



Nios II IDE Help System



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

Nios II IDE Version:	6.0
Document Version:	1.0
Document Date:	May 2006

UG-N2IDEHELP-1.0

Table Of Contents

About This Document	1
Welcome to the Nios II IDE	3
What's New in the Nios II IDE v6.0	6
Tutorials	16
About Tutorials	16
Creating a C/C++ Application Project	17
Building the Project	19
Running the Project	20
Debugging the Project	23
Editing the Project Properties	32
Creating Projects	34
About Nios II IDE Projects	34
About the Nios II IDE Managed-Make Build Environment	36
Creating a New Project	42
Importing, Exporting and Sharing Projects and Files	46
Configuring Projects	48
About Project Properties	48
Configuring Project Properties	49
Choosing and Configuring an Operating System	50
Choosing and Configuring Middleware Software Components	51
Configuring Project Dependencies	52
Editing Code	54
About Editing Code	54
Building Projects	55
About Building Projects	55
Building a Project	56
Running and Debugging Projects	57
About Running and Debugging Projects	57
Configuring the FPGA	59
Running and Debugging on Hardware	61
Running and Debugging on the ISS	63
Running on the ModelSim Simulator	65
Running and Debugging Multiprocessor Collections	66
Viewing Execution Trace	68
Viewing Disassembly	70
Profiling Execution Performance	72
About Profiling with the Nios II IDE	72
Profiling C Code	73

Storing Firmware on the Target Board	75
About Storing Firmware	75
Programming Flash	76
Features and Terms Reference	78
Advanced Debugging Features by FS2	78
C-to-Hardware Acceleration (C2H) Compiler	79
Hardware Abstraction Layer (HAL)	80
Hardware Simulation with ModelSim	81
Hardware Target	82
Host-Based File System	83
Instruction Set Simulator (ISS)	85
Lightweight TCP/IP Stack	87
MicroC/OS-II RTOS	88
Multiprocessor Nios II Systems	89
Run/Debug Configuration	90
Valid Project Names	91
Zip Read-Only File System	92
GUI Reference	93
Flash Programmer Dialog Box	93
Import Wizard	95
New Project Wizard	96
New Project Wizard	96
New C/C++ Application (New Project Wizard)	97
New System Library (New Project Wizard)	99
New Managed Library (New Project Wizard)	100
New Advanced C/C++ Project (New Project Wizard)	101
Preferences Dialog Box	102
Preferences Dialog Box	102
Nios II Page (Preferences Dialog Box)	103
New Projects Page (Preferences Dialog Box)	104
Trace Page (Preferences Dialog Box)	105
Profiling Perspective	106
Profiling Perspective	106
Call Hierarchy View (Profiling Perspective)	107
Editor View (Profiling Perspective)	108
Samples - Function Total View (Profiling Perspective)	109
Samples - Line By Line View (Profiling Perspective)	110
Project Properties Dialog Box	111
Properties Dialog Box	111
Associated System Library Page (Properties Dialog Box)	113

Builders Page (Properties Dialog Box)	114
C/C++ Build Page (Properties Dialog Box)	115
C/C++ Documentation Page (Properties Dialog Box)	117
Project References Page (Properties Dialog Box)	118
System Library Page (Properties Dialog Box)	119
RTOS Options Dialog Box (System Library Properties Page)	122
Software Components Dialog Box (System Library Properties Page)	123
Run/Debug Dialog Box	124
Run/Debug Dialog Box	124
Common Tab (Run/Debug Dialog Box)	126
Debugger Tab (Run/Debug Dialog Box)	127
ISS Settings Tab (Run/Debug Dialog Box)	129
Launch ModelSim Tab (Run Dialog Box)	131
Main Tab (Run/Debug Dialog Box)	132
Source Tab (Run/Debug Dialog Box)	133
Target Connection Tab (Run/Debug Dialog Box)	134
Views	135
C/C++ Projects View (C/C++ Perspective)	135
Call Hierarchy View (Profiling Perspective)	139
Disassembly View (Debug Perspective)	140
Editor View (Profiling Perspective)	141
Samples - Function Total View (Profiling Perspective)	142
Samples - Line By Line View (Profiling Perspective)	143
Trace View (Debug Perspective)	144
Workspace Launcher Dialog Box	145
Troubleshooting	146



About This Document

This document provides complete reference for the Nios II integrated development environment (IDE), including details of the IDE design flow and tutorials on using the IDE. This document is based on the HTML content of the Nios II IDE help system. Altera provides this content as a PDF file to make it accessible as a stand-alone document outside of the Nios II IDE.

The content for this document was developed as an interactive help system. However, the PDF file does not provide the interactive functionality. In specific, this document does not provide hyperlinks to related documentation. Use the following guidelines to find related documentation listed in individual topics:

- **Related Nios II IDE Help Topics** are included in this PDF file.
- **Related Eclipse Workbench User Guide Help Topics** are available at <http://help.eclipse.org/help30>.
- **Related C/C++ Development (CDT) User Guide Help Topics** are available at <http://www.eclipse.org/cdt>.
- **Related Topics on the Web** provide specific URLs to locate the documents on the Internet.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide website at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/ (800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	www.altera.com/mysupport/ +1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	www.altera.com	www.altera.com
Altera literature services	literature@altera.com	literature@altera.com
Non-technical customer service	(800) 767-3753	+1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.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.



Welcome to the Nios II IDE

The Nios II integrated development environment (IDE) is the primary software development tool for the Nios II family of embedded processors. The Nios II IDE provides a consistent development platform that works for all Nios II processor systems. You can accomplish all software development tasks within the Nios II IDE, including editing, building, debugging, and profiling programs. The IDE allows you to create single-threaded programs as well as complex applications based on a real-time operating system (RTOS) and middleware libraries available from Altera and third-party vendors.

► Design Flow:

A typical design flow with the Nios II IDE comprises the following stages:

- Create a project
- Configure the project properties
- Edit code
- Build the project
- Run and debug the project
- Profile execution performance
- Store the project firmware on a target board

Not all stages are required, depending on the particular design. At any stage, you can return to a previous stage.

► Getting Started:

The following help topics and cheat sheets help you to understand and start using the Nios II IDE quickly.

- **Quick-Start Tutorial**
This cheat sheet guides you through the process of creating a new project, compiling it, and running it on a Nios II development board.
To start the Nios II Quick-Start Tutorial, do the following:
 1. On the Help menu, click **Cheat Sheets...**
 2. Click **Nios II IDE Quick-Start Tutorial**.
 3. Click **OK**.
- **Software Development Tutorial**
This tutorial guides you through the complete software development process in detail. You will compile, run, debug, set breakpoints, edit project properties, and more.
- **Nios II IDE Tour Cheat Sheet**
This cheat sheet introduces you to the significant features of the IDE.
To start the Nios II IDE Tour cheat sheet, do the following:
 1. On the Help menu, click **Cheat Sheets...**
 2. Click **Nios II IDE Tour**.
 3. Click **OK**.

▶ **Workbench, Perspectives, and Views:**

The Nios II IDE is based on the popular Eclipse IDE framework and the Eclipse C/C++ Development Toolkit (CDT) plug-ins. The Nios II IDE inherits much of its behavior from Eclipse, including the concepts of workbench, perspectives, and views.

The Eclipse graphical interface is called the *workbench*. Each Eclipse workbench window contains one or more *perspectives*. Each perspective provides a set of capabilities aimed at accomplishing a specific type of task. For example, the C/C++ perspective provides facilities for editing and compiling C/C++ projects. Perspectives in the workbench comprise one or more *views*. Views help you to organize and navigate the information in your workbench. For example, the Registers view in the Debug perspective allows you to inspect and edit the values of processor registers while debugging a project.

The Nios II IDE primarily uses the C/C++ and Debug perspectives, and also provides a profiling perspective.

▶ **Projects and Workspace:**

Nios II IDE *projects* are groups of files treated as a unit, containing source code, makefiles, object files, libraries, and other related files. Projects contain the resources you need to create, build, run, and debug within the Nios II IDE.

The Nios II IDE stores your projects in a directory called a *workspace*. You can define one or more workspaces, as well as select the workspace to use for the current IDE session using the **Workspace Launcher** dialog box.

▶ **Customizing Nios II IDE Preferences:**

The Nios II IDE provides *preferences* to customize the IDE. The Eclipse IDE framework offers user preference pages for all the Eclipse C/C++ Development Toolkit (CDT) plug-ins. User preference pages let you define and control the look, feel, and behavior of your workbench. Some preference pages are part of the standard Eclipse environment and other preference pages are specific to the Nios II IDE.

The following Nios II IDE preferences pages affect how the Nios II IDE builds, stores, runs, and debugs projects.




- Nios II Page (Preferences Dialog Box)
- New Projects Page (Preferences Dialog Box)
- Trace Page (Preferences Dialog Box)

Getting Help

The Nios II IDE provides an extensive help system that covers all aspects of the Nios II IDE. This page you are reading right now is part of the IDE help system.

There are two ways to open the help system:

- Click **Help Contents** on the IDE Help menu to launch the help system, then click **Nios II IDE Help** in the **Contents** pane. You can browse through topics in the **Contents** pane, or use the **Search** box to search for a specific term across all help topics.
- On Windows systems you can also press **F1** at any point in the IDE to display context sensitive help for the current window. A tool-tip appears with a simple description of the window. If help is available for the window, you will see a **Nios II Help** link. Click the link to jump to the topic in the help system.

 **Note:** The Nios II IDE help system might not function properly on browsers older than Firefox 1.0.7, Internet Explorer 6.0, Mozilla 1.7.12, Netscape Navigator 8.1, and Opera 8.5.1. In some older browsers, expandable text indicated by  does not work. Click  **Show All** in the upper-right corner to expand all text in the topic.

In addition to Altera-specific content, the Nios II IDE help system includes the Eclipse *Workbench User Guide* and the *CDT C/C++ Development User Guide*. Altera-specific content overrides any information found in the Eclipse or CDT user guides. Specifically, Eclipse and CDT help topics relating to creating, building, and debugging projects are invalid. See the Nios II IDE Help topics instead.

The help browser only works while the Nios II IDE is running. If the browser stays open after you close the IDE, the navigation stops working.

Related Nios II IDE Help Topics

- About Tutorials

Related Eclipse and CDT Help Topics

- Workbench User Guide > Concepts > Help system
- C/C++ Development User Guide > Concepts > Perspectives available to C/C++ developers
- Workbench User Guide > Tasks > Working with perspectives
- Workbench User Guide > Reference > Preferences
- C/C++ Development User Guide > Reference > C/C++ preferences

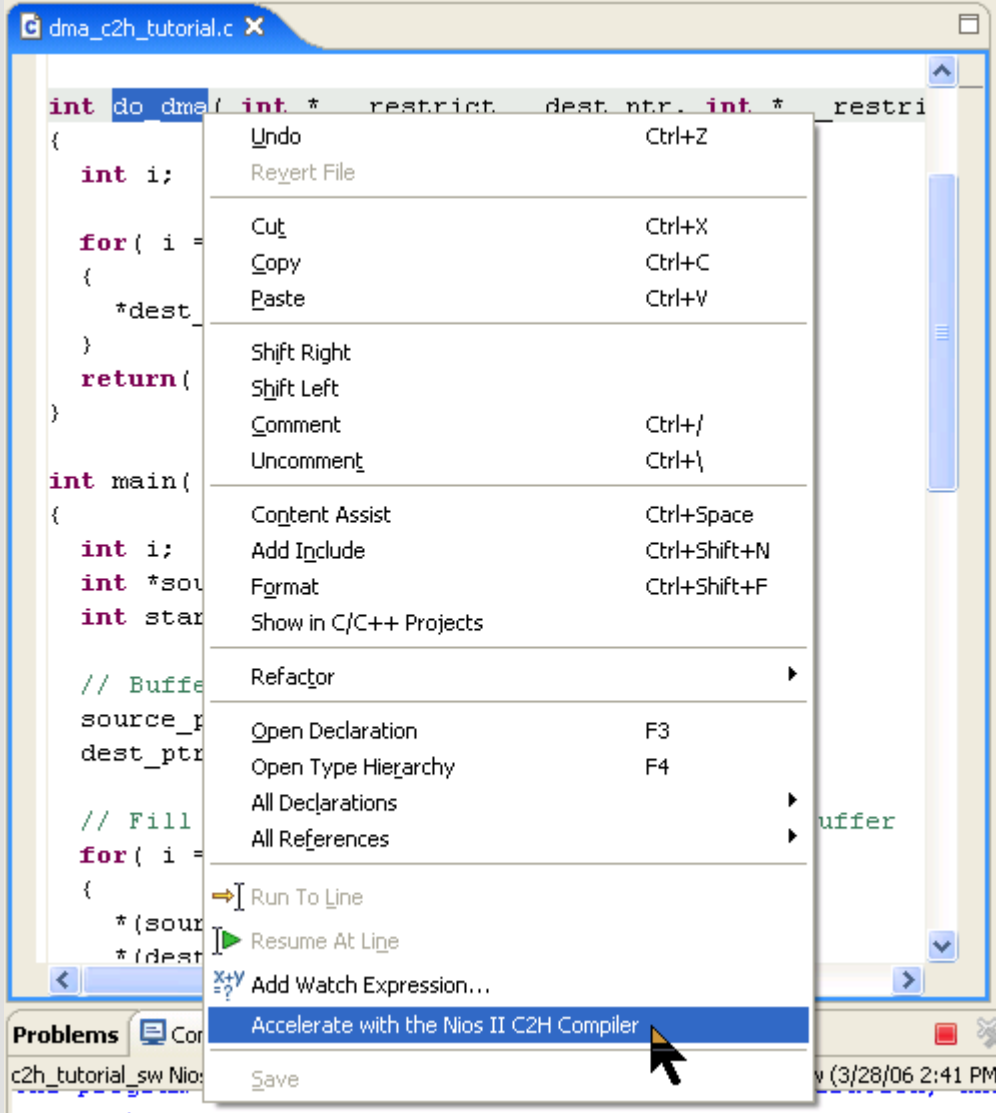
Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on writing programs for the Nios II processor.
- Nios II literature web page at www.altera.com/literature/lit-nio2.jsp - Contains all documentation related to the Nios II processor.
- Eclipse Homepage at www.eclipse.org



What's New in the Nios II IDE v6.0

Below are the most significant changes to the Nios II IDE for the v6.0 release.

Feature	Description
Nios II C-to-Hardware Acceleration (C2H) Compiler support	<p>The Nios II IDE provides the user interface for Altera's new C2H Compiler, enabling you to easily convert processor-intensive C functions to hardware accelerators.</p> 
Floating-point instruction software support	<p>The Nios II processor 6.0 provides hardware-accelerated floating-point instructions. The compiler recognizes if the Nios II processor core has floating-point hardware enabled, and emits appropriate instructions to take advantage of the hardware.</p>

Refer to the Nios II C2H Compiler User Guide for more information.

Help System Improvements

Refer to the Using Nios II Floating-Point Custom Instructions Tutorial for more information.

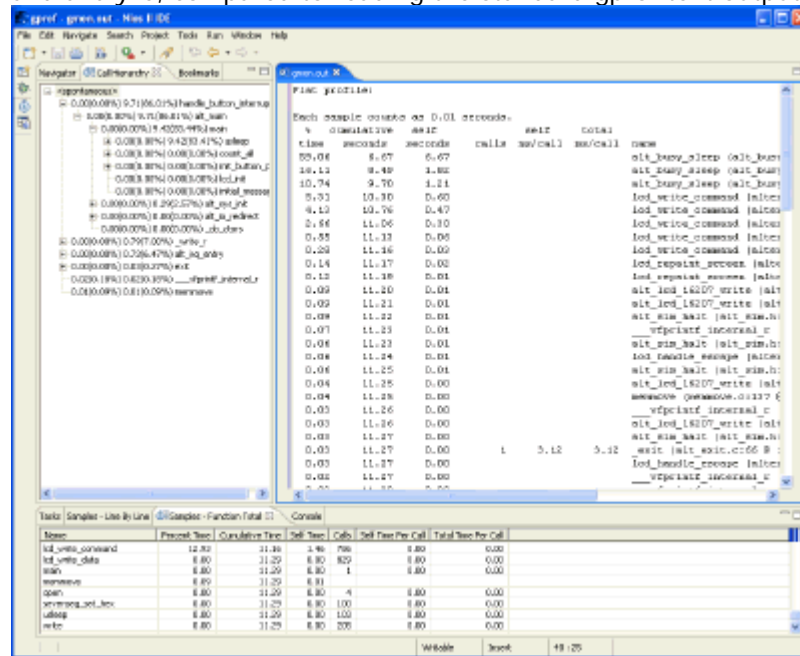
The help system for the Nios II IDE is reorganized, and has improved usability to help you quickly find the information you need. (The page you are reading now is part of the help system.) You can browse through topics in the **Contents** pane, which is now arranged in order of the development flow, or use the **Search** box to search for a specific term across all help topics.

What's New in the Nios II IDE v5.1:

Below are the most significant changes to the Nios II IDE for the v5.1 release:

Profiling perspective

The Nios II IDE Profiling perspective allows you to conveniently analyze GNU profiling (gprof) data stored in a **gmon.out** data file. The display features of the Profiling perspective make the data much easier to read and analyze, compared to reading the standard gprof text output.

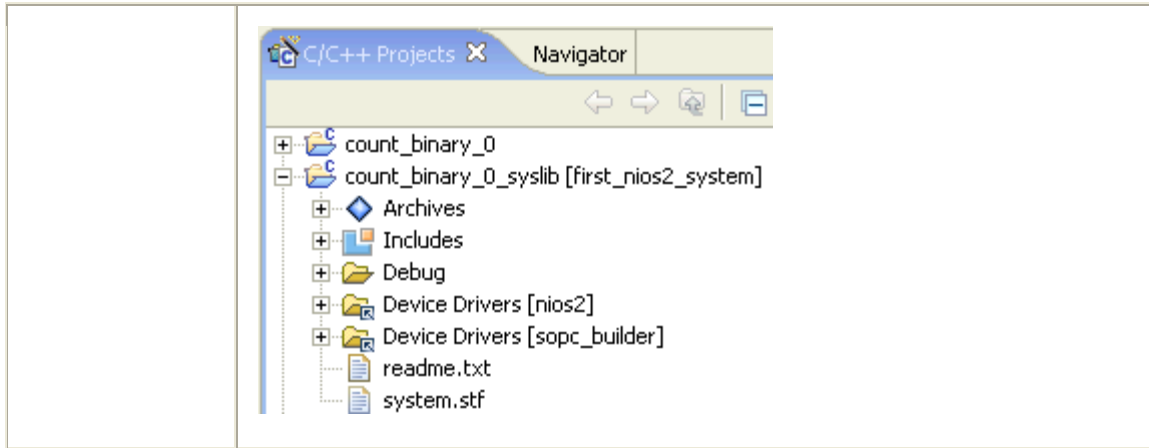


Flash programmer

The Nios II flash programmer has improved usability making it easier to use the flash programmer on custom systems. While the flash programmer GUI did not change, the background operation has changed significantly. See the Nios II Flash Programmer User Guide.

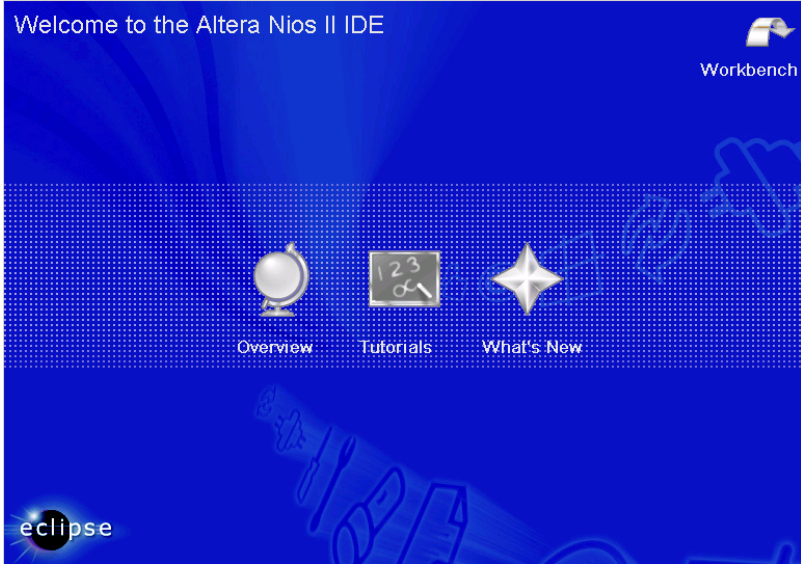
Nios II Device Drivers project location change


Device drivers for installed SOPC Builder components are now referenced in linked folders within each system library project. The Nios II IDE accommodates drivers for components, regardless of where the components are installed. The system library contains one Device Drivers folder for each path containing SOPC Builder components.

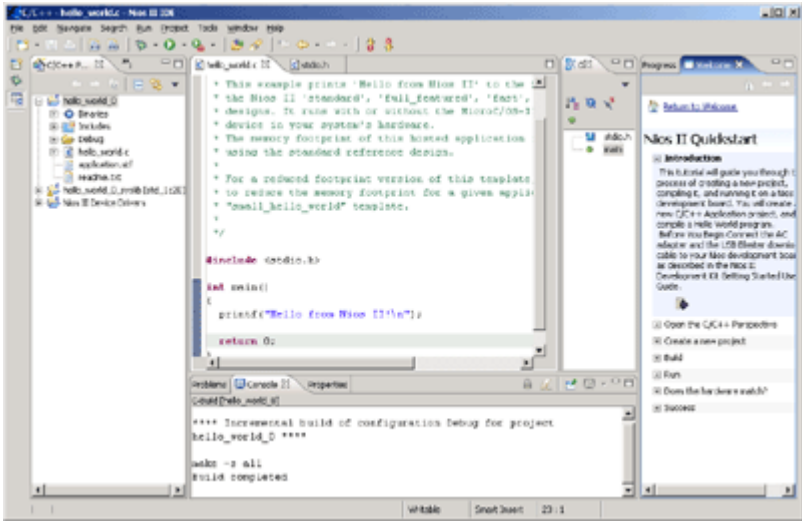
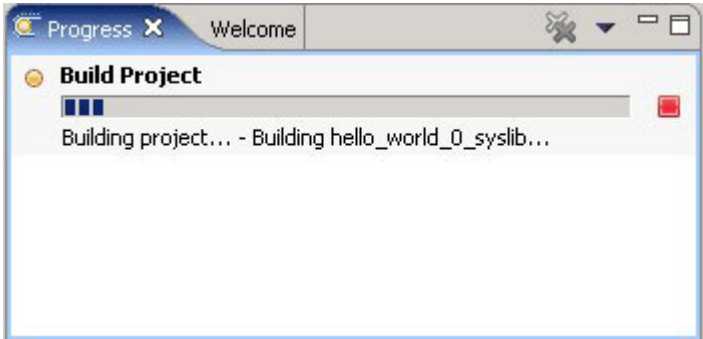


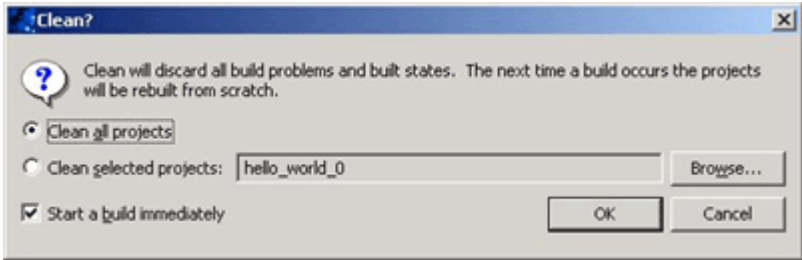
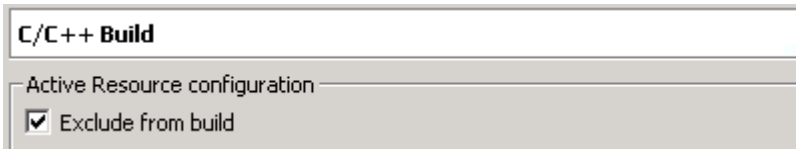
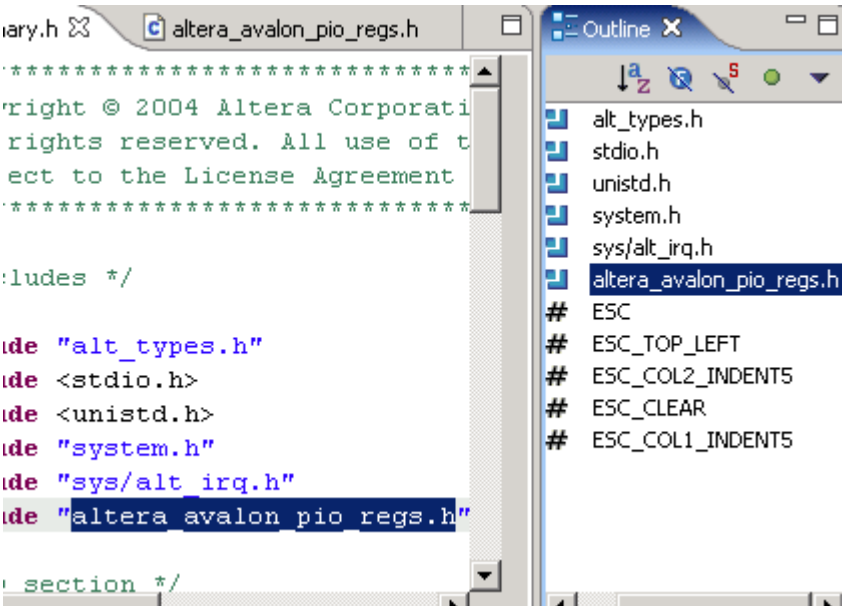
What's New in the Nios II IDE v5.0:

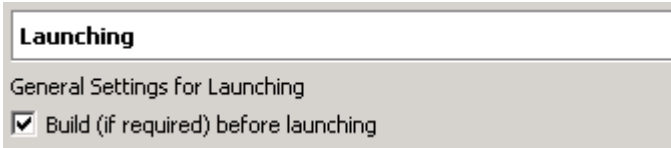
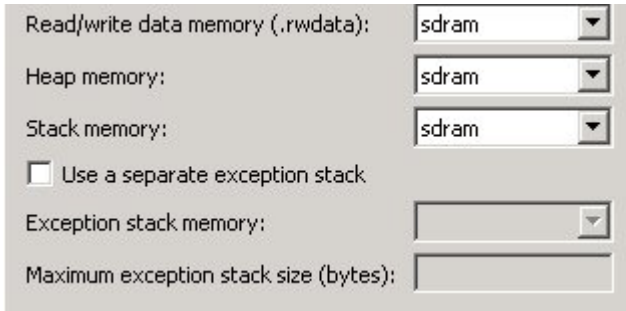
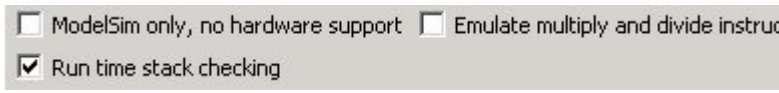
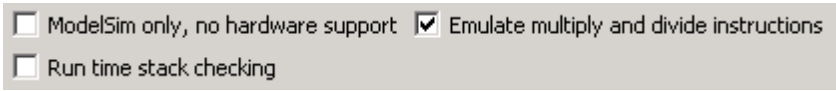
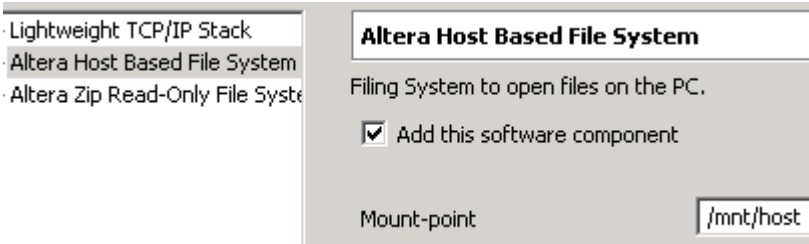
Below are the most significant changes to the Nios II IDE for the v5.0 release:

