	297	16
	64	0	Double	7	256.48	330	0	--	0	451	18
	64	16	Single	7	282.01	217	0	--	0	297	16
	64	16	Double	7	256.48	330	0	--	0	451	18
	64	32	Single	7	282.01	217	0	--	0	297	16
	64	32	Double	7	256.48	330	0	--	0	451	18

Family	Input Width	Input Fraction	Output Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Cyclone V (5CGXFC7D6F31C7)	32	0	Single	8	230.04	168	0	--	0	264	21
	32	0	Double	7	292.74	180	0	--	0	258	23
	32	16	Single	8	230.04	168	0	--	0	264	21
	32	16	Double	7	292.74	180	0	--	0	258	23
	32	32	Single	8	237.14	166	0	--	0	262	20
	32	32	Double	7	268.6	179	0	--	0	258	29
	64	0	Single	9	248.51	219	0	--	0	391	18
	64	0	Double	10	176.87	338	0	--	0	648	18
	64	16	Single	9	248.51	219	0	--	0	391	18
	64	16	Double	10	176.87	338	0	--	0	648	18
	64	32	Single	9	248.51	219	0	--	0	391	18
	64	32	Double	10	176.87	338	0	--	0	648	18

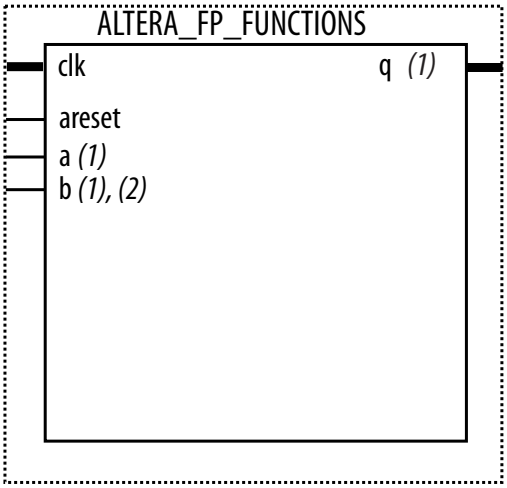
Family	Input Width	Input Fraction	Output Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Stratix V (5SGXEA7K2F40C2)	32	0	Single	3	579.71	148	--	0	0	97	1
	32	0	Double	2	547.95	161	--	0	0	72	1
	32	16	Single	3	550.66	148	--	0	0	97	1
	32	16	Double	2	536.19	160	--	0	0	72	0
	32	32	Single	3	558.66	145	--	0	0	96	1
	32	32	Double	2	496.28	154	--	0	0	72	1
	64	0	Single	3	454.55	194	--	0	0	125	0
	64	0	Double	3	434.22	304	--	0	0	194	3
	64	16	Single	3	454.55	194	--	0	0	125	0
	64	16	Double	3	434.22	304	--	0	0	194	3
	64	32	Single	3	454.55	194	--	0	0	125	0
	64	32	Double	3	434.22	304	--	0	0	194	3

Family	Input Width	Input Fraction	Output Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
										Primary	Secondary
Arria 10 (10AX115H4F34I3SP)	32	0	Single	3	464.9	147	--	0	0	97	0
	32	0	Double	2	458.93	161	--	0	0	72	0
	32	16	Single	3	464.9	147	--	0	0	97	0
	32	16	Double	2	432.15	160	--	0	0	72	0
	32	32	Single	3	451.67	145	--	0	0	96	0
	32	32	Double	2	419.99	154	--	0	0	72	0
	64	0	Single	3	417.54	193	--	0	0	124	3
	64	0	Double	3	407.33	305	--	0	0	193	3
	64	16	Single	3	417.54	193	--	0	0	124	3
	64	16	Double	3	407.33	305	--	0	0	193	3
	64	32	Single	3	417.54	193	--	0	0	124	3
	64	32	Double	3	407.33	305	--	0	0	193	3

Family	Input Precision	Output Precision	Latency	f_{MAX}	ALMs	M10K	M20K	DSP Blocks	Logic Registers	
									Primary	Secondary
Arria V (5AGXFB3H4F40C5)	Single	Double	2	371.61	93	0	—	0	71	0
	Double	Single	2	370.64	127	0	—	0	74	1
Cyclone V (5CGXFC7D6F31C7)	Single	Double	2	346.14	93	0	—	0	72	1
	Double	Single	3	349.9	126	0	—	0	111	2
Stratix V (5SGXEA7K2F40C2)	Single	Double	0	—	76	—	0	0	0	0
	Double	Single	0	717.36	126	—	0	0	34	0
Arria 10 (10AX115H4F34I3SP)	Single	Double	0	—	75	—	0	0	0	0
	Double	Single	0	—	126	—	0	0	34	0

ALTERA_FP_FUNCTIONS Signals

Figure 18-1: ALTERA_FP_FUNCTIONS Signals



1) The floating point and fixed point data widths determine the port width of this port.
2) This port is not relevant for convert and square root functions.

