

This chapter familiarizes you with the main features of the Nios® II integrated development environment (IDE).

In most cases, you should create new projects using either the Nios II Software Build Tools (SBT) for Eclipse™ or the SBT command line. IDE support is for the following situations:

- Working with pre-existing Nios II IDE software projects
- Creating new projects for the Nios II C2H compiler
- Debugging with the FS2 console

You can use the Nios II IDE only if your underlying hardware is generated with the SOPC Builder system integration tool.

If your hardware design is created with SOPC Builder 7.0 or earlier, you must either use the Nios II IDE development flow, or update your hardware design.

The Nios II IDE is an optional feature. It is available only if you enable **Legacy Package: Nios II IDE / GCC3 Toolchain / C2H Compiler** when you install the Altera® Complete Design Suite.

The Nios II IDE uses GCC 3, and IDE projects can only be built with GCC 3.



For detailed information about installing the Altera Complete Design Suite, refer to the *Altera Software Installation and Licensing Manual*.

This chapter contains the following sections:

- “Differences from the Nios II Software Build Tools”
- “Getting Started with the Nios II IDE” on page A-4
- “Developing Software with the Nios II IDE” on page A-7
- “Porting Nios II IDE Projects to the SBT” on page A-18
- “Archiving Nios II IDE Software Projects” on page A-21
- “Help System” on page A-22



For more information on all topics related to the Nios II IDE, refer to the Nios II IDE help system.

## Differences from the Nios II Software Build Tools

The Nios II Embedded Design Suite (EDS) offers two software development tool flows, as described in “Nios II Software Development Environment” in the *Overview* chapter of the *Nios II Software Developer’s Handbook*. The Nios II IDE is the key part of the Nios II IDE development flow. This section describes some importance differences between the SBT development flow and the Nios II IDE development flow.



Table A-1 describes the tools provided by the Nios II IDE user interface.

**Table A-1. The Nios II IDE and Associated Tools**

Tools	Description
The Nios II IDE	The Nios II IDE is a software development user interface for the Nios II processor. All software development tasks can be accomplished in the IDE, including editing, building, and debugging programs. For more information, refer to the Nios II IDE help system.
Flash programmer	<p>The Nios II IDE includes a flash programmer utility that allows you to program flash memory chips on a target board. The flash programmer supports programming flash on any board, including Altera® development boards and your own custom boards. The flash programmer facilitates programming flash for the following purposes:</p> <ul style="list-style-type: none"> <li>■ Executable code and data</li> <li>■ Bootstrap code to copy code from flash to RAM, and then run from RAM</li> <li>■ Hardware Abstraction Layer (HAL) file subsystems</li> <li>■ FPGA hardware configuration data</li> </ul> <p>For more information, refer to the <i>Nios II Flash Programmer User Guide</i>.</p>
Instruction set simulator	Altera provides an instruction set simulator (ISS) for the Nios II processor. The ISS is available in the Nios II IDE, and the process for running and debugging programs on the ISS is the same as for running and debugging on target hardware. For more information, refer to the Nios II IDE help system.
Quartus® II Programmer	The Quartus II programmer is part of the Altera Complete Design Suite, however the Nios II IDE can start the Quartus II programmer directly. The Quartus II programmer allows you to download new FPGA configuration files to the board. For more information, refer to the Nios II IDE help system, or to the Quartus II help system.

## Nios II IDE Makefiles

A major difference between the Nios II IDE software development flow and the Nios II SBT flow is the difference in makefile implementation. The Nios II SBT generates user-managed makefiles that you can read, and modify in detail using the SBT. In the Nios II IDE development flow, the IDE creates and manages your project makefiles for you.

The key differences between user-managed makefiles and Nios II IDE makefiles are as follows:

- The Nios II IDE has control over the contents of a makefile in an IDE project.
- In a Nios II IDE makefile, the structure and syntax are optimized for automation rather than for human readability.
- It is not normally necessary or recommended for you to read or modify a Nios II IDE makefile.

## Nios II IDE Terminology

The Nios II SBT and the Nios II IDE are described with somewhat different project terminology. Where the meaning is unambiguous, this handbook uses the SBT terminology for both development flows. The IDE terminology is used where needed to distinguish the Nios II IDE development flow from the SBT development flow.

The terminology differences are listed in [Table A-2](#).

**Table A-2. Nios II IDE Terminology**

Nios II IDE Terminology	Nios II SBT Terminology
Nios II C/C++ application	Nios II application
Nios II C/C++ library	Nios II user library
System library	Board support package (BSP)
System library option	BSP setting
Software component	Software package

## Altera Nios II Instruction Set Simulator

The Nios II Instruction Set Simulator (ISS) allows you to begin developing programs before the target hardware platform is ready. The Nios II IDE allows you to run programs on the ISS as easily as running on a real hardware target.

## Command-Line Tools

Although the Nios II IDE is primarily a GUI, it includes some commands for use at the Nios II Command Shell. This section describes those commands.

### Nios II IDE Command-Line Tools

[Table A-3](#) shows the command-line utilities that form the basis of the Nios II IDE. These tools can create and build Nios II IDE projects without launching the Nios II IDE GUI. However, Altera recommends that you use the Nios II SBT to address command-line needs, as well as for all new projects.



For detailed information about the Nios II SBT, refer to the [Nios II Software Build Tools](#) chapter of the *Nios II Software Developer's Handbook*.

Each of the Nios II IDE command-line tools launches the Nios II IDE in the background, without displaying the GUI. You cannot use these utilities while the IDE is running, because only one instance of the Nios II IDE can be active at a time.

The Nios II IDE command-line tools are in the `<Nios II EDS install path>/bin/` directory.