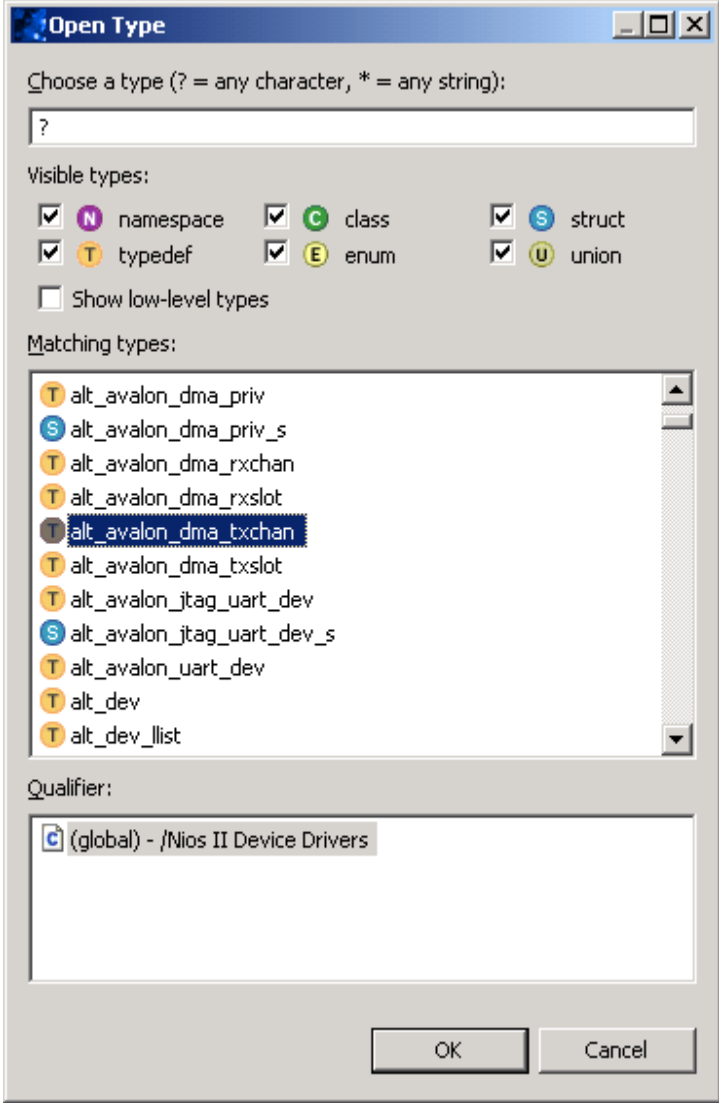
<p>New welcome page</p>	<p>The Nios II IDE now opens with a new Welcome page. It contains several pages for you to learn about the Nios II IDE, read tutorials, and learn what's new in this release.</p>  <p>The screenshot shows the 'Welcome to the Altera Nios II IDE' page. It features a blue background with a grid pattern. At the top right, there is a 'Workbench' logo. In the center, there are three icons: a globe labeled 'Overview', a chalkboard labeled 'Tutorials', and a star labeled 'What's New'. The 'eclipse' logo is visible in the bottom left corner.</p>
<p>Cheat sheets</p>	<p>You can now learn about the Nios II IDE with cheat sheets. Cheat sheets provide step by step guidance on how to use the Nios II IDE, and can automatically perform actions for you. To view the cheat sheets, choose Window > Help > Cheat Sheets in the Nios II IDE.</p>

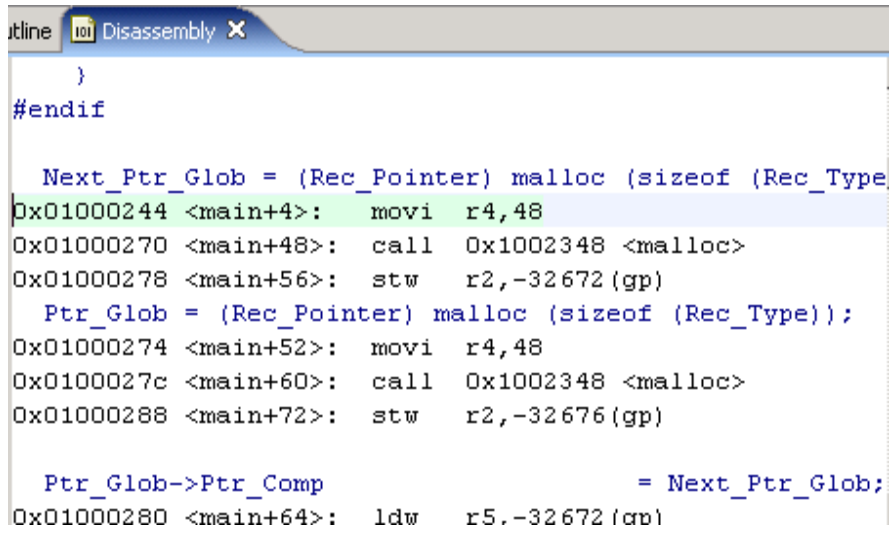
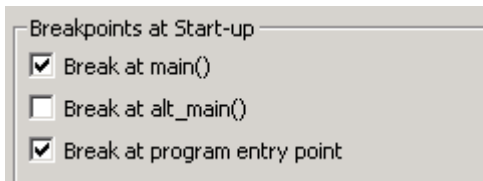
	 <p>The screenshot shows a window titled 'Cheat Sheets' with a sub-header 'Nios II IDE Quick-Start Tutorial'. Under the 'Introduction' section, it explains the tutorial's goal: creating a new C/C++ Application project and compiling a Hello World program. It provides instructions on connecting hardware (AC adapter and USB Blaster) and clicking a 'Click to Begin' icon. A note mentions that cheat sheets can perform some steps. Below the text is a list of tutorial steps, each with a plus icon and a question mark icon:</p> <ul style="list-style-type: none"> + Create a New Project ? + Navigate the Project ? + Build the Project ? + Run the Project ? + Configure the FPGA ? + Success ?
<p>New look and feel</p>	<p>The workbench has a new look and feel. Here are some of the things to notice:</p> <ul style="list-style-type: none"> • Title bars and tabs for views and editors look different. • Title bars and tabs for views and editors include maximize and restore options. • Views also include a minimize option. • The perspectives toolbar has greater flexibility. <ul style="list-style-type: none"> ○ The toolbar can be docked on the top right (default), top left or left. ○ Perspective buttons include text for quickly identifying the current perspective. <p>Right-click on a perspective icon to set these preferences.</p> • Drag and drop now offers better feedback while dragging. • The workbench includes many other minor items such as fastview style, status bar style, border widths, shading, etc...

	
<p>Run tasks in the background</p>	<p>The Nios II IDE now offers a higher level of responsiveness, with support for performing builds, searches, and launches in the background.</p> <p>Productivity improvements include the following:</p> <ul style="list-style-type: none"> • Progress view • Status line entry showing what's running in the background • Dialog box for showing operations that you can optionally run in the background 
<p>Open external files</p>	<p>You can now open and edit files that are not in the IDE workspace, using the File > Open External File... feature.</p>
<p>Simplified build commands & new "make clean..." option</p>	<p>There is now a simplified set of build commands in the Project menu. The new Clean... command replaces the previous Rebuild All and Rebuild Project commands. The Build Project command in the Project menu is smarter, and builds out-of-date prerequisite projects used by the selected project. A new Build Working Set submenu lets you choose a set of projects to be built; this command brings all projects in that working set up to date, building any prerequisite projects that are not in the working set if (and only if) required. You can</p>

	<p>quickly toggle auto-build on and off with Build Automatically.</p> 
<p>Exclude source files from a C/C++ Nios II project build</p>	<p>You can now control which source files are compiled when building a C/C++ project by using the new Exclude from Build feature. To exclude a file, select a C/C++ file in a Nios II C/C++ project, right click, and turn on Properties > C/C++ Build > Exclude from build.</p> 
<p>System library source code navigation improvements</p>	<p>You can now open header files, typedefs and structs declared in system library source files, from the C/C++ Outline view. This is now possible because the system library include paths are added to the Nios II IDE search list.</p> 
<p>Build before launch is now a global preference setting</p>	<p>The setting to automatically build a C/C++ application before launching a run/debug session has moved from the individual run/debug configuration dialog box to the global preference setting Window > Preferences > Run/Debug > Launch > Build (if required) before launching.</p>

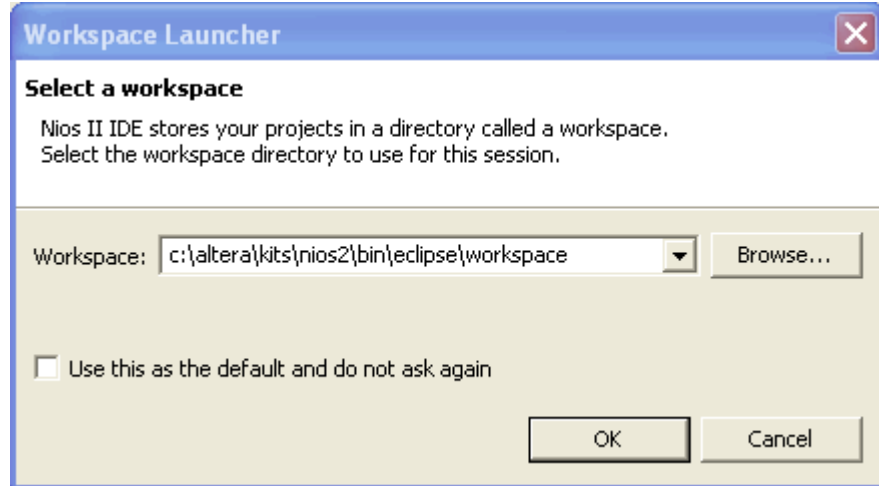
	
<p>Stack, heap & exception stack memory assignment</p>	<p>You now can individually specify memory sections for the stack, heap and exception stack. This setting is available on the system library project's Properties > System Library page.</p> 
<p>Stack overflow checking</p>	<p>The Nios II IDE and compiler now allow you to perform run-time stack memory overflow checks. Turn on Run time stack checking in the system library project System Library properties page.</p> 
<p>Emulate multiply and divide instructions</p>	<p>You can now disable the software emulation of multiply and divide instructions, resulting in reduced code size. Turn on Emulate multiply and divide instructions in the system library project System Library properties page.</p> 
<p>Host based file system</p>	<p>The host based file system is a new software component that allows allows the target processor to read from and write to files on the host computer in the C/C++ application project directory.</p> 

<p>Open type</p>	<p>Use Open Type to open up the declaration of C/C++ classes, structures, unions, typedefs, enumerations and namespaces. Specifically, you can open declarations from the Nios II Device Drivers project.</p> 
<p>Mixed C/C++ source disassembly debug view</p>	<p>You can now view interleaved C/C++ source code and disassembly code. To open the Disassembly view, choose Window > Show View > Other... > Debug > Disassembly.</p>

	 <pre>) #endif Next_Ptr_Glob = (Rec_Pointer) malloc (sizeof (Rec_Type)); 0x01000244 <main+4>: movi r4,48 0x01000270 <main+48>: call 0x1002348 <malloc> 0x01000278 <main+56>: stw r2,-32672(gp) Ptr_Glob = (Rec_Pointer) malloc (sizeof (Rec_Type)); 0x01000274 <main+52>: movi r4,48 0x0100027c <main+60>: call 0x1002348 <malloc> 0x01000288 <main+72>: stw r2,-32676(gp) Ptr_Glob->Ptr_Comp = Next_Ptr_Glob; 0x01000280 <main+64>: ldw r5,-32672(cpu) </pre>
<p>Start debugging from the program entry point</p>	<p>You can now start a debug session and pause the program at the program entry point. This is typically <code>_start</code> or the reset address. To break at the program entry point, open the appropriate run/debug configuration, from the Run > Debug... menu, select the Debug tab, and turn on Break at program entry point.</p> 
<p>Persistent settings across debug sessions</p>	<p>Many debug settings now persist between debug sessions, including expressions, breakpoints, global data watchpoints, and global variables.</p>
<p>Disabling and skipping breakpoints</p>	<p>The Breakpoints view now has checkboxes for quickly disabling and re-enabling breakpoints. The new Skip All Breakpoints button turns off all breakpoints in the workspace. The Window > Preferences > Run/Debug page also offers a setting to skip breakpoints during run-to-line operations.</p>
<p>New Nios II IDE Executable (nios2-ide.exe)</p>	<p>The new Nios II IDE executable uses the environment variables <code>SOPC_KIT_NIOS2</code> and <code>QUARTUS_ROOTDIR</code> to construct the runtime environment when starting up.</p> <p>As a result, the Tools Location page (Window > Preferences > Nios II > Tool Locations) is no longer required, and has been removed.</p>

**Session
workspace
path**

The Nios II IDE stores your projects in a directory called a workspace. Each user can now define a separate workspace to keep their environments separated as desired. Additionally, workspaces allow more than one instance of the Nios II IDE to run simultaneously, with each instance pointing to a different workspace.



Tutorials



About Tutorials

The Nios II IDE provides tutorials so beginning Nios II developers can get familiar with the development environment, user interface, and software development process. The following sections describe tutorials available in the Nios II IDE. The Altera website also provides tutorials that Nios II developers might find useful.

▶ Quick-Start Tutorial:

This tutorial guides you through the process of creating a new project, compiling it, and running it on a Nios development board. You will create a new C/C++ application project, and compile a hello world program.

To start the Nios II quick-start tutorial, do the following:

1. On the Help menu, click **Cheat Sheets**....
2. Click **Nios II Quick-Start Tutorial**.
3. Click **OK**.

▶ Software Development Tutorial:

This tutorial provides the information you need to create, build, and debug a C/C++ application and its associated system library. The tutorial steps you through running and debugging Nios II software on a target board and the instruction set simulator. It also explains various options available for configuring your project.

The tutorial is part of this help system and contains the following topics:

- Creating a C/C++ Application Project - Create a C/C++ application project containing your application code and corresponding build settings.
- Building the Project - Build your C/C++ application project.
- Running the Project - Set up a run/debug configuration and run your application code on a target board or the instruction set simulator.
- Debugging the Project - Debug your code, set breakpoints, step through your code, and view memory contents.
- Editing the Project Properties - Edit your application project and system library properties.

Use the links near the bottom of each tutorial page to advance to the next and previous tutorial pages.



Related Topics on the Web


- Nios II Hardware Development Tutorial at www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf
- Literature: Nios II Processor at www.altera.com/literature/lit-nio2.jsp - Contains links to several tutorials and other "how to" information.




Creating a C/C++ Application Project


A C/C++ application project contains your program code and corresponding build settings. In this tutorial you create a C/C++ application project for the full featured hardware design provided in the Nios II Embedded Design Suite (EDS). The example C program exercises the visible output devices on a Nios development board, such as the LEDs and LCD screen.


Follow the steps below to create a C/C++ application project using the **New Project** wizard.

1. On the File menu, point to **New**, and then click  **C/C++ Application**. The **New Project** wizard appears.
2. Type `tutorial` as the name of the project in the **Name** box.
3. Accept the default location for the Nios II C/C++ application project by leaving **Use Default Location** on. The default location is shown in the **Location** box. The name of the project directory is the project name, `tutorial`.


 **Note:** When on, the Nios II IDE creates the new project directory in the default location specified in the **New Projects** preference page. At installation, the default location is the `<SOPC Builder system path>/software` directory. Altera recommends using this location because it keeps software files in proximity to the system hardware files. Turning off **Use Default Location** allows you to specify an alternative project location in the **Location** box.

4. Click **Browse** next to the **SOPC Builder System** box, and browse to the following directory:
`<Nios II EDS install path>/examples/verilog/<Nios development board>/full_featured`
5. Select the SOPC Builder system file (`.ptf`) in this directory (e.g. `full_1c20.ptf`) and click **Open**. You return to the **New Project** wizard, with data appearing in the **SOPC Builder System** and **CPU** boxes.

 **Note:** The SOPC Builder system file describes the CPUs, memories, and peripherals contained in a Nios II hardware system. SOPC Builder generates the file, and the Nios II IDE uses it when building projects, downloading code, and communicating with the target hardware. For designs targeting custom hardware, obtain an SOPC Builder system from the hardware designer.

 **Note:** If your SOPC Builder system has more than one Nios II CPU, you can select a specific CPU for the project in the **CPU** box. Since the `full_featured` example design contains only one CPU (named "cpu"), only one CPU is available in the list.

7. Select the **Count Binary** project template in the **Select Project Template** list. The Count Binary example continuously sends a counting pattern to the LEDs, the seven segment display, and the LCD display on the Nios development board.

 **Note:** Each template is a collection of software files and project settings that serve as a base for the new project. The Nios II IDE automatically copies the source files into the new project's directory. You can add your own source code to the project later.

8. Click **Finish** to exit the wizard. The **New Project** wizard closes, and you return to the Nios II IDE workbench.

7.



Note: The **New Project** wizard actually creates two projects when you click **Finish**:

- The C/C++ application project **tutorial** contains your source code.
- The system library project **tutorial_syslib** contains drivers and library files for the selected SOPC Builder system. The system library project serves as a board support package for the target hardware.

8.

In the IDE workbench you can browse the software files in a project using the C/C++ Projects view in the left-hand pane of the IDE. Double-clicking a file in the C/C++ Projects view opens the file in the IDE editor.



Note: The left-hand pane in the IDE has two views: C/C++ Projects view and the Navigator view. Click the C/C++ Projects tab at the top of the pane to display the C/C++ Projects view. This view is appropriate for most C/C++ development activity.

Now, you are ready to build your C/C++ application project.

Next: Building the Project



Related Nios II IDE Help Topics

- Importing and Exporting Files and Projects - Contains details on importing C files into the Nios II IDE.



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Tasks > Writing code - Describes how to customize and use the C/C++ editor.



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on writing programs for the Nios II processor.



Building the Project

Building a project in the Nios II IDE compiles and links all the source code associated with the project and the system library project. The result of building is an executable file (**.elf**) that you can run or debug.

Use one of the following methods to build the project:

- Right-click on the project in the C/C++ Projects view, and click **Build Project**.
- Click the project in the C/C++ Projects view, then click **Build Project** on the Project menu.



Note: Make sure you build the application project **tutorial**, *not* the system library **tutorial_syslib**. Building the system library does not create an executable file.

When building a project, the Nios II IDE first builds all other projects that the current project references. Therefore, the system library project is usually built first. Building a project the first time might take a few minutes while the Nios II IDE builds the system library. Subsequent builds are faster.

In the IDE workbench, the Console view displays messages generated by the GCC tool-chain during compilation. The Problems view displays any warnings or errors. (In this example, there are no errors.)



Note: If building the project generates warnings or errors, the Problems view appears automatically. You can double-click each row to jump to the line of code that caused the warning/error.



Note: The information displayed in the Console and Problems views depends on the project selected in the C/C++ Projects view. If you accidentally select a different project, you will see unexpected (or no) text in the Console and Problems views.

After successfully building the project, you can download and run the application on your development board. Refer to [Running the Project](#) for instructions. To debug your application on the board, refer to [Debugging the Project](#).

Next: [Running the Project](#)

Previous: [Creating a C/C++ Application Project](#)



Related Nios II IDE Help Topics

- [C/C++ Projects View \(C/C++ Perspective\)](#) - Contains details on options available by right-clicking in the C/C++ Projects view.



Running the Project

Running a project in the Nios II IDE executes the project code so you can analyze the output. You can run a project on these different targets using the Nios II IDE:

- Nios II hardware
- Nios II instruction set simulator (ISS)
- ModelSim hardware simulator

This tutorial discusses running on a Nios development board (i.e., a Nios II hardware target) and on the ISS.

Running has the following basic steps.


▶ **To configure the hardware:** (not necessary for the ISS)

When targeting Nios II hardware, you must configure the FPGA on the development board with your project's associated SOPC Builder system. The factory-programmed SOPC Builder system (i.e. the FPGA hardware design) on the Nios development board is different than your project's SOPC Builder system. Therefore, your executable code will not run unless you configure the FPGA with the expected SOPC Builder system.




Note: You only need to configure the FPGA when you reset the board or if the SOPC Builder system file changes. Normally, you configure the FPGA once after you apply power to the board, and the configuration persists through the duration of the Nios II IDE session.

To configure the FPGA, perform the following steps:

1. Click the **tutorial** project in the C/C++ Projects view.
2. Perform one of the following actions to launch the Quartus II Programmer. The Quartus II Programmer is a tool for configuring Altera FPGAs via a JTAG download cable, such as the USB Blaster.
 - On Windows, click  **Quartus II Programmer...** on the Tools menu.
 - On Linux, launch the Quartus II software, and click **Programmer** on the Tools menu.
3. Click **Add File...** in the Quartus II Programmer to browse to the FPGA configuration file (**.sof**) for your project. The **Select Programming File** dialog box appears.
4. Browse to *<Nios II EDS install path>/examples/verilog/<Nios development board>/full_featured*, which is the location of the **full_featured** example hardware design that corresponds to your Nios development board.
5. Click the file **full_featured.sof**, then click **Open**. You return to the Quartus II Programmer.
6. Turn on the **Program/Configure** checkbox for the target device.
7. Click **Start**. When the progress meter progresses to 100%, configuration is complete.

 **Note:** If **Start** is not enabled, click **Hardware Setup** to configure your Altera download cable.

- Close or minimize the Quartus II Programmer and return to the Nios II IDE.

 **Note:** If you are targeting a board other than the Nios development board, you need to configure the Quartus II programmer differently, as specified by the board designer. You can save the Quartus II configuration settings to a chain descriptor file (.cdf), eliminating the need to configure the Quartus II programmer every time you configure the device.


The board is now configured, and ready to run the project's executable code.

▶ To run the project:

You can run your project on your target hardware, or on the ISS.

Running on Hardware


After configuring the target hardware, perform these steps to download and run the executable code:

- Right-click the **tutorial** project in the C/C++ Projects view.
- Point to **Run As**, and then click  **Nios II Hardware**. After a moment, the board's LEDs, seven-segment display, and LCD screen count from 0x00 to 0xff, pause, and then repeat.

The Console view in the IDE also displays the following:


```
*****
* Hello from Nios II!      *
* Counting from 00 to ff *
*****
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e, 1f,
```

Congratulations! The program is running on your development board.

 **Note:** If nothing displays on the LCD screen, disconnect the CompactFlash card, and repeat the steps. The CompactFlash and LCD screen share lines on the Nios development board, sometimes causing conflicts.


When targeting Nios II hardware, the **Run As** command does the following:

- Creates a default run/debug configuration for the target board.

 **Note:** This step usually completes automatically without user intervention. If it cannot (the most common cause is that you have multiple JTAG download cables installed), the IDE displays an error message, and you must manually set up a run configuration.


- Builds the project. If the project is not up-to-date, then the IDE builds it first to generate an up-to-date executable file.


3. Establishes communication with the target board, and verifies that the expected SOPC Builder system is configured in the FPGA. If the FPGA is not configured properly, you should repeat the steps to configure the hardware.
4. Downloads the executable file (.elf) to memory on the target board.
5. Instructs the Nios II CPU to begin executing the code.

After using the **Run As** command once, to run again click  **Run** on the toolbar.

Running using the ISS


To run using the ISS, perform these steps to run the executable code:

1. Right-click the **tutorial** project in the C/C++ Projects view.
2. Point to **Run As**, and then click  **Nios II Instruction Set Simulator**. After a moment, output displays in the Console view. The count output appears very slowly because there are delay loops (usleep function calls) in the code.

 **Note:** The ISS does not model the LED, seven-segment display, or LCD peripherals. Only the console output displays in the Console view.

▶ To analyze the output:

Program output appears on the development board and in the Console view of the Nios II IDE. The Console view maintains a terminal I/O connection with a communication device connected to the Nios II processor in the SOPC Builder system, such as a JTAG UART. When the Nios II program writes to stdout or stderr, the Console view displays the text. The Console view can also accept character input from the host keyboard, which is sent to the CPU and read as stdin. (The count_binary.c program does not read any input, so typing in the Console view when running this project has no effect.)

To terminate the terminal connection to the target, click  **Terminate** in the Console view. Terminating only disconnects the host from the target; the target CPU continues executing.

After successfully running the project, you are ready to learn how to use the debugger in the next section of the tutorial.

Next: Debugging the Project

Previous: Building the Project

Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Instruction Set Simulator (ISS) - Contains details on the capabilities and limitations of the ISS.
- C/C++ Projects View (C/C++ Perspective) - Contains details on options available by right-clicking in the C/C++ Projects view.


Related Topics on the Web

- AN 351: Simulating Nios II Embedded Processor Designs at www.altera.com/literature/an/an351.pdf - Contains details on ModelSim simulation.



Debugging the Project

The Nios II IDE contains an integrated debugger that allows you to debug your program on Nios II hardware or on the instruction set simulator (ISS). This topic introduces the main features of the debugger. The tutorial assumes you are working with a Nios II hardware target. The process is the same for debugging on the ISS, with the exception of steps that involve the target hardware.


 **Note:** When debugging on Nios II hardware, the FPGA on the development board must be configured with your project's associated SOPC Builder system. If you configured the FPGA before running your project in the previous tutorial step, you do not have to reconfigure the FPGA again unless you reset the board or remove power. If you need to configure the FPGA, follow the steps to configure the hardware in the Running the Project tutorial topic.



These sections describe how to control the flow of a debug session.

▶ To display line numbers next to each line of code in the editor:

1. Click **Preferences** on the Window menu.
2. Expand **C/C++**, and click **Editor**.
3. Click the **Appearance** tab.
4. Turn on **Show line numbers**.
5. Click **OK**.

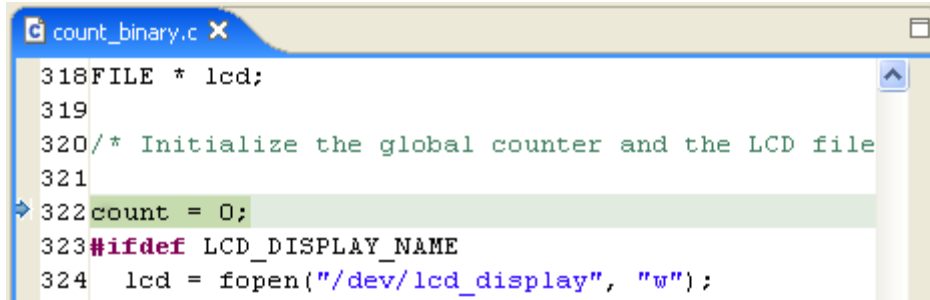
▶ To download executable code and start the debugger:

1. Right-click the **tutorial** project in the C/C++ Projects view, point to **Debug As**, and then click  **Nios II Hardware**.

 **Note:** To debug on the ISS instead, point to **Debug As**, and then click  **Nios II Instruction Set Simulator**. The `count_binary.c` program used in this tutorial primarily interacts with hardware on the development board which cannot be simulated using the ISS. Some of the steps below do not work with the ISS for this program.

2. If the **Confirm Perspective Switch** dialog box appears, click **Yes**.

After a moment, you see the `main()` function of the Count Binary design in the editor. There is a blue arrow next to the first line of code, as shown below, indicating that execution is stopped on this line. Note that the exact line numbers might vary on your screen.



```


318 FILE * lcd;
319
320 /* Initialize the global counter and the LCD file
321
322 count = 0;
323 #ifdef LCD_DISPLAY_NAME
324  lcd = fopen("/dev/lcd_display", "w");

```


Notice that the perspective of the Nios II IDE changed from the C/C++ Development perspective to the Debug perspective. A perspective is a different configuration of the Nios II IDE workbench. Refer to Related Topics for more information about perspectives. You can switch between perspectives anytime by pointing to **Open Perspective** on the Window menu or by clicking on the shortcut buttons on the far-left column of the Nios II IDE window border.

When targeting Nios II hardware, the **Debug As** command does the following:




1. Creates a default run/debug configuration for the target board.


 **Note:** This step usually completes automatically without user intervention. If it cannot, the IDE displays an error message and you must manually set up a run/debug configuration. The most common reason for manual intervention is having multiple JTAG download cables installed. In this case you need to select one manually.

