Using Nios II Floating-Point Custom Instructions

May 2006 | Tutorial

The Nios® II floating-point custom instructions accelerate arithmetic functions executed on `float` types.

This tutorial guides you through the basics of using the Nios II floating-point custom instructions. It is a good starting point if you are considering the floating-point custom instructions for inclusion in your own project. In this tutorial you add the floating-point custom instructions to a Nios II example design, and create a software program to exhibit floating-point performance. When you complete this tutorial, you will know:

- How to add the floating-point custom instructions to a Nios II processor
- How to use the floating-point custom instructions in a C program, and how the custom instructions work with the Nios II Embedded Design Suite (EDS).
- The advantages and disadvantages of the floating-point custom instructions, and how best to use them in your own system

Table of Contents

Document Conventions 2
About the Floating-Point Custom Instructions 2
Getting Ready 2
- Prerequisites 2
- Hardware & Software Requirements 3
- Getting the Example Design 3
- Getting the Software Files 3

Building and Programming the Hardware 4
Building and Running the Software 5
- Creating the Software Project 5
- Building and Running the Software and Analyzing the Results 5
- Tutorial Implementation 6

Moving On to Your Own System 7
- Assessing Your Floating-Point Optimization Needs 7
- Floating-Point Divide Considerations 8
- Floating Point Constants 8
- Simulation 9
- Device Resource Usage 9

Document Conventions

This document uses the variables shown in Table 1 to represent file paths on your system.

Table 1. File path conventions

Symbol	Meaning
<Nios II EDS Install Path>	The location where the Nios II EDS is installed. On a Windows system, by default, that location is **c:\altera\kits\nios2_*nn***, where ***nn*** represents the current version number.
<HDL>	The selected hardware description language: **verilog** or **vhdl**
<Nios II development board>	The directory name identifying a Nios II development board, e.g. **niosII_stratixII_2s60**

About the Floating-Point Custom Instructions

The floating-point custom instructions, optionally available on the Nios II processor, implement single precision floating-point arithmetic operations. You can use the custom instructions to accelerate floating-point operations in your Nios II C/C++ application program. This set of custom instructions is available on every Nios II core implementation. The basic set of floating-point custom instructions includes single precision floating-point addition, subtraction, and multiplication. Floating-point division is available as an extension to the basic instruction set.

Table 2 lists approximate acceleration factors afforded by the floating-point custom instructions.

Table 2. Sample Floating-Point Custom Instruction Acceleration Factors(1)

Target device	Addition	Subtraction	Multiplication	Division
EP1C20	21 x	18 x	27 x	15 x
EP1S40	20 x	20 x	19 x	18 x
EP2S60	14 x	15 x	12 x	14 x

Note to Table 2:

(1) For each floating-point custom instruction, these figures show typical speed increases over the equivalent software implementation. For each target device, these results were obtained with the full_featured example design provided with the EDS. You might see different acceleration results, depending on your hardware design and target device, and on the details of your software application.

When the floating-point custom instructions are present in your target hardware, the Nios II IDE compiles your code to use the custom instructions for floating-point operations, including the four primitive arithmetic operations (addition, subtraction, multiplication and division) and the ANSI C math library. The ANSI C math functions are listed in Table 3 on page 8.

The floating point custom instructions substantially comply with the IEEE 754-1985 floating point standard. For details, refer to *Floating Point Instructions* under *Arithmetic Logic Unit*, in the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

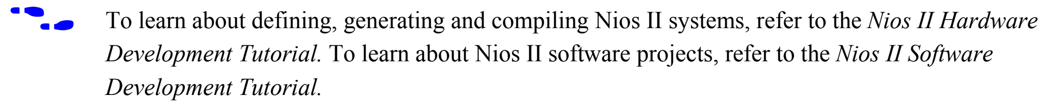
Getting Ready

Prerequisites

To make effective use of this tutorial, you should be familiar with the following topics:

- Defining and generating Nios II hardware systems with SOPC Builder

- Compiling Nios II hardware systems with the Quartus® II development software
- Creating, compiling, and running Nios II software projects