**Table A-3. Nios II IDE Command-Line Tools**

Tool	Description
<code>nios2-create-system-library</code>	Creates a new system library project.
<code>nios2-create-application-project</code>	Creates a new C/C++ application project.
<code>nios2-build-project</code>	Builds a project using the Nios II IDE managed-make facilities. Creates or updates the makefiles to build the project, and optionally runs make. <code>nios2-build-project</code> operates only on projects that exist in the current Nios II IDE workspace.

**Table A-3. Nios II IDE Command-Line Tools**

Tool	Description
<code>nios2-import-project</code>	Imports a previously-created Nios II IDE project into the current workspace.
<code>nios2-delete-project</code>	Removes a project from the Nios II IDE workspace, and optionally deletes files from the file system.



The Nios II IDE command-line tools must be supplied with a workspace location. This location is supplied by means of the `-data <path to workspace>` command-line argument. The path to the workspace must not contain whitespace. Otherwise, any valid disk location can be used for the workspace. The workspace shown in [Example A-1](#) is the default workspace which is used by the IDE.

#### Example A-1. Specifying a Workspace on the Command Line

```
nios2-create-project \
    -data c:/altera/80/nios2eds/bin/eclipse/nios2-ide-workspace-8.0 \
    <other arguments>
```

### FS2 Command-Line Interface

The `nios2-console` command opens the FS2 command-line interface, connects to the Nios II processor, and optionally downloads code.



The FS2 console is not compatible with the Nios II SBT for Eclipse.

## Getting Started with the Nios II IDE

This section describes the key components of the Nios II IDE, and describes how to create and debug a software project.

### The Nios II IDE Workbench

The term “workbench” refers to the desktop development environment for the Nios II IDE. The workbench is where you edit, compile and debug your programs in the IDE.

#### Perspectives, Editors, and Views

Each workbench window contains one or more perspectives. Each perspective provides a set of capabilities for accomplishing a specific type of task.

Most perspectives in the workbench comprise an editor area and one or more views. An editor allows you to open and edit a project resource (i.e., a file, folder, or project). Views support editors, provide alternative presentations, and ways to navigate the information in your workbench.

Any number of editors can be open at once, but only one can be active at a time. The main menu bar and toolbar for the workbench window contain operations that are applicable to the active editor. Tabs in the editor area indicate the names of resources that are currently open for editing. An asterisk (\*) indicates that an editor has unsaved changes. Views can also provide their own menus and toolbars, which, if present, appear along the top edge of the view. To open the menu for a view, click the drop-down arrow icon at the right of the view's toolbar or right-click in the view. A view might appear on its own, or stacked with other views in a tabbed notebook.

## EDS Development Flows and the Nios II IDE

The main distinction between the two development flows is in the management of the project.

### Nios II IDE Projects and Makefiles

In the Nios II IDE development flow, the IDE manages Nios II C/C++ application and board support package (BSP) projects and makefiles that you create with the **New Project** wizard in Nios II IDE. The best way to modify and build an IDE project is through the IDE. You manage the BSP project settings with the **System Library** page of the **Properties** dialog box.



In the Nios II IDE, the term “system library” is used for a BSP project.

### SBT Projects and Makefiles

In the Nios II SBT development flow, you manage Nios II application, user library, and BSP projects and makefiles, giving you total control. Typically, you create SBT projects outside of the Nios II IDE and then import them into the IDE for debugging.



SBT projects and Nios II IDE projects are not interchangeable. However, you can manually convert an IDE project to an SBT project.



For details, refer to “[Porting Nios II IDE Projects to the SBT](#)” on page A-18.

## Creating a New Nios II IDE-Managed Project

The Nios II IDE provides a **New Project** wizard that guides you through the steps to create new IDE projects. To start the **New Project** wizard for Nios II C/C++ application projects, on the File menu in the Nios II C/C++ perspective, point to **New**, and then click **Nios II C/C++ Application**.

The Nios II C/C++ application **New Project** wizard prompts you to specify:

1. A name for your new Nios II project.
2. The target hardware.
3. A template for the new project.

Project templates are ready-made, working software projects that serve as examples to show you how to structure your own Nios II projects. It is often easier to start with a working “Hello World” project, than to start a blank project from scratch.

When the Nios II IDE creates the new application project, it also creates a BSP project. If the name of the application project is `<name>`, the default name of the BSP project is `<name>*_syslib` (for example, `dhystone_0_syslib`). These projects appear in the Nios II C/C++ Projects view of the workbench.



The first time you create or build a Nios II project, the Nios II IDE automatically creates a project in your workspace called **altera.components**. This project contains links to the source code files for all Altera-provided device drivers and software packages, enabling you to step through system code in the debugger, set breakpoints, and use other debugger features. The **altera.components** project appears in the Nios II C/C++ Projects view. The Nios II C/C++ view protects the source files in **altera.components** from accidental deletion, because they are shared among all software projects. Do not attempt to circumvent this protection.

## Building and Managing Projects

Right-clicking on any resource (a file, folder, or project) opens a context-sensitive menu containing commands that you can perform on the resource. Right-clicking is usually the quickest way to find the command you need, though commands are also available in menus and toolbars.

To compile a Nios II project, right-click the project in the Nios II C/C++ Projects view, and click **Build Project**. When building, the Nios II IDE first builds the BSP project (and any other project dependencies), and then compiles the main project. Any warnings or errors are displayed in the Tasks view.

Right-clicking a project in the Nios II C/C++ Projects view also allows you to access the following important options for managing the project:

- **Properties**—Manage the dependencies on target hardware and other projects
- **System Library Properties**—Manage hardware-specific settings, such as communication devices and memory partitioning
- **Build Project**—i.e., make
- **Run As**—Run the program on hardware or under simulation
- **Debug As**—Debug the program on hardware or under simulation

## Debug and Release Configurations

You can select a Debug or Release configuration in the **Project Properties** dialog box, under **C/C++ Build**. The project configuration controls the optimization level and debug compiler options.