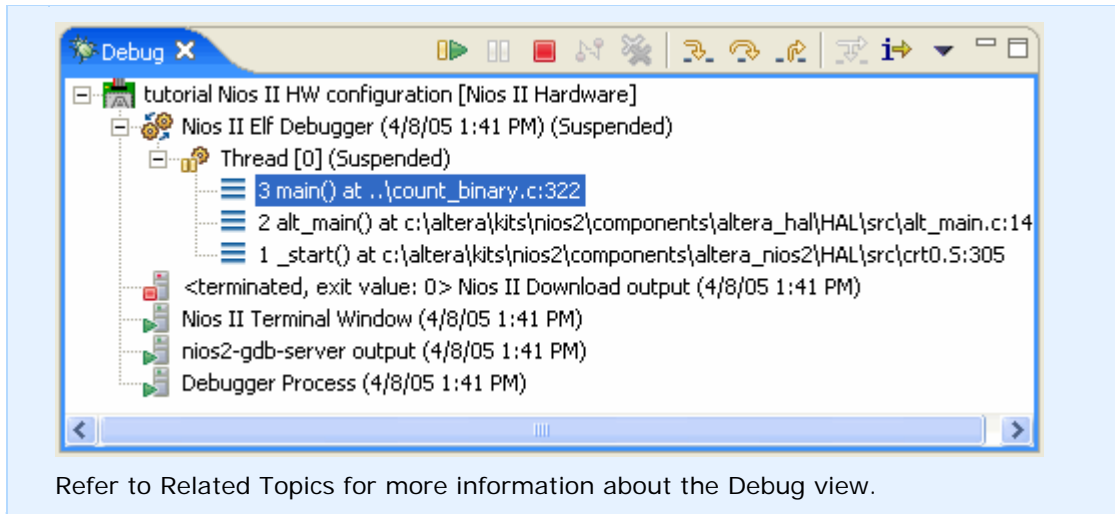
2. Builds the project. If the project is not up-to-date, then the IDE builds it first to generate an up-to-date executable file.
3. Establishes communication with the target board, and verifies that the expected SOPC Builder system is configured in the FPGA.
4. Downloads the executable file (**.elf**) to memory on the target board.
5. Sets a breakpoint at `main()`.
6. Instructs the Nios II CPU to begin executing the code.


After using the **Debug As** command once, you can click  **Debug** on the toolbar to start the debugger again.

 To resume and suspend execution:




- Click  **Resume** in the Debug view to resume execution. You can also resume execution by pressing **F8**.
- Click  **Suspend** in the Debug view to suspend execution. If the processor suspends outside the scope of the current file, the IDE opens the source file corresponding to the current program counter.
- Click  **Terminate** in the Debug view to end the debug session and disconnect from the target.

 **Note:** The Debug buttons are context sensitive, depending on the currently highlighted selection in the Debug view. Make sure you select an item under **Nios II Elf Debugger**, as shown below, to ensure you are debugging the desired thread.



If you accidentally terminate the debug session, or the download cable connection is interrupted, you can easily start a new debugging session by clicking  **Debug** on the toolbar.



▶ **To step through the C/C++ code line by line:**

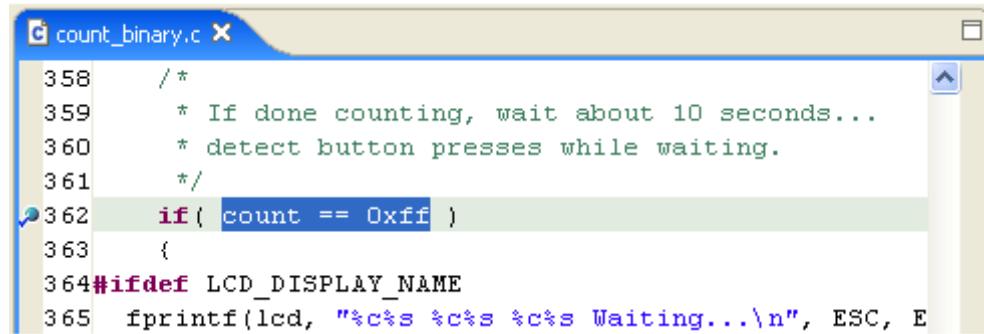
- Click  **Step Into**. If executing a line of code that calls a function, the debugger steps into the function. Otherwise it executes the line of code and suspends on the next line in the current function. You can also step into a function by pressing **F5**.
- Click  **Step Over**. If executing a line of code that calls a function, the debugger executes the entire called function and suspends on the next line in the current function. Otherwise it executes the line of code and suspends on the next line in the current function. You can also step over a line of code by pressing **F6**.
- Click  **Step Return**. The debugger finishes executing the current function, returns to the calling function, and suspends on the next line in the calling function. You can also step return from a function by pressing **F7**.

▶ **To use breakpoints and watchpoints:**

You can set breakpoints on specific lines of code, remove breakpoints, or disable them temporarily. Enabled breakpoints suspend execution when the processor reaches that line of code.

To set a breakpoint for this example:


1. If the processor is running, select **Thread [0] (Running)** in the Debug view and click  **Suspend**. You can only add breakpoints while the processor is suspended.
2. Click the **count_binary.c** tab in the editor.
3. On the Edit menu, click **Find/Replace....**
4. Type `count == 0xff` in the **Find** box, then click **Find**. The editor displays the appropriate line of code.
5. Click **Close** in the **Find/Replace** dialog box.
6. Double-click in the margin next to the line `if(count == 0xff)` to set a breakpoint. You can also right-click the margin and click **Toggle Breakpoint**. The  breakpoint symbol appears in the margin, as shown below.




```



count_binary.c
358  /*
359  * If done counting, wait about 10 seconds...
360  * detect button presses while waiting.
361  */
362  if( count == 0xff )
363  {
364  #ifdef LCD_DISPLAY_NAME
365  fprintf(lcd, "%c%s %c%s %c%s Waiting...\n", ESC, E

```


 **Note:** You must click in the margin to the left of the line of code. Clicking within the editor does not affect breakpoints.

- Click  **Resume** in the Debug view. The processor resumes and then suspends just before executing the line of code with the breakpoint. The editor displays an arrow in the margin next to the suspended line of code. It might take a moment for the program to execute to the breakpoint. Resume again to iterate through the loop another time.

To remove a breakpoint:

- Double-click the  breakpoint symbol in the margin. You can also right-click the  breakpoint symbol, and then click **Toggle Breakpoint**.

To disable a breakpoint:

- Right-click the  breakpoint symbol, and then click **Disable Breakpoint**. Disabling temporarily prevents a breakpoint from suspending the processor while leaving it in place for future reference.

To use Breakpoints view:



- Click the **Breakpoints** tab in the upper-right pane of the Debug perspective to display the Breakpoints view. This view displays the location and status of all breakpoints you have previously set on specific lines in the code every time the processor hits a breakpoint or suspends. Values that have changed since the last time the processor suspended display in red.
- Right-click a breakpoint in the list, and then click **Enable**, **Disable**, or **Remove** to change the status of the breakpoint.


The Breakpoints view also displays watchpoints. Refer to Related Topics for more information about debugging with watchpoints.

Several default views in the Debug perspective help you to organize, navigate, and analyze your project during a debug session. The Nios II IDE updates each view every time the processor hits a breakpoint or suspends. Values that have changed since the last time the processor suspended display in red.

▶ To view disassembly:

When the processor suspends, Disassembly view automatically appears. You can also open the Disassembly view from the Window menu by pointing to **Show View**, and clicking **Disassembly**. This view displays the assembly language instructions interleaved with the C/C++ source code.

- Click  **Instruction Stepping Mode** in the Debug view toolbar to allow single stepping through the individual assembly instructions. Stepping through assembly code advances the instruction pointer in the Disassembly view. Because multiple assembly instructions represent a single line of C/C++ code, the instruction pointer might not advance in the C/C++ Editor view with each step through the assembly instructions.
- Click  **Instruction Stepping Mode** a second time to return to single stepping in the Editor view at the C/C++ statement level.

 **Note:** If you do not have the source code for a function, stepping through the code automatically uses Disassembly view regardless of whether instruction stepping mode is on or off.

▶ To view stack trace:

The Debug view displays the program execution stack and dynamically updates it as you step through code. Any time the processor suspends, the Debug view displays the name of the suspended function, and the sequence of function calls that led up to the current program counter. This view provides a snapshot of your current position within the program execution.

Refer to Related Topics for more information about the Debug view.

▶ To view execution trace:

On the Window menu, point to **Show View**, and then click **Trace**. When executing a program on a Nios II hardware target, the Trace view displays the exact execution trace of the program running in hardware. This view provides a snapshot of the specific code that executed to arrive at the current position.


▶ To track variables:


Click the **Variables** tab in the upper-right pane of the Debug perspective to display the Variables view. You can also point to **Show View** on the Window menu, and then click **Variables**.

Local Variables

The Variables view automatically displays all variables local to the scope where the processor is suspended. Use the Variables view to track and change variable values on-the-fly during a debug session. This is useful to test your program's response to specific conditions, or to force a loop index to skip over a loop.




The Variables view is context sensitive, depending on the currently selected function in the Debug view's stack trace display. Selecting different functions allows you to see the variables (and their current values) defined at each level of the stack trace.

 **Note:** Hovering the mouse over a variable in the source code displays the variable's value as a tool-tip. This is often the easiest way to see the value of a variable in the current scope.

To change the value of a variable, right-click the variable name and then click  **Change Value....** This opens the **Set Value** dialog box, which allows you to specify a new value for the variable.

Global Variables

You can also selectively display global variables, which are variables defined outside the scope of all functions, but are available from within any function. In the **count_binary.c** example, to track the global variable `count` in the Variables view, do the following:

1. If the processor is running, click **Thread [0] (Running)** in the Debug view, and click  **Suspend**.
2. Right-click in the Variables view and then click  **Add Global Variables...**, or click  **Add Global Variables** on the Variables view toolbar.
3. Scroll down and turn on **count**.
4. Click **OK**. The variable `count` and its current value appears in the Variables view. Because `count` is declared as a `char` type, it displays in ASCII format by default in the Variables view.

Now, every time execution suspends, the Variables view displays the value for the variable `count`.

Variables Display Format

You can change the display format of variable values appearing in the Variables view in two ways. For example, to change the display format of a single variable to hexadecimal, do the following:

1. Right-click the variable, point to **Format**, and then click **Hexadecimal**. The format of the value changes.

To change the display format of all variables to hexadecimal, do the following:

1. On the Window menu, click **Preferences**.
2. Expand **C/C++**, **Debug**.
3. Select **Hexadecimal** in the **Default variable format** list.
4. Click **OK** to close the **Preferences** dialog box.

The format of the values for local variables changes the next time you resume execution. The format of the values for global variables changes the next time you restart the debug session.

Refer to Related Topics for more information about variables.

▶ To track watch expressions:

Click the **Expressions** tab in the upper-right pane of the Debug perspective to display the Expressions view. You can also point to **Show View** on the Window menu, and then click **Expressions**. This view displays user-specified C expressions evaluated at the current scope. When the processor suspends, the Expressions view evaluates each of the expressions, and displays the value.

In the **count_binary.c** example, to track the arbitrary expression `count==5` in the Expressions view, do the following:

1. If necessary, use a breakpoint to stop execution on the line `if(count == 0xff)`, as described in the breakpoints section of this topic.


```

358  /*
359  * If done counting, wait about 10 seconds...
360  * detect button presses while waiting.
361  */
362  if( count == 0xff )
363  {
364  #ifdef LCD_DISPLAY_NAME
365  fprintf(lcd, "%c%s %c%s %c%s Waiting...\n", ESC, E

```

2. Highlight the expression `count == 0xff` in the code.
3. Right-click and then click **Add Watch Expression...** The **Add Watch Expression** dialog box appears with the **Expression to watch** box automatically filled in with the selected text: `count==0xff`.
4. Change **Expression to watch** to `count==5`.
5. Click **OK**. The expression `"count==5" = 0` appears in the **Expressions** view in the upper-right window.

Now, every time execution suspends at a breakpoint, the Expressions view evaluates the expression `count==5`. If execution suspends when `count` equals 5, then `count==5` evaluates true, signified by the expression `count==5=1`.

You can assign an IDE preference to show expressions values in a different format, such as hexadecimal, as described in variables section of this topic. However, you must restart the debug session to force the format to change. Refer to Related Topics for more information about expressions.

▶ To view and edit registers:

Click the **Registers** tab in the upper-right pane of the Debug perspective to display the Registers view. You can also point to **Show View** on the Window menu, and then click **Registers**. This view displays the contents of registers.

To edit the contents of a particular register, right-click it in the Registers view and click **Change Value...**

Note: Altera recommends that you do not manually edit register values, because it could cause your program to behave unpredictably.


You can assign an IDE preference to show register values in a different format, such as hexadecimal, as described in variables section of this topic. However, you must restart the debug session to force the format to change. Refer to Related Topics for more information about registers.


▶ To view and edit memory:


Click the **Memory** tab in the upper-right pane of the Debug perspective to display the Memory view. You can also point to **Show View** on the Window menu, and then click **Memory**. This view displays the contents of memory.

The Memory view has four tabs, **Memory 1** to **Memory 4**, which allows you to track multiple locations in memory without having to continually type in addresses by hand. The **Address** box accepts constants and expressions, which allows you to search for a memory location by symbolic name.

The following steps demonstrate various uses of the Memory view.

1. If the processor is running, click **Thread [0] (Running)** in the Debug view, and then click  **Suspend**.
2. In the Memory view, click the **Memory 1** tab.
3. Type 0x00 in the **Address** box and click **Evaluate**. Note the contents.
4. Right-click in the memory contents area, point to **Memory Unit Size**, and then click **4 bytes**.
5. Click the **Memory 2** tab.
6. Type &count in the **Address** box and click **Evaluate**. This displays memory contents at the location where variable `count` is stored.
7. Click on the left edge of the top-left-most byte displayed in the Memory view. (Clicking sets the insertion point. Do not highlight the byte.)
8. Type a value, such as 66. This updates the memory contents as you type. To move to the next byte, use the right-arrow key. You can also scroll to the right edge of the Memory view, click in the ASCII field, and type characters.

 **Note:** When you use the Memory view to edit memory contents, the Nios II IDE writes each hexadecimal digit to memory as you type it, i.e., 4 bits at a time. Or, if you click in the ASCII area and type, data is written 8 bits at a time. While this is acceptable for modifying memory, it is generally not suitable for accessing a peripheral's registers. Currently, there is no way to write 32 bits at a time.

If the memory region displayed is located in read-only memory, the Nios II IDE does not respond to attempts to edit the memory content. The Memory view is set to refresh itself automatically; however, you can also manually refresh by right-clicking in the Memory view, and then clicking  **Refresh**.

Refer to Related Topics for more information about memory.

Next: Editing the Project Properties

Previous: Running the Project



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Instruction Set Simulator (ISS) - Contains details on the capabilities and limitations of the ISS.
- Advanced Debugging Features by FS2



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Reference > C/C++ Views and Editors > Debug view > Debug view
- C/C++ Development User Guide > Tasks > Running and Debugging > Adding expressions
- C/C++ Development User Guide > Tasks > Running and Debugging > Debugging a program > Adding watchpoints
- C/C++ Development User Guide > Tasks > Running and Debugging > Working with memory
- C/C++ Development User Guide > Tasks > Running and Debugging > Working with registers

- [C/C++ Development User Guide > Tasks > Running and Debugging > Working with variables](#)
- [Workbench User Guide > Tasks > Working with perspectives](#)



Editing the Project Properties

A project's **Properties** dialog box controls how the program interacts with the system hardware and how the Nios II IDE builds the application. Settings available vary for each project type. Due to the way the Nios II IDE separates and associates C/C++ application projects and system library projects, settings in a system library project can significantly impact the C/C++ application project.

▶ To view and modify C/C++ application project properties:

Right-click on the project in the C/C++ Projects view and click **Properties**, or click the project in the C/C++ Projects view and then click **Properties** on the Project menu. The Nios II IDE automatically sets most C/C++ application properties correctly for you.



▶ To view and modify system library properties associated with a C/C++ application project:

Right-click a C/C++ application project, and then click **System Library Properties**. Alternatively, you can right-click the system library project directly, and then click **Properties**. System library properties specify how your program interacts with the underlying hardware.

▶ To reduce code footprint by editing system library properties:


This example reduces the code size of your executable program by adjusting the system library properties. A significant reduction in code size is possible beyond the reduction shown here, and the system library properties control more than code footprint. This section simply demonstrates one way that editing system library properties can impact your system.

First, rebuild the project and determine the original code size. Perform the following steps:

1. If a debug session is still running, click  **Terminate** in the Debug view to disconnect from the target.
2. Click the  C/C++ Development perspective icon in the margin near the left edge of the window.
3. Click the **tutorial** project in the C/C++ Projects view, and then click **Clean....** on the Project menu. The **Clean?** dialog box appears.
4. Select **Clean selected projects**.
5. Turn off **Start a build immediately**.
6. Click **OK**.
7. Right-click the **tutorial** project in the C/C++ Projects view, and choose **Build Project**.

When the build completes, you can see the size of the resulting executable code, displayed in the Console view:

```
Info: (tutorial_project.elf) 78 KBytes program size (code + initialized data).
```

 **Note:** The program size you see might differ depending on your target SOPC Builder system.

Next, change the system library settings to reduce code size. Perform the following steps:

1. Right-click the **tutorial** project in the C/C++ Project view, and click **System Library Properties**. The **Properties** dialog box appears.
2. Click **System Library** in the left-hand pane. The **System Library** page appears.
3. Turn off **Clean exit (flush buffers)** - This option affects how the program behaves after `main()` returns. This example program, similar to most embedded programs, never returns from `main()`. Therefore, the program has no need for any post-`main()` code.
4. Turn on **Reduced device drivers** - This option causes each device driver in the system library to link in a reduced foot-print version of its driver, if it has one. In this example, the JTAG UART switches to a smaller, polled-operation driver (by default it is interrupt-driven), which executes slower but has a smaller code foot-print. The LCD driver's response to this option is to include no driver, and therefore the LCD will stop functioning.
5. Click **OK** to accept these settings. You return to the Nios II IDE workbench.
6. Right-click the **tutorial** project in the C/C++ Projects view, and click **Build Project**.

When the build completes, you can see the size of the resulting executable code, displayed in the Console view:

```
Info: (tutorial_project.elf) 53 KBytes program size (code + initialized data).
```

Congratulations! You have completed the Software Development Tutorial.

You can learn more about the Nios II IDE in the Nios II IDE Help, and by reading the Nios II documentation located at `<Nios II EDS install path>/documents/index.htm`. On Windows, you can click **Start**, point to **Programs, Altera, Nios II**, and then click **Nios II Documentation**.

Previous: Debugging the Project



Related Nios II IDE Help Topics

- Properties Dialog Box
- System Library Page (Properties Dialog Box)



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on writing programs for the Nios II processor.

Creating Projects



About Nios II IDE Projects

An individual Nios II IDE project is a group of files treated as a unit, containing source code, makefiles, binaries, and other related files. The Nios II IDE provides four project types.

▶ C/C++ application project:

A C/C++ application project contains a C/C++ program, usually including a project's `main()` function. Building a C/C++ application project results in an executable file (`.elf`) that you can run on target hardware, the Nios II instruction set simulator (ISS), and the ModelSim hardware simulator. A C/C++ application project depends on a single system library project and might reference functions in a managed library project.

A C/C++ application project is a Nios II IDE managed-make project. The Nios II IDE creates the necessary makefiles and manages the project for you.

▶ System library project:

A system library project serves as a board-support package for the target hardware. A system library project contains all settings that affect how a program interacts with the target, including the hardware abstraction layer (HAL) code. C/C++ application projects depend on a system library project, making the application code portable to other Nios II systems. Multiple C/C++ applications can share a single system library.

A system library project contains automatically-generated code based on hardware-specific information contained in two files provided by the hardware designer. Both of these files are generated during the normal hardware development flow. SOPC Builder generates the files, and the Nios II IDE uses them to create and build projects.

- The FPGA configuration file (`.sof`) - This binary file contains the hardware design for the target FPGA. After downloading the `.sof` file to the board, the FPGA behaves as specified by the hardware design, which in this case, includes a Nios II processor system.
- The SOPC Builder system file (`.ptf`) - This file contains a description of the Nios II processor system, including CPU cores, memories, and peripherals. The hardware image of the SOPC Builder system is included in the FPGA configuration file.

After building a system library project, the **Device Drivers** directories found in the C/C++ Projects view under the system library project contain symbolic links to shared source code, for example, the HAL system library and peripheral device drivers. These links allow the Editor view to display shared code, so you can set breakpoints in the code. Do not alter or delete anything in these directories.

▶ Managed library project:

A managed library project can contain reusable, general purpose functions that multiple C/C++ application projects can share. A library containing common arithmetical functions is one example. Building a managed library project results in a `.a` library file. The Nios II IDE manage these library projects for you. Managed library projects typically do not have dependencies on a system library project. You can reference managed library projects from C/C++ application projects or from other managed library projects.

▶ Advanced C/C++ project:

Using an advanced C/C++ project gives you total control over the build process. However, creating and managing the makefile becomes your responsibility. The advanced C/C++ project can contain any files required by the makefile, and the result of building the project is whatever you specify in the makefile. An Advanced C/C++ project is the same as a standard make project in the Eclipse C/C++ Development Toolkit (CDT).

A typical executable program consists of two or more individual projects working in conjunction. The most common combination is a C/C++ application project that depends on a system library project.

Project properties control how the project builds and functions. Project dependencies allow a project to reference other projects that reside in your workspace.

**Related Nios II IDE Help Topics**

- About the Nios II IDE Managed-Make Build Environment
- Creating a New Project

**Related Topics on the Web**

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - The chapters related to the hardware abstraction layer (HAL) system library contain details on system libraries and the structure of Nios II software projects.
- Nios II Hardware Development Tutorial at www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf



About the Nios II IDE Managed-Make Build Environment

This topic describes the Nios II IDE build environment for building programs based on the hardware abstraction layer (HAL) system library. If you are a typical user, the Nios II IDE completely manages the build environment for you, relieving you of the need to understand the inner workings of this managed-make environment.

Internally, the Nios II IDE uses a standard GNU GCC compiler tool chain to compile projects. The Nios II IDE managed make environment translates your settings and actions in the GUI into variables and rules in makefiles. The Nios II IDE manages the contents of the makefiles associated with a project so that you do not need to know anything about makefiles. The Nios II IDE provides a helpful interface for building complex embedded programs, but there is no hidden magic to how the Nios II IDE builds projects; in the end, executable software files are generated by running **make**.

The Nios II IDE manages the complex task of building software for the Nios II soft-core processor architecture, which can change from project to project. You might find it useful to understand how the Nios II IDE manipulates makefiles if you need to separate the process of building your project from the graphical user interface.

This topic assumes that you are familiar with:

- Using the IDE to build projects and browse through project source files
- HAL development concepts described in the Nios II Software Developer's Handbook
- Makefile usage and syntax
- The GCC compiler toolchain

For a full understanding of the inner workings of the build environment, opening and reading the makefiles is as important as the information contained in this topic. Examples of all of the makefiles discussed in this topic are located in the directory of any existing project created with the Nios II IDE, or in the directory where the Nios II Embedded Design Suite (EDS) is installed.

The following sections provide more details about the Nios II IDE managed-make build environment.

▶ **HAL-Based Projects:**

An executable program based on the HAL is constructed using two projects:

- A system library project - The system library project is used to build a library which contains all of the system-specific device drivers, and HAL system routines. This library is automatically configured to match the associated SOPC Builder system hardware.
- An application project - The application project builds the user's application software, and links it with a system library project.

The Nios II IDE manages the build process for both of these classes of project using the makefiles described in this document.

Configuration information is communicated to the build process using the following files:

- **SOPC Builder system file (.ptf)** - The SOPC Builder system describes the hardware configuration, and is generated by SOPC Builder. Every system library is associated to exactly one SOPC Builder system.

- **.stf file** - The **.stf** file describes a project's software configuration, and is generated by the Nios II IDE. The **.stf** file contains the information captured through the **System Library** properties page in the IDE.
- **.cdtbuild file** - The **.cdtbuild** file describes a project's tool chain options, and is generated by the Nios II IDE. The **.cdtbuild** file contains the information captured through the **C/C++ Build** properties page in the IDE.

The **.ptf**, **.stf**, and **.cdtbuild** files are machine-generated files. You should not edit them by hand.

▶ **HAL Source Code:**

HAL source code is provided in a *distributed* form, meaning that the source is distributed through a number of components. There is no single directory that contains all of the source used to build the HAL system library. Which components are used to build the system library is dependent on the system configuration (i.e. the SOPC Builder system and **.stf** files).

All components provided with the Nios II Embedded Design Suite (EDS) are located in the `<Nios II EDS install path>/components` and `<SOPC Builder path>/components` directories.

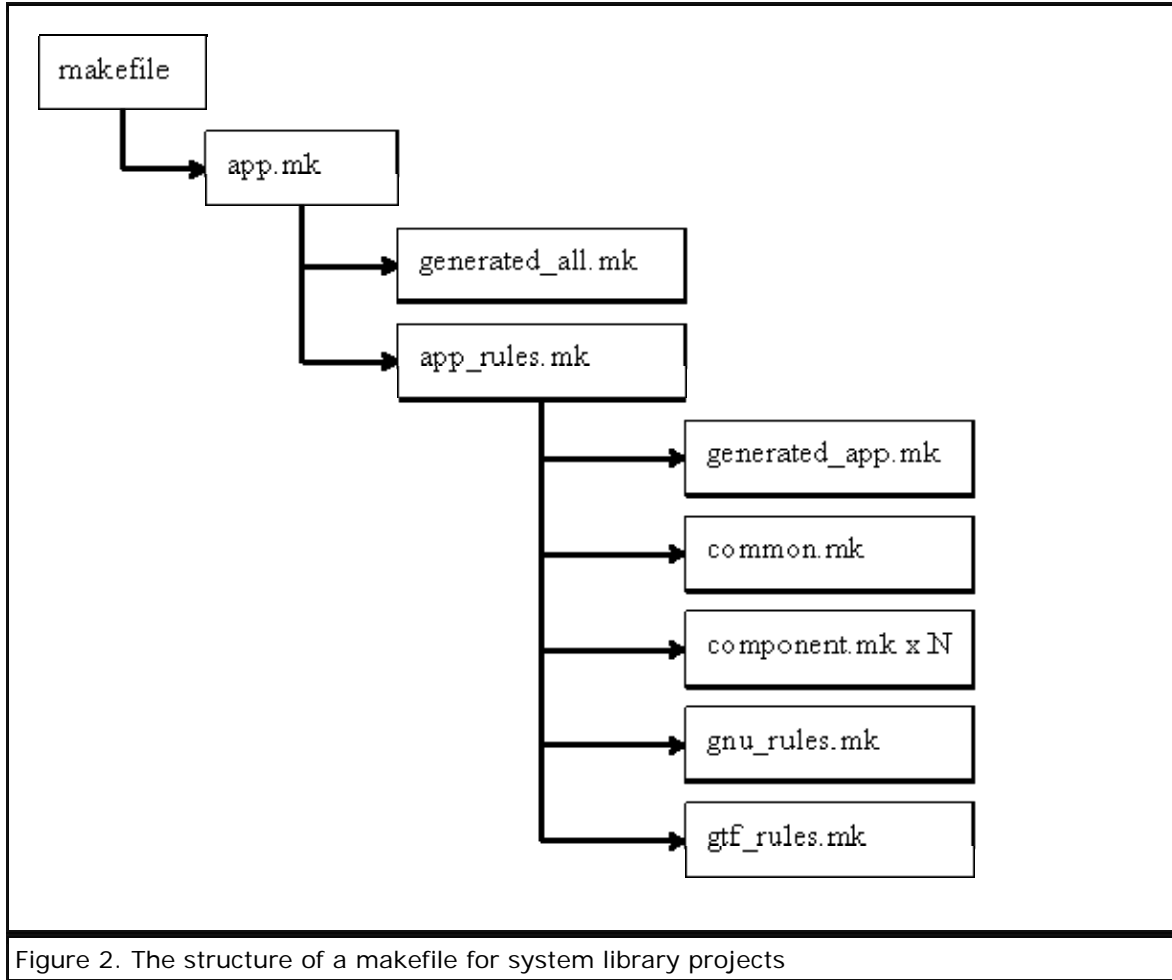
All system libraries are built using the `altera_hal` component, which provides the core HAL functionality, and the processor component, `altera_nios2`, which contains processor-specific definitions for accessing the hardware. Additional components might also be included which provide device drivers, operating system extension, and/or additional system software (e.g. file systems).

