2. Getting Started with the Graphical User Interface

NII52017-13.1.0

The Nios® II Software Build Tools (SBT) for Eclipse™ is a set of plugins based on the Eclipse™ framework and the Eclipse C/C++ development toolkit (CDT) plugins. The Nios II SBT for Eclipse provides a consistent development platform that works for all Nios II embedded processor systems. You can accomplish all Nios II software development tasks within Eclipse, including creating, editing, building, running, debugging, and profiling programs.

This chapter familiarizes you with the features of the Nios II SBT for Eclipse. This chapter contains the following sections:

Getting Started with Nios II Software in Eclipse

Writing software for the Nios II processor is similar to writing software for any other microcontroller family. The easiest way to start designing effectively is to purchase a development kit from Altera that includes documentation, a ready-made evaluation board, a getting-started reference design, and all the development tools necessary to write Nios II programs.

Modifying existing code is a common, easy way to learn to start writing software in a new environment. The Nios II Embedded Design Suite (EDS) provides many example software designs that you can examine, modify, and use in your own programs. The provided examples range from a simple "Hello world" program, to a working RTOS example, to a full TCP/IP stack running a web server. Each example is documented and ready to compile.

This section guides you through the most fundamental operations in the Nios II SBT for Eclipse in a tutorial-like fashion. It shows how to create an application project for the Nios II processor, along with the board support package (BSP) project required to interface with your hardware. It also shows how to build the application and BSP projects in Eclipse, and how to run the software on an Altera® development board.

The Nios II SBT for Eclipse Workbench

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

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, and 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.

For detailed information about the Eclipse workbench, perspectives, and views, refer to the Eclipse help system.

Before you create a Nios II project, you must ensure that the Nios II perspective is visible. To open the Nios II perspective, on the Window menu, point to **Open Perspective**, then **Other**, and click **Nios II**.

The Altera Bytestream Console

The workbench in Eclipse for Nios II includes a bytestream console, available through the Eclipse **Console** view. The Altera bytestream console enables you to see output from the processor's `stdout` and `stderr` devices, and send input to its `stdin` device. For information about the Altera bytestream console, see "Using the Altera Bytestream Console" on page 2–8.

Creating a Project

In the Nios II perspective, on the File menu, point to **Nios II Application and BSP from Template**. The **Nios II Application and BSP from Template** wizard appears. This wizard provides a quick way to create an application and BSP at the same time.

Alternatively, you can create separate application, BSP and user library projects.

Specifying the Application

In the first page of the **Nios II Application and BSP from Template** wizard, you specify a hardware platform, a project name, and a project template. You optionally override the default location for the application project, and specify a processor name if you are targeting a multiprocessor hardware platform.

You specify a BSP in the second page of the wizard.

Specifying the Hardware Platform

You specify the target hardware design by selecting a SOPC Information File (**.sopcinfo**) in the **SOPC Information File name** box.

Specifying the Project Name

Select a descriptive name for your project. The SBT creates a folder with this name to contain the application project files.

Letters, numbers, and the underscore (_) symbol are the only valid project name characters. Project names cannot contain spaces or special characters. The first character in the project name must be a letter or underscore. The maximum filename length is 250 characters.

The SBT also creates a folder to contain BSP project files, as described in "Specifying the BSP".

Specifying the Project Template

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 project than to start a blank project from scratch.

You select the project template from the **Templates** list.

The hello_world template provides an easy way to create your first Nios II project and verify that it builds and runs correctly.

Specifying the Project Location

The project location is the parent directory in which the SBT creates the project folder. By default, the project location is under the directory containing the .**sopcinfo** file, in a folder named **software**.

To place your application project in a different folder, turn off **Use default location**, and specify the path in the **Project location** box.

Specifying the Processor

If your target hardware contains multiple Nios II processors, **CPU name** contains a list of all available processors in your design. Select the processor on which your software is intended to run.

Specifying the BSP

When you have finished specifying the application project in the first page of the **Nios II Application and BSP from Template** wizard, you proceed to the second page by clicking **Next**.

On the second page, you specify the BSP to link with your application. You can create a new BSP for your application, or select an existing BSP. Creating a new BSP is often the simplest way to get a project running the first time.

You optionally specify the name and location of the BSP.

Specifying the BSP Project Name

By default, if your application project name is <*project*>, the BSP is named <*project*>**_bsp**. You can type in a different name if you prefer. The SBT creates a directory with this name, to contain the BSP project files. BSP project names are subject to the same restrictions as application project names, as described in "Specifying the Project Name".