## Running and Debugging Programs

Run and debug operations are available by right-clicking the Nios II project. The Nios II IDE allows you to run or debug the project either on a target board, under the Nios II ISS, or using the ModelSim® logic simulator. For example, to run the program on a target board, right-click the project in the Nios II C/C++ Projects view, point to Run As, and then click **Nios II Hardware**. Character I/O to stdout and stderr are displayed in the Console view.

Starting a debug session is similar to starting a run session. For example, to debug the program on the ISS, right-click the project in the Nios II C/C++ Projects view, point to **Debug As**, and then click **Nios II Instruction Set Simulator**.

Launching the debugger changes the workbench perspective to the debug perspective. You can easily switch between the debug perspective and the Nios II C/C++ development perspective by clicking on the **Open Perspective** icon at the upper right corner of the workbench window.

After you start a debug session, the debugger loads the program, sets a breakpoint at `main()`, and begins executing the program. You use the usual controls to step through the code: Step Into, Step Over, Resume, Terminate, etc. To set a breakpoint, double click in the left-hand margin of the code view, or right-click in the margin and then click **Add Breakpoint**.

The Nios II IDE offers many views that allow you to examine the status of the processor while debugging, such as the Variables, Expressions, Registers, and Memory views.

## Programming Flash

Many Nios II processor systems use external flash memory to store one or more of the following items:

- Program code
- Program data
- FPGA configuration data
- File systems

The Nios II IDE provides a Flash Programmer utility to help you manage and program the contents of flash memory.



To program an SBT C/C++ application to flash memory, you must first specify an SOPC Builder System File, as follows:

1. Click **Browse** at the right of the **SOPC Builder System PTF File** box.
2. Locate the SOPC Builder System File on which the application's BSP is based. For example, if you are using a Nios II SBT example, the SOPC Builder System File is three levels up in the directory tree from the software project.

## Developing Software with the Nios II IDE

In many ways, Nios II software development with the Nios II IDE is the same as development with the SBT. The processor architecture, Hardware Abstraction Layer (HAL), software packages and drivers are the same. However, there are a few limitations and differences in tool flow details. This section discusses those differences and limitations.

## Using the HAL in an IDE Project

Like the Nios II SBT development flow, the Nios II IDE flow can automatically keep your system library up to date with the SOPC Builder system. In an IDE project, the Nios II IDE manages the system library and updates the driver configurations to accurately reflect the system hardware. If the SOPC Builder system changes — i.e., the SOPC Builder system file (.ptf) is updated — the IDE rebuilds the system library the next time you build or run your C/C++ application program.

### Generated Files

The Nios II IDE development flow uses different file name and directory structure conventions for some generated system library files, as described in this section.

#### **generated.x**

In a Nios II IDE project, the **generated.x** file is the same as the **linker.x** file created by the Nios II build tools.

#### **generated.gdb**

In a Nios II IDE project, the **generated.gdb** file is the same as the **memory.gdb** file created by the Nios II build tools.

#### **alt\_main.c**

In a Nios II IDE project, you can find **alt\_main.c** in  
<Nios II EDS install path>/components/altera\_hal/HAL/src.

### System Library Settings

In a Nios II IDE project, you manage the system library project settings with the **System Library** page of the **Properties** dialog box.



For details of how to control system library settings, refer to the Nios II IDE help system.

### Reducing Code Footprint

The basic techniques for reducing code footprint are the same in the Nios II IDE flow as in the SBT flow, but you use a different procedure to specify the system library options. You control the following system library options through the Nios II IDE system properties dialog box:

**Table A-4. System Library Options for Reducing Code Footprint**

Technique	System Library Option Name
Use Reduced Device Drivers	Reduced device drivers
Reduce the File Descriptor Pool	Max file descriptors
Use a Smaller File I/O Library	Small C library
Use the Lightweight Device Driver application programming interface (API)	Lightweight device driver API
Eliminate Clean Exit	Clean exit (flush buffers)



**Table A-4. System Library Options for Reducing Code Footprint**

Technique	System Library Option Name
Eliminate All Exit Code	Program never exits
Turn off C++ Support	Support C++

## Paths to Hardware Abstraction Layer Files

In Nios II IDE projects, HAL source files are in several directories. You can find HAL-related files in the following locations:

- The *<Nios II EDS install path>/components* directory contains most HAL source files.
- *<Nios II EDS install path>/components/altera\_hal/HAL/inc/sys* contains header files defining the HAL generic device models. In a `#include` directive, reference these files relative to *<Nios II EDS install path>/components/altera\_hal/HAL/inc/*. For example, to include the direct memory access (DMA) drivers, use `#include sys/alt_dma.h`
- *<Nios II EDS install path>/bin* contains the newlib ANSI C library header files.
- The Altera Complete Design Suite includes HAL drivers for SOPC Builder components distributed with the Altera Complete Design Suite. For example, if the design suite is installed in `c:\altera\80`, you can find the drivers under `c:\altera\80\ip\sopc_builder_ip`.

## Overriding HAL Functions

In the Nios II IDE build flow, you can override any HAL source file, including `alt_sys_init.c`, by placing your own implementation in your system project directory. When building the executable, the Nios II IDE finds your function, and uses it in place of the HAL version.

## Device Drivers for Nios II IDE Projects

HAL device drivers work the same in the Nios II IDE flow as in the SBT flow. However, there are slight differences in how you create a device driver.

Compared with the Nios II IDE, Nios II SBT provides a less rigid set of file naming and location requirements for your drivers. However, Altera recommends using the Nios II IDE conventions to maintain build-flow compatibility. Provided you use the file hierarchy described in “Integrating a Device Driver in the HAL” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*, your device driver is compatible with the Nios II IDE development flow.

This section describes how to develop device drivers for Nios II IDE projects.

## Integrating a Device Driver in the HAL

This section discusses how to take advantage of the HAL's ability to instantiate and register device drivers during system initialization. You can take advantage of this service, whether you created a device driver for one of the HAL generic device models, or you created a peripheral-specific device driver. Taking advantage of the automation provided by the HAL is mainly a process of placing files in the appropriate place in the HAL directory structure.