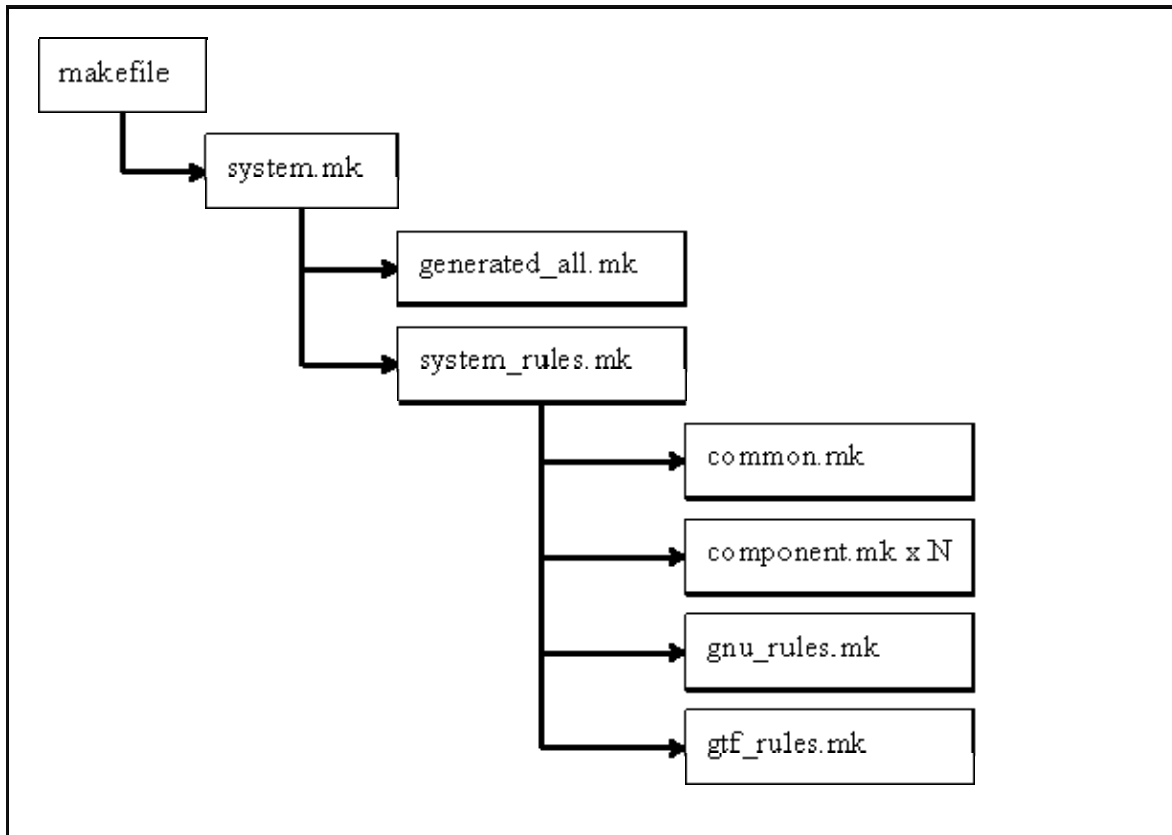
Each component provides a makefile fragment, **component.mk**, which described the source files it provides, and how they should be built into the system. This is described in more detail below.

▶ **Makefile Overview:**

The HAL build system locates the source code supplied by the components using the SOPC Builder system, and information supplied by the user through the Nios II IDE. This is achieved using a number of individual makefiles. At the top level of the build process are the makefiles generated by the Nios II IDE. These include makefile fragments supplied with the `altera_hal` component. These fragments define the source code to build, and the rules which are to be applied. Figure 1 and Figure 2 below illustrate which files are included into the top level makefile for application projects and system library projects.

Figure 1. The structure of a makefile for application projects





The following section describes the function of each of these included files. For detailed implementation details, refer to the individual file concerned.

makefile

Both application and system projects contain at their top level a **makefile** generated by the Nios II IDE. This **makefile** is stored within the project.

This file contains the information supplied through the IDE in the form of **make** variables. For example, compiler flags defined by the user are added to the `CFLAGS` variable, e.g.
`CFLAGS = -O0 -g --Wall`

See the comments in the generated **makefile** for a complete list of the available variables.

The intention is that these variables should be generically applicable regardless of which operating system is being used. The rules which combine and use these variables are supplied in two files: **system.mk** and **app.mk**.

The last line of a Nios II IDE generated **makefile** includes one of these two files. An application project includes **app.mk**, and a system library project includes **system.mk**. These are described below.

app.mk

This file is included into the top level **makefile** for application projects, and is supplied in the build directory of the `altera_hal` component; **app.mk** is responsible for defining the rules used to build the project. In practice this file defers the rule definitions to the included file: **app_rules.mk**. **app.mk** only configures variables required by **app_rules.mk**.

The key feature of this file is that it includes the auto-generated file, **generated_all.mk**, to obtain a list of the components built into the library. **generated_all.mk** defines four

lists of components being used, which are combined here to form the `COMPONENTS` variable.

`COMPONENTS` is then used to construct: the include search path, and a list of the makefile fragments supplied by the various components. These are then used by **app_rules.mk**.

system.mk

This file is included into the top level **makefile** for system library projects, and is supplied in the build directory of the `altera_hal` component. It is responsible for defining the rules used to build the system library project.

In practice this file defers the rule definitions to the included file, **system_rules.mk**; **system.mk** restricts itself to simply configuring variables required by **system_rules.mk**.

The key feature of this file is that it includes the auto-generated file, **generated_all.mk**, to obtain a list of the components to build into the library. **generated_all.mk** defines four lists of components being used, which are combined here to form the `COMPONENTS` variable.

`COMPONENTS` is then used to construct the source search path, the include search path, and a list of the makefile fragments supplied by the various components. These are then used by **system_rules.mk**.

generated_all.mk

The file **generated_all.mk** is generated from the contents of the SOPC Builder system and `.stf` files, and is stored within the system library project. The rule used to generate **generated_all.mk** is defined in **gtf_rules.mk**.

This file defines a list of **make** variables which are used by both application and system projects. In particular this file defines the following variables:

- `COMPONENTS_PROCESSOR`
- `COMPONENTS_OS`
- `COMPONENTS_SOFTWARE`
- `COMPONENTS_DEVICE_DRIVERS`

These four variables provide a white space separated list of all of the components which are to be used to build this system.

These lists are used by both **app.mk** and **system.mk** to construct search paths for include and source files, and also to locate all of the **component.mk** files that are to be included into the **makefile**. See below for a description of the **component.mk** files.

app_rules.mk

This file provides the common rules which are shared between HAL application projects and HAL-based operating system (e.g. MicroC/OS-II) application projects. It defines the **all** and **clean** rules, and then includes the files **gnu_rules.mk** and **gtf_rules.mk**.

app_rules.mk includes the auto-generated file **generated_app.mk** which defines all of the "post `.elf`" build rules, such as the rules used to build flash programming files. The content of **generated_app.mk** is dependent upon the system configuration and is generated based on the settings in the SOPC Builder system and `.stf` files.

app_rules.mk is also responsible for ensuring that the build for the associated system library project is up to date before proceeding with the build for the application project.

This file is located in the build directory of the `altera_hal` component.

system_rules.mk

This file provides the common rules which are shared between HAL application projects and HAL-based operating system (e.g. MicroC/OS-II) system projects. It defines the **all** and **clean** rules, and includes the files **gnu_rules.mk** and **gtf_rules.mk**.

This file is located in the build directory of the `altera_hal` component.

generated_app.mk

The file **generated_app.mk** is generated from the contents of the SOPC Builder system and `.stf` files, and is stored in the system library project. The rule used to generate **generated_app.mk** is defined in **gtf_rules.mk**.

This file defines all of the rules that are to be run as a part of the build process after the `.elf` file has been created, for example the rules to build flash programming files.

common.mk

This file defines some useful `define` statements common to both system library and application projects. It redefines the `SOPC_KIT_NIOS2` environment variable so that it is defined with a UNIX rather than DOS path as required by the GNU tool chain. It also supplies the location of the `.stf` file to use.

This file is located in the build directory of the `altera_hal` component.

component.mk

Each component supplies a makefile fragment named **component.mk** within the component's **HAL/src** directory. This fragment defines a list of the source files that are to be built into the system library. In addition it optionally defines any additional rules and/or make variables that are required for the component to be integrated into the build process. See the description of **component.mk** in the Nios II Software Developer's Handbook.

gnu_rules.mk

This file defines the rules for the GNU tools that are used for compiling, archiving, linking, and generating the **objdump** file. It is located in the build directory of the `altera_hal` component.

gtf_rules.mk

This file defines the rules for generating files which are dependent on the `.stf` and SOPC Builder system, i.e. **generated_app.mk**, **generated_all.mk**, **generated.sh**, **generated.x**, **alt_sys_init.c** and **system.h**. It is located in the build directory of the `altera_hal` component.




Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - The chapters related to the hardware abstraction layer (HAL) system library contain details on system libraries and the structure of Nios II software projects.



Creating a New Project

To create a new project in the Nios II IDE:

1. On the File menu, point to **New**, and then click  **Project...** The **New Project** wizard appears and presents a list of all project types available.
2. Expand **Altera Nios II** in the **Wizards** list. A list of the four Nios II project types appears.
3. Click the desired project type.
4. Click **Next**.

Each of the four types of projects has a different set of wizard pages. The flow of the wizard differs, depending on which project type you select.

▶ [To create a C/C++ application project:](#)

Follow these steps using the **C/C++ Application** pages of the **New Project** wizard.

1. Type a meaningful valid project name in the **Name** box. If you do not supply a name, the Nios II IDE creates one for you when you select a project template in step 5.
2. Specify a location for the Nios II C/C++ application project by doing one of the following.
 - Turn on **Use Default Location**. The default location is shown in the **Location** box. The name of the project directory is the project name. The Nios II IDE creates the new project directory in the default location specified in the **New Projects** preference page.



Note: After installing the Nios II IDE, the default location is the **<SOPC Builder system path>/software** directory. Altera recommends using this location, because it keeps software files in proximity to the system hardware files. This can help when sharing project files with other people.

- Turn off **Use Default Location** and specify an alternative project location in the **Location** box. You can click the **Browse...** button next to the **Location** box to browse the file system.
3. Specify the SOPC Builder system file (**.ptf**) that describes your target hardware in **SOPC Builder System**. You can click the **Browse...** button next to the **SOPC Builder System** list to browse the file system for a **.ptf** file. Previously used systems appear in the drop-down list.
 4. Select a specific CPU for the project in the **CPU** list. If your SOPC Builder system has only one CPU, the Nios II IDE sets **CPU** automatically.
 5. Select a project template in the **Select Project Template** list. As you click the project templates, information about each template appears in the **Description** and **Details** boxes. Each template is a collection of software files and project settings that serve as a base for the new project. You can add your own source code to the project later.



Note: Use the **Blank Project** template to avoid copying any files into the new

project.

6. If you want the Nios II IDE to create a default associated system library for you, skip to step 9.
7. Click **Next**. The next page of the wizard appears, showing options for an associated system library project.
8. Specify how you want to associate the C/C++ application project to a system library by doing one of the following.
 - Click **Create a new system library named: <application project name>_syslib**. This option creates a default HAL system library project to accompany your C/C++ application project. This option is usually desirable for single-threaded Nios II programs.
 - Click **Select or create a system library**, then select an existing system library in the **Available System Library Projects** list. Alternatively, you can click **New System Library Project...** to create a new system library project.



Note: Multiple C/C++ application projects can depend on the same system library. This can be desirable, for example, if the system library includes a large operating system that takes effort to configure once, and you rarely configure it again.

9. Click **Finish**. The Nios II IDE creates new directories for your C/C++ application project and system library project. The IDE copies the template source files into the project directories, and adds your projects to the list of available projects in the C/C++ Projects view.

▶ To create a system library project:

Follow these steps using the **System Library** page of the **New Project** wizard.


1. Type a meaningful valid project name in the **Name** box.
2. Specify a location for the Nios II system library project by doing one of the following.
 - Turn on **Use Default Location**. The default location is shown in the **Location** box. The name of the project directory is the project name. The Nios II IDE creates the new project directory in the default location specified in the **New Projects** preference page.



Note: After installing the Nios II IDE, the default location is the **<SOPC Builder system path>/software** directory. Altera recommends using this location because it keeps software files in proximity to the system hardware files. This can help when sharing project files with other people.

- Turn off **Use Default Location** and specify an alternative project location in the **Location** box. You can click the **Browse...** button immediately adjacent to the **Location** box to browse the file system.
3. Specify the SOPC Builder system file (**.ptf**) that describes your target hardware in **SOPC Builder System**. The SOPC Builder system file (**.ptf**) defines the CPUs and peripherals included in the SOPC Builder system. You can click the **Browse...** button next to the **SOPC Builder System** list to browse the file system for a **.ptf** file. Previously used systems appear in the drop-down list.


4. Select a specific CPU for the project in the **CPU** list. If your SOPC Builder system has only one CPU, the Nios II IDE sets **CPU** automatically.
5. Select an RTOS in the **Type of RTOS** list.
6. Click **Finish**. The Nios II IDE creates a new directory for your system library project, and adds your project to the list of available projects in the C/C++ Projects view.

 **Note:** After creating a new system library project, you generally need to use the **System Library** page of the **Properties** dialog box to configure the system library to interact with the target hardware appropriately.

▶ **To create a managed library project:**

Follow these steps using the **Managed Library** page of the New Project wizard.

1. Type a meaningful valid project name in the **Name** box.
2. Specify a location for the Nios II system library project by doing one of the following.
 - Turn on **Use Default Location**. The default location is shown in the **Location** box. The name of the project directory is the project name. The Nios II IDE creates the new project directory in the Nios II IDE workspace folder, unless you have previously set another custom default location on the **New Projects** preference page.
 - Turn off **Use Default Location** and specify an alternative project location in the **Location** box. You can click the **Browse...** button immediately adjacent to the **Location** box to browse the file system.
3. Click **Finish**. The Nios II IDE creates a new directory for your managed library project, and adds your project to the list of available projects in the C/C++ Projects view.

 **Note:** After creating a new managed library project, you need to use the **Project References** page of the **Properties** dialog box for a C/C++ application project to associate the library with the application.

▶ **To create an advanced C/C++ project:**

An advanced C/C++ project is the same as a standard make project in the Eclipse C/C++ Development Toolkit (CDT). Refer to Related Topics for more information about standard make projects.

Once your new project exists, you can edit the code, create new files within the Nios II IDE, or import files into your project from outside the Nios II IDE.

 **Related Nios II IDE Help Topics**

- About Nios II IDE Projects
- About the Nios II IDE Managed-Make Build Environment
- Configuring Project Dependencies - Contains details on referencing libraries.

- New Project Wizard
- New C/C++ Application (New Project Wizard)
- New System Library (New Project Wizard)
- New Advanced C/C++ Project (New Project Wizard)
- New Managed Library (New Project Wizard)
- New Projects Page (Preferences Dialog)

**Related Eclipse and CDT Help Topics**

- C/C++ Development User Guide > Getting Started > CDT Standard Make Tutorial

**Related Topics on the Web**

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - The chapters related to the hardware abstraction layer (HAL) system library contain details on system libraries and the structure of Nios II software projects.



Importing, Exporting and Sharing Projects and Files

The Nios II IDE allows you to import and export existing projects, folders, and files. Importing and exporting projects allows you to archive projects and share projects with other designers. Importing files allows you to associate files on your hard drive with a Nios II IDE project.

▶ To import files and folders:

Use an external file management tool (such as Windows Explorer) to "drag and drop" files and folders onto the project folder in the C/C++ Projects view of the Nios II IDE. The Nios II IDE automatically recognizes files and folders in the project folder and associates them with the project. After copying, you might have to right-click the project in the C/C++ Projects view, and then click **Refresh** for the Nios II IDE to recognize the files.

Alternatively, you can copy files and folders into the project directory with an external file management tool, and then click **Refresh** in the C/C++ Projects view to have the Nios II IDE to recognize the files.

▶ To import projects:

1. Copy the project directory where you want it to exist on the host file system.
2. In the Nios II IDE on the File menu, click **Import....** The **Import** wizard appears.
3. Select **Existing Altera Nios II Project into Workspace** in the **Select an import source** list.
4. Click **Next**. The **Import Project From File System** page appears.
5. Click **Browse....**
6. Navigate to the directory that contains the **.project** file for the project you wish to import. Nios II project directories contain a **.stf** file and a **.project** file.
7. Click **OK**.
8. Click **Finish**. The Nios II IDE adds the project to the list of available projects in the C/C++ Projects view.

Importing projects from an earlier version of the Nios II IDE might prompt you to convert the project files to the new version. Converted projects will no longer load in previous versions of Nios II IDE.

Importing previously built projects might prompt you to delete the build contents of the project. Click **Yes** to prepare for a clean build when you next build the project.




Note: Avoid spaces in project names. Build problems might occur later if the project name contains spaces.

▶ To export projects:

Use an external file management tool (such as Windows Explorer) to copy the project directory elsewhere. Files in the project directory and its subdirectories contain related information about a project. You export a project by copying all of these files to another

location. If you are concerned about size, you do not need to copy the Release or Debug directories, because the build process will recreate them.

To see where a project resides, right-click on the project in the C/C++ Projects view and then click **Properties**. The **Info** page of the **Properties** dialog box appears, displaying the project's location.


 **Note:** Common files that the project references, such as the software components directories, are not exported as part of the project. If you want to archive them, you have to copy them manually.

▶ To share software projects:

You can share Nios II IDE projects with other Nios II IDE users by either copying files, or using the CVS source control system. Refer to Related Topics for more information about the CVS source control system.

To share software projects by copying files, use the import and export steps described in this topic. Passing project files to another Nios II IDE user requires that you:

- Have the same version of the Nios II Embedded Design Suite (EDS) installed as the other IDE user.
- Use the default locations for the C/C++ application project and system library project when you create the projects. The default location keeps the software files in a fixed location relative to the SOPC Builder system file (**.ptf**), so that the Nios II IDE can find everything it needs to build the project.
- Include both the C/C++ application project and the system library project directories.
- Include the FPGA configuration file (**.sof**) and SOPC Builder system file (**.ptf**), which describe the target hardware.
- Optionally include any run/debug configurations associated with the project.

 **Note:** Run/debug configurations by default reside separate from the project files to keep project directories clutter free. To store a run/debug configuration with its project, making it easier to track when sharing projects, select **Shared** on the **Common** tab of the desired run/debug configuration and specify the project's path in the **Location** box.

- **Do not** include the build directory, typically named "Debug" or "Release". The contents of these directories will be re-generated on the new machine.



Related Nios II IDE Help Topics

- About Nios II IDE Projects



Related Eclipse and CDT Help Topics

- Workbench User Guide > Reference > Import Wizard
- Workbench User Guide > Concepts > Team Programming with CVS

Configuring Projects



About Project Properties

Project properties control how the Nios II IDE builds a project and how programs interact with the system hardware at runtime. The Nios II IDE offers multiple project types, and project properties depend on the project type.

For executable application projects, properties such as preprocessor and compiler settings determine how the project builds. For system library projects, properties affect how the application program interacts with the target. Because all C/C++ application projects depend on a system library project, system library project properties significantly impact the compiled results of a C/C++ application project. For this reason, the system library properties are typically the most important settings for a Nios II IDE project.

You configure RTOS and middleware software component settings as part of the system library properties.



Related Nios II IDE Help Topics

- [Configuring Project Properties](#)
- [Choosing and Configuring an Operating System](#)
- [Choosing and Configuring Middleware Software Components](#)
- [Configuring Project Dependencies](#)
- [Properties Dialog Box](#)



Configuring Project Properties

Project properties control how the Nios II IDE builds the project and how the program interacts with the system hardware at runtime.

▶ To configure a project's properties:

1. Right-click on the project in the C/C++ Projects view, and click **Properties**. The **Properties** dialog box appears.
2. Click the page titles in the left-hand pane to view and edit project properties, as described in the **Properties** dialog box topics.



Note: For C/C++ application projects, the most important project settings generally relate to the application's interaction with the hardware. You specify these settings in the System Library page of the **Properties** dialog box for the system library project associated with your application.

3. Click **OK** to close the **Properties** dialog box.



Related Nios II IDE Help Topics

- About Project Properties
- Properties Dialog Box



Choosing and Configuring an Operating System

By default, the Nios II IDE uses the hardware abstraction layer (HAL) single-threaded runtime environment. You can optionally include a real-time operating system (RTOS) as part of your system library project. Altera provides the MicroC/OS-II RTOS, and other vendors can provide their own OS as a plug-in.

You add and configure OS options using the **System Library** page of the **Properties** dialog box.

▶ To enable MicroC/OS-II for your project:

1. Right-click the C/C++ application project in the C/C++ Projects view.
2. Click **System Library Properties**. The **Properties** dialog box appears.
3. Click **System Library** in the left-hand pane of the **Properties** dialog box.
4. Select **MicroC/OS-II** in the **RTOS** drop-down list.
5. Click **Apply** to save the new **RTOS** setting. A warning appears to confirm that you really want to reset the RTOS options to their defaults.
6. Click **Yes**.
7. Click **RTOS Options...** The **MicroC/OS-II RTOS Options** dialog box appears. The Nios II Software Developer's Handbook briefly describes the RTOS options. For full details on configuring the MicroC/OS-II kernel, see *Chapter 17: MicroC/OS-II Configuration Manual* of the book *MicroC/OS-II: The Real-Time Kernel, Second Edition* by Jean Labrosse (CMP Books).
8. Click **OK** to save the settings and close the **MicroC/OS-II RTOS Options** dialog box.
9. Click **OK** to close the **Properties** dialog box.

The system library will include the MicroC/OS-II kernel the next time you build the project. Your program code can use the MicroC/OS-II API.



Related Nios II IDE Help Topics

- MicroC/OS-II RTOS
- System Library Page (Properties Dialog Box)



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on writing Nios II programs based on MicroC/OS-II.
- Using MicroC/OS-II RTOS with the Nios II Processor Tutorial at www.altera.com/literature/tt/tt_nios2_MicroC_OSII_tutorial.pdf - Contains step-by-step instructions on creating MicroC/OS-II applications.



Choosing and Configuring Middleware Software Components

Software components are middleware software modules that the Nios II IDE recognizes and can automatically link into a software project. You add and configure software components using the **System Library** page of the **Properties** dialog box.

The following software components are available from Altera:

- Host Based File System
- Lightweight TCP/IP Stack
- Zip Read-Only File System

Other vendors can provide their own software components as plug-ins.

▶ [To enable a software component:](#)

1. Right-click the system library project in the C/C++ Projects view, and click **Properties**. The **Properties** dialog box for the system library project appears.
2. Click **System Library** in the left-hand pane.
3. Click **Software Components....** The **Software Components** dialog box appears.
4. Click the software component you wish to enable in the left-hand pane.
5. Turn on **Add this software component**.
6. Specify the software component settings.
7. Click **OK** to save the settings and close the **Software Components** dialog box.

The system library includes the software component the next time you build the project.



Related Nios II IDE Help Topics

- Host-Based File System
- Lightweight TCP/IP Stack
- Zip Read-Only File System
- System Library Page (Properties Dialog Box)
- Software Components Dialog Box (System Library Properties Page)



Configuring Project Dependencies

Projects in the Nios II IDE can be dependent on other Nios II IDE projects. C/C++ application projects inherently depend on a single system library project. C/C++ application projects and advanced C/C++ projects can also depend on managed library projects. Managed library projects can depend on other managed library projects. During the build process for a project, the Nios II IDE first builds all dependent projects.

The Nios II IDE automatically establishes a dependency between the C/C++ application project and the system library project at the time you create the C/C++ application project. You do not and should not create project dependencies between C/C++ application projects and system library projects manually. The **Associated System Library** page of the **Properties** dialog box for your C/C++ application project contains details about this special dependency. Once the dependency exists, you can change it to reference a different system library project.

Managed library projects are the most common project dependencies. Making an application project dependent on a managed library project allows you to do the following:




- Ensure your managed library builds and is up to date when you compile your application project.
- Call functions in the managed library from your application project.
- Include header files located in a managed library.
- Debug and step into source code located in the managed library.

▶ To change dependence on a system library project:

1. Right-click your C/C++ application project in the C/C++ Projects view, and then click **Properties**. The **Properties** dialog box appears.
2. Click **Associated System Library** in the left-hand pane.
3. Click **Browse...**. A list of system library projects appears.
4. Select a system library project.
5. Click **OK** to close the list of system library projects.
6. Click **OK** to close the **Properties** dialog box.

▶ To configure dependence on a managed library project:

1. Right-click your project in the C/C++ Projects view, and then click **Properties**. The **Properties** dialog box appears.
2. Reference the managed library project from your C/C++ application project. This step ensures that your managed library project builds before your C/C++ application project. It also ensures you can easily step into the managed library code when debugging your application project.
 1. Click **Project References** in the left-hand pane.
 2. Turn on the managed library project you want to reference.
3. Set up the linker options so your C/C++ application project can locate your library. The C/C++ library of a managed library project is not automatically detected when you reference the project; you have to explicitly add it.

1. Click **C/C++ Build** in the left-hand pane.
 2. Click the **Tool Settings** tab.
 3. Expand **Linker**. The **General** option appears.
 4. Click **General**. The **Libraries** and **Library Paths** panes appear.
 5. In the **Libraries** pane, click  **Add Library** and type the name of the managed library project. Do not press the **Browse...** button.
 6. In the **Library Paths** pane, click  **Add Library Path** and add the path to the *.a library file. The library path should include the **Debug** or **Release** folder. Note that you will have to update the library path if you change the configuration (i.e. Release or Debug) of the managed library project.
4. Set up the include paths to locate headers from your managed library project. If your managed library project includes header files you would like to access from your C/C++ application project, you need to specifically enter the paths to the header files.
 1. On the **Tool Settings** tab, expand **Nios II Compiler**. The **General** option appears.
 2. Click **General**. The **Include Paths** pane appears.
 3. In the **Include Paths** pane, click  **Add Include Path** and add the full or relative path to the folder containing the header files.
 5. Click **OK** to close the **Properties** dialog box.



Related Nios II IDE Help Topics

- Properties Dialog Box
- Associated System Library Page (Properties Dialog Box)



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - The chapters related to the hardware abstraction layer (HAL) system library contain details on system libraries and the structure of Nios II software projects.

Editing Code



About Editing Code

The C/C++ editor, part of the C/C++ Development Toolkit (CDT), provides specialized features for editing C/C++ files. The editor includes the following features:

- Syntax highlighting
- Code completion
- Code templates

Normally, you use the C/C++ editor in the C/C++ perspective. It is also available in the Debug perspective. You can invoke the C/C++ editor by opening a file from the C/C++ Projects view.



Related Nios II IDE Help Topics

- C/C++ Projects View (C/C++ Perspective)



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Concepts > Editing C/C++ files > C/C++ Editor
- C/C++ Development User Guide > Concepts > Code aids > Content Assist - Describes code completion.
- C/C++ Development User Guide > Concepts > Code aids > Templates - Describes code templates.
- C/C++ Development User Guide > Concepts > Views in the C/C++ perspective - Describes the C/C++ perspective.
- C/C++ Development User Guide > Concepts > Debug > Debug information - Describes the Debug perspective.
- C/C++ Development User Guide > Tasks > Write code - Describes how to customize and use the C/C++ editor.
- Workbench User Guide > Tasks > Navigating and finding resources - Describes how to search for text spanning multiple files.

Building Projects



About Building Projects

Building a project in the Nios II IDE compiles and links all the source code associated with the project. Associated code can include other projects that your project depends on, such as a system library project associated with a C/C++ application project. When building a project, Nios II IDE first builds all dependent projects that the current project references.

The build process runs a makefile in the background that produces the project output. Project output depends on the project type:

- C/C++ application projects create executable files (**.elf**) that you can run or debug.
- System library projects create pre-compiled libraries (**.a**).
- Managed library projects create pre-compiled libraries (**.a**).
- Advanced C/C++ projects output depends on your makefile.

When building a C/C++ application project, the Nios II IDE references the SOPC Builder system file (**.ptf**) to create executable code that matches the target hardware. After creating a project based on a specific SOPC Builder system, if the **.ptf** file changes (as a result of modifications to the hardware design in SOPC Builder), the system library project must be rebuilt. The build process will detect an out-of-date **.ptf** file, and automatically rebuild the system library project.

If you want to re-target your project to a different SOPC Builder system, you need to create a new system library project. Edits to your program source code might be necessary if, for example, the peripherals are named differently in the new SOPC Builder system.



Related Nios II IDE Help Topics

- Building a Project
- Configuring Project Dependencies
- Nios II Page (Preferences Dialog Box) - Contains settings that affect how the Nios II IDE builds and runs projects.



Building a Project

Building a project in the Nios II IDE compiles and links all the source code and libraries associated with the project. Associated code and libraries might exist in other projects that your project depends on. You can specify a single project to build or build all projects in your workspace.


▶ **To build a project and update its dependencies:**

1. Right-click on the project in the C/C++ Projects view, and click **Build Project**. The Nios II IDE first builds any out-of-date projects your project depends on, and then builds your project.