Specifying the BSP Project Location

The BSP project location is the parent directory in which the SBT creates the folder. The default project location is the same as the default location for an application project. To place your BSP in a different folder, turn off **Use default location**, and specify the BSP location in the **Project location** box.

Selecting an Existing BSP

As an alternative to creating a BSP automatically from a template, you can associate your application project with a pre-existing BSP. Select **Select an existing BSP project from your workspace**, and select a BSP in the list. The **Create** and **Import** buttons to the right of the existing BSP list provide convenient ways to add BSPs to the list.

Creating the Projects

When you have specified your BSP, you click **Finish** to create the projects. The SBT copies required source files to your project directories, and creates makefiles and other generated files. Finally, the SBT executes a **make clean** command on your BSP.

For details about what happens when Nios II projects are created, refer to "Nios II Software Projects" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*. For details about the **make clean** command, refer to "Makefiles" in the same chapter.

Navigating the Project

When you have created a Nios II project, it appears in the **Project Explorer** view, which is typically displayed at the left side of the Nios II perspective. You can expand each project to examine its folders and files.

For an explanation of the folders and files in a Nios II BSP, refer to "Nios II Software Projects" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

Building the Project

To build a Nios II project in the Nios II SBT for Eclipse, right-click the project name and click **Build Project**. A progress bar shows you the build status. The build process can take a minute or two for a simple project, depending on the speed of the host machine. Building a complex project takes longer.

During the build process, you view the build commands and command-line output in the Eclipse **Console** view.

For details about Nios II SBT commands and output, refer to the *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer's Handbook*.

When the build process is complete, the following message appears in the **Console** view, under the **C-Build [<*project name*>]** title:

```
[<project name> build complete]
```

If the project has a dependency on another project, such as a BSP or a user library, the SBT builds the dependency project first. This feature allows you to build an application and its BSP with a single command.

Configuring the FPGA

Before you can run your software, you must ensure that the correct hardware design is running on the FPGA. To configure the FPGA, you use the Quartus® II Programmer.

In the Windows operating system, you start the Quartus II Programmer from the Nios II SBT for Eclipse, through the Nios II menu. In the Linux operating system, you start Quartus II Programmer from the Quartus II software.

The project directory for your hardware design contains an SRAM Object File (**.sof**) along with the **.sopcinfo** file. The **.sof** file contains the hardware design to be programmed in the FPGA.

For details about programming an FPGA with Quartus II Programmer, refer to the *Quartus II Programmer* chapter in *Volume 3: Verification* of the *Quartus II Handbook*.

Running the Project on Nios II Hardware

This section describes how to run a Nios II program using the Nios II SBT for Eclipse on Nios II hardware, such as an Altera development board.

☞ If your project was created with version 10.1 or earlier of the Nios II SBT, you must re-import it to create the Nios II launch configuration correctly.

A Nios II instruction set simulator is available through the Lauterbach GmbH website (www.lauterbach.com).

To run a software project, right-click the application project name, point to **Run As**, and click **Nios II Hardware**. This command carries out the following actions:

- Creates a Nios II run configuration. For details about run configurations, refer to "Run Configurations in the SBT for Eclipse" on page 2–20.
- Builds the project executable. If all target files are up to date, nothing is built.
- Establishes communications with the target, and verifies that the FPGA is configured with the correct hardware design.
- Downloads the Executable and Linking Format File (**.elf)** to the target memory
- Starts execution at the **.elf** entry point.

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

To disconnect the terminal from the target, click the **Terminate** icon in the Nios II Console view. Terminating only disconnects the host from the target. The target processor continues executing the program.

Debugging the Project on Nios II Hardware

This section describes how to debug a Nios II program using the Nios II SBT for Eclipse on Nios II hardware, such as an Altera development board.

If your project was created with version 10.1 or earlier of the Nios II SBT, you must re-import it to create the Nios II launch configuration correctly.

To debug a software project, right-click the application project name, point to **Debug As**, and click **Nios II Hardware**. This command carries out the following actions:

- Creates a Nios II run configuration. For details about run configurations, refer to "Run Configurations in the SBT for Eclipse" on page 2–20.
- Builds the project executable. If all target files are up to date, nothing is built.
- If debugging on hardware, establishes communications with the target, and verifies that the FPGA is configured with the correct hardware design.
- Downloads the **.elf** to the target memory.
- Sets a breakpoint at the top of `main()`.
- Starts execution at the **.elf** entry point.