### Device Driver Files for the HAL

This section describes how to provide appropriate files to integrate your device driver into the HAL.

- **A Device's HAL Header File and `alt_sys_init.c`**—At the heart of the HAL is the autogenerated source file, `alt_sys_init.c`. `alt_sys_init.c` contains the source code that the HAL uses to initialize the device drivers for all supported devices in the system. In particular, this file defines the `alt_sys_init()` function, which is called before `main()` to initialize all devices and make them available to the program.



Refer to “Creating a Custom Device Driver for the HAL” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer's Handbook* for more information about `alt_sys_init.c`.

- **A Device's HAL Header File and `alt_sys_init.c`**—In the Nios II IDE development flow, for each device visible to the processor, the generator utility searches for an associated header file in the device's HAL/inc directory. The name of the header file depends on the SOPC Builder component name. For example, for Altera's JTAG UART component, the generator finds the file `altera_avalon_jtag_uart/HAL/inc/altera_avalon_jtag_uart.h`. If the generator utility finds such a header file, it inserts code into `alt_sys_init.c` to perform the following actions:
  - Include the device's header file.
  - Call the macro `<name of device>_INSTANCE` to allocate storage for the device.
  - Call the macro `<name of device>_INIT` inside the `alt_sys_init()` function to initialize the device.
- **Device Driver Source Code**—Place any required source code in the `HAL/src` directory. In addition, you must include a makefile fragment, `component.mk`. The `component.mk` file lists the source files to include in the system library. You can list multiple files by separating filenames with a space. [Example A-2](#) shows an example makefile fragment for Altera's JTAG UART device.

The Nios II IDE includes the `component.mk` file into the top-level makefile when compiling system library projects and application projects. `component.mk` can only modify the make variables listed in [Table A-5](#).

`component.mk` can add additional make rules and macros as required, but interoperability macro names must conform to the namespace rules.



For details about namespace rules, refer to “HAL Namespace Allocation” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

#### Example A-2. component.mk for a UART Driver

```
C_LIB_SRCS += altera_avalon_uart.c
ASM_LIB_SRCS +=
INCLUDE_PATH +=
```

Table A-5. Make Variables Defined in component.mk

Make Variable	Meaning
C_LIB_SRCS	The list of C source files to build into the system library.
ASM_LIB_SRCS	The list of assembler source files to build into the system library (these are preprocessed with the C preprocessor).
INCLUDE_PATH	A list of directories to add to the include search path. The directory <code>&lt;component&gt;/HAL/inc</code> is added automatically and so does not need to be explicitly defined by the component.

### Overriding the Default Device Drivers

The Nios II IDE locates all include and source files using search paths. The system library project directory is always searched first. If you place an alternative driver in the system library project directory, it overrides drivers installed with the Nios II EDS. For example, if a component provides the header file `alt_my_component.h`, and the system library project directory also contains a file `alt_my_component.h`, the version provided in the system library project directory is used at compile time. This same mechanism can override C and assembler source files.

## Exception Handling in a Nios II IDE Project

Exception handling with the internal interrupt controller in Nios II IDE projects is largely the same as in SBT projects. This section discusses the differences.



The Nios II IDE development flow does not support external interrupt controllers.

### Software Trap Handling

If your software is compiled for release, the exception handler makes a distinction between traps and other exceptions. If your software is compiled for debug, traps and other exceptions are handled identically, by executing a break instruction.



For more information about HAL software exception handling, refer to “HAL Exception Handling System Implementation” in the *Exception Handling* chapter of the *Nios II Software Developer’s Handbook*.



The instruction-related exception handler is unavailable in Nios II IDE projects.

## Advanced Exceptions

Advanced exception support, including the instruction-related exception handler, is not available in the Nios II IDE development flow.

## Using the Unimplemented Instruction Handler

To include the unimplemented instruction handler in a Nios II IDE project, turn on **Emulate multiply and divide instructions** on the **System properties** page of the Nios II IDE.



You do not normally need the unimplemented instruction handler, because the HAL includes software emulation for unimplemented instructions from its run-time libraries if you are compiling for a Nios II processor that does not support the instructions.



For further information about the unimplemented instruction handler, refer to “HAL Exception Handling System Implementation” in the *Exception Handling* chapter of the *Nios II Software Developer’s Handbook*.

## Configuring MicroC/OS-II Projects with the Nios II IDE

In the Nios II IDE, the displayed MicroC/OS-II setting names are different from the equivalent BSP setting names. This section lists the meanings of the IDE setting names.

For step-by-step instructions on how to create a MicroC/OS-II project in the Nios II IDE, refer to *Using the MicroC/OS-II RTOS with the Nios II Processor Tutorial*.

## MicroC/OS-II General Options

Table A-6 shows the general MicroC/OS-II options available through the Nios II IDE.

**Table A-6. General Options (Part 1 of 2)**

Option	Description
<b>Maximum number of tasks</b>	Specifies the value of the <code>OS_MAX_TASKS</code> preprocessor symbol. Must be at least 2
<b>Lowest assignable priority</b>	Specifies the value of the <code>OS_LOWEST_PRIO</code> preprocessor symbol. Maximum allowable value is 63.
<b>Thread safe C library</b>	Enable thread-safe C library
<b>Enable code for event flags</b>	Specifies the value of the <code>OS_FLAG_EN</code> preprocessor symbol. When this option is disabled (set to 0), event flag settings are also disabled. Refer to “ <a href="#">Event Flag Settings</a> ” on page A-13.
<b>Enable code for mutex semaphores</b>	Specifies the value of the <code>OS_MUTEX_EN</code> preprocessor symbol. When this option is disabled (set to 0), mutual exclusion semaphore settings are also disabled. Refer to “ <a href="#">Mutex Settings</a> ” on page A-13
<b>Enable code for semaphores</b>	Specifies the value of the <code>OS_SEM_EN</code> preprocessor symbol. When this option is disabled (set to 0), semaphore settings are also disabled. Refer to “ <a href="#">Semaphore Settings</a> ” on page A-14.