▶ **To build all projects in your workspace:**

1. Click **Build All** on the Project menu. The Nios II IDE builds all projects in your workspace.

During the build process, the Console view displays messages generated by the GCC tool-chain during compilation. The Problems view displays any warnings or errors. If building the project generates warnings or errors, the Problems view appears automatically.

 **Note:** The information displayed in the Console and Problems views depends on the project selected in the C/C++ Projects view.



Related Nios II IDE Help Topics

- About Building Projects
- Configuring Project Dependencies

Running and Debugging Projects



About Running and Debugging Projects

Running or debugging a project in the Nios II IDE executes the project code so you can analyze the output. Running a project consists of loading the executable code into the target environment, and then turning control over to the target environment for code execution. Debugging consists of loading the executable code into the target environment, and then interactively communicating with the target environment to pause code execution for step-by-step analysis using the debugger integrated in Nios II IDE. Refer to the Debugging the Project tutorial topic to learn about the Nios II IDE integrated debugger and the many views available in the Debug perspective.

In the Nios II IDE, run and debug sessions are controlled by run/debug configurations. Each configuration is a group of settings that specifies which project to run or debug, and defines the target environment. Some additional global run and debug preferences are also available.

The actions required to run or debug a project depend on the type of run/debug configuration. Configurations are categorized by target type. The following sections describe running and debugging on the various target types.

▶ Nios II Hardware System:

Running and debugging on a Nios II hardware target allows you to execute code on a Nios II processor system in hardware. Executing code on a hardware target allows you to analyze the real-time behavior of your project. When debugging on a hardware target, you can view execution trace and view disassembly of the program's execution.

Running and debugging on a Nios II hardware target involves the following steps:

1. Creating a run/debug configuration
2. Downloading the FPGA configuration file (**.sof**) to the target board to configure the FPGA with the desired SOPC Builder system
3. Downloading the executable software file (**.elf**) to memory on the target board
4. Executing the code

▶ Nios II Instruction Set Simulator:

Running and debugging on a Nios II instruction set simulator (ISS) target allows you to execute code in simulation on a host PC. Executing code on the ISS allows you to analyze the behavior of your project without the need for actual hardware. When debugging on the ISS, you can view execution trace and view disassembly of the program's execution in the Console view.

Running and debugging on the ISS involves the following steps:

1. Creating a run/debug configuration
2. Passing the executable software file (**.elf**) to the ISS
3. Executing the code

▶ ModelSim Simulation of a Nios II System:

Running on a Nios II ModelSim target allows you to execute code on a hardware simulation of a Nios II processor system. Hardware simulation allows you to simulate

cycle-accurate behavior of a Nios II processor system. ModelSim is only available as a run target. You cannot interactively debug executable code using the ModelSim simulator.

Running on a Nios II ModelSim target involves the following steps:

1. Creating a run/debug configuration
2. Running the ModelSim simulator with a memory model of the executable software file (**.elf**)
3. Viewing the simulation results in ModelSim

▶ **Nios II Multiprocessor Collection:**

Running and debugging on a Nios II multiprocessor collection target allows you to execute code simultaneously on multiple Nios II processors in hardware. Executing code on a hardware target allows you to analyze the real-time behavior of your project. When debugging on a hardware target, you can view execution trace and view disassembly of the program's execution.

Running and debugging on a Nios II multiprocessor collection target involves the following steps:

1. Creating a run/debug configuration for each project you wish to include in the collection
2. Creating a run/debug configuration for the multiprocessor collection
3. Downloading the FPGA configuration file (**.sof**) to the target board to configure the FPGA with the desired SOPC Builder system
4. Downloading the executable software files (**.elf**) for all the included projects to memory on the target board
5. Executing the code



Related Nios II IDE Help Topics


- Running and Debugging on Hardware
- Running and Debugging on the ISS
- Running on the ModelSim Simulator
- Running and Debugging Multiprocessor Collections
- Run/Debug Configuration
- Run/Debug Dialog Box
- Debugging the Project
- Viewing Disassembly
- Viewing Execution Trace



Configuring the FPGA

When targeting Nios II hardware, you must configure the FPGA on the target board with your project's associated SOPC Builder system. The factory-programmed SOPC Builder system (i.e. the FPGA hardware design) on targets like the Nios development boards is different than your project's SOPC Builder system. Therefore, your executable code will not run unless you configure the FPGA with the expected SOPC Builder system.

You only need to configure the FPGA when you reset the board or if the SOPC Builder system file changes. Normally, you configure the FPGA once after you apply power to the board, and the configuration persists through the duration of the Nios II IDE session.

 **Note:** If you are targeting a board other than the Nios development board, you need to configure the Quartus II programmer differently, as specified by the board designer. In the Quartus II software, you can save the configuration settings to a chain descriptor file (.cdf), eliminating the need to configure the Quartus II programmer every time you configure the device.

To configure the FPGA, use the Quartus II Programmer. The Quartus II Programmer is Altera's tool for configuring FPGAs via a JTAG download cable, such as the USB Blaster.

▶ [To launch the Quartus II Programmer:](#)

- On Windows:
 1. Switch to the C/C++ Development perspective, if necessary.
 2. Click your project in the C/C++ Projects view.
 3. On the Tools menu, click **Quartus II Programmer...**
- On Linux:
 1. Launch the Quartus II software.
 2. On the Tools menu, click **Programmer**.

▶ [To configure the FPGA from the Quartus II Programmer:](#)

1. Click **Add File...** in the Quartus II Programmer. The **Select Programming File** dialog box appears.
2. Browse to the location of the hardware design that corresponds to your Nios development board. For example, the directory used in the software development tutorial is `<Nios II EDS install path>/examples/verilog/<Nios development board>/full_featured`. On Windows, you are likely already in the proper directory.
3. Select the FPGA configuration file (.sof) for your project and click **Open**. The **Select Programming File** dialog box closes.
4. Turn on the **Program/Configure** checkbox for the target device.
5. Click **Start**. When the progress meter progresses to 100%, configuration is complete.



Note: If **Start** is not enabled, click **Hardware Setup** to configure your Altera download cable.

6. Close or minimize the Quartus II Programmer and return to the Nios II IDE.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Running and Debugging on Hardware




Related Topics on the Web

- Nios II Processor Reference Handbook at www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf - Contains details on the JTAG debug module on the Nios II processor.





Running and Debugging on Hardware

Running and debugging a Nios II project on a Nios II hardware target involves creating a run/debug configuration, and then starting the run or debug session. The Nios II IDE can automatically create a default configuration for you, or you can manually create a configuration.

 **Note:** Running or debugging on a hardware requires that you first download an FPGA configuration file (.sof) to the target board to configure the FPGA.

▶ To automatically create a run/debug configuration and start running or debugging:

1. Right-click your project in the C/C++ Projects view.
2. Point to **Run As** or **Debug As**, and then click  **Nios II Hardware**. The Nios II IDE automatically performs the following tasks:
 - Creates a default run/debug configuration for the target board, placing it in the **Configurations** list under **Nios II Hardware** in the **Run/Debug** dialog box.
 - Selects a default JTAG download cable.

 **Note:** This step usually completes automatically without user intervention. If it cannot (the most common cause is that you have multiple JTAG download cables installed), the IDE displays an error message, and you must manually set up a run configuration.

- Initiates a run or debug session.

▶ To manually create a run/debug configuration and start running or debugging:



1. Right-click your project in the C/C++ Projects view.
2. Click **Run...** or **Debug...** on the Run menu. The **Run/Debug** dialog box appears.
3. Select the **Nios II Hardware** target type in the **Configurations** list.
4. Click **New**. The Nios II IDE performs the following tasks:
 - Creates a new hardware target configuration, placing it in the **Configurations** list under **Nios II Hardware**.
 - Selects a default JTAG download cable on the **Target Connection** tab.
 - Turns on **break at main** on the **Debugger** tab.
5. Click **Run** or **Debug** in the lower-right corner of the **Run/Debug** dialog box to initiate the run or debug session.

The Nios II IDE performs the following operations when starting a run or debug session:

- Builds the project. If the project is not up-to-date, then the IDE builds it first to generate an up-to-date executable file.

- Establishes communication with the target board, and verifies that the expected SOPC Builder system is configured in the FPGA. If the FPGA is not configured properly, you should repeat the steps to configure the hardware.
- Downloads the executable file (**.elf**) to memory on the target board.
- Sets a breakpoint at `main()`. (*debugging only*)
- Instructs the Nios II CPU to begin executing the code.
- Suspends code execution at `main()`. (*debugging only*)



Note: After using **Run As**, **Run**, **Debug As**, or **Debug** once, click  **Run** or  **Debug** on the toolbar to run or debug again.

Program output can appear in the Console view of the Nios II IDE. The Console view maintains a terminal I/O connection with a communication device, such as a JTAG UART, connected to the Nios II processor in the SOPC Builder system. When the Nios II program writes to `stdout` or `stderr`, the Console view displays the text. The Console view can also accept character input from the host keyboard, which is sent to the CPU and read as `stdin`.



Related Nios II IDE Help Topics


- About Running and Debugging Projects
- Configuring the FPGA
- Run/Debug Configuration
- Run/Debug Dialog Box
- Viewing Disassembly
- Viewing Execution Trace



Running and Debugging on the ISS

Using the Nios II IDE, the process to run or debug on the Nios II instruction set simulator (ISS) is nearly identical to running and debugging on a target hardware. You typically invoke the ISS as a run or debug target from the Nios II IDE. Alternatively, you can invoke **nios2-iss.exe** from a Nios II command shell, although Altera recommends command-line usage only to advanced users.

▶ To automatically create a run/debug configuration and start running or debugging:

1. Right-click your project in the C/C++ Projects view.
2. Point to **Run As** or **Debug As**, and then click  **Nios II Instruction Set Simulator**. The Nios II IDE automatically performs the following tasks:
 - Creates a default run/debug configuration for the ISS target, placing it in the **Configurations** list under **Nios II Instruction Set Simulator** in the **Run/Debug** dialog box.
 - Initiates a run or debug session.



▶ To manually create a run/debug configuration and start running or debugging:

1. Right-click your project in the C/C++ Projects view.
2. Click **Run...** or **Debug...** on the Run menu. The **Run/Debug** dialog box appears.
3. Select the **Nios II Instruction Set Simulator** target type in the **Configurations** list.
4. Click **New**. The Nios II IDE performs the following tasks:
 - Creates a new ISS target configuration, placing it in the **Configurations** list under **Nios II Instruction Set Simulator**.
 - Turns on **break at main** on the **Debugger** tab.
5. Click the **ISS Settings** tab. The **ISS Settings** tab appears.
6. Adjust any settings you desire to change.
7. Click **Run** or **Debug** in the lower-right corner of the **Run/Debug** dialog box to initiate the run or debug session.

The Nios II IDE performs the following operations when starting a run or debug session:

- Builds the project. If the project is not up-to-date, then the IDE builds it first to generate an up-to-date executable file.
- Passes the executable software file (**.elf**) to the ISS.
- Sets a breakpoint at `main()`. (*debugging only*)
- Instructs the Nios II CPU to begin executing the code.
- Suspends code execution at `main()`. (*debugging only*)



Note: After using **Run As**, **Run**, **Debug As**, or **Debug** once, click  **Run** or  **Debug** on the toolbar to run or debug again.

When you start a debug session, the Nios II IDE invokes the **nios2-iss.exe** application. The Nios II ISS supports all GDB debug facilities, such as the ability to set breakpoints, set watchpoints, and view memory. Reading uninitialized memory or empty regions in the memory map returns zero. Depending on the run/debug configuration, the ISS can generate a warning or an error whenever a read occurs from uninitialized memory. Fetching an instruction from these locations returns a zero instruction word (equivalent to `call 0x0`), which generally causes your program to fail.

The IDE uses **nios2-elf-gdb** to communicate with **nios2-iss.exe** over an automatically-selected TCP/IP port. Invoking **nios2-elf-gdb** from the command-line also communicates over a TCP/IP port.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Run/Debug Configuration
- Run/Debug Dialog Box
- Viewing Disassembly
- Viewing Execution Trace



Running on the ModelSim Simulator

Running on a Nios II ModelSim target allows you to execute code on a hardware simulation of a Nios II processor system. Hardware simulation allows you to simulate cycle-accurate behavior of a Nios II processor system. ModelSim is only available as a run target. You cannot interactively debug executable code using the ModelSim simulator.

ModelSim requires hardware design files, and therefore the ModelSim simulation process is linked closely with SOPC Builder. You can only run a ModelSim run configuration in the Nios II IDE if the target SOPC Builder system was generated with the ModelSim simulation option enabled in SOPC Builder.

Full details of using the ModelSim simulator are beyond the scope of this help system, and are available online. Refer to Related Topics for more information on ModelSim.



Related Nios II IDE Help Topics

- About Running and Debugging Projects



Related Topics on the Web

- AN351: Simulating Nios II Embedded Processor Designs at www.altera.com/literature/an/an351.pdf - Contains details for ModelSim.



Running and Debugging Multiprocessor Collections

Multiprocessor collections provide an easy way to run or debug a group of Nios II C/C++ application projects on a Nios II hardware target as a unit. Running or debugging a multiprocessor collection allows you to download executable code for each project in the collection and start execution for all processors in a single command.

To run or debug a multiprocessor collection, you must first create a run/debug configuration for each project you wish to include in the collection and download the FPGA configuration file (.sof) to the target board to configure the FPGA with the desired SPOC Builder system.



The following steps allow you to create and work with a multiprocessor collection.

▶ **To instruct the Nios II IDE to allow multiple active run and debug sessions:**

Before you begin using multiprocessor collections, you must turn on the following Nios II IDE setting. You only have to perform this action once.

1. Click **Preferences** on the Window menu. The **Preferences** dialog box appears.
2. Click **Nios II** in the left-hand pane.
3. Turn on **Allow multiple active run/debug sessions**.



Note: This setting has side-effects on the behavior of single-processor run and debug sessions. With this setting on, clicking  **Run** or  **Debug** does not auto-terminate existing sessions, which can cause start-up errors. You must manually terminate existing run and debug sessions before starting a new one.

▶ **To create a new multiprocessor collection configuration and start program execution:**

1. Click **Run...** or **Debug...** on the Run menu.
2. Select **Nios II Multiprocessor Collection** in the **Configurations** list.
3. Click **New** to create a new multiprocessor collection.
4. On the **Main** tab, turn on the Nios II hardware target configurations you want to include in the multiprocessor collection.
5. You can change the name of your new multiprocessor collection in the **Name** box.
6. Click **Run** or **Debug** to start running or debugging all Nios II hardware target configurations in the multiprocessor collection.


Each processor begins executing code as soon as code finishes downloading. In the case of a debug session, each processor breaks at the start of `main()`. The Debug view displays a separate process for the multiprocessor collection in addition to a process for each of the individual Nios II hardware target configurations.

▶ **To manage multiprocessor debug sessions:**


The following commands are available to control all processors simultaneously in the debug session:


- To start all processes simultaneously, click  **Resume**.

- To terminate all processes simultaneously, select the multiprocessor collection process in the Debug view, and click  **Terminate**.

 **Note:** A debug process for a multiprocessor collection remains active as long as any of the associated Nios II hardware target configuration processes are active. If one of the debug processes fails to start correctly or terminates early, the other processes will continue running. You must terminate all processes before attempting to start debugging the multiprocessor collection again.

Other debug commands, such as **Suspend** and **Step**, are not available for multiprocessor collection process. You can control individual processors by selecting the appropriate process in the Debug view and then using the normal debug commands, such as **Resume**, **Suspend**, **Terminate**.

 **Note:** The buttons in the Debug view are context sensitive, depending on the process selected. Be sure you have the correct process selected when using debug commands.

Clicking  **Debug** does not work to restart multiprocessor collections. The button only restarts the most recently launched Nios II hardware target configuration, not the whole multiprocessor collection.



▶ To navigate source code in the debugger:

The Debug view provides a convenient mechanism to jump directly to the line of code where a processor suspends. When you suspend a process, the editor automatically displays the source file in which the processor stopped.

Note that selecting a different process in the Debug view does not automatically change the file displayed in the editor. You can double-click on the top-most function in the stack trace to open the source file in the editor and display the suspended line of code.

▶ To switch the Console view between active target connections:

Depending on the target connection settings for each of the Nios II hardware target configurations, each processor can communicate character I/O with the Console view. An independent terminal connection exists for each target processor, but the Console view displays only one connection at a time. To switch between the active target connections:

1. Click  **Unpin Console** from the Console view's toolbar.
2. Click  **Display Selected Console** and select a **Nios II Terminal Window** item corresponding to the desired processor.

Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Multiprocessor Nios II Systems
- Run/Debug Configuration

Related Topics on the Web

- Nios II Processor Reference Handbook at www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf - Contains details on the JTAG debug module on the Nios II processor.



Viewing Execution Trace

The Nios II IDE provides methods to trace code execution for Nios II hardware and Nios II instruction set simulator (ISS) targets during debug sessions. Trace methods display a sequential list of the instructions executed prior to the processor suspending.

When debugging a program on a Nios II hardware target, the Trace view in the Debug perspective displays the exact execution trace of the program running in hardware. The Nios II trace preferences affect how the Trace view collects and displays data.

When debugging a program on a ISS target, the ISS can display execution trace in the Console view, or you can optionally redirect the output to a file. The ISS trace preferences affect how the ISS collects and displays data.

▶ To view trace data for Nios II hardware targets:

1. Start a debug session with a Nios II hardware target. Code displays in the Debug perspective and processor execution suspends.
2. On the Window menu, point to **Show View**, and then click **Trace**. The Trace view appears.
3. Continue your debug session. The Trace view automatically displays trace data for the most recently executed instructions whenever processor execution suspends.

To copy trace data onto the clipboard for Nios II hardware targets:

1. Select the desired trace data output in the Trace view. You can either highlight a portion of the text, or right-click in the Trace view and click **Select All**.
2. Right-click in the Trace view, and then click **Copy**.

▶ To view trace data for Nios II instruction set simulator targets:

The following steps require that you first create an ISS run/debug configuration for your project.

1. On the Run menu, click **Debug...** The **Run/Debug** dialog box appears.
2. Expand **Nios II Instruction Set Simulator** in the **Configurations** list. The list of your ISS run/debug configurations appears.
3. Select your ISS project's run/debug configuration from the **Configurations** list.
4. Click the **ISS Settings** tab. The **ISS Settings** tab appears. The **Trace Options** control what trace information to display.
5. Turn on **Enable Tracing**.
6. Turn on the trace options you want to use.
7. Click **Debug** in the lower-right corner of the **Run/Debug** dialog box. Trace data will display in the Console view.

The following is example output:

```
[19] 0x010002c8: 0xd839883a mov fp, sp [dstData=0x1fffff0 dstReg=fp]
```


This line shows the executed instruction has the value 0xd839883a at program counter location 0x010002c8. The instruction moves the value in `sp` (0x1ffff0) to `fp`. When **Instruction Count** is on, the ISS counts each instruction executed, and displays the instruction count (such as [19]).

▶ [To direct trace data for Nios II instruction set simulator targets to a file:](#)

The following steps require that you first create an ISS run/debug configuration for your project.

1. On the Run menu, click **Debug...** The **Run/Debug** dialog box appears.
2. Expand **Nios II Instruction Set Simulator** in the **Configurations** list. The list of your ISS run/debug configurations appears.
3. Select your ISS project's run/debug configuration from the **Configurations** list.
4. Click the **ISS Settings** tab. The **ISS Settings** tab appears. The **Trace Options** control what trace information to display.
5. Turn on **Enable Tracing**.
6. Turn on the trace options you want to use.
7. Turn on **Send trace output to file**.
8. Specify the name of your output file in the **Trace File** box. The file will reside in the application project directory or use **Browse...** to specify a different location.
9. Click **Debug** in the lower-right corner of the **Run/Debug** dialog box. Initial information and warnings appear in the Console view. All subsequent output writes to the trace file.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Trace View (Debug Perspective)
- Trace Page (Preferences Dialog Box)
- Run/Debug Configuration





Viewing Disassembly


Viewing disassembly allows you to analyze the exact instruction-by-instruction execution during a debug session. You can use the Disassembly view to see mixed C/C++ and assembly source code, set breakpoints, and single step through your code.

▶ To open Disassembly view:

To view disassembly, start a debug session. The Disassembly view displays in the Debug perspective automatically.

▶ To step through code:

By default, stepping through the code executes entire C/C++ statements. To step through individual assembly instructions, click  **Instruction Stepping Mode** in the Debug view toolbar. In instruction stepping mode, stepping through code advances the instruction pointer in the Disassembly view to the next assembly instruction. Because multiple assembly instructions represent a single line of C/C++ code, the instruction pointer might not advance in the C/C++ Editor view with each instruction step. Click  **Instruction Stepping Mode** a second time to return to stepping at the C/C++ statement level.

 **Note:** If the IDE cannot locate source code for a function, stepping through the code automatically uses Disassembly view regardless of whether instruction stepping mode is on or off.

▶ To set breakpoints on assembly instructions:

Double-click in the margin to the left of the instruction to toggle an address breakpoint on and off. You can also right-click in the margin, and then click **Toggle Breakpoint**.

▶ To copy text from the Disassembly view:

You can highlight and copy text in the Disassembly view by using the keyboard shortcut (e.g. Type `ctrl-c` to copy on Windows).



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Debugging the Project - This page of the Software Development Tutorial contains debugging instructions.
- Disassembly View



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Concepts > Debug > Breakpoints - Contains details on adding breakpoints.



Related Topics on the Web

- Nios II Processor Reference Handbook at www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf - Contains details on the Nios II instruction set.

Profiling Execution Performance



About Profiling with the Nios II IDE

The GNU profiler (gprof) collects information about which functions call other functions during program execution, and tracks the time spent in each function. Profiling tells you information about the efficiency of your program by showing you where and how your program spends its time. Profiling can help you determine where to optimize your code to improve execution performance.

The Profiling perspective in the Nios II IDE provides a convenient and useful way to analyze the GNU profiling data. Many display features of the perspective's views make the data much easier to read and analyze, compared to reading the standard gprof text output.



Related Nios II IDE Help Topics

- Profiling C Code
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.



Profiling C Code

To use the GNU profiler, you must compile and link your program with the profiling library, execute the program to generate a **gmon.out** profile data file, and run **gprof** to parse the profile data. The following sections describe how to accomplish the tasks.

▶ To collect profiling data:

1. Switch to the C/C++ Development perspective, if necessary.
2. Right-click your project in the C/C++ Projects view, and then click **System Library Properties**.
3. In the **Properties** dialog box, click **System Library**.
4. Turn on **Link with profiling library**.
5. Click **OK**.
6. Right-click your project in the C/C++ Projects view, point to **Run As**, and then click **Nios II Hardware**.

After program execution completes, the file **<project directory>/<build configuration directory>/gmon.out** contains the profiling information.



Note: Execution must return from `main()` to create the file; if you terminate the run or debug session, all profiling information is lost.

▶ To analyze profiling data:

The **nios2-elf-gprof** utility converts the binary **gmon.out** data file and displays the profiling information in a readable format. You can launch the **nios2-elf-gprof** utility automatically with the Profiling perspective in the Nios II IDE.

To view the profiling data, perform the following steps:

1. In the C/C++ Projects view, locate the generated **gmon.out** file in the **<build configuration directory>** (typically Release or Debug) of your project.
2. Double-click the **gmon.out** file to automatically switch to the Profiling perspective and display the **gmon.out** data.

You can also launch the **nios2-elf-gprof** utility from the command line manually.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective
- Call Hierarchy View (Profiling Perspective)
- Editor View (Profiling Perspective)
- Samples - Function Total View (Profiling Perspective)
- Samples - Line By Line View (Profiling Perspective)



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.

Storing Firmware on the Target Board



About Storing Firmware

After you have successfully built and debugged an application, you might want to store the executable file (**.elf**) as firmware in your target hardware. You might store firmware in the target at the end of the development process when the firmware is ready to release to manufacturing. Alternatively, you might store firmware during the development process as part of a test procedure.

The following sections discuss the two broad approaches to storing the firmware in the hardware.

▶ Storing Firmware in On-Chip Memory

Depending on the target FPGA architecture, you can design Nios II systems to initialize on-chip memory immediately after FPGA configuration. If you configure the system library project to place program code or data into on-chip memory, the Nios II IDE automatically creates memory initialization files for the FPGA. The Nios II IDE stores these files in the same location as the SOPC Builder system file (**.ptf**). When you are ready to release your project's executable code, you must send the memory initialization file(s) back to the hardware designer to include them in the FPGA design.

Memory initialization file names take the form: *<name of memory>.hex*.

▶ Programming Firmware in Flash Memory

You can program your Nios II executable file directly to a flash memory in your development hardware.

The Nios II IDE flash programmer provides an easy way to program flash memory devices on the target board using an Altera download cable, such as the USB Blaster. The flash programmer programs executable software files (**.elf**) and FPGA hardware configuration files (**.sof**) into flash memory. The flash programmer can also program binary data files.

The flash programmer can program two kinds of devices:

- Common flash interface (CFI) compliant flash memories
- Altera EPCS serial configuration devices



Related Nios II IDE Help Topics

- Programming Flash
- Flash Programmer Dialog



Related Topics on the Web


- Nios II Flash Programmer User Guide at www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf



Programming Flash

You can program flash memory with software files, FPGA configuration files, and data files. Programming flash memory enables your hardware to load software and FPGA configurations at startup time. You can program flash memory connected to an FPGA using the Nios II IDE flash programmer.

You manage programming flash memory using flash programmer configurations. A flash programmer configuration is a group of settings that affect the flash programming process for a specific hardware target. You can create multiple flash programmer configurations, each of which has its own programming parameters. This is useful if you are working on multiple projects or targeting multiple boards. You can also set up a flash programmer configuration to program a combination of file types, allowing you to program two or three files in one operation.


 **Note:** To run the flash programmer, the files to program into flash must reside in a Nios II application project. The associated SOPC Builder system must include flash memory and specify a target board.

▶ To open the flash programmer and create a flash programmer configuration:

1. On the Tools menu, click **Flash Programmer....** The **Flash Programmer** dialog box opens.
2. Click **New**, which creates a new flash programmer configuration. A new flash programmer configuration appears in the **Configurations** list.
3. Enter a unique, meaningful name for the new configuration in the **Name** box.

▶ To specify files to program into flash memory:

1. In the **Configurations** list, click a flash programmer configuration.
2. Click the **Main** tab.
3. Turn on **Program software into flash memory**.
4. Specify the desired project in the **Project** box. The flash programmer will automatically find the **Nios II ELF Executable** for your project.

 **Note:** In order for the flash programmer to program an FPGA configuration, you must specify a software project, even if you are not programming an executable file into flash memory. The flash programmer uses the target board setting in the SOPC Builder system associated with the software project to determine the available flash memories and FPGA configuration locations.

5. If you are not programming an executable file into flash memory, turn off **Program software into flash memory**.
6. If you have an FPGA configuration to program into flash memory, use the following steps:
 1. Turn on **Program FPGA configuration data into hardware-image region of flash memory**.
 2. Specify the data file in the **FPGA configuration (.sof)** box.

3. If your hardware supports multiple FPGA configurations, specify the FPGA configuration location in **Hardware Image**.
7. If you have a data file to program into flash memory, use the following steps:
 1. Turn on **Program a file into flash memory**.
 2. Specify the file to program to flash in the **File** box.
 3. In the **Memory** list, select the flash memory device.
 4. In the **Offset** box, type the offset within the flash memory device to place the base of the data.

▶ **To specify programming cable and target flash memory device:**

1. In the **Configurations** list, click a flash programmer configuration.
2. Click the **Target Connection** tab.
3. In the **JTAG cable** list, select the JTAG cable attached to your target board. If you only have one cable, the **automatic** value automatically identifies your cable. If your cable is not shown in the list, make sure that it is installed correctly, and click **Refresh** to add it to the list.
4. In the **JTAG device** list, select the Nios II system to program. If you only have one Nios II system connected to your JTAG cable, the **automatic** value automatically identifies your system. If your Nios II system is not shown in the list, make sure that the hardware is installed correctly, and click **Refresh** to add it to the list.

▶ **To program flash memory on the target board:**

1. In the **Configurations** list, click a flash programmer configuration. Settings on the **Main** and **Target Connection** tabs must be legal values to proceed.
2. Click **Program Flash** in the lower-right corner of the dialog box to begin programming the flash. The flash programming process might take several minutes, depending on the size of the data to download.

If your project is not up to date, the Nios II IDE automatically builds your project before programming it into flash memory. To turn off the automatic build, on the Window menu, click **Preferences**, then expand **Run/Debug**, click **Launching**, and turn off **Build (if required) before launching**.