To learn about defining, generating and compiling Nios II systems, refer to the *Nios II Hardware Development Tutorial.* To learn about Nios II software projects, refer to the *Nios II Software Development Tutorial.*

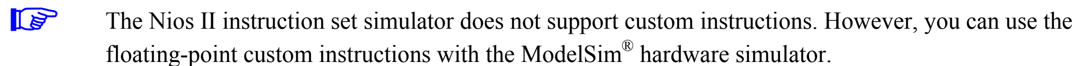
Hardware & Software Requirements

This tutorial requires you to have the following software and hardware:

- Altera® Quartus II development software version 6.0 or later, installed on a Windows or Linux computer.
- Nios II EDS version 6.0 or later
- Nios II target hardware.
- A JTAG download cable compatible with your target hardware: for example, a USB-Blaster™ cable.

Your target hardware can be a Nios II development board for the EP1C20, EP2C35, EP1S40 or EP2S60, or any hardware that meets these criteria:

- It must include an Altera FPGA supporting the Nios II processor.
- The target hardware must support a full_featured example design (distributed with the EDS) with the following characteristics:
  - Includes a performance counter component.
  - Leaves enough unused logic to support the floating-point custom instructions. For details, see Table 4. Approximate Device Resource Usage.
- An oscillator must drive a constant clock frequency to an FPGA pin. The maximum frequency limit depends on the speed grade of the FPGA.
- The board must have a 10-pin header connected to the dedicated JTAG pins on the FPGA to provide a communication link to the Nios II system.



The Nios II instruction set simulator does not support custom instructions. However, you can use the floating-point custom instructions with the ModelSim® hardware simulator.

Getting the Example Design

As the basis for this tutorial, use the full-featured example design corresponding to your target hardware. The full-featured example designs reside at *<Nios II EDS Install Path>/examples/<HDL>/<Nios II development board*>/full_featured, installed with the Nios II EDS.

Do not modify the full-featured design in your Nios II EDS installation. Make a copy of it in a working directory.

Getting the Software Files

The tutorial software files are available on the Nios II literature page. A hyperlink to the software files appears next to this document, at **www.altera.com/literature/lit-nio2.jsp.**

The software files are distributed in a zip file. Unzip this file to a temporary directory. There are four files:

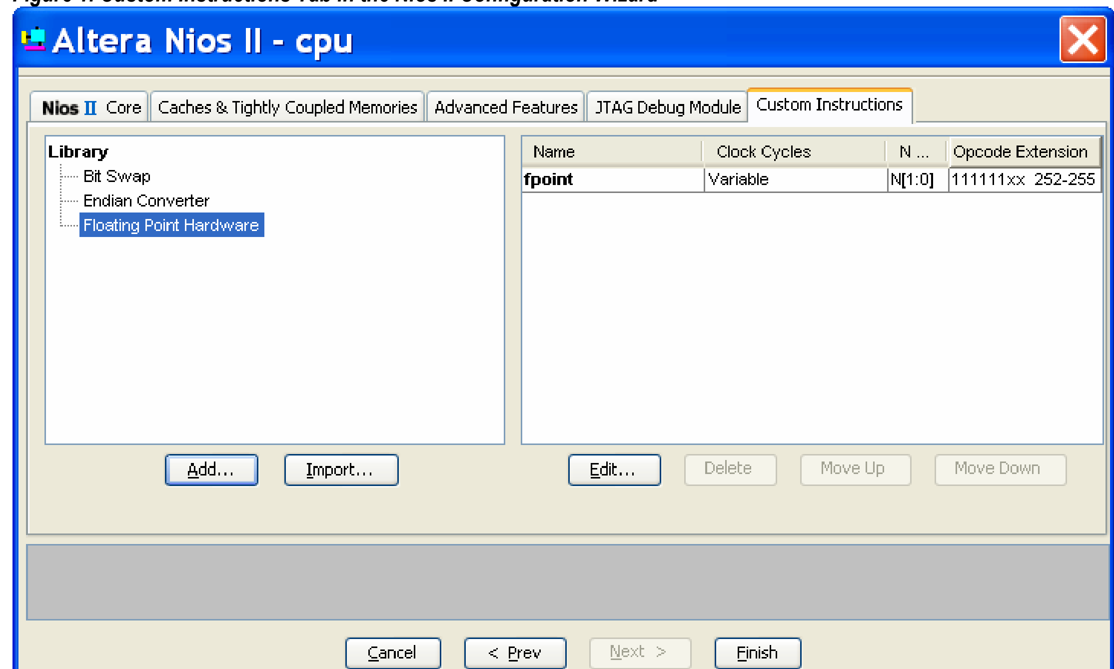
- **floating_point.c** – the main program
- **floating_point.h** – global definitions
- **floating_point_CI.c** – functions to exercise the floating-point custom instructions
- **floating_point_SW.c** – functions to exercise the software-implemented floating-point operations