**Table A-6. General Options (Part 2 of 2)**

Option	Description
Enable code for mailboxes	Specifies the value of the <code>OS_MBOX_EN</code> preprocessor symbol. When this option is disabled (set to 0), mailbox settings are also disabled. Refer to “Mailbox Settings” on page A-14.
Enable code for queues	Specifies the value of the <code>OS_Q_EN</code> preprocessor symbol. When this option is disabled (set to 0), queue settings are also disabled. Refer to “Queue Settings” on page A-14.
Enable code for memory management	Specifies the value of the <code>OS_MEM_EN</code> preprocessor symbol. When this option is disabled (set to 0), memory management settings are also disabled. Refer to “Memory Management Settings” on page A-15.
Enable code for timers	Enable code for timers

## Event Flag Settings

Table A-7 shows the event flag settings available through the Nios II IDE.

**Table A-7. Event Flags Settings**

Setting	Description
Include code for wait on clear in the event flags	Specifies the value of the <code>OS_FLAG_WAIT_CLR_EN</code> preprocessor symbol. This setting includes code to wait for the specified bits to be cleared in the event flag group.
Include code for <code>OSFlagAccept()</code>	Specifies the value of the <code>OS_FLAG_ACCEPT_EN</code> preprocessor symbol.
Include code for <code>OSFlagDel()</code>	Specifies the value of the <code>OS_FLAG_DEL_EN</code> preprocessor symbol.
Include code for <code>OSFlagQuery()</code>	Specifies the value of the <code>OS_FLAG_QUERY_EN</code> preprocessor symbol.
Maximum number of event flag groups	Specifies the value of the <code>OS_MAX_FLAGS</code> preprocessor symbol.
Size of name of event flags group	Specifies the value of the <code>OS_FLAG_NAME_SIZE</code> preprocessor symbol.
Event flag bits (8, 16, 32)	Specifies the number of event flag bits

## Mutex Settings

Table A-8 shows the mutex settings available through the Nios II IDE.

**Table A-8. Mutex Settings**

Setting	Description
Include code for <code>OSMutexAccept()</code>	Specifies the value of the <code>OS_MUTEX_ACCEPT_EN</code> preprocessor symbol.
Include code for <code>OSMutexDel()</code>	Specifies the value of the <code>OS_MUTEX_DEL_EN</code> preprocessor symbol.
Include code for <code>OSMutexQuery()</code>	Specifies the value of the <code>OS_MUTEX_QUERY_EN</code> preprocessor symbol.

## Semaphore Settings

Table A-9 shows the semaphore settings available through the Nios II IDE.

**Table A-9. Semaphores Settings**

Setting	Description
Include code for <b>OSSemAccept()</b>	Specifies the value of the <code>OS_SEM_ACCEPT_EN</code> preprocessor symbol.
Include code for <b>OSSemSet()</b>	Specifies the value of the <code>OS_SEM_SET_EN</code> preprocessor symbol.
Include code for <b>OSSemDel()</b>	Specifies the value of the <code>OS_SEM_DEL_EN</code> preprocessor symbol.
Include code for <b>OSSemQuery()</b>	Specifies the value of the <code>OS_SEM_QUERY_EN</code> preprocessor symbol.

## Mailbox Settings

Table A-10 shows the mailbox settings available through the Nios II IDE.

**Table A-10. Mailboxes Settings**

Setting	Description
Include code for <b>OSMboxAccept()</b>	Specifies the value of the <code>OS_MBOX_ACCEPT_EN</code> preprocessor symbol.
Include code for <b>OSMBoxDel()</b>	Specifies the value of the <code>OS_MBOX_DEL_EN</code> preprocessor symbol.
Include code for <b>OSMboxPost()</b>	Specifies the value of the <code>OS_MBOX_POST_EN</code> preprocessor symbol.
Include code for <b>OSMboxPostOpt()</b>	Specifies the value of the <code>OS_MBOX_POST_OPT_EN</code> preprocessor symbol.
Include code for <b>OSMBoxQuery()</b>	Specifies the value of the <code>OS_MBOX_QUERY_EN</code> preprocessor symbol.

## Queue Settings

Table A-11 shows the queue settings available through the Nios II IDE.

**Table A-11. Queues Settings (Part 1 of 2)**

Setting	Description
Include code for <b>OSQAccept()</b>	Specifies the value of the <code>OS_Q_ACCEPT_EN</code> preprocessor symbol.
Include code for <b>OSQDel()</b>	Specifies the value of the <code>OS_Q_DEL_EN</code> preprocessor symbol.
Include code for <b>OSQFlush()</b>	Specifies the value of the <code>OS_Q_FLUSH_EN</code> preprocessor symbol.
Include code for <b>OSQPost()</b>	Specifies the value of the <code>OS_Q_POST_EN</code> preprocessor symbol.
Include code for <b>OSQPostFront()</b>	Specifies the value of the <code>OS_Q_POST_FRONT_EN</code> preprocessor symbol.
Include code for <b>OSQPostOpt()</b>	Specifies the value of the <code>OS_Q_POST_OPT_EN</code> preprocessor symbol.

**Table A-11. Queues Settings (Part 2 of 2)**

Setting	Description
Include code for OSQQuery()	Specifies the value of the OS_Q_QUERY_EN preprocessor symbol.
Maximum number of Queue Control blocks	Specifies the value of the OS_MAX_QS preprocessor symbol.

## Memory Management Settings

Table A-12 shows the memory management settings available through the Nios II IDE.