The flash programmer automatically appends boot-loader code to the front of the programming data if it is needed.



Related Nios II IDE Help Topics

- About Storing Firmware
- Flash Programmer Dialog



Related Topics on the Web

- Nios II Flash Programmer User Guide at www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf

Features and Terms Reference



Advanced Debugging Features by FS2

The FS2 console from First Silicon Solutions, Inc. (FS2) provides additional advanced debugging features for Windows users. During a debug session, the Nios II IDE automatically launches the FS2 console when the Debugger Tab (Run/Debug Dialog Box) **Use FS2 console window for trace and watchpoint support** setting is on. This feature is not available on Linux.



Related Topics on the Web

- First Silicon Solutions, Inc. at www.fs2.com



C-to-Hardware Acceleration (C2H) Compiler

The Nios II C-to-Hardware Acceleration (C2H) Compiler is a tool that allows you to create custom hardware accelerators directly from ANSI C source code. A hardware accelerator is a block of logic that implements a C function in hardware, which often improves the execution performance by an order of magnitude. Using the C2H Compiler, you can develop and debug an algorithm in C targeting a Nios II processor, and then quickly convert the algorithm to a hardware accelerator implemented in a field programmable gate array (FPGA).



Related Topics on the Web

- Nios II C2H Compiler User Guide at www.altera.com/literature/ug/ug_nios2_c2h_compiler.pdf



Hardware Abstraction Layer (HAL)

The Nios II hardware abstraction layer (HAL) runtime library is a lightweight runtime environment that provides a simple device driver interface for programs to communicate with the underlying hardware. The HAL application program interface (API) is integrated with the ANSI C standard library. The HAL API allows you to access devices and files using familiar C library functions, such as `printf()`, `fopen()`, `fwrite()`, etc.

The HAL serves as a board-support package for Nios II processor systems, providing a consistent interface to the peripherals in your embedded systems. The Nios II integrated development environment (IDE) generates the HAL system library for you automatically. The Nios II IDE generates a custom HAL system library to match the hardware configuration. Changes in the hardware configuration automatically propagate to the HAL device driver configuration, preventing bugs that might otherwise appear due to subtle changes in the underlying hardware.



Related Nios II IDE Help Topics

- System Library (Properties Dialog Box)
- About the Nios II IDE Managed-Make Build Environment



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains information on the HAL system library.



Hardware Simulation with ModelSim

The Nios II development environment is integrated with the ModelSim hardware simulator, which allows the Nios II IDE to launch programs on a hardware simulation of a Nios II processor system. Hardware simulation allows you to simulate cycle-accurate behavior of a Nios II processor system. Using your PC, you can see how an SOPC Builder system will behave in hardware, including signals internal to the processor and interface signals to the outside world.

ModelSim simulation is generally the domain of hardware engineers. ModelSim requires hardware design files, and therefore the ModelSim simulation process is linked closely with SOPC Builder. You can only launch ModelSim from the Nios II IDE if the target SOPC Builder system was generated in SOPC Builder with the ModelSim simulation option enabled.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Running on the ModelSim Simulator



Related Topics on the Web

- AN351: Simulating Nios II Embedded Processor Designs at www.altera.com/literature/an/an351.pdf - Contains details for ModelSim.



Hardware Target

A Nios II hardware target is a printed circuit board (PCB) with an Altera FPGA containing a Nios II system. Altera and other companies provide a number of Nios II hardware targets in the form of development boards. Many other companies develop their own proprietary hardware targets.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Running and Debugging on Hardware



Related Topics on the Web

- Nios II home page at www.altera.com/nios2 - Contains all Nios II processor-related information, including details on Altera development boards.



Host-Based File System

The host-based file system enables programs executing on a target board to read and write files stored on the host computer. The Nios II IDE transmits file data over the Altera download cable. Your program accesses the host based file system using the ANSI C standard library I/O functions, such as `fopen()` and `fread()`. The host-based file system is a software component which you add to your system library.

The following features and restrictions apply to the host based file system:

- The host-based file system makes the C/C++ application project directory and its subdirectories available to the hardware abstraction layer (HAL) file system on the target hardware.
- The target processor can access any file in the project directory. Be careful not to corrupt project source files.
- The host-based file system only operates while debugging a project. It cannot be used for run sessions.
- Host file data travels between host and target serially through the Altera download cable, and therefore file access time is relatively slow. It takes approximately 10 ms per call to the host. For higher performance, use buffered I/O (`fread()`, `fwrite()`, etc.), and increase the buffer size for large files.

You configure the host-based file system using the **Software Components** dialog box. The host-based file system has the following setting:

- **Mount-point** box - Specifies the mount point within the HAL file system. For example, if you name the mount point `/mnt/host` and the project directory on you host computer is `/software/project1`, the code `fopen("/mnt/host/foo.dat", "r");` in a HAL-based program opens the file `/software/project1/foo.dat`.

▶ Example Programs:

The **Host File System** project template in the **New Project** wizard contains one example of using the host-based file system.

Below is another example program that uses the host-based file system to read and display its own source code.

```
/*
 * "hello_hostfs.c" example.
 *
 * This example reads the file "hello_hostfs.c" (ie this file) from the
 * project directory on the host and writes it to the STDOUT stream.
 * It runs on any design which includes a hostfs software component,
 * with any RTOS (or no RTOS) and requires a STDOUT device in your
 * system's hardware.
 */

#include <stdio.h>

int main()
{
    FILE * hostfile = fopen("/mnt/host/hello_hostfs.c", "r");
    char buffer[1024];

    while (fgets(buffer, sizeof(buffer), hostfile) != NULL)
    {
```

```
    fprintf(stdout, "%s", buffer);  
  }  
  
  fclose(hostfile);  
  
  printf("That's all folks\n");  
  
  return 0;  
}
```



Related Nios II IDE Help Topics

- Choosing and Configuring Middleware Software Components
- Software Components Dialog Box (System Library Properties Page)



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on accessing files using the HAL API and the ANSI C standard library.



Instruction Set Simulator (ISS)

The Nios II instruction set simulator (ISS) allows you to execute and debug Nios II programs in simulation on a host PC. The ISS simulates software executing on a Nios II processor core connected to a limited set of peripherals. The simulation is at the functional level, and all operations complete in one cycle. It is not a cycle-accurate simulation, and therefore performance benchmarking on the ISS gives optimistic results. On a modern Windows PC, the ISS runs at about 300K instructions per second when simulating code on the **fast** example design provided in the Nios II Embedded Design Suite.

The ISS can produce an execution trace. The trace output appears in the Console view, and you can optionally redirect it to a file. It is common to output trace data to a file, because trace tends to produce a large amount of information.

▶ ISS-supported SOPC Builder components:

- All Nios II processor cores: Nios II/f, Nios II/s, Nios II/e
- Interval timer core
- JTAG UART core
- UART core
- On-chip memory (RAM/ROM)
- SDRAM controller core
- IDT71V416 SRAM (1 MB SRAM mounted on Nios development board)
- EPCS serial flash controller core, with limitations.

If any unsupported components are present in the system, the ISS displays a warning message at the start of the run or debug session. The ISS ignores writes to unsupported components during simulation. Reading from an unsupported component during simulation returns zero.

▶ SOPC Builder system requirements:

The Nios II ISS simulates a Nios II processor system described by an SOPC Builder system file (**.ptf**). The Nios II ISS makes the following assumptions about the SOPC Builder system:

- SOPC Builder successfully generated the **.ptf** file.
- All memories with initialized content are initialized from one **.elf** file.
- The system contains exactly one Nios II CPU. The ISS does not support multiprocessor systems.
- The system has one clock domain.
- The system has one address map. (This is true for all Nios II systems created by SOPC Builder.)

▶ ISS limitations:

- Simulations are functional only, and not cycle-accurate.

- The ISS does not model Nios II instruction and data caches, and will not find bugs involving cache initialization, flushing, or bypassing.
- The ISS does not support reading or writing tightly coupled memories connected to the Nios II processor.
- The ISS does not support custom instructions.
- The ISS models the Nios II `ienable` register as a complete 32-bit register. In hardware (both on a target board and in HDL simulation), all bits associated with unused interrupt inputs are always zero.
- The EPCS Serial Flash Controller core only supports boot-from-flash behavior. If the SOPC Builder system contains an EPCS Serial Flash Controller core, the simulation does not model the full behavior of the EPCS device. The ISS only models the first 1 Kbytes of the controller's register map as a block of ROM. In the case that the processor resets to the EPCS controller address (the typical boot-from-flash scenario), the simulation relies on the fact that RAMs are pre-initialized. Therefore, the controller's boot-loader does not need to copy code from EPCS memory to RAM. Instead, the controller simply jumps directly to RAM.

You can run or debug on the ISS from the Nios II IDE or from the Nios II Command Shell command line, although Altera recommends command-line usage only to advanced users.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Running and Debugging on the ISS



Lightweight TCP/IP Stack

The lightweight IP (lwIP) TCP/IP stack is a small-footprint implementation of the transmission control protocol/internet protocol (TCP/IP) suite. The focus of the lwIP TCP/IP implementation is to reduce resource usage while providing a full scale TCP/IP. Altera provides the Lightweight IP (lwIP) TCP/IP Stack as a software component plug-in for the Nios II IDE.

The lwIP TCP/IP stack is a software component which you add to your system library. You configure the lwIP TCP/IP stack using the **Software Components** dialog box. In order to use the lwIP TCP/IP stack, you must base your system library project on the MicroC/OS-II RTOS. If your system library project is not based on the MicroC/OS-II RTOS, the **Software Components** dialog box will not allow you to add the lwIP software component.

Your program code can use the sockets API to send data using the Ethernet hardware.



Related Nios II IDE Help Topics

- Choosing and Configuring Middleware Software Components
- Software Components Dialog Box (System Library Properties Page)
- Choosing and Configuring an Operating System
- MicroC/OS-II RTOS



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on writing Nios II programs using lwIP, including descriptions of the settings available on this page.
- Using Lightweight IP for the Nios II Processor Tutorial at www.altera.com/literature/tt/tt_nios2_lwip_tutorial.pdf - Contains step-by-step instructions on creating lwIP applications.



MicroC/OS-II RTOS

MicroC/OS-II is a real-time, multitasking kernel for microprocessors and microcontrollers. Altera provides the MicroC/OS-II real-time operating system (RTOS) as part of the Nios II Embedded Design Suite. Evaluation versions of the Nios II Embedded Design Suite do not include MicroC/OS-II. The Nios II IDE makes it easy to base your C/C++ application project on MicroC/OS-II. When enabled, the MicroC/OS-II real-time kernel is part of the system library.



Note: You can evaluate MicroC/OS-II at no charge using the Nios development board. However, you must purchase a license to ship a commercial product based on MicroC/OS-II.

▶MicroC/OS-II Thread-Aware Debugging:

When debugging a MicroC/OS-II application, the debugger displays the current state of all threads within the application, including backtraces and register values. You cannot use the debugger to change the current thread, so it is not possible to use the debugger to change threads or to single step a different thread. Thread-aware debugging does not change the behavior of the application in any way.



Related Nios II IDE Help Topics

- Choosing and Configuring an Operating System



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on writing Nios II programs based on MicroC/OS-II.
- Using MicroC/OS-II RTOS with the Nios II Processor Tutorial at www.altera.com/literature/tt/tt_nios2_MicroC_OSII_tutorial.pdf - Contains step-by-step instructions on creating MicroC/OS-II applications.



Multiprocessor Nios II Systems

The Nios II IDE is capable of running and debugging multiple Nios II processors simultaneously. Altera hardware development tools automatically connect the JTAG debug circuitry to the Nios II processor(s). Regardless of where the Nios II processor(s) reside in the JTAG chain, the Nios II IDE can connect to each processor, download code, and run and debug. For example, the Nios II IDE supports all of the following cases:

- 1 processor in an FPGA in a JTAG chain of 1 device - This is the simplest single-processor case. This is the structure used on Nios development boards.
- 1 processor in an FPGA in a JTAG chain of 3 devices - This is a common case, in which an Altera FPGA co-exists on a board with other devices in the JTAG test chain.
- 2 processors in an FPGA in a JTAG chain of 1 device - This is a common case for multi-processor Nios II systems, in which multiple CPUs reside on a single FPGA.
- 2 processors, 1 each in 2 FPGAs in a JTAG chain of 2 devices
- 2 processors in separate FPGAs in separate JTAG chains - The two Nios II processor systems can be on one board, or they can be in entirely separate systems. This case requires multiple download cables to connect to the separate JTAG chains.

Multiprocessor systems require extra consideration during both hardware and software development. If multiple processors exist in a single SOPC Builder system, the hardware designer must consider which memory device(s) to share between the processors. Furthermore, you must make sure that one processor doesn't store data in the same memory space that another processor uses for code.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Running and Debugging Multiprocessor Collections



Related Topics on the Web

- Creating Multiprocessor Nios II Systems Tutorial at www.altera.com/literature/tt/tt_nios2_multiprocessor_tutorial.pdf
- Quartus II Handbook Volume 5: Embedded Peripherals at www.altera.com/literature/hb/nios2/n2cpu_nii5v3.pdf - Contains details on the mailbox and mutex peripherals for coordinating multiprocessor systems.
- Nios II Processor Reference Handbook at www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf - Contains details on the JTAG debug module on the Nios II processor.



Run/Debug Configuration

A run/debug configuration is an IDE-managed file that stores the settings necessary to run or debug a specific project on a specific target. The available target types are: Nios II hardware, Nios II instruction set simulator (ISS), Nios II ModelSim, and Nios II multiprocessor collection. Configurations are categorized by target type and contain settings specific to the target type. The help topics for the individual tabs in the Run/Debug dialog box describe all of the settings available.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Run/Debug Dialog Box



Valid Project Names

Nios II IDE project names must be unique and conform to the following rules:

- Alphabetic characters, numeric characters, and the underscore symbol are the only valid project name characters. Project names cannot contain spaces or special characters, such as \, +, =.
- The first character in the project name must be alphabetic or the underscore symbol.
- The minimum filename length is one character.
- The maximum filename length is 250 characters.



Zip Read-Only File System

The zip read-only file system provides access to a simple file system stored in flash memory. Your program accesses the Zip file system using the ANSI C standard library I/O functions, such as `fopen()` and `fread()`. The zip read-only file system is a software component which you add to your system library.

The zip read-only file system requires an uncompressed **.zip** file that contains all of the files for the zip file system. This zip file must exist in the system library project.



Note: You can drag-and-drop a zip file from the host file system onto your system library project in the C/C++ Projects view to automatically import the zip file into the system library project. Alternatively, on the File menu, click **Import...**, then use **File system** (not **Zip File**) to import the **.zip** file (not the contents of the zip file) to your system library.

You configure the zip read-only file system using the **Software Components** dialog box. The zip read-only file system has the following settings:

- **Flash memory device** list - Displays the available target flash devices and allows you to select a device to program.
- **Offset** box - Specifies an offset into the flash memory, indicating where to write the zip file system contents.
- **Mount-point** box - Specifies the mount point within the hardware abstraction layer (HAL) file system. For example, if you name the mount point **/mnt/zipfs**, the code `fopen("/mnt/zipfs/foo", "r");` in a HAL-based program opens the file **foo** within the zip file.
- **Zip file (must be uncompressed)** - Specifies the zip file.

For your program to access files in the zip file system, you must write the project to flash memory using the flash programmer. As part of the project build process, the Nios II IDE creates a **.flash** file that includes the data for the zip file system, and stores it in the Release or Debug configuration directory of your project.



Related Nios II IDE Help Topics

- Choosing and Configuring Middleware Software Components
- Software Components Dialog Box (System Library Properties Page)



Related Topics on the Web

- Nios II Flash Programmer User Guide at www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf

GUI Reference



Flash Programmer Dialog Box

You open this dialog box by clicking **Flash Programmer...** on the Tools menu.

The **Flash Programmer** dialog box lets you program data to flash memory with the flash programmer. The flash programmer uses a flash programmer configuration, which contains all the parameters that affect the flash programming process. The **Flash Programmer** dialog box includes controls to create flash programmer configurations and set up all the parameters they contain.

The **Flash Programmer** dialog box contains the following controls.

- **Configurations** - Is a list of available flash programmer configurations and allows you to select an active configuration.
- **New** - Creates a new flash programmer configuration.
- **Delete** - Deletes the selected flash programmer configuration.
- **Name** - Allows you to enter a unique name for a configuration. **Name** appears only after you select an existing configuration.
- **Apply** - Saves your settings in the selected flash programmer configuration.
- **Revert** - Discards your setting changes, and returns the selected flash programmer configuration to its previous settings.
- **Program Flash** - Programs the flash memory using the settings in the selected flash programmer configuration.
- **Close** - Closes the **Flash Programmer** dialog box.

When you select a flash programmer configuration in the **Configurations** list, tabs appear in the right side of the dialog box. Each tab presents a group of settings in the selected configuration. The **Flash Programmer** dialog box includes the following tabs:

▶ **Main Tab:**

The **Main** tab allows you to specify the application project, the content to program, and other settings for the selected configuration.

The **Main** tab contains the following settings.

- **Program software project into flash memory** - When on, the flash programmer programs the software project into flash memory. When the project is selected, the dialog box shows you the following read-only values.
 - **Nios II ELF Executable** - The executable file (**.elf**) to program to flash. This is the **.elf** file associated with the project.
 - **Target Hardware** - The target SOPC Builder system and CPU for the selected project.
- **Program FPGA configuration data into hardware-image region of flash memory** - When on, the flash programmer programs FPGA configuration data into flash memory. The following options specify how the FPGA configuration is programmed.
 - **FPGA Configuration (.sof)** - Specifies the file that contains the FPGA configuration data.

- **Hardware Image** - Specifies the hardware image location in which to store the FPGA configuration. The contents of this drop-down list depend on the SOPC Builder target board, which specifies where it expects FPGA configuration data to reside in flash.
- **Program a file into flash memory** - When on, the flash programmer programs a data file into the chosen flash memory device at a given offset. The following options affect how the file is programmed into flash.
 - **File** - The file to program to flash.
 - **Memory** - The flash memory device.
 - **Offset** - The offset within the flash memory.

▶ **Target Connection Tab:**

The **Target Connection** tab specifies the download cable and device for the selected configuration.

The **Target Connection** tab contains the following settings.

- **JTAG cable** - Specifies the Altera download cable (such as the USB Blaster) to use for communicating with the Nios II processor. If you only have one cable, the **automatic** value selects your cable.
 - **Refresh** - Updates the list if you change your JTAG cables while in the flash programmer dialog box.
- **JTAG device** - Specifies the device on the selected JTAG chain.
 - **Refresh** - Updates the list if you change your JTAG devices while in the flash programmer dialog box.



Related Nios II IDE Help Topics

- About Storing Firmware
- Programming Flash



Related Topics on the Web

- Nios II Flash Programmer User Guide at www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf



Import Wizard

You open this wizard by clicking **Import...** on the File menu.

You use the **Import** wizard to import an existing project that you, or someone else, created in a different IDE workspace. The first page of the **Import** wizard presents a list of all import sources available. The only one of concern to Nios II IDE users is:

- **Existing Altera Nios II Project into Workspace** - Allows you to browse your hard drive for existing projects to import.



Note: The Nios II IDE includes the **Import** and **Export** wizards, which are part of the standard Eclipse IDE framework. Do not use these wizards to import or export files, or to export a project. There are easier methods for importing, exporting and sharing projects and files.



Related Nios II IDE Help Topics

- [Importing, Exporting and Sharing Projects and Files](#)

New Project Wizard



New Project Wizard

You open this wizard by pointing to **New** on the File menu, and then clicking  **Project.....**

The **New Project** wizard guides you through the process of creating a new project. The wizard consists of several pages that present an ordered sequence of actions to create a project.

The wizard provides the following controls on all pages:

- **Back** - Goes back to the previous page of the wizard.
- **Next** - Advances to the next page of the wizard.
- **Finish** - Completes the wizard, using default values for any remaining pages.
- **Cancel** - Cancels the work in progress and closes the **New Project** wizard.

The first page of the **New Project** wizard presents a list of all project types available. Project types specific to the Nios II processor are under the **Altera Nios II** category. The remaining pages of the wizard depend on the project type specified. The following **New Project** wizard pages correspond to the project types available for the Nios II processor:

- New C/C++ Application - Creates a new project for developing C/C++ programs.
- New System Library - Creates a new system library project for a particular SOPC Builder system.
- New Advanced C/C++ Project - Creates a skeleton project for developing C/C++ programs, which requires you to create and manage the makefiles.
- New Managed Library Project - Creates a new project for developing C/C++ libraries.




Related Nios II IDE Help Topics

- About Nios II IDE Projects
- Creating a New Project
- New C/C++ Application (New Project Wizard)
- New System Library (New Project Wizard)
- New Advanced C/C++ Project (New Project Wizard)
- New Managed Library (New Project Wizard)



New C/C++ Application (New Project Wizard)


You open this wizard page by clicking  **C/C++ Application** in the **New Project** wizard, and then clicking **Next**.

The **C/C++ Application** pages of the **New Project** wizard allow you to create a new C/C++ application project and an associated system library project.

▶ **The first page allows you to specify settings for the C/C++ application project:**

The following controls are available.

- **Name** - Specifies a valid project name for the new project.
- **Use Default Location** - When on, the IDE creates the new project directory in the default location specified on the **New Projects** preference page. Turning off **Use Default Location** allows you to specify an alternative project location in the **Location** box.

 **Note:** Project directories cannot be nested. You cannot create a new project inside the directory of an existing project.

- **Location** - Specifies the base directory where the new project contents will reside.
- **Select Target Hardware** - These options specify the target hardware.
 - **SOPC Builder System** - Specifies the target SOPC Builder system. The SOPC Builder system file (**.ptf**) defines the CPUs and peripherals included in the SOPC Builder system. Select from the drop-down list of recently used SOPC Builder systems, or click **Browse...** to find a specific **.ptf** file.
 - **CPU** - Specifies the target CPU in the SOPC Builder system. If there is only one CPU in the system, the Nios II IDE selects it automatically. If there are multiple CPUs, you must select one from the list.
- **Select Project Template** - The Nios II Embedded Design Suite (EDS) offers several ready-made example designs. You can use these as reference designs, or as the basis for your own projects. The Nios II IDE copies the design files for the selected template into your project. Each template provides a **readme.txt** file that describes the purpose and usage of the design files. Using **Blank Project** avoids copying any files into the new project.

▶ **The second page allows you to specify settings for the associated system library project:**

The following controls are available.

- **Create a new system library named: <application project name>_syslib** - When selected, the Nios II IDE creates a new HAL system library project to accompany your C/C++ application project. This is the default action, which is usually desirable for single-threaded Nios II programs.
- **Select or create a system library** - When selected, you can specify an existing system library in the **Available System Library Projects** list, or create a new system library via the **New System Library Project...** button. Multiple projects can use the system library.

- **New System Library Project...** - Launches a wizard to create a new system library.



Related Nios II IDE Help Topics

- Creating a New Project
- New Project Wizard




Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - The chapters related to the hardware abstraction layer (HAL) system library contain details on system libraries and the structure of Nios II software projects.



New System Library (New Project Wizard)

You open this wizard page by clicking  **System Library** in the **New Project** wizard, and then clicking **Next**.

The **System Library** page of the **New Project** wizard allows you to create a new system library for an SOPC Builder system.

The **System Library** page contains the following controls.

- **Name** - Specifies a valid project name for the new project.
- **Use Default Location** - When on, the IDE creates the new project directory in the default location specified on the **New Projects** preference page. Turning off **Use Default Location** allows you to specify an alternative project location in the **Location** box.



Note: Project directories cannot be nested. You cannot create a new project inside the directory of an existing project.

- **Location** - Specifies the base directory where the new project contents will reside.
- **Select Target Hardware** - These options specify the target hardware.
 - **SOPC Builder System** - Specifies the target SOPC Builder system. Select from the drop-down list of recently used SOPC Builder systems, or click **Browse...** to find a specific **.ptf** file.
 - **CPU** - Specifies the target CPU in the SOPC Builder system. If there is only one CPU in the system, the Nios II IDE selects it automatically. If there are multiple CPUs, you must select one from the list.
- **Select Type of RTOS** - These options specify the RTOS to build into the system library. The following are the options available from Altera.
 - **None (single-threaded)** - The Nios II IDE bases the system library on the Altera hardware abstraction layer (HAL) providing a single-threaded environment.
 - **MicroC/OS-II** - The Nios II IDE includes both the MicroC/OS-II real-time kernel and the Altera HAL in the system library. This option is not available in evaluation versions of the Nios II Embedded Design Suite.



Related Nios II IDE Help Topics

- About Nios II IDE Projects
- Creating a New Project



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - The chapters related to the hardware abstraction layer (HAL) system library contain details on system libraries and the structure of Nios II software projects.
- Nios II home page at www.altera.com/nios2 - Contains an up-to-date list of RTOS vendors that support the Nios II processor.



New Managed Library (New Project Wizard)

You open this wizard page by clicking  **Managed Library Project** in the **New Project** wizard, and then clicking **Next**.

The **Managed Library Project** page of the **New Project** wizard allows you to create a new managed library project for developing C/C++ libraries. Managed Library projects typically contain independent utility functions, and do not have dependencies on a system library project. You can reference managed library projects from C/C++ application projects or from other managed library projects.

The **Managed Library Project** page contains the following controls.

- **Name** - Specifies a valid project name for the new project.
- **Use Default Location** - When on, the IDE creates the new project directory in the Nios II IDE workspace folder, unless you have previously set another custom default location on the **New Projects** preference page. Turning off **Use Default Location** allows you to specify an alternative project location in the **Location** box.



Note: Project directories cannot be nested. You cannot create a new project inside the directory of an existing project.

- **Location** - Specifies the base directory where the new project contents will reside.



Related Nios II IDE Help Topics

- [Creating a New Project](#)
- [New Project Wizard](#)



New Advanced C/C++ Project (New Project Wizard)

You open this wizard page by clicking  **Advanced C/C++ Project** in the **New Project** wizard, and then clicking **Next**.

The **Advanced C/C++ Project** page of the **New Project** wizard allows you to create advanced C/C++ projects. Advanced C/C++ projects are projects that do not manage the project makefiles for you. This gives you total control over the build process; creating and managing the makefile becomes your responsibility.

An Advanced C/C++ project is the same as a standard make project in the Eclipse C/C++ Development Toolkit (CDT).

The **Advanced C/C++ Project** page contains the following controls.

- **Name** - Specifies a valid project name for the new project.
- **Use Default Location** - When on, the IDE creates the new project directory in the default location specified on the **New Projects** preference page. Turning off **Use Default Location** allows you to specify an alternative project location in the **Location** box.



Note: Project directories cannot be nested. You cannot create a new project inside the directory of an existing project.

- **Location** - Specifies the base directory where the new project contents will reside.



Note: Altera does not recommend using advanced C/C++ projects. Instead, let the Nios II IDE manage your project for you by using New C/C++ Application.



Related Nios II IDE Help Topics

- About Nios II IDE Projects
- Creating a New Project



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Concepts > CDT Projects - Contains details on standard make projects.

Preferences Dialog Box



Preferences Dialog Box

You open this dialog box by clicking **Preferences** on the Window menu.

The **Preferences** dialog box allows you to customize the look, feel, and behavior of the Nios II IDE workbench.

Certain preference pages are specific to the Nios II IDE. Nios II IDE preference pages affect how the Nios II IDE builds, stores, runs, and debugs projects. Other preference pages are part of the standard Eclipse environment. The Eclipse IDE framework offers user preference pages for all the Eclipse C/C++ Development Toolkit (CDT) plug-ins.



Related Nios II IDE Help Topics

- [Nios II Page \(Preferences Dialog Box\)](#)
- [New Projects Page \(Preferences Dialog Box\)](#)
- [Trace Page \(Preferences Dialog Box\)](#)



Related Eclipse and CDT Help Topics

- [Workbench User Guide > Reference > Preferences](#)
- [C/C++ Development User Guide > Reference > C/C++ preferences](#)
- [C/C++ Development User Guide > Reference > C/C++ preferences > Debug preferences](#)
- [C/C++ Development User Guide > Reference > C/C++ editor preferences](#)



Nios II Page (Preferences Dialog Box)

You open this dialog box by clicking **Preferences** on the Window menu, and then clicking **Nios II** in the left-hand pane.

The **Nios II** preferences page contains settings that affect how the Nios II IDE builds and runs projects. The settings are common for all projects.

The **Nios II** preferences page contains the following settings.