Table 18-6: ALTERA_FP_FUNCTIONS Input Signals

Port Name	Required	Description
`clk`	Yes	All input signals must be synchronous to this clock.
`areset`	Yes	Asynchronous active-high reset. Deassert this signal synchronously to the input clock to avoid metastability issues.
`en`	No	Optional port. Allow calculation to take place when asserted. When deasserted, no operation will take place and the outputs are unchanged.
`a`	Yes	Data input signal.
`b`	Yes	Data input signal (where applicable).
`s`	Yes	Select port for Add/Sub function.
`c`	Yes	Data port for integer exponent port for LDExp function.

Table 18-7: ALTERA_FP_FUNCTIONS Output Signals

Port Name	Required	Description
`q`	Yes	Data output signal.

ALTERA_FP_FUNCTIONS Parameters

These tables list the ALTERA_FP_FUNCTIONS parameters.

Table 18-8: ALTERA_FP_FUNCTIONS Parameters: Functionality Tab

Category	Parameter	Values	Descriptions
	Family	All Arithmetic Comparisons Conversions Exp/Log/Pow Roots Trigonometry	Allows you to chose which functions will be displayed in the Function Name Parameter list. The default value is All.
Function	**Name**	• Add • Subtract • Add/Sub • Multiply • Divide • Reciprocal • Absolute • Scalar Product • Multiply Accumulate • Accumulate • Multiply Add • Complex Multiply • Sin • Cos • Tan • Arcsin • Arccos • Arctan • Arctan2 • Exponent • Exponent base 2 • Exponent base 10 • Log • Log_2 • Log_{10} • $Log_{(1+x)}$ • Power • Square Root • Reciprocal Square Root • Cube Root • 3D Hypotenuse • Minimum • Maximum • Less Than Equal • Equal • Not Equal • Greater Than	Allows you to choose your desired function. **Note:** this parameter will only display the options you have selected from the Family Parameter

Category	Parameter	Values	Descriptions
Floating Point Data	**Format**	Single Double Custom	Allows you to choose the floating point format of the data values. The default value is single.
	Exponent	5 to 11	Allows you to specify the width of the exponent. This parameter is only available when the Format parameter is set to custom. The default value is 8.
	Mantissa	10 to 52	Allows you to specify the width of the mantissa. This parameter is only available when the Format parameter is set to custom. The default value is 23.
	Input Vector Dimension	Integer	Provide the desired the number of inputs to compute the vector dimension.
	Input Format	Single Double Custom	Allows you to choose the floating point format of the input data values. The default value is single. **Note:** Only available for **Floating to Floating Point** function.
	Input Exponent	Integer	Allows you to specify the width of the input exponent. This parameter is only available when the Format parameter is set to custom. The default value is 8. **Note:** Only available for **Floating to Floating Point** function.
	Input Mantissa	Integer	Allows you to specify the width of the mantissa. This parameter is only available when the Format parameter is set to custom. The default value is 23. **Note:** Only available for **Floating to Floating Point** function.
	Output Format	Single Double Custom	Allows you to choose the floating point format of the output data values. The default value is single.
	Output Exponent	Integer	Allows you to specify the

Category	Parameter	Values	Descriptions
Fixed Point Data	**Width**	16 to 128	The bit width of the fixed point data port. This parameter is only available when the Name parameter is set to **Fixed to Floating Point**. The default value is 32.
	Fraction	-128 to 128	The bit width of the fraction. This parameter is only available when the Name parameter is set to **Fixed to Floating Point**.
	Sign	Signed , Unsigned	Choose if the fixed point data is signed or unsigned. This parameter is only available when the Name parameter is set to Convert. The default value is signed.
Rounding	**Mode**	• nearest with tie breaking to even	The rounding mode.
	Relax rounding to round up or down to reduce resource usage	—	Choose if the nearest rounding mode should be relaxed to faithful rounding, where the result may be rounded up or down, to reduce resource usage. Only available for arithmetic functions
Ports	Generate Enable Port	—	Choose if the ALTERA_FP_FUNCTION IP core should have an enable signal.