The Eclipse debugger with the Nios II plugins provides a Nios II perspective, allowing you to perform many common debugging tasks. Debugging a Nios II program with the Nios II plugins is generally the same as debugging any other C/C++ program with Eclipse and the CDT plugins.

For information about debugging with Eclipse and the CDT plugins, refer to the Eclipse help system.

The debugging tasks you can perform with the Nios II SBT for Eclipse include the following tasks:

- Controlling program execution with commands such as:
  - Suspend (pause)
  - Resume
  - Terminate
  - Step Into
  - Step Over
  - Step Return
- Setting breakpoints and watchpoints
- Viewing disassembly
- Instruction stepping mode
- Displaying and changing the values of local and global variables in the following formats:
  - Binary
  - Decimal
  - Hexadecimal
- Displaying watch expressions
- Viewing and editing registers in the following formats:
  - Binary
  - Decimal
  - Hexadecimal
- Viewing and editing memory in the following formats:
  - Hexadecimal
  - ASCII
  - Signed integer
  - Unsigned integer
- Viewing stack frames in the **Debug** view

Just as when running a program, Eclipse displays program output in the Console view of Eclipse. The Console view maintains a terminal I/O connection with a communication device connected to the Nios II processor in the hardware 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 processor and read as stdin.

To disconnect the terminal from the target, click the **Terminate** icon in the Console view. Terminating only disconnects the host from the target. The target processor continues executing the program.

If your project was created with version 10.1 or earlier of the Nios II SBT, you must re-import it to create the Nios II launch configuration correctly.

Using the Altera Bytestream Console

The Altera bytestream console enables you to see output from the processor's `stdout` and `stderr` devices, and send input to its `stdin` device. The function of the Altera bytestream console is similar to the **nios2-terminal** command-line utility.

Open the Altera bytestream console in the Eclipse **Console** view the same way as any other Eclipse console, by clicking the **Open Console** button.

When you open the Altera bytestream console, the **Bytestream Console Selection** dialog box shows you a list of available bytestreams. This is the same set of bytestreams recognized by System Console. Select the bytestream connected to the processor you are debugging.

For information about how System Console recognizes bytestreams, refer to the *Analyzing and Debugging Designs with the System Console* chapter in *Volume 3: Verification* of the *Quartus II Handbook*.

You can send characters to the processor's `stdin` device by typing in the bytestream console. Be aware that console input in buffered on a line-by-line basis. Therefore, the processor does not receive any characters until you press the Enter key.

A bytestream device can support only one connection at a time. You must close the Altera bytestream console before attempting to connect to the processor with the **nios2-terminal** utility, and vice versa.

Creating a Simple BSP

You create a BSP with default settings using the **Nios II Board Support Package** wizard. To start the wizard, on the File menu, point to **New** and click **Nios II Board Support Package**.

The **Nios II Board Support Package** wizard enables you to specify the following BSP parameters:

- The name
- The underlying hardware design
- The location

- The operating system and version

  ☞ You can select the operating system only at the time you create the BSP. To change operating systems, you must create a new BSP.

- Additional arguments to the **nios2-bsp** script

  If you intend to run the project in the Nios II ModelSim® simulation environment, use the **Additional arguments** parameter to specify the location of the testbench Simulation Package Descriptor File (**.spd**). The **.spd** file is located in the Quartus II project directory. Specify the path as follows:

  ```
  --set QUARTUS_PROJECT_DIR=<relative path>
  ```

  Altera recommends that you use a relative path name, to ensure that the location of your project is independent of the installation directory.

  For details about **nios2-bsp** command arguments, refer to "Details of BSP Creation" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

After you have created the BSP, you have the following options for GUI-based BSP editing:

- To access and modify basic BSP properties, right-click the BSP project, point to **Properties** and click **Nios II BSP Properties**.
- To modify parameters and settings in detail, use the Nios II BSP Editor, described in "Using the BSP Editor".

Makefiles and the Nios II SBT for Eclipse

The Nios II SBT for Eclipse creates and manages the makefiles for Nios II software projects. When you create a project, the Nios II SBT creates a makefile based on the source content you specify and the parameters and settings you select. When you modify the project in Eclipse, the Nios II SBT updates the makefile to match.

Details of how each makefile is created and maintained vary depending on the project type, and on project options that you control. The authoritative specification of project contents is always the makefile, regardless how it is created or updated.

By default, the Nios II SBT manages the list of source files in your makefile, based on actions you take in Eclipse. However, in the case of applications and libraries, you have the option to manage sources manually. Both styles of source management are discussed in the following sections.