**Table A-12. Memory Management Settings**

Setting	Description
Include code for OSMemQuery()	Specifies the value of the OS_MEM_QUERY_EN preprocessor symbol.
Maximum number of memory partitions	Specifies the value of the OS_MAX_MEM_PART preprocessor symbol.
Size of memory partition name	Specifies the value of the OS_MEM_NAME_SIZE preprocessor symbol.

## Miscellaneous Settings

Table A-13 shows the miscellaneous settings available through the Nios II IDE.

**Table A-13. Miscellaneous Settings**

Setting	Description
Enable argument checking	Specifies the value of the OS_ARG_CHK_EN preprocessor symbol.
Enable uCOS-II hooks	Specifies the value of the OS_CPU_HOOKS_EN preprocessor symbol.
Enable debug variables	Specifies the value of the OS_DEBUG_EN preprocessor symbol.
Include code for OSSchedLock() and OSSchedUnlock()	Specifies the value of the OS_SCHED_LOCK_EN preprocessor symbol.
Enable tick stepping feature for uCOS-View	Specifies the value of the OS_TICK_STEP_EN preprocessor symbol.
Enable statistics task	Specifies the value of the OS_TASK_STAT_EN preprocessor symbol.
Check task stacks from statistics task	Specifies the value of the OS_TASK_STAT_STK_CHK_EN preprocessor symbol.
Statistics task stack size	Specifies the value of the OS_TASK_STAT_STK_SIZE preprocessor symbol.
Idle task stack size	Specifies the value of the OS_TASK_IDLE_STK_SIZE preprocessor symbol.
Maximum number of event control blocks	Specifies the value of the OS_MAX_EVENTS preprocessor symbol.
Size of semaphore, mutex, mailbox, or queue name	Specifies the value of the OS_EVENT_NAME_SIZE preprocessor symbol.

## Task Management Settings

Table A-14 shows the task management settings available through the Nios II IDE.

**Table A-14. Task Management Settings**

Setting	Description
Include code for OSTaskChangePrio()	Specifies the value of the OS_TASK_CHANGE_PRIO_EN preprocessor symbol.
Include code for OSTaskCreate()	Specifies the value of the OS_TASK_CREATE_EN preprocessor symbol.
Include code for OSTaskCreateExt()	Specifies the value of the OS_TASK_CREATE_EXT_EN preprocessor symbol.
Include code for OSTaskDel()	Specifies the value of the OS_TASK_DEL_EN preprocessor symbol.
Include variables in OS_TCB for profiling	Specifies the value of the OS_TASK_PROFILE_EN preprocessor symbol.
Include code for OSTaskQuery()	Specifies the value of the OS_TASK_QUERY_EN preprocessor symbol.
Include code for OSTaskSuspend() and OSTaskResume()	Specifies the value of the OS_TASK_SUSPEND_EN preprocessor symbol.
Include code for OSTaskSwHook()	Specifies the value of the OS_TASK_SW_HOOK_EN preprocessor symbol.
Size of task name	Specifies the value of the OS_TASK_NAME_SIZE preprocessor symbol.

## Time Management Settings

Table A-15 shows the time management settings available through the Nios II IDE.

**Table A-15. Time Management Settings**

Setting	Description
Include code for OStimeDlyHMSM()	Specifies the value of the OS_TIME_DLY_HMSM_EN preprocessor symbol.
Include code for OStimeDlyResume()	Specifies the value of the OS_TIME_DLY_RESUME_EN preprocessor symbol.
Include code for OStimeGet() and OStimeSet()	Specifies the value of the OS_TIME_GET_SET_EN preprocessor symbol.
Include code for OStimeTickHook()	Specifies the value of the OS_TIME_TICK_HOOK_EN preprocessor symbol.



## Timer Management Settings

Table A-16 shows the timer management settings available through the Nios II IDE.

**Table A-16. Timer Management Settings**

Setting	Description
Maximum number of timers	Specifies the maximum number of timers
Determine the size of a timer name	Specifies the size of a timer name
Size of timer wheel (#Spokes)	Specifies the size of the timer wheel
Rate at which timer management task runs (Hz)	Specifies the rate at which the timer management task runs
Stack size for timer task	Specifies the stack space allocated for the timer task
Priority of timer task (0=highest)	Specifies the timer task priority

## Using NicheStack in a Nios II IDE Project

This section discusses the details of how to use the NicheStack TCP/IP Stack in the Nios II IDE.

### get\_mac\_addr() and get\_ip\_addr()

The NicheStack TCP/IP Stack system code calls `get_mac_addr()` and `get_ip_addr()` during the device initialization process. These functions are necessary for the system code to set the media access control (MAC) and IP addresses for the network interface, which you select through **MAC interface** in the **NicheStack TCP/IP Stack** tab of the **Software Components** dialog box.

`INICHE_DEFAULT_IF`, defined in **system.h**, identifies the network interface that you defined in SOPC Builder. In the Nios II IDE, you can set `INICHE_DEFAULT_IF` through the **MAC interface** control in the **NicheStack TCP/IP Stack** tab of the **Software Components** dialog box.

`DHCP_CLIENT`, also defined in **system.h**, specifies whether to use the dynamic host configuration protocol (DHCP) client application to obtain an IP address. You can set or clear this setting in the Nios II IDE (with the **Use DHCP to automatically assign IP address** check box)

### Configuring the NicheStack TCP/IP Stack in the Nios II IDE

The Nios II IDE allows you to configure certain options (i.e. modify the `#defines` in **system.h**) without editing source code. The most commonly accessed options are available through the **NicheStack TCP/IP Stack** tab of the **Software Components** dialog box.



If you modify the **ipport.h** file directly, be careful not to select the **Clean Project** build option in the Nios II IDE. Selecting **Clean Project** results in your modified **ipport.h** file being replaced with the starting template version of this file.

## Porting Nios II IDE Projects to the SBT

The Nios II SBT uses a different directory structure and settings file format than the Nios II IDE. Therefore, if you wish to take advantage of the Nios II SBT, you need to port your IDE projects to the Nios II SBT development flow.