- **Show command lines when running 'make' (i.e. Don't use '-s' flag on make)** - When on, the Nios II IDE runs make in verbose mode, displaying all the command lines that the Nios II IDE executes. The default is off.
- **Generate objdump file** - When on, the Nios II IDE creates an objdump file when it builds a project. The default is off.
- **Allow multiple active run/debug sessions** - When on, you can debug multiple CPUs at the same time. The default is off.
- **Confirm before starting the flash programmer** - When on, a confirmation dialog box appears before the flash programmer writes to flash memory. The default is on.
- **Warn about launches in Run mode for C/C++ Application projects using Host-based File System** – When on, the Nios II IDE gives a warning if you attempt to run (instead of debug) a project containing a host-based file system. The default is on.



Related Nios II IDE Help Topics

- Preferences Dialog Box
- Multiprocessor Nios II Systems



New Projects Page (Preferences Dialog Box)

You open this dialog box by clicking **Preferences** on the Window menu, expanding **Nios II** in the left-hand pane, and then clicking **New Projects**.

The Nios II **New Projects** preferences page contains settings that specify the default location where the Nios II IDE saves new projects. The settings on this page affect only the default choice for new project location; the **New Project** wizard can override this location later.

The Nios II **New Projects** preferences page contains the following settings.

- **Workspace folder** - Uses the Nios II IDE workspace directory you specified in the **Workspace Launcher** dialog box at startup. The workspace is the default area for Nios II IDE projects and other IDE-related settings files. Storing projects in the workspace directory allows you to upgrade versions of the IDE while keeping the same projects in your workspace.
- **Software folder in the SOPC Builder system folder** - Uses the **<SOPC Builder system path>/software** directory. This option stores projects with hardware-specific dependencies in a subdirectory of the SOPC Builder system directory. This option only applies to system library and C/C++ application projects. Other project types do not have associated SOPC Builder systems, and default to the **Workspace folder** option.
- **Other location** - Uses a custom location. When selected, you must specify the desired directory path in the **Location** box.



Note: A Nios II IDE project cannot reside in a subdirectory of another Nios II IDE project. You must specify a location that does not overlap with any existing projects.

In all cases, the default location for a new project is the specified path plus the name of the new project.



Related Nios II IDE Help Topics

- Preferences Dialog Box
- New Project Wizard



Trace Page (Preferences Dialog Box)

You open this dialog box by clicking **Preferences** on the Window menu, expanding **Run/Debug** in the left-hand pane, and then clicking **Trace**.

The Nios II **Trace** preferences page contains settings that affect how the Nios II IDE Trace view collects and displays data. The settings apply to all projects.

The **Trace** preferences page contains the following settings.

- **Stop trace collection when the buffer is full** - When on, trace data collection stops if the on-chip trace buffer fills up before a breakpoint or watchpoint allows trace data to be read from the buffer. When off, new trace data overwrites old trace data when the buffer overflows. The default is off.
- **Include load addresses** - When on, the data in the Trace view shows data addresses associated with load instructions. The default is off.
- **Include store addresses** - When on, the data in the Trace view shows data addresses associated with store instructions. The default is off.
- **Include data values for loads and stores** - When on, the data in the Trace view shows the data values after load or store instructions. This setting is available only if **Include load addresses** or **Include store addresses** are on. The default is off.



Related Nios II IDE Help Topics

- Preferences Dialog Box
- Viewing Execution Trace
- Trace View (Debug Perspective)



Related Topics on the Web

- Nios II Processor Reference Handbook at www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf - Contains details on the JTAG debug module on the Nios II processor.

Profiling Perspective



Profiling Perspective

You open this perspective by pointing to **Open Perspective** on the Window menu, clicking **Other...**, and then double-clicking **Profiling** in the **Select Perspective** list.

The Nios II IDE Profiling perspective allows you to conveniently capture and analyze profiling data stored in a **gmon.out** data file. The display features of the Profiling perspective make the data much easier to read and analyze, compared to reading the standard gprof text output.

By default the Profiling perspective displays four views that work in conjunction with each other and are the most useful views for analyzing the **gmon.out** data.

- Editor view which runs gprof on the **gmon.out** data and displays the results
- Call Hierarchy view
- Samples - Line By Line view
- Samples - Function Total view



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling C Code
- Call Hierarchy View (Profiling Perspective)
- Editor View (Profiling Perspective)
- Samples - Function Total View (Profiling Perspective)
- Samples - Line By Line View (Profiling Perspective)



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.



Call Hierarchy View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective. You can also open this view from the Window menu by pointing to **Show View**, clicking **Other...**, expanding **Profiling**, and double-clicking **Call Hierarchy**.

The Call Hierarchy view displays the time spent in each function, based on the standard **gmon.out** data, in an easy-to-read tree format. This view also calculates and displays the percentage of time spent in each function. In this view you can follow the function call sequences much more easily than reading the gprof output.

There are two ways to view the call hierarchy data.

▶ Top down:

Top down call direction lists the calling functions, with the functions they called nested below.

The first entry in the Call Hierarchy view is **spontaneous**. This is a term gprof uses when it cannot determine what the calling function is. There are two sets of actual time and percentage time figures for each function. The first set represents the time spent within the function. The second set represents the total time spent within the function plus all functions called by that function. Each indented line in the view drills down into the details of time and percentage spent in each called function. In the top-down mode, when cycles of recursion exist in your code, a --> symbol appears to the left of the function name indicating that the called function cycles.

▶ Inverted:

Inverted call direction lists the called functions, with the functions that called them nested below.

The first entry in the Call Hierarchy view is **inverted calls**. Actual time and percentage time figures for each function represent time spent within the function.

To reverse call direction display, right-click in the Call Hierarchy view, and click **Toggle Call Direction**.



Note: You can load the source code for functions displayed in this view into the Editor view. To do this, right-click the file name, and then click **Show Source**.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.



Editor View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective.

When you open the **gmon.out** file with the Nios II IDE, the IDE automatically calls `gprof` and displays the standard `gprof` text output. This Editor view is for viewing only.



Note: Saving the file with the default **gmon.out** filename overwrites the original **gmon.out** data. If you save the output as a text file, be sure to rename the saved file.

You can change the `gprof` command line arguments and rerun the **nios2-elf-gprof** utility. To do so, right-click anywhere in the Editor view and click **Change gprof arguments**. Type comma separated values, such as `--help`, in the command line box.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU `gprof` documentation found at www.gnu.org.



Samples - Function Total View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective. You can also open this view from the Window menu by pointing to **Show View**, clicking **Other...**, expanding **Profiling**, and double-clicking **Samples - Function Total**.

This view uses the **gmon.out** profiling data to show a breakdown of program execution by function executed. An entry for each function sampled during profiling appears in a table.

The table consists of the following columns for each table entry. Refer to GNU Profiler documentation for explanations of each of the columns.

- Function name
- Percent time
- Cumulative time
- Self time
- Number of times called
- Self time per call
- Total time per call

Clicking on the title of any column sorts the data by that column. Clicking a second time inverts the sort order.



Note: You can load the source code for functions displayed in this view into the Editor view. To do this, right-click the file name, and then click **Show Source**.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.



Samples - Line By Line View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective. You can also open this view from the Window menu by pointing to **Show View**, clicking **Other...**, expanding **Profiling**, and double-clicking **Samples - Line By Line**.

This view uses the **gmon.out** profiling data to show a breakdown of program execution by line of source code executed. An entry for each code line sampled during profiling appears in a table.

The table consists of the following columns for each table entry. Refer to GNU Profiler documentation for explanations of each of the columns.

- Function name
- Filename
- Line number
- Percent time
- Cumulative time
- Self time
- Number of times called
- Self time per call
- Total time per call

Clicking on the title of any column sorts the data by that column. Clicking a second time inverts the sort order.



Note: You can load the source code for functions displayed in this view into the Editor view. To do this, right-click the file name, and then click **Show Source**.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.

Project Properties Dialog Box




Properties Dialog Box

You open this dialog box by right-clicking a project in the C/C++ Projects view, and clicking **Properties**.

This dialog box lets you adjust properties settings for your project. The **Properties** dialog box groups similar settings together on pages, and the pages that appear depend on the project type. The following table describes which properties pages are available for each project type. Links to Nios II IDE specific topics appear in the table. Click a hyperlink under **Properties Page** for more information on a specific page. Properties pages without a hyperlink are native to the C/C++ Development Toolkit (CDT). Refer to the Related Topics for more information.

Properties Page	Project Type			
	C/C++ Application	System Library	Advanced C/C++	Managed Library
Info	o	o	o	o
Associated System Library	o			
Builders	o	o	o	o
C/C++ Build	o	o		o
C/C++ Documentation	o	o	o	o
C/C++ File Types	o	o	o	o
C/C++ Include Paths and Symbols			o	
C/C++ Indexer	o	o	o	o
C/C++ Make Project			o	
C/C++ Project Paths			o	
Project References	o	o	o	o
System Library		o		

 **Note:** For C/C++ application projects, the most important project settings generally relate to the application's interaction with the hardware. You specify these settings in the **System Library** page of the **Properties** dialog box for the system library project associated with your application.



Related Nios II IDE Help Topics

- [About Nios II IDE Projects](#) - Contains more information on the four project types.



Related Eclipse and CDT Help Topics

- [C/C++ Development User Guide > Reference > C/C++ Project Properties > Managed Make Projects > Info](#)
- [C/C++ Development User Guide > Reference > C/C++ Project Properties > Managed Make Projects > File Types](#)
- [C/C++ Development User Guide > Reference > C/C++ Project Properties > Standard Make Projects > C/C++ Include Paths and Symbols](#)
- [C/C++ Development User Guide > Reference > C/C++ Project Properties > Managed](#)

- Make Projects > Indexer
- C/C++ Development User Guide > Reference > C/C++ Project Properties > Standard Make Projects > C/C++ Make Project
 - C/C++ Development User Guide > Reference > C/C++ Project Properties > Standard Make Projects > C/C++ Project Paths



Associated System Library Page (Properties Dialog Box)

You open this properties page by clicking **Associated System Library** in the left-hand pane of the **Properties** dialog box for C/C++ application projects.

This page allows you to specify the system library for a C/C++ application project.

- **Target** - Displays the project type **Nios II Application**. (This value is read only.)
- **System Library** - Specifies the associated system library for this application project. Click **Browse** to select from the open system library projects.



Note: If you change the system library, you must perform a clean build on the application project.

- **System Library Properties...** - Opens the **System Library** properties page for the specified system library.



Related Nios II IDE Help Topics

- Properties Dialog Box



Builders Page (Properties Dialog Box)

You open this properties page by clicking **Builders** in the left-hand pane of the **Properties** dialog box for C/C++ application, system library, advanced C/C++, and managed library projects.

This page lists the builders the Nios II IDE uses to build a project. For C/C++ application projects, system library projects, and managed make projects, **Altera Generated Makefile Builder** is always present and must be selected. **Altera Generated Makefile Builder** is the tool that manages makefiles for Nios II system libraries. For advanced C/C++ projects, refer to Related Topics for more information about builders for Eclipse standard make projects.



Related Nios II IDE Help Topics

- Properties Dialog



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Reference > C/C++ Project Properties > Standard Make Projects > Builders



C/C++ Build Page (Properties Dialog Box)

You open this properties page by clicking **C/C++ Build** in the left-hand pane of the **Properties** dialog box for C/C++ application, system library, and managed library projects.

This page allows you to specify compilation, preprocessor, and linking options for the **nios2-elf-gcc** compiler and linker. The page is organized in the following sections.

▶ Active Configuration:

This section identifies the current configuration.

- **Project Type** - Displays the project type **Nios II Executable**. (This value is read only.)
- **Configuration** - Specifies the configuration to build. The Nios II IDE allows you to control groups of related compilation options as a single *build configuration*. By default, there are two build configurations: Release and Debug.
- **Manage...** - Allows you to add or delete build configurations.

▶ Configuration Settings:

This section describes the configuration settings tabs.

Tool Settings Tab


Clicking on an item on the left-hand side of the tab displays options for that item in the right half of the tab.

- **Nios II Compiler** - Displays flags configured within this category for the **nios2-elf-gcc** compiler command.
 - **Preprocessor** - Defines symbols for the **nios2-elf-gcc** preprocessor. Several advanced HAL system library options are specified as preprocessor options.
 - **Defined Symbols** - Specifying a symbol here is equivalent to a `#define` macro in source code, or supplying `-D` arguments on the command line. You can define symbols directly, for example `THING_TO_DEFINE`, or as equivalents, such as `PI=3.14159`.
 - **Undefined Symbols** - Specifying a symbol here is equivalent to a `#undef` macro in source code.
 - **General** - Configures known flags for the compiler. For a complete list of flags, type `nios2-elf-gcc -v --help` on a command line or refer to the GNU tools documentation included with the Nios II Embedded Design Suite (EDS).
 - **Compiler Flags** - Allows additional command line flags to pass to the compiler.
 - **Optimization Level** - Configures the compiler optimization level. Options are `-O0` to `-O3`. The compiler optimizes for both size and speed.
 - **Debug Level** - Configures the debug level.

- **Include Paths** - Lists where the compiler searches for C header files. The Nios II IDE creates a search path including the project directory, device driver directories, the Newlib standard C library, etc. Use **Include Paths** to specify additional paths to find files in other locations. To add an include path, click **New** and enter the path. The paths are searched in the order they appear from top to bottom. To modify the search order, click on a list entry then click **Move Up** and **Move Down** accordingly.
- **Enable All Warnings (-Wall)** - Enables additional warnings when compiling.
- **Linker** - Displays linker flags configured within this category for the **nios2-elf-gcc** compiler command.
 - **General** - Configures known flags for the linker. For additional information on linker flags, type `nios2-elf-ld --help` on a command line or see the Nios II documentation located at `<Nios II EDS install path>/documents/gnu-tools/binutils/ld.html`.


Error Parser Tab

This tab gives advanced users control over which types of build errors the Nios II IDE displays in the Problems view.

 **Note:** Altera strongly recommends leaving these settings at their default values.

Binary Parser Tab

This tab allows advanced users to choose which binary parser to use to read build files in the IDE.

 **Note:** Altera strongly recommends using the GNU Elf Parser.



Related Nios II IDE Help Topics

- Properties Dialog Box



C/C++ Documentation Page (Properties Dialog Box)

You open this properties page by clicking **C/C++ Documentation** in the left-hand pane of the **Properties** dialog box for C/C++ application, system library, advanced C/C++, and managed library projects.

This page lists available HTML-based C/C++ Help Books which provide context-sensitive help for C/C++ code.



Related Nios II IDE Help Topics

- Properties Dialog Box



Project References Page (Properties Dialog Box)

You open this properties page by clicking **Project References** in the left-hand pane of the **Properties** dialog box for C/C++ application, system library, advanced C/C++, and managed library projects.

This page allows you to specify project dependencies, i.e. other projects that must be built prior to building the application project. In general, if the current project references another project, then the other project must be built first.



Important: Do not use this page to link to a system library. The dependency between the C/C++ application project and the system library project is handled by the Associated System Library page.



Related Nios II IDE Help Topics

- Properties Dialog Box



System Library Page (Properties Dialog Box)

You open this properties page by clicking **System Library** in the left-hand pane of the **Properties** dialog box for system library projects.

This page allows you to configure compiler and linker options that affect how your program interacts with hardware, such as:

- Specifying a device for the stdin, stdout, and stderr channels
- Specifying which sections of code belong in which physical memory
- Adding and configuring software components, such as an RTOS or file system

This page contains groups of settings related to target hardware, system library contents, and linker script. The following sections describe each group in detail.

▶ **Target Hardware:**

This section displays the target **CPU** and **SOPC Builder System**. These values cannot be modified. You must make a new system library project if you wish to change these values.


▶ **System Library Contents:**

This bottom-left group of settings affects the contents included in the system library. These settings configure the behavior of the HAL system library and the system RTOS, if any.

- **RTOS** - Specifies the operating system to base the system on. The **MicroC/OS-II** option configures the system library to use the MicroC/OS-II RTOS. Other RTOS choices are available from other vendors. **none (single threaded)** configures the system library with a single-threaded HAL runtime environment. If an option other than **none (single threaded)** is selected, the **RTOS Options...** button is enabled.
- **RTOS Options** - Displays configuration options that are specific to the selected **RTOS**. Consult the RTOS vendor's documentation regarding these options.
- **stdout, stderr, stdin** - Allow you to associate the C stdout, stderr, and stdin streams to devices in the SOPC Builder system. You can choose any character mode device for each stream. Setting unused streams to **null** might reduce the memory footprint.
- **System clock timer** - Associates the system clock driver with a timer device in the SOPC Builder system.
- **Timestamp timer** - Associates the timestamp driver to a timer device in the SOPC Builder system. The **Timestamp timer** and the **Periodic system timer** cannot specify the same physical device.
- **Max file descriptors** - Specifies the maximum number of file descriptors that can be open simultaneously for accessing character mode devices and file subsystems. The default is 32. Using a smaller number might reduce the memory footprint.
- **Clean exit (flush buffers)** - When on, the system library calls `exit()` upon returning from `main()`, which flushes I/O buffers and then calls `_exit()`. When off, the system library calls only `_exit()`. This option increases memory footprint, and is often undesirable for embedded programs that never return from `main()`.

- **Small C library** - When on, the system library uses a reduced implementation of the Newlib ANSI C standard library. Notably, the `printf()` family of routines (`printf()`, `fprintf()`, `sprintf()`, etc.) will not support floating-point values. The reduced library is optimized for smaller memory footprint, although the implementation might be less time efficient.
- **ModelSim only, no hardware support** - Turning on this feature allows the program to take short-cuts for faster ModelSim simulation. For example, in hardware simulation, memories are pre-initialized with data, making it unnecessary to load RAM contents from non-volatile storage. This saves millions of simulated CPU cycles, and saves considerable time in hardware simulation iterations.

When off, code runs properly on hardware. Turning off this feature also allows you to simulate true hardware behavior, such as booting from flash.


 **Important:** Only turn on this option if you are simulating a Nios II system on the ModelSim RTL simulator. When on, your program will not run in hardware.

- **Run time stack checking** - This option is only available when using the HAL runtime environment. When on, the compiler inserts a test for stack overflow upon function entry and when allocating memory on the stack. If an allocation results in the stack pointer exceeding the stack limit then a break instruction executes. Enabling this option is equivalent to supplying the `-mstack-check` option to `nios2-elf-gcc` on the command line.
- **Reduced device drivers** - When on, the compiler includes the reduced version of device drivers for all devices that provide small drivers. This reduces memory footprint at the expense of functionality. See documentation associated with each peripheral for more information.
- **Link with profiling library** - When on, the build system links in a profiling library to collect information about which functions call other functions and the amount of time spent in each function. Refer to Related Topics for more information about profiling.
- **Emulate multiply and divide instructions** - When on, the linker includes an exception handler to emulate multiply and divide instructions on Nios II systems that do not include hardware multiply or divide hardware. This option increases memory footprint and might decrease system performance.
- **Software Components...** - Opens a page listing available software components that can be built into the software library. See the following documentation for information on these Altera-provided software components:
 - Host Based File System
 - Lightweight TCP/IP Stack
 - Zip Read-Only File System

▶ Linker Script:

These settings configure the linker script used when building the project.

- **Custom linker script** - When selected, you create and manage your own linker script, and specify it here.

 **Note:** Altera strongly recommends that you use the auto-generated linker script.

- **Auto-generated linker script** - When selected, the Nios II IDE automatically creates and manages a linker script that is sufficient for most system compilation needs. When using this option, you must specify the following:
 - **Program memory (.text)** - Specifies where executable code resides in physical memory.
 - **Read-only data memory (.rodata)** - Specifies where read-only data resides in physical memory.
 - **Read/write data memory (.rwdata)** - Specifies where read/write data resides in physical memory.
 - **Heap memory** - Specifies where the heap resides in physical memory.
 - **Stack memory** - Specifies where the stack resides in physical memory.
 - **Use a separate exception stack** - When on, the exception stack resides in separate physical memory. Placing the exception stack in a fast memory improves the performance of exception handling.
 - **Exception stack memory** - Specifies where the exception stack resides in physical memory, when **Use a separate exception stack** is on.
 - **Maximum exception stack size (bytes)** - Specifies the maximum size of the exception stack, when **Use a separate exception stack** is on.

You can also use the `__attribute__` declaration in C code to specify which memory to use for a specific block of code. The Nios II IDE creates a corresponding memory section for each memory device defined in the SOPC Builder system (see **system.h**). For example, a memory device named "on_chip_memory" has an associated memory section named ".on_chip_memory". The follow examples demonstrate how to force a function or a variable to reside in a specific memory.

```
/* data using the section attribute should be initialized */
int foo_var __attribute__ ((section (".on_chip_memory"))) = 0;
void bar_func __attribute__ ((section (".sdram"))) (void)
{
    foo++;
}
```



Related Nios II IDE Help Topics

- Properties Dialog Box
- About Profiling with the Nios II IDE



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains information on the HAL system library and the MicroC/OS-II RTOS.
- Using MicroC/OS-II RTOS with the Nios II Processor Tutorial at www.altera.com/literature/tt/tt_nios2_MicroC_OSII_tutorial.pdf - Contains step-by-step instructions on creating MicroC/OS-II applications.



RTOS Options Dialog Box (System Library Properties Page)

You open this dialog box by clicking **RTOS Options...** on the **System Library** properties page. The **RTOS Options...** button is only enabled when something other than **none (single threaded)** is selected in the **RTOS** drop-down list.

The **RTOS Options** dialog box lets you adjust settings for the RTOS specified in the **RTOS** drop-down list on the **System Library** properties page.

The **RTOS Options** dialog box contains the following controls.

- **RTOS options list** - Displays pages of options for the RTOS in the left-hand pane of the **RTOS Options** dialog box.
- **Restore Defaults** - Discards your changes, and returns the dialog box to its previous settings.
- **Apply** - Saves your settings.
- **OK** - Closes the dialog box, saving any settings you changed.
- **Cancel** - Closes the dialog box, ignoring any settings you changed.
- **Help** - Displays help information.

When you select a page in the RTOS options list, settings specific to that page appear in the right side of the dialog box.



Related Nios II IDE Help Topics

- Choosing and Configuring an Operating System
- MicroC/OS-II RTOS
- System Library Page (Properties Dialog Box)



Related Topics on the Web

- Nios II Software Developer's Handbook at www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf - Contains details on writing Nios II programs based on MicroC/OS-II.
- Using MicroC/OS-II RTOS with the Nios II Processor Tutorial at www.altera.com/literature/tt/tt_nios2_MicroC_OSII_tutorial.pdf - Contains step-by-step instructions on creating MicroC/OS-II applications.



Software Components Dialog Box (System Library Properties Page)

You open this dialog box by clicking **Software Components...** on the **System Library** properties page.

The **Software Components** dialog box lets you add middleware software components to your system library.

The **Software Components** dialog box contains the following controls.

- **Software components list** - Displays the available software components and allows you to select a component.
- **Add this software component** - Indicates that you want to add the software component to your system library. Turning on this setting enables all the other settings specific to the selected software component.
- **Restore Defaults** - Discards your changes, and returns the dialog box to its previous settings.
- **Apply** - Saves your settings.
- **OK** - Closes the dialog box, saving any settings you changed.
- **Cancel** - Closes the dialog box, ignoring any settings you changed.
- **Help** - Displays help information.

When you select an page in the software components list, settings specific to that component appear in the right side of the dialog box.



Related Nios II IDE Help Topics

- Choosing and Configuring Middleware Software Components
- Host-Based File System
- Lightweight TCP/IP Stack
- Zip Read-Only File System
- System Library Page (Properties Dialog Box)

Run/Debug Dialog Box



Run/Debug Dialog Box

You open this dialog box by clicking **Run...** or **Debug...** on the Run menu.

The **Run** dialog box allows you to configure and start run sessions, and the **Debug** dialog box allows you to configure and start debug sessions. The **Run** and **Debug** dialog boxes are distinct entities, but their controls are nearly identical. This help topic describes both dialog boxes.

The **Run** and **Debug** dialog boxes allow you to manage run/debug configurations. A run/debug configuration is a group of settings that specify the target type and which project to run or debug. The available target types are: Nios II hardware, Nios II instruction set simulator (ISS), Nios II ModelSim, and Nios II multiprocessor collection.

The following sections describe the controls and tabs available on this dialog box.

▶ **Run and Debug Controls:**

The **Run** and **Debug** dialog boxes contains the following controls:


- **Configurations** - Is a list of available run/debug configurations, grouped by target type.
- **New** - Creates a new run/debug configuration. You must first select a target type in the **Configurations** list before clicking **New**.
- **Delete** - Deletes the selected configuration.
- **Name** - Allows you to enter a unique name for a configuration. **Name** appears only after you select an existing configuration.
- **Apply** - Saves your settings for the selected configuration.
- **Revert** - Discards changes for the selected configuration, and returns the configuration to its previously-saved state.
- **Run** or **Debug** - Starts a run or debug session with the selected configuration. The **Run** button appears only on the **Run** dialog box, and the **Debug** button appears only on the **Debug** dialog box.
- **Close** - Closes the dialog box.

▶ **Run and Debug Configuration Tabs:**

If you select a specific run/debug configuration in the **Configurations** list, tabs appear on the right side of the dialog box. Each tab presents a group of settings related to the selected configuration. The tabs that appear depend on the target type. The following table lists which tabs appear for which target type. Click a hyperlink under **Tab Name** for more information on a specific tab.

Tab Name	Target Type			
	Nios II Hardware	Nios II ISS	ModelSim Simulator	Multiprocessor Collection
Main	o	o		o
Target Connection	o			

ISS Settings		<input type="radio"/>	
Launch			<input type="radio"/>
ModelSim			<input type="radio"/>
Debugger	<input type="radio"/>	<input type="radio"/>	
Source	<input type="radio"/>	<input type="radio"/>	
Common	<input type="radio"/>	<input type="radio"/>	

 **Note:** The **Launch ModelSim** tab is not available on the **Debug** dialog box.

Perspectives Tab:

If you select a target type in the **Configurations** list, the **Perspectives** tab appears. The Perspectives tab allows you to associate a workbench perspective with run and debug sessions for each target type. When you launch a run/debug configuration, the IDE automatically switches to the perspective specified here.

This tab has the following settings:

- **Debug** - Identifies the perspective to use when you start a run session.
- **Run** - Identifies the perspective to use when you start a debug session.



Related Nios II IDE Help Topics

- About Running and Debugging Projects
- Main Tab (Run/Debug Dialog)
- Target Connection Tab (Run/Debug Dialog)
- ISS Settings Tab (Run/Debug Dialog)
- Launch ModelSim Tab (Run/Debug Dialog)
- Debugger Tab (Run/Debug Dialog)
- Source Tab (Run/Debug Dialog)
- Common Tab (Run/Debug Dialog)



Common Tab (Run/Debug Dialog Box)

You open this tab by clicking the **Common** tab on the **Run/Debug** dialog box for Nios II Hardware and Nios II Instruction Set Simulator (ISS) configurations.

The **Common** tab contains the following settings.

- **Type of launch configuration** - Specifies where to store the run/debug configuration. The following options are available:
 - **Local** - Saves the run/debug configuration under the workspace directory.
 - **Shared** - Saves the run/debug configuration within a project in your workspace, which allows you to commit it to CVS and share it with other users.
 - **Location of shared configuration** - Specifies where to save the shared configuration.
- **Display in favorites menu** - Specifies which run/debug configurations to list in the special "favorites" section at the top of the **Run, Run History** and **Run, Debug History** menus.
 - **Run** - When on, the run/debug configuration appears in the top section of the **Run, Run History** menu.
 - **Debug** - When on, the run/debug configuration appears in the top section of the **Run, Debug History** menu.
- **Launch in background** - Specifies whether the run/debug configuration executes in the background. When on, you can continue working in the Nios II IDE while your project is building.



Related Nios II IDE Help Topics

- [Run/Debug Dialog Box](#)



Debugger Tab (Run/Debug Dialog Box)

You open this tab by clicking the **Debugger** tab on the **Run/Debug** dialog box for Nios II Hardware and Nios II Instruction Set Simulator (ISS) configurations.


The **Debugger** tab contains settings that affect the IDE behavior during debug sessions. Some settings also apply to run sessions. Most settings on this tab are grouped by category.

- **Debugger** - Allows you to specify the application to use for debugging. The only choice is the **Nios II Elf Debugger**.

▶ Download and Reset:

These settings determine if the IDE downloads code to the target and resets the target. These settings apply to both run and debug sessions.