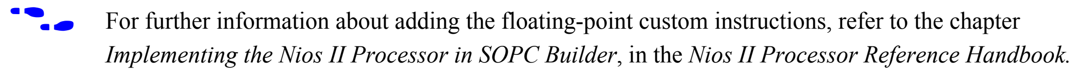
Building and Programming the Hardware

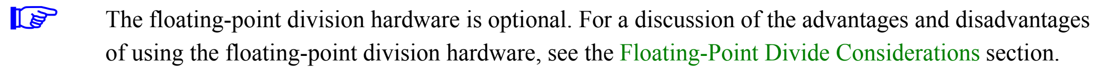
Follow these steps to add the floating-point custom instructions to the Nios II processor in the example design:

1. Launch the Quartus II development software, and open your working copy of the hardware example design.
2. Launch SOPC Builder.
3. Edit the Nios II processor component. This launches the Nios II configuration wizard.
4. Select the **Custom Instructions** tab.
5. Select **Floating Point Hardware** from the **Library** list, and click **Add**.
6. Turn on the **Use floating point division hardware** option. Click **Finish** to exit the **Nios II Floating Point Hardware** dialog.
7. Click **Finish** to exit the Nios II Wizard. Figure 1 shows the SOPC Builder **Custom Instructions** Tab with floating-point hardware inserted.

Figure 1. Custom Instructions Tab in the Nios II Configuration Wizard




For further information about adding the floating-point custom instructions, refer to the chapter *Implementing the Nios II Processor in SOPC Builder*, in the *Nios II Processor Reference Handbook.*


The floating-point division hardware is optional. For a discussion of the advantages and disadvantages of using the floating-point division hardware, see the Floating-Point Divide Considerations section.

8. Generate the HDL for your SOPC Builder system. When the generation process is complete, exit SOPC Builder.

9. Compile the Quartus II project.

10. Program your target hardware with the resulting FPGA configuration file (**.sof**).

Building and Running the Software

Creating the Software Project

Follow these steps to create and build the software project:

1. Launch the Nios II IDE.

2. Create a new Nios II C/C++ application, based on the **Blank Project** template. Under **Select Target Hardware**, select the SOPC Builder project that you generated in the Building and Programming the Hardware section.

3. Import the tutorial software files into your Nios II C/C++ application project. The easiest way to do this is to select the files in an application such as Windows Explorer, and drag them into the Nios II C/C++ application project folder in the **C/C++ Projects** view of the Nios II IDE.

4. Open the **Properties** dialog box for your Nios II C/C++ application project. In the **C/C++ Build** category, set **Configuration** to **Release**. This enables compiler optimization.

Building and Running the Software and Analyzing the Results

1. Build the software project.

2. Run the software on your Nios II hardware.

The Nios II IDE detects the presence of the floating-point custom instructions at build time, and uses them for all single precision floating-point arithmetic.

The program runs four tests, one each for the add, subtract, multiply, and divide operations. In each test, the program carries out the floating-point operation on 1000 pairs of random operands. It executes both the floating-point custom instruction and the equivalent software implementation. Using the performance counter component, the tutorial software compares the hardware and software execution times.