This appendix describes the steps required to port a Nios II IDE project to the Nios II SBT development flow. The Nios II EDS includes a utility to convert Nios II IDE projects to the SBT flow.



You do not need to rewrite your Nios II IDE project's C/C++ code for use with the SBT development flow.

The Nios II SBT development flow provides a number of advantages over the Nios II IDE development flow. You might want to port an IDE project to the SBT to take advantage of the following improvements:

- Fully repeatable control over all build options using command line options, Tcl scripts, or both
- Simplified project file management and naming
- Simplified makefiles
- Versioned device drivers
- Independence from Eclipse code and Eclipse projects
- Self-contained BSPs, making hand-off and version control easier than is possible with Nios II IDE-created BSPs (system library projects)
- Upwards compatibility with future releases of the Nios II EDS
- GCC toolchain upgraded to version 4.1.2

## Converting a Nios II IDE Project

This section describes how to convert a Nios II IDE project to an SBT project using the **nios2-convert-ide2sbt** utility.

1. Build the original project in the Nios II IDE, using either the Debug or the Release configuration, depending on your preference. Ensure that the project builds without errors.



The Nios II SBT flow does not include separate Debug and Release builds as implemented in the Nios II IDE development flow.

2. Launch the Nios II Command Shell.



For details about the Nios II Command Shell, refer to the *Getting Started from the Command Line* chapter of the *Nios II Software Developer's Handbook*.

3. Run **nios2-convert-ide2sbt**. The command syntax is as follows:

```
nios2-convert-ide2sbt --input-dir=<source directory> \  
--output-dir=<target directory> \  
--build-config=<configuration>
```

The command arguments are as follows:

- *<source directory>*—directory containing the original Nios II IDE project.
- *<target directory>*—directory where **nios2-convert-ide2sbt** places the converted SBT project. If *<target directory>* does not exist, **nios2-convert-ide2sbt** creates it.
- *<configuration>*—Debug or Release, designating the Nios II IDE project configuration.

Nios II IDE project types are converted as shown in [Table A-17](#).

**Table A-17. Conversion of Project Types by nios2-convert-ide2sbt**

Nios II IDE Project Type	SBT Conversion
Nios II C/C++ system library	BSP
Nios II C/C++ library	User library (1)
Nios II C/C++ application	Application project (1)
<b>Note to Table A-17:</b> (1) At the same time, <b>nios2-convert-ide2sbt</b> converts any Nios II C/C++ system library or Nios II C/C++ libraries on which the converted project depends.	

For example, suppose you have a Nios II C/C++ application in the Release configuration, located in the **./software/hello\_world** directory. To convert the project and its associated system library, and put the resulting SBT project in the **./software\_sbt** directory, type:

```
nios2-convert-ide2sbt --input-dir=software/hello_world \
  --output-dir=software_sbt --build-config=Release ↵
```



If you need to reconvert a project you previously converted, you must delete the previous target directory, or specify a new target directory.

**nios2-convert-ide2sbt** converts your Nios II IDE software project to equivalent SBT projects. **nios2-convert-ide2sbt** ports compiler flags, like the optimization level, debug, and custom instruction options, to the new project. During conversion, **nios2-convert-ide2sbt** displays the converted project settings on the console, in the form of a sample SBT command line.



For details about SBT command usage, refer to the [Nios II Software Build Tools](#) and [Nios II Software Build Tools Reference](#) chapters of the *Nios II Software Developer's Handbook*.

## Other Software Modules

This section describes how to convert and incorporate the following kinds of software modules that you might need to include in your converted BSP:

- Custom device drivers and software components
- Precompiled libraries and non-HAL device drivers

## Custom Device Drivers and Software Components

In the Nios II IDE development flow, a makefile fragment named **component.mk** specifies device drivers. By contrast, in the Nios II SBT development flow, a Tcl script defines the device driver structure. If you have custom device drivers and software components, including third-party device drivers, convert them to Tcl scripts manually.



For more information about implementing device drivers and software packages for the Nios II SBT, refer to “Device Drivers and Software Packages” in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer’s Handbook*.

## Precompiled Libraries and Non-HAL Device Drivers

If you have precompiled libraries and non-HAL device drivers, including third-party libraries and device drivers, convert them manually.

The best way to convert a typical precompiled library is to create a software package. If the precompiled library is dependent on a specific hardware device, it is better to incorporate the library into the device driver. Library archive files (**.a**) can be incorporated into a device driver just as C source files (**.c**) and header files (**.h**) are.



Non-HAL device drivers do not support initialization through `alt_sys_init()`.



For information about creating software packages and drivers, refer to “Integrating a Device Driver in the HAL” in the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

**nios2-convert-ide2sbt** does not convert GNU Compiler Collection (GCC) command-line options for external include file paths (**-I**) or external library file paths (**-L**, **-l**). You must handle these cases manually.

To add include paths to a BSP, use the `hal.make.bsp_inc_dirs` BSP setting in your Tcl script.

To add libraries to an application, add or modify one of the following variables in the application makefile:

- **APP\_LIBRARY\_DIRS**—Specifies a list of paths to directories in which **lib<libname>.a** files reside.
- **APP\_LIBRARY\_NAMES**—Specifies a list of the names of the libraries being added. If the library file is named **lib<libname>.a**, specify **<libname>** as the name of the library.

## Using Your Converted Project

When you have finished porting your project, you can manage the makefiles, build and run the project, and perform all other project tasks exactly as if the project were created with the SBT. You can also import the project to the SBT for Eclipse for debugging.

The Nios II EDS includes two versions of the GCC toolchain: GCC 3.4.6 and GCC 4.1.2. GCC 4, introduced with the Nios II EDS version 10.0, is fully backwards-compatible with GCC 3, and provides substantially faster build times.

After conversion, your project is configured to use GCC 3 by default. To upgrade to GCC 4, import the project to the Nios II SBT for Eclipse and change the toolchain in the **Properties** dialog box, or simply build the project in the GCC 4 Nios II Command Shell.