Eclipse Source Management

Nios II application and user library makefiles are based on source files and properties that you specify directly. Eclipse source management allows you to add and remove source files with standard Eclipse actions, such as dragging a source file into and out of the Project Explorer view and adding a new source file through the File menu.

You can examine and modify many makefile properties in the **Nios II Application Properties** or **Nios II Library Properties** dialog box. To open the dialog box, right-click the project, click **Properties**, and click **Nios II Application Properties** or **Nios II Library Properties**.

Table 2–1 lists GUI actions that make changes to an application or user library makefile under Eclipse source management.

Table 2–1. Modifying a Makefile with Eclipse Source Management

Modification	Where Modified
Specifying the application or user library name	**Nios II Application Properties** or **Nios II Library Properties** dialog box.
Adding or removing source files	Refer to the Eclipse help system.
Specifying a path to an associated BSP	**Project References** dialog box.
Specifying a path to an associated user library	**Project References** dialog box.
Enabling, disabling or modifying compiler options	**Nios II Application Properties** or **Nios II Library Properties** dialog box.

After the SBT has created a makefile, you can modify the makefile in the following ways:

- With the Nios II SBT for Eclipse, as described in Table 2–1.
- With Nios II SBT commands from the Nios II Command Shell.

When modifying a makefile, the SBT preserves any previous nonconflicting modifications, regardless how those modifications were made.

After you modify a makefile with the Nios II Command Shell, in Eclipse you must right-click the project and click **Update linked resource** to keep the Eclipse project view in step with the makefile.

When the Nios II SBT for Eclipse modifies a makefile, it locks the makefile to prevent corruption by other processes. You cannot edit the makefile from the command line until the SBT has removed the lock.

If you want to exclude a resource (a file or a folder) from the Nios II makefile temporarily, without deleting it from the project, you can use the **Remove from Nios II Build** command. Right-click the resource and click **Remove from Nios II Build**. When a resource is excluded from the build, it does not appear in the makefile, and Eclipse ignores it. However, it is still visible in the Project Explorer, with a modified icon. To add the resource back into the build, right-click the resource and click **Add to Nios II Build**.

Do not use the Eclipse **Exclude from build** command. With Nios II software projects, you must use the **Remove from Nios II Build** and **Add to Nios II Build** commands instead.

Absolute Source Paths and Linked Resources

By default, the source files for an Eclipse project are stored under the project directory. If your project must incorporate source files outside the project directory, you can add them as linked resources.

An Eclipse linked resource can be either a file or a folder. With a linked folder, all source files in the folder and its subfolders are included in the build.

When you add a linked resource (file or folder) to your project, the SBT for Eclipse adds the file or folder to your makefile with an absolute path name. You might use a linked resource to refer to common source files in a fixed location. In this situation, you can move the project to a different directory without disturbing the common source file references.

A linked resource appears with a modified icon (green dot) in the Project Explorer, to distinguish it from source files and folders that are part of the project. You can use the Eclipse debugger to step into a linked source file, exactly as if it were part of the project.

You can reconfigure your project to refer to any linked resource either as an individual file, or through its parent folder. Right-click the linked resource and click **Update Linked Resource**.

You can use the **Remove from Nios II Build** and **Add to Nios II Build** commands with linked resources. When a linked resource is excluded from the build, its icon is modified with a white dot.

You can use Eclipse to create a path variable, defining the location of a linked resource. A path variable makes it easy to modify the location of one or more files in your project.

For information about working with path variables and creating linked resources, refer to the Eclipse help system.

User Source Management

You can remove a makefile from source management control through the **Nios II Application Properties** or **Nios II Library Properties** dialog box. Simply turn off **Enable source management** to convert the makefile to user source management. When **Enable source management** is off, you must update your makefile manually to add or remove source files to or from the project. The SBT for Eclipse makes no changes to the list of source files, but continues to manage all other project parameters and settings in the makefile.

Editing a makefile manually is an advanced technique. Altera recommends that you avoid manual editing. The SBT provides extensive capabilities for manipulating makefiles while ensuring makefile correctness.

In a makefile with user-managed sources, you can refer to source files with an absolute path. You might use an absolute path to refer to common source files in a fixed location. In this situation, you can move the project to a different directory without disturbing the common source file references.

Projects with user-managed sources do not support the following features:

- Linked resources
- The **Add to Nios II Build** command
- The **Remove from Nios II Build** command

Table 2–2 lists GUI actions that make changes to an application or user library makefile under user source management.