As shown in Figure 2, the results include a report for each test. In each report, the section `FP CI` … lists the performance of the custom instruction, and the section `FP SW` … lists the performance of the software implementation. The `Time (sec)` and `Time (clock)` columns represent the aggregate time spent executing

the floating-point operations, in seconds and in Nios II clock cycles. `Total Time` represents the duration of the test, expressed both in seconds and in Nios II clock cycles. The column labeled with a percent sign (`%`) represents the time spent executing the floating-point operation, as a percentage of the test total.

Figure 2. Sample Software Results

```
--Performance Counter Report--
Total Time: 0.01222420 seconds  (611210 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|FP CI ADD      | 2.29|    0.00030|          14000|       1000|
+---------------+-----+-----------+---------------+-----------+
|FP SW ADD      | 50.2|    0.00610|         306640|       1000|
+---------------+-----+-----------+---------------+-----------+

--Performance Counter Report--
Total Time: 0.00987798 seconds  (493899 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|FP CI SUBTRACT | 2.83|    0.00028|          14000|       1000|
+---------------+-----+-----------+---------------+-----------+
|FP SW SUBTRACT | 50.8|    0.00502|         250975|       1000|
+---------------+-----+-----------+---------------+-----------+


--Performance Counter Report--
Total Time: 0.0110131 seconds  (550654 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|FP CI MULTIPLY | 2.18|    0.00024|          12000|       1000|
+---------------+-----+-----------+---------------+-----------+
|FP SW MULTIPLY |   59|    0.00650|         325076|       1000|
+---------------+-----+-----------+---------------+-----------+


--Performance Counter Report--
Total Time: 0.0142152 seconds  (710758 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|FP CI DIVIDE   |  4.5|    0.00064|          32000|       1000|
+---------------+-----+-----------+---------------+-----------+
|FP SW DIVIDE   | 67.8|    0.00963|         481698|       1000|
+---------------+-----+-----------+---------------+-----------+
```

You might see different speed results, depending on your target hardware and on the actual values of the random operands.

Tutorial Implementation

The tutorial software uses `#pragma` directives to compare hardware and software implementations of the floating-point instructions. These `#pragmas` direct the Nios II compiler to ignore the floating-point instructions and generate software implementations. The `#pragma` directives are:

- `#pragma no_custom_fadds` — forces software implementation of floating-point add
- `#pragma no_custom_fsubs` — forces software implementation of floating-point subtract
- `#pragma no_custom_fmuls` — forces software implementation of floating-point multiply

- `#pragma no_custom_fdivs` — forces software implementation of floating-point divide

The scope of these `#pragmas` is the entire C file.

The tutorial software uses the Nios II performance counter component to collect timing information on the floating-point operations. For more detail on the performance counter component, refer to the *Performance Counter Core with Avalon Interface* section in the *Quartus II Development Software Handbook, Volume 5: Embedded Peripherals*.

Moving On to Your Own System

Congratulations! You have built and demonstrated a Nios II system using the floating-point custom instructions. Through this tutorial, you have familiarized yourself with the steps for integrating the floating-point custom instructions into a Nios II system:

- Modifying and generating Nios II system hardware in SOPC Builder
- Compiling the Quartus II project
- Creating a new project in the Nios II IDE
- Compiling the project
- Running the software on the target hardware

This section can help you determine how to use the floating-point custom instructions in your own project.

Assessing Your Floating-Point Optimization Needs

The best choice for your hardware design depends on a balance among floating-point usage, hardware resource usage, and performance. While the floating-point custom instructions speed up floating-point arithmetic, they add substantially to the size of your hardware project. If resource usage is an issue, before using the floating-point custom instructions, consider these possibilities:

- Have you identified your performance bottlenecks? Make sure your performance issues are caused by floating-point arithmetic before you try to fix them with floating-point acceleration.

Refer to *AN391: Profiling Nios II Systems* for detailed information about Nios II performance profiling.

- Can you use integer arithmetic? While the floating-point custom instructions are faster than software-implemented floating-point, they are slower than hardware-based integer arithmetic. A common integer technique is to represent numerical values with an implicit scaling factor. As a simple example, if you are calculating milliamps, you might represent your values internally as microamps.
- Are you taking full advantage of compiler optimization? You can increase the Nios II compiler optimization level through the **Properties** dialog box for your Nios II C/C++ application and system library projects.