For information about managing GCC toolchains in the SBT for Eclipse, refer to “Managing Toolchains in Eclipse” in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer’s Handbook*. For information about selecting the toolchain on the command line, refer to the *Getting Started from the Command Line* chapter of the *Nios II Software Developer’s Handbook*.



Nios II IDE projects cannot be directly imported to the SBT for Eclipse. You must first convert the project to the SBT, according to the procedures in this section.

## Archiving Nios II IDE Software Projects

This section helps you identify the files you must include when archiving a Nios II IDE software project. With this information, you can archive a Nios II application project and its associated Nios II BSP project.

You might want to archive your projects for one of the following reasons:

- To place them under source control
- To create backups
- To bundle the projects for transfer to another location

This section covers the following information:

- How to find and identify the files that you must include in an archived Nios II IDE software project.
- Which files must have write permission to allow the software projects to be built.

### Required Files

This section describes the files required by Nios II IDE software projects. This is the minimum set of files needed to completely rebuild a software project, including the Executable and Linking Format File (.elf).

Archive your Nios II IDE software projects together with the SOPC Builder system on which they are based. You cannot rebuild a Nios II IDE software project without its associated SOPC Builder system.

## Nios II Application Project Files

The files listed in [Table A-18](#) are located in the Nios II application project directory.

**Table A-18. Files Required for a Nios II Application Project**

File Description	File Name	Write Permission Required? <a href="#">(1)</a>
All source files	for example: <b>app.c</b> , <b>header.h</b> , <b>assembly.s</b> , <b>lookuptable.dat</b>	No
Eclipse project file	<b>.project</b>	No
C/C++ Development Toolkit project file	<b>.cdtproject</b>	Yes
C/C++ Development Toolkit option file	<b>.cdtbuild</b>	No
Software configuration file	<b>application.stf</b>	No

**Note to [Table A-18](#):**

(1) For further information about write permissions, refer to “[File Write Permissions](#)”.

## Nios II BSP Project

The files listed in [Table A-19](#) are located in the Nios II BSP (system library) project directory.

**Table A-19. Files Required for a Nios II BSP Project**

File description	File name	Write permission required? <a href="#">(1)</a>
Eclipse project file	<b>.project</b>	Yes
C/C++ Development Toolkit project file	<b>.cdtproject</b>	Yes
C/C++ Development Toolkit option file	<b>.cdtbuild</b>	No
System software configuration file	<b>system.stf</b>	Yes

**Note to [Table A-19](#):**

(1) For further information about write permissions, see “[File Write Permissions](#)”.

## File Write Permissions

You must have write permission for certain files, shown in [Table A-18](#) and [Table A-19](#). The tools write to these files as part of the build process. If the files are not writable, the tool chain fails.

Many source control tools mark local files read-only by default. In this case, you must override this behavior. You do not have to check the files out of source control unless you are modifying the Nios II software project.

## Help System

The Nios II IDE help system provides documentation on all IDE topics. To launch the help system, click **Help Contents** on the Help menu. You can also press F1 on Windows (Shift-F1 on Linux) at any time for context-sensitive help. The Nios II IDE help system contains hands-on tutorials that guide you step-by-step through the process of creating, building, and debugging Nios II projects.

## Document Revision History

Table A-20 shows the revision history for this document.

**Table A-20. Document Revision History**

Date	Version	Changes
May 2011	11.0.0	The GCC 3 toolchain is an optional feature.
February 2011	10.1.0	Removed “Referenced Documents” section.
July 2010	10.0.0	<ul style="list-style-type: none"> <li>■ Introduction of GCC 4 toolchain for Nios II Software Build Tools.</li> <li>■ Nios II IDE projects limited to GCC 3.</li> <li>■ Upgrading converted projects to GCC 4.</li> </ul>
November 2009	9.1.0	<ul style="list-style-type: none"> <li>■ Nios II Software Build Tools for Eclipse introduced.</li> <li>■ <i>Nios II Integrated Development Environment</i> chapter combined with <i>Appendix A, Porting Nios II IDE Projects to the Software Build Tools</i>. Chapter replaced by <i>Getting Started with the Graphical User Interface</i> chapter).</li> <li>■ Include Nios II IDE-specific content from the following chapters: <ul style="list-style-type: none"> <li>■ Overview</li> <li>■ Developing Programs Using the Hardware Abstraction Layer</li> <li>■ Developing Device Drivers for the Hardware Abstraction Layer</li> <li>■ Exception Handling</li> <li>■ MicroC/OS-II Real-Time Operating System</li> <li>■ Ethernet and the NicheStack TCP/IP Stack - Nios II Edition</li> </ul> </li> </ul>
March 2009	9.0.0	<ul style="list-style-type: none"> <li>■ Reorganized and updated information and terminology to clarify role of Nios II Software Build Tools.</li> <li>■ Corrected minor typographical errors.</li> </ul>
May 2008	8.0.0	Maintenance release.
October 2007	7.2.0	<b>altera.components</b> project added.
May 2007	7.1.0	<ul style="list-style-type: none"> <li>■ Nios II Software Build Tools introduced.</li> <li>■ Added instructions for importing Software Build Tools projects.</li> <li>■ Changed chapter title.</li> <li>■ Added table of contents to “Introduction” section.</li> <li>■ Added Referenced Documents section.</li> </ul>
March 2007	7.0.0	Maintenance release.
November 2006	6.1.0	Updated look and feel based on Eclipse 3.2, including Nios II C/C++ perspective and Nios II C/C++ Projects views, renamed project types.
May 2006	6.0.0	Maintenance release.
October 2005	5.1.0	Updated for the Nios II IDE version 5.1.
May 2005	5.0.0	Maintenance release.
September 2004	1.1	Updated screen shots.
May 2004	1.0	Initial release.