Table 18-9: ALTERA_FP_FUNCTIONS Parameters: Performance Tab

Category	Parameter	Values	Descriptions
Target	**Goal**	• Frequency • Latency • Combined • Manually Specify DSP Registers	If the **Goal** is the frequency, then the **Target** is the desired frequency in MHz. This, together with the target device family, will determine the amount of pipelining. If the **Goal** is **Combined** then two Targets are displayed, one is the desired frequency in MHz, one is the target latency in cycles. When you set the **Goal** parameter to frequency, the default value is 200 MHz When you set the **Goal** parameter to latency, the default value is 2. If the **Goal** is **Latency**, then the **Target** is the desired latency. The report generates the achievable latency if it can't meet target latency. If the **Goal** is set to **Manually Specify DSP Registers**, you can manually select the register and function subblocks within the DSP IP core.
	Target	Any Positive Integer	Specify your target frequency and latency.
Report	**Latency on Arria 10 is <x> cycles**	—	This report shows the latency of the function.
	Resource Estimates: • **Multiplies** • **LUTs** • **Memory Bits** • **Memory Blocks**	—	This report shows the number of multipliers, LUTs, memory bits, and memory blocks utilized by the IP core.
	Check Performance	—	Click this to check if the design can achieve the target latency. **Note:** Only available when **Goal** is set to **Latency**.

Floating-Point IP Cores User Guide Document Archives

A

2016.12.09

UG-01058 Subscribe Send Feedback

If an IP core version is not listed, the user guide for the previous IP core version applies.

IP Core Version	User Guide
15.0	**Floating-Point IP Cores User Guide**

101 Innovation Drive, San Jose, CA 95134

Document Revision History

B

2016.12.09

UG-01058 Subscribe Send Feedback

Document Revision History

This table lists the document revision history for the Floating-Point IP Cores user guide.

Date	Document Version	Changes Made
December 2016	2016.12.09	• Added simple description about the IP core on each introduction page. • Added descriptions for each function supported by ALTERA_FP_FUNCTIONS IP core. • Clarified that ALTERA_FP_FUNCTIONS IP core replaces all ALTFP IP cores in Arria 10 devices. • Added new parameters in ALTERA_FP_FUNCTIONS Parameter: Functionality Tab table. • Clarified the functionality of en signal for ALTERA_FP_FUNCTIONS.
July 2015	2015.07.30	• Updated link to Floating-Point IP Cores Design Examples. • Updated Memory Blocks numbers in ALTERA_FP_MATRIX_MULT Resource Utilization and Performance for the Arria 10 and Stratix V Devices table. • Added notes on default settings for WIDTH_INT and WIDTH_DATA.
December 2014	2014.12.19	• Remove all references to the complex mode in ALTFP_MATRIX_MULTIPLY. • Updated ALTERA_FP_MATRIX_MULT and ALTERA_FP_FUNCTIONS sections.

Date	Document Version	Changes Made
November 2013	7.0	• Added "ALTERA_FP_FUNCTIONS" on page 3–1. • Added "ALTERA_FP_ACC_CUSTOM" on page 2–1. • Updated Table 1–1 on page 1–1 to list ALTERA_FP_FUNCTIONS and ALTERA_FP_ACC_CUSTOM. • Updated the "ALTFP_MATRIX_INV" on page 17–1 section to include 4 x 4 and 6 x 6 dimensions. • Updated "Rounding" on page 1–4 to clarify that the code for round-to-nearest-even mode is `TO_NEAREST`. • Removed Design Example section for "ALTFP_MATRIX_MULT" on page 18–1. • Removed device family support for HardCopy III, HardCopy IV, Stratix II, and Stratix II GX devices from "Device Family Support" on page 1–2.
November 2011	6.0	Updated "General Features" on page 1–2.
May 2011	5.0	Added "ALTFP_ATAN" on page 12–1.
January 2011	4.0	Added "ALTFP_SINCOS" on page 13–1.
July 2010	3.0	• Updated architecture information for the following sections: ALTFP_MATRIX_MULT ALTFP_MATRIX_INV. • Added specification information in all sections.
November 2009	2.0	• Updated resource utilization information for the following sections: ALTFP_ADD_SUB ALTFP_DIV ALTFP_MULT ALTFP_SQRT ALTFP_EXP ALTFP_INV ALTFP_INV_SQRT ALTFP_LOG ALTFP_COMPARE ALTFP_CONVERT ALTFP_MATRIX_MULT • Added the ALTFP_MATRIX_INV section. • Updated the Ports and Parameters section for all floating-point megafunctions.

Date	Document Version	Changes Made
March 2009	1.0	Initial release.