For details, refer to the *Reducing Code Footprint* section of the chapter *Developing Programs using the HAL* in the *Nios II Software Developer's Handbook*.

- Have you hand-optimized your mathematical operations? Numerical analysis textbooks offer simple, effective techniques for performing accurate calculations with the minimum number of floating-point operations.

If you have followed these suggestions, and you need further acceleration, the floating-point custom instructions are probably an appropriate solution.

Floating-Point Divide Considerations

The floating-point division hardware requires more resources than the other instructions, so you might opt to omit it if your Nios II C/C++ application does not make heavy use of floating-point division.

In some cases, you can rewrite your code to minimize or even eliminate divide operations. For example, if your algorithm requires division by a constant value, you can precalculate its inverse and use a multiply operation in the speed-critical section of your code.

Table 3 indicates which math library functions use floating-point, and of those, which use floating-point division. If a function uses floating-point, it runs faster with floating-point hardware. If a function uses floating-point division, it runs still faster with floating-point division hardware.

Table 3. Math Library Floating-Point Usage

Math function	Uses floating-point?	Uses floating-point division?	Math function	Uses floating-point?	Uses floating-point division?
`acos()`	Yes	Yes	`frexp()`	Yes	
`asin()`	Yes	Yes	`ldexp()`	Yes	
`atan()`	Yes	Yes	`log()`	Yes	Yes
`atan2()`	Yes	Yes	`log10()`	Yes	Yes
`cos()`	Yes		`modf()`	Yes	
`cosh()`	Yes	Yes	`pow()`	Yes	Yes
`sin()`	Yes		`sqrt()`	Yes	Yes
`sinh()`	Yes	Yes	`ceil()`	Yes	
`tan()`	Yes	Yes	`fabs()`		
`tanh()`	Yes	Yes	`floor()`	Yes	
`exp()`	Yes	Yes	`fmod()`	Yes	Yes

When you omit the floating-point divide instruction, the Nios II IDE implements floating-point division in software.

For details of selecting the floating-point division hardware in the Nios II Wizard, refer to the *Implementing the Nios II Processor in SOPC Builder* chapter of the *Nios II Processor Reference Handbook.*

Floating Point Constants

The Nios II compiler treats floating point constants as double precision. Therefore, if you want the floating point custom instructions to be used for an operation with a floating point constant, you need to append an `f` to the constant, as in Figure 1. This tells the compiler to treat it is a single precision float.

Figure 3. Float and Double Constants

```
y = x * 4.67;   // Double precision.  Does NOT use floating point custom instructions.
y = x * 4.67f;  // Single precision.  Does use floating point custom instructions.
```

Simulation

The Nios II instruction set simulator does not support custom instructions. If you need to run your software on the instruction set simulator, you can disable the floating-point custom instructions in software with the `#pragma` directives described in Tutorial Implementation on page 6.

You can use the floating-point custom instructions with the ModelSim hardware simulator.

Device Resource Usage

The floating-point custom instructions are available on all Altera devices that support the Nios II processor. Table 4 on page 9 shows approximate resource usage in each supported device.

If the target device includes on-chip multiplier elements, the floating-point hardware incorporates them as needed. If there are no on-chip multiplier elements, the floating-point custom instructions are implemented entirely with general-purpose logic elements.

Table 4. Approximate Device Resource Usage

Target device family	LEs or ALUTs (1)		Multiplier elements (2)
	without divide	with divide	
Cyclone	2500	6500	N/A
Cyclone II	2000	5400	7
Stratix	1800	5500	8
Stratix II	1600	4900	8

Notes to Table 4:

(1) For the Stratix II, the numbers in these columns represent adaptive look-up tables (ALUTs.) For other devices, the numbers represent logic elements (LEs).

(2) In Cyclone II devices, a "multiplier element" is an embedded multiplier 9-bit element. In Stratix and Stratix II devices, a "multiplier element" is a DSP 9-bit element. Cyclone devices do not have hardware multipliers.

Resource usage in your own project might differ considerably from the values shown in Table 4, depending on the details of Quartus II routing and fitting.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com

© 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.