- **Download Program to RAM** - Downloads the executable software file (.elf) to RAM at the start of a debug session.
- **Attach to existing program on target (no download)** - Attaches the debugger and console to a process that is already running in hardware.
- **Reset target and execute from reset vector (no download)** - Allows you to reset the target, attach the debugger and console to the process already executing in hardware, and pause the debug session at the reset vector.

 **Important:** This option is explicitly for debugging boot code from flash.

▶ Breakpoints at Start-up:

These settings determine where the debugger suspends at the start of a debug session. If no checkboxes are on, the debug session executes until it encounters a user-inserted breakpoint.

- **Break at main()** - Causes the debugger to break at `main()`. This is equivalent to inserting a breakpoint at the first instruction of `main()`.
- **Break at alt_main()** - Causes the debugger to break at `alt_main()`. This is equivalent to inserting a breakpoint at the first instruction of `alt_main()`. Use this setting when debugging freestanding C/C++ application projects.
- **Break at program entry point** - Causes the debugger to break at the program's entry point. This is equivalent to inserting a breakpoint at the program's entry point, typically `_start()`.

▶ Advanced:

These settings provide access to features that are not standard in the Nios II IDE debug flow.

- **Use FS2 console window for trace and watchpoint support** - Turning this option on launches the FS2 console (provided by First Silicon Solutions, Inc.) at the start of debug sessions. This option is available only on Windows and is always disabled on Linux.

- **Use the Altera generated initialization script for faster debug performance**
 - Turning this option off allows you to specify a custom GDB initialization script to run at the start of debug sessions. This setting has little impact on Nios II instruction set simulator targets. If you turn this setting off, you must also specify the **GDB command file**.



Note: Altera does not recommend turning off this setting for Nios II hardware targets, because doing so slows down debugging.

- **GDB command file** - Specifies the GDB initialization script file to use when the **Use the Altera generated initialization script for faster debug performance** setting is off.



Note: If you provide your own script, the commands can interfere with the normal startup operation of the debugger. For example, do not use the "run" command in your script.



Related Nios II IDE Help Topics

- Run/Debug Dialog Box
- About Running and Debugging Projects



Related Topics on the Web

- First Silicon Solutions, Inc. at www.fs2.com



ISS Settings Tab (Run/Debug Dialog Box)

You open this tab by clicking the **ISS Settings** tab on the **Run/Debug** dialog box for Nios II Instruction Set Simulator configurations.

The **ISS Settings** tab allows you to configure settings specific to the Nios II instruction set simulator (ISS). Most settings on this tab are grouped by category.

▶ Trace Options:

The following options control how much trace information the ISS reports.

- **Enable Tracing** - Enables tracing the instruction-by-instruction execution of the CPU in simulation. The trace information output depends on the remaining trace options.
- **Information** - Outputs information about the system and debugging process.
- **Disassembly** - Outputs the program counter, assembly instructions, and values of affected registers.
- **Registers** - Outputs the CPU registers. Registers include the program counter (PC), r0, at(r1), r2-r23, et, bt, gp, sp, fp, ea, ba, ra, status, estatus, bstatus, ienable, and ipending.
- **Warning** - Outputs ISS warnings encountered while simulating.
- **Instruction Count** - Outputs the instruction number of the currently executing instruction. This option only has an effect if the **Disassembly** or **Registers** options are on.
- **Send trace output to file** - When on, the ISS writes trace output to the specified file.
 - **Trace File** - Specifies where to save trace data. This uses the application project directory if you do not specify a different location.
- **Start tracing from** - Specifies the trace entry-point function. The choices are: `main()`, `alt_main()` or the program entry point. This setting defaults to `main()`, which allows you to ignore any pre-`main()` activity. Note that from the program entry point to the start of `main()` or `alt_main()`, as many as several hundred thousand instructions occur.

▶ Host Communications Devices:

The following options control where the ISS directs the `stdio`, `stdout`, and `stderr` character streams.

- **Use default host communication devices** - When on, the ISS uses the communication devices specified in the system library project. You can turn this setting off to manually select the host communication device for `stdio`, `stdout`, and `stderr`. The ISS settings for `stdio`, `stdout`, and `stderr` override the system library project settings.
 - **stdout** - Specifies the communication device for `stdout`.
 - **stderr** - Specifies the communication device for `stderr`.
 - **stdin** - Specifies the communication device for `stdin`.

▶ Memory Dump:

The following options allow you to save the contents of memory to a file after running a program on the ISS.

- **Enable memory dump** - When on, memory contents are dumped to a file after program execution terminates. The resulting file name is **<processor name>_memdump.out**, located in the application project directory.
 - **Start Address** - Specifies the starting address of the memory range to dump.
 - **End Address** - Specifies the ending address of the memory range to dump.

▶ Other Settings:

The following are options not associated with any other group of options:

- **Summarize system components** - When on, the ISS reports details at startup about the SOPC Builder system components connected to the processor. This report can be a useful reminder of what components are present in the system, and which components will not simulate accurately.
- **Additional nios2-iss arguments** - Specifies the command line arguments to pass to **nios2-iss** when launching. This option is for advanced users only.



Note: For details on **nios2-iss** command line arguments, type `nios2-iss --help` from the Nios II command shell. Specify the arguments as if running the ISS in a Nios II command shell. For example, to launch the terminal window as a separate window, type `-w` in the **Additional nios2-iss arguments** box.

- **Uninitialized memory reads** - Specifies the ISS behavior when a read from uninitialized memory occurs. The following options are available.
 - **Generate Error** - Stops execution and generates an error.
 - **Generate Warning** - Generates a warning message, and continues execution. This option is useful if you know that the data read will not be used by the processor.
 - **Ignore** - Continues execution without generating any warning.



Related Nios II IDE Help Topics

- Run/Debug Dialog Box
- Instruction Set Simulator (ISS)
- Running and Debugging on the ISS



Launch ModelSim Tab (Run Dialog Box)

You open this tab by clicking the **Launch ModelSim** tab on the **Run** dialog box for Nios II ModelSim configurations.

The **Launch ModelSim** tab contains general settings about the application project, the executable code (**.elf**), and additional settings required for hardware simulation with ModelSim.

The **Launch ModelSim** tab contains the following settings.

- **Project** - Specifies the source application project. The Nios II IDE associates each run configuration with exactly one application project.
- **Target Hardware** - Displays the target **SOPC Builder System** and **CPU** for the selected project. These values are read-only for C/C++ Application projects and changeable for Advanced C/C++ projects.
- **ModelSim Path** - Displays the location of the ModelSim executable. This display is read-only. You set the ModelSim path in SOPC Builder by specifying the **ModelSim Directory** in the **SOPC Builder Setup** dialog box found on the SOPC Builder File menu.



Related Nios II IDE Help Topics

- Run/Debug Dialog Box
- Running on the ModelSim Simulator



Related Topics on the Web

- AN351: Simulating Nios II Embedded Processor Designs at www.altera.com/literature/an/an351.pdf



Main Tab (Run/Debug Dialog Box)

You open this tab by clicking the **Main** tab on the **Run/Debug** dialog box for Nios II Hardware, Nios II Instruction Set Simulator (ISS), and Nios II Multiprocessor Collection configurations.

The options available on the **Main** tab depend on the configuration type. The following sections describe the options for each configuration type.

▶ Nios II Hardware and Nios II ISS Configurations:

For Nios II Hardware configurations and Nios II ISS configurations, the **Main** tab contains the following settings.

- **Project** - Specifies the source application project. The Nios II IDE associates each run/debug configuration with exactly one application project.
- **Nios II ELF Executable** - Specifies which executable file (**.elf**) to download to the target. This setting is read-only for C/C++ Application projects and changeable for Advanced C/C++ projects.
- **Target Hardware** - Displays the target **SOPC Builder System** and **CPU** for the selected project. These values are read-only for C/C++ Application projects and changeable for Advanced C/C++ projects.
- **Validate Nios II system ID before software download** - This option is only enabled if there is a system ID component in your SOPC Builder system. When on, the Nios II IDE verifies the system ID in the target before downloading code. If the ID expected by your executable does not match the actual hardware ID, a mismatch error appears in the Console view.



Note: **Validate Nios II System ID before software download** is not available for Nios II ISS configurations.

▶ Nios II Multiprocessor Collection Configurations:

For Nios II Multiprocessor Collection configurations, the **Main** tab contains the following settings.

- **Select Nios II Hardware configurations to run concurrently** - Specifies multiple run/debug configurations to launch simultaneously.



Related Nios II IDE Help Topics

- Run/Debug Dialog Box
- Hardware Target
- Instruction Set Simulator (ISS)
- Hardware Simulation with ModelSim
- Multiprocessor Nios II Systems



Source Tab (Run/Debug Dialog Box)

You open this tab by clicking the **Source** tab on the **Run/Debug** dialog box for Nios II Hardware and Nios II Instruction Set Simulator (ISS) configurations.

The **Source** tab specifies where the debugger searches for source files associated with an executable file. You can specify Nios II IDE project folders and other folders.



Note: If you do not have the source code for a file, you can step through the assembly instructions in the Disassembly view.



Related Nios II IDE Help Topics

- Run/Debug Dialog Box



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Reference > C/C++ Run and Debug > Source - Contains details on the Source tab.



Target Connection Tab (Run/Debug Dialog Box)

You open this tab by clicking the **Target Connection** tab on the **Run/Debug** dialog box for Nios II Hardware configurations.

The **Target Connection** tab specifies how the IDE communicates with the target hardware.

The **Target Connection** tab contains the following settings.

- **JTAG cable** - Specifies the Altera download cable (such as the USB Blaster) to use for communicating with the Nios II processor. If you have only one cable, the **automatic** value displays your cable. If more than one cable is connected, you must select a specific cable.
 - **Refresh** - Allows you to update the **JTAG cable** list if you change your JTAG cables while the **Run** or **Debug** dialog box is open.
- **JTAG device** - Specifies the device in the JTAG chain that contains the Nios II CPU. If there are multiple devices containing Nios II systems in the JTAG chain, you must select a specific device.
 - **Refresh** - Allows you to update the **JTAG device** list if you change your JTAG cables while the **Run** or **Debug** dialog box is open.
- **Nios II Terminal communication device** - Specifies the component (such as a JTAG UART) in the SOPC Builder system to use for terminal communication. When the selected device is a UART device, you must also specify which **Host COM port** to use.
- **Host COM port** - Specifies which host COM port to use when the selected **Nios II terminal communication device** is a UART device.



Related Nios II IDE Help Topics

- Run/Debug Dialog Box
- Hardware Target

Views



C/C++ Projects View (C/C++ Perspective)

This view displays as part of the C/C++ perspective by default. You can also open this view from the C/C++ perspective manually by pointing to **Show View** on the Window menu, and then clicking **C/C++ Projects**.

The C/C++ Projects view presents project resources in the context of C/C++ project development, providing visual cues and hierarchy to organize the various files in your projects. Icons in the C/C++ Projects view are not necessarily associated with files in the file system (i.e. on your hard drive). For example, object file resources display the names of functions defined in the file.

The resource selected in the C/C++ Projects view affects the information displayed in other views. Refer to Related Topics for more information about the C/C++ Projects view.


Right-clicking on a project or project resource displays a context-sensitive menu of available commands for the selected project or project resource. Most tasks related to developing Nios II programs are available from this context-sensitive menu. The sections below define the commands available on the context-sensitive menu for projects.

▶ **New:**

The **New** submenu allows you to create a new resource (file, folder, or project). **New** is also available in the File menu. The following choices are available.


- **Project** - Opens the **New Project** wizard, which allows you to create a new project of any kind.
- **Folder** - Opens the **New Folder** wizard to create a new folder.
- **File** - Opens the **New File** wizard to create a new file.
- **Header File** - Opens the **New Header File** wizard to create a new header file.
- **C/C++ Application** - Opens directly to the **New Project** wizard for creating Nios II C/C++ application projects. This is the recommended way to start a new Nios II user application.
- **Other...** - Opens the **New** wizard to allow you to create the resource of your choice.

▶ **Go Into:**

This command limits the C/C++ Projects view display to just the selected project. To return to displaying all projects, click  **Back** in the C/C++ Projects view toolbar.


▶ **Open in New Window:**

This command opens a new Nios II IDE window.

 **Important:** Altera does not recommend working on the same project simultaneously in separate windows. Values set in one window might not propagate to the other window, and undefined results could occur.


▶ Run As:

The **Run As** submenu allows you to run the selected project on a Nios II hardware, Nios II instruction set simulator (ISS), or Nios II ModelSim target. **Run As** works as an easy "build and run" shortcut. When **Build (if required) before launching** is on (on the Window menu's **Preferences, Run/Debug, Launching** page), **Run As** executes the makefile for the selected project, rebuilding any source files or project dependencies which have changed, before launching the software executable. The **Run As** submenu is also available from the Run menu.

 **Note:** If you have more than one programming cable connected to your system, and have not created a run/debug configuration for your project, choose **Run...** from the Run menu (instead of **Run As**) to first create a run/debug configuration.

▶ Debug As:

The **Debug As** submenu allows you to debug the selected project on a Nios II hardware or Nios II instruction set simulator (ISS) target. **Debug As** works as an easy "build and debug" shortcut. When **Build (if required) before launching** is on (on the Window menu's **Preferences, Run/Debug, Launching** page), **Debug As** executes the makefile for the selected project, rebuilding any source files or project dependencies which have changed, before launching the software executable in the debugger. The **Debug As** submenu is also available from the Run menu.


 **Note:** If you have more than one programming cable connected to your system, and have not created a run/debug configuration for your project, choose **Debug...** from the Run menu (instead of **Debug As**) to first create a run/debug configuration.

▶ Build Project:

This command builds the selected project (i.e. runs **make**). You can modify build settings on the **C/C++ Build** page of the **Properties** dialog box. **Build Project** is also available from the Project menu.

▶ Copy & Paste:

These commands allow you to copy and paste file and folder resources. **Copy** and **Paste** are also available from the Edit menu.


 **Important:** Do not paste entire projects. Projects created by pasting might fail to build.

▶ Delete:

This command deletes the currently selected resource and removes it from the hard drive. If the resource is a project, you are given a choice to delete or not delete the project's contents. **Delete** is also available from the Edit menu.

▶ Rename & Move...:

These commands allow you to rename and move resources. Renaming or moving a resource might have repercussions on other projects that reference the resource. **Rename** and **Move...** are also available from the File menu.


 **Important:** If you change the name of a system library project, application projects that depend on that system library will fail to build.

▶ **Import:**

The **Import...** command opens the **Import** wizard, which allows you to import an existing Nios II IDE project into the workbench. **Import...** is also available from the File menu.

▶ **Export:**



The **Export...** command opens the **Export** wizard, which is part of the standard Eclipse IDE framework.

 **Note:** Do not use this wizard to export files or projects. There are easier methods for importing, exporting and sharing projects and files.

▶ **Refresh:**

This command refreshes folders in the C/C++ Projects view to reflect the contents of the file system (i.e. the hard drive). This is sometimes necessary to force the Nios II IDE to recognize project files you created or changed outside of the Nios II IDE user interface. **Refresh** is also available from the File menu.

▶ **Close Project:**

You can close projects on the Nios II IDE workbench. The project folder icon changes to , indicating the project is currently closed. Resources of closed projects do not appear in and are not changeable from the workbench, but they do still reside on the local file system. Closed projects require less memory. Closing projects can improve build time, because they are not examined during builds. You can filter closed projects from the C/C++ Projects view by choosing  **View** from the C/C++ Projects view toolbar, clicking **Filters...**, turning on **Closed Projects**, and clicking **OK**. **Close Project** is also available from the Project menu.

▶ **Open Project:**

When a project is closed on the Nios II IDE workbench, **Close Project** changes to **Open Project** on this context-sensitive menu. The **Open Project** command reinstates the project to open status. **Open Project** is also available from the Project menu.

▶ **Team:**

The Eclipse IDE framework supports team-based development using CVS. The **Team** submenu provides access to the Eclipse team development features. Refer to Related Topics for more information about CVS.

▶ **Compare with, Replace with & Restore from Local History...:**

The Eclipse IDE framework provides advanced tools to track history and compare changes made to source files. The **Compare With** and **Replace With** submenus provide access to these features. Refer to Related Topics for more information about local history.

▶ **Properties:**

This command opens the **Properties** dialog box for the selected project.

▶ **System Library Properties:**

This command only appears for C/C++ application projects. It opens the **Properties** dialog box for the system library project associated with the C/C++ application project and displays the **System Library** page.



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Reference > C/C++ Views and Editors > C/C++ Projects view
- Workbench User Guide > Concepts > Team Programming with CVS
- Workbench User Guide > Tasks > Working with local history



Call Hierarchy View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective. You can also open this view from the Window menu by pointing to **Show View**, clicking **Other...**, expanding **Profiling**, and double-clicking **Call Hierarchy**.

The Call Hierarchy view displays the time spent in each function, based on the standard **gmon.out** data, in an easy-to-read tree format. This view also calculates and displays the percentage of time spent in each function. In this view you can follow the function call sequences much more easily than reading the gprof output.

There are two ways to view the call hierarchy data.

▶ Top down:

Top down call direction lists the calling functions, with the functions they called nested below.

The first entry in the Call Hierarchy view is **spontaneous**. This is a term gprof uses when it cannot determine what the calling function is. There are two sets of actual time and percentage time figures for each function. The first set represents the time spent within the function. The second set represents the total time spent within the function plus all functions called by that function. Each indented line in the view drills down into the details of time and percentage spent in each called function. In the top-down mode, when cycles of recursion exist in your code, a --> symbol appears to the left of the function name indicating that the called function cycles.

▶ Inverted:

Inverted call direction lists the called functions, with the functions that called them nested below.

The first entry in the Call Hierarchy view is **inverted calls**. Actual time and percentage time figures for each function represent time spent within the function.

To reverse call direction display, right-click in the Call Hierarchy view, and click **Toggle Call Direction**.



Note: You can load the source code for functions displayed in this view into the Editor view. To do this, right-click the file name, and then click **Show Source**.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.



Disassembly View (Debug Perspective)

This view is available in the Debug perspective, and automatically displays when execution suspends at a breakpoint. You can also open this view from the Debug perspective manually by pointing to **Show View** on the Window menu, and then clicking **Disassembly**.

The Disassembly view, also known as "mixed source" view, displays assembly instructions interspersed with the associated C/C++ source code. The Disassembly view allows you to analyze the exact assembly output of the compiler.

Whenever processor execution suspends during a debug session, the Disassembly view displays mixed C/C++ and assembly source code when available. When the processor suspends on an instruction with no debugging information available, the Disassembly view only shows the assembly instructions, omitting the interleaved C/C++ source code. The Disassembly view highlights the next instruction to execute with an arrow to the left of the instruction.

The Disassembly view displays the following information:

- Lines of C/C++ source code followed by their assembly translation
- The absolute address and the relative address within the function of each assembly instruction
- The assembly instructions with their source and destination operands



Related Nios II IDE Help Topics

- Viewing Disassembly



Related Topics on the Web

- Nios II Processor Reference Handbook at www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf - Contains details on the Nios II instruction set.



Editor View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective.

When you open the **gmon.out** file with the Nios II IDE, the IDE automatically calls `gprof` and displays the standard `gprof` text output. This Editor view is for viewing only.



Note: Saving the file with the default **gmon.out** filename overwrites the original **gmon.out** data. If you save the output as a text file, be sure to rename the saved file.

You can change the `gprof` command line arguments and rerun the **nios2-elf-gprof** utility. To do so, right-click anywhere in the Editor view and click **Change gprof arguments**. Type comma separated values, such as `--help`, in the command line box.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU `gprof` documentation found at www.gnu.org.



Samples - Function Total View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective. You can also open this view from the Window menu by pointing to **Show View**, clicking **Other...**, expanding **Profiling**, and double-clicking **Samples - Function Total**.

This view uses the **gmon.out** profiling data to show a breakdown of program execution by function executed. An entry for each function sampled during profiling appears in a table.

The table consists of the following columns for each table entry. Refer to GNU Profiler documentation for explanations of each of the columns.

- Function name
- Percent time
- Cumulative time
- Self time
- Number of times called
- Self time per call
- Total time per call

Clicking on the title of any column sorts the data by that column. Clicking a second time inverts the sort order.



Note: You can load the source code for functions displayed in this view into the Editor view. To do this, right-click the file name, and then click **Show Source**.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.



Samples - Line By Line View (Profiling Perspective)

This view automatically displays as part of the Profiling perspective. You can also open this view from the Window menu by pointing to **Show View**, clicking **Other...**, expanding **Profiling**, and double-clicking **Samples - Line By Line**.

This view uses the **gmon.out** profiling data to show a breakdown of program execution by line of source code executed. An entry for each code line sampled during profiling appears in a table.

The table consists of the following columns for each table entry. Refer to GNU Profiler documentation for explanations of each of the columns.

- Function name
- Filename
- Line number
- Percent time
- Cumulative time
- Self time
- Number of times called
- Self time per call
- Total time per call

Clicking on the title of any column sorts the data by that column. Clicking a second time inverts the sort order.



Note: You can load the source code for functions displayed in this view into the Editor view. To do this, right-click the file name, and then click **Show Source**.



Related Nios II IDE Help Topics

- About Profiling with the Nios II IDE
- Profiling Perspective



Related Topics on the Web

- AN 391: Profiling Nios II Systems at www.altera.com/literature/an/an391.pdf
- The GNU Profiler - GNU gprof documentation found at www.gnu.org.



Trace View (Debug Perspective)

You open this view from the Debug perspective by pointing to **Show View** on the Window menu, and then clicking **Trace**.

The Nios II IDE Trace view displays a disassembly trace of the instructions executed prior to the current breakpoint or watchpoint, and optionally displays load/store addresses and the associated data. The Trace view allows you to analyze execution details that are not available in the source disassembly listing, such as branches taken at runtime, and precisely when exceptions occurred.

The Trace view is only available in the Debug perspective, and does not automatically display as part of the Debug perspective. The Trace view only functions when debugging Nios II hardware targets. If the hardware does not support trace, then the Trace view does not function. The JTAG debug module on the target processor must be configured to support trace collection.

During a debug session, the Trace view automatically displays trace data whenever processor execution suspends. The most recently executed instruction displays at the bottom of the view.

The Trace view displays the following information:

- Address of the instruction in hex
- C/C++ function containing the suspended instruction, plus a (hex) offset
- The instruction word in hex
- Disassembly of the instruction word
- Load and/or store addresses, and associated data (depending on trace preferences)

Ellipses in the disassembly indicate that the JTAG debug module might have dropped some instructions while collecting trace data. The trace might not be reliable at that point. The following disassembly is an example.

```
0x01000068 <foo+0x24>: 0xdec00204 addi sp, sp, 8
...
0x0100006c <foo+0x28>: 0xf800283a ret
```



Related Nios II IDE Help Topics

- Viewing Execution Trace
- Trace Page (Preferences Dialog Box)



Related Topics on the Web

- Nios II Processor Reference Handbook at www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf - Contains details on the JTAG debug module on the Nios II processor.



Workspace Launcher Dialog Box

This dialog box automatically displays when you start the Nios II IDE. You can also open this dialog box by clicking **Switch Workspace...** on the File menu.

The **Workspace Launcher** dialog box lets you select the Nios II IDE workspace to use for the current IDE session. The Nios II IDE stores your projects in a directory called a workspace. The workspace is the default area for Nios II IDE projects and other IDE-related settings files.

Each user can define one or more workspaces to keep their environments separated as desired. Additionally, workspaces allow more than one instance of the Nios II IDE to run simultaneously, with each instance pointing to a different workspace. If an instance of the Nios II IDE is already running when you launch another, a warning appears instructing you to choose a different workspace than the one currently in use.

Storing projects in the workspace directory allows you to upgrade versions of the IDE while keeping the same projects in your workspace.

The **Workspace Launcher** dialog box contains the following controls.

- **Workspace** - Is a list of the previously-identified workspaces.
- **Browse...** - Allows you to navigate the file system and select a workspace directory.
- **Use this as the default and do not ask again** - Signals the IDE to use the selected workspace on future launches.
- **OK** - Sets the workspace to use and closes the **Workspace Launcher** dialog box.
- **Cancel** - Closes the **Workspace Launcher** dialog box without changing the workspace. When displayed as part of system startup, cancelling also exits the Nios II IDE.



Related Nios II IDE Help Topics

- [New Projects Page \(Preferences Dialog\)](#)



Troubleshooting

This topic contains troubleshooting tips for questions and issues that might arise due to non-intuitive behavior of the Nios II IDE.

Q: How do I get my C/C++ or Debug perspectives to display correctly after upgrading to the latest version?

A: Switch to the affected perspective, and then click **Reset Perspective** on the Window menu.

Q: Sometimes I get the message "An error has occurred. See error log for more details." Where is this error log located?

A: The error log is located in your workspace directory in the `.metadata` subdirectory in the `.log` file. You can view the error log from the Nios II IDE. Click **About Nios II IDE** on the Help menu, then click **Configuration Details**, then click **View Error Log**.

Q: Why do I get "Error creating project. Reason: Invalid project description" in the New Project wizard when I specify a custom project location and click Finish?

A: Nios II IDE project directories can contain only one project, and cannot be nested inside other project directories. For example, if you have `project_1` in directory `/software/project_1`, then you cannot create a `project_2` in directory `/software/project_1/project_2`. If you want to specify a particular directory, but a project already exists there, you must first delete the old project from the IDE workspace. Right-click the old project in the C/C++ Projects view, and then click **Delete**.

When **Use Default Location** is on, the **New Project** wizard always creates a new, valid project directory.

Q: Why doesn't Make Targets view appear in the C/C++ perspective?

A: Since Make Targets view only pertains to advanced projects, the C/C++ and Debug perspectives do not automatically display the view by default. To display Make Targets view, point to **Show View** on the Window menu, click **Other...**, expand **Make**, and then double-click **Make Targets**.

Q: Why does the source code not display when stepping in the debugger?

A: If you compile a project without the `-g` compiler option, the IDE cannot display source code during debug sessions. To add debug information to your project, right-click your project and then click **Properties**. Click **C/C++ Build**. On the **Tool Settings** tab, expand **Nios II Compiler** and click **General**. Select **Default (-g)** from the **Debug Level** list on the right side.

Q: Why can't I debug systems containing an active watchdog timer?

A: While paused in debug mode, the processor does not stop the watchdog timer. This is a deliberate choice because in deployed systems the watchdog timer should trigger even if the processor has hit a break instruction. Unfortunately the paused processor is not able to refresh the watchdog timer, so the watchdog timer resets the system and your debug connection is lost. To avoid this, do not enable watchdog timers while debugging.

Q: Why do I see the message "Unable to connect to JTAG UART because another application is using it" when I try to start a run or debug session?

A: The IDE does not automatically terminate previous run or debug sessions if **Allow multiple active run/debug sessions** is on. In this case, you must manually terminate existing sessions that use the JTAG cable before starting a new session. Alternately, you can click **Preferences** on the Window menu, then click **Nios II**, then turn off **Allow multiple active run/debug sessions**.

Q: Why does my project fail to compile after I make changes in SOPC Builder? How do I get the Nios II IDE to recognize component name changes made in SOPC Builder?

A: If you change the names of components in the SOPC Builder system, you must update the system library project to reflect the new component names. Right-click your system library project and then click **Properties**. Review the **System Library** page of the **Properties** dialog box to verify that any referenced component names appear correctly, and then rebuild your system library project. If you renamed the CPU in SOPC Builder, then you must recreate a system library project targeted to the system with the new CPU. Alternatively, you can delete the system library project (without deleting the contents) and re-import the project, selecting the new CPU in the process. You might also have to update your application code to reflect the new component name(s).

Q: Why is the Nios II IDE GUI responsiveness slow?

A: If you have a lot of open projects in the C/C++ Projects view, you might want to disable the automatic refresh feature. On the Window menu, click **Preferences**, then click **Workbench**, then turn off **Refresh workspace automatically**. You can manually refresh the workspace by choosing **Refresh** on the File menu or pressing F5.

The GUI responsiveness might also be slower while the C/C++ Indexer creates a database of source and header files when you first start the IDE or after you create a new project.

Q: Why do I get the message "Launch failed. Error starting gdbserver - see console for details" with no other details on the console, when launching a debug session with the Nios II ISS target?

A: If you are running an antivirus program with a firewall, it might be blocking the listening tcp port opened by the **nios2-iss** executable. You must unblock the **nios2-iss.exe** program to enable it to operate through the firewall. Please consult the help section of your antivirus program, for instructions on how to do this.