Table 2–2. Modifying a Makefile with User Source Management

Modification	Where Modified
Specifying the application or user library name	**Nios II Application Properties** or **Nios II Library Properties** dialog box
Specifying a path to an associated BSP	**Project References** dialog box
Specifying a path to an associated user library	**Project References** dialog box
Enabling, disabling or modifying compiler options	**Nios II Application Properties** or **Nios II Library Properties** dialog box

With user source management, the source files shown in the Eclipse Project Explorer view do not necessarily reflect the sources built by the makefile. To update the Project Explorer view to match the makefile, right-click the project and click **Sync from Nios II Build**.

BSP Source Management

Nios II BSP makefiles are handled differently from application and user library makefiles. BSP makefiles are based on the operating system, BSP settings, selected software packages, and selected drivers. You do not specify BSP source files directly.

BSP makefiles must be managed by the SBT, either through the BSP Editor or through the SBT command-line utilities.

For further details about specifying BSPs, refer to "Using the BSP Editor".

Using the BSP Editor

Typically, you create a BSP with the Nios II SBT for Eclipse. The Nios II plugins provide the basic tools and settings for defining your BSP. For more advanced BSP editing, use the Nios II BSP Editor. The BSP Editor provides all the tools you need to create even the most complex BSPs.

Tcl Scripting and the Nios II BSP Editor

The Nios II BSP Editor provides support for Tcl scripting. When you create a BSP in the BSP Editor, the editor can run a Tcl script that you specify to supply BSP settings.

You can also export a Tcl script from the BSP Editor, containing all the settings in an existing BSP. By studying such a script, you can learn about how BSP Tcl scripts are constructed.

Starting the Nios II BSP Editor

You start the Nios II BSP Editor in one of the following ways:

- Right-click an existing project, point to **Nios II**, and click **BSP Editor**. The editor loads the BSP Settings File (**.bsp**) associated with your project, and is ready to update it.
- On the Nios II menu, click **Nios II BSP Editor**. The editor starts without loading a **.bsp** file.
- Right-click an existing BSP project and click **Properties**. In the **Properties** dialog box, select **Nios II BSP Properties**, and click **BSP Editor**. The editor loads your **.bsp** file for update.

The Nios II BSP Editor Screen Layout

The Nios II BSP Editor screen is divided into two areas. The top area is the command area, and the bottom is the console area. The details of the Nios II BSP Editor screen areas are described in this section.

Below the console area is the **Generate** button. This button is enabled when the BSP settings are valid. It generates the BSP target files, as shown in the **Target BSP Directory** tab.

The Command Area

In the command area, you specify settings and other parameters defining the BSP. The command area contains several tabs:

- The **Main** tab
- The **Software Packages** tab
- The **Drivers** tab
- The **Linker Script** tab
- The **Enable File Generation** tab
- The **Target BSP Directory** tab

Each tab allows you to view and edit a particular aspect of the **.bsp**, along with relevant command line parameters and Tcl scripts.

The settings that appear on the **Main**, **Software Packages** and **Drivers** tabs are the same as the settings you manipulate on the command line.

For detailed descriptions of settings defined for Altera-provided operating systems, software packages, and drivers, refer to the *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer's Handbook*.

The Main Tab

The **Main** tab presents general settings and parameters, and operating system settings, for the BSP. The BSP includes the following settings and parameters:

- The path to the **.sopcinfo** file specifying the target hardware
- The processor name
- The operating system and version

  ☞ You cannot change the operating system in an existing BSP. You must create a new BSP based on the desired operating system.

- The BSP target directory—the destination for files that the SBT copies and creates for your BSP.
- BSP settings

BSP settings appear in a tree structure. Settings are organized into **Common** and **Advanced** categories. Settings are further organized into functional groups. The available settings depend on the operating system.

When you select a group of settings, the controls for those settings appear in the pane to the right of the tree. When you select a single setting, the pane shows the setting control, the full setting name, and the setting description.

Software package and driver settings are presented separately, as described in "The Software Packages Tab" and "The Drivers Tab".

The Software Packages Tab

The **Software Packages** tab allows you to insert and remove software packages in your BSP, and control software package settings.

At the top of the **Software Packages** tab is the software package table, listing each available software package. The table allows you to select the software package version, and enable or disable the software package.

The operating system determines which software packages are available.

Many software packages define settings that you can control in your BSP. When you enable a software package, the available settings appear in a tree structure, organized into **Common** and **Advanced** settings.

When you select a group of settings, the controls for those settings appear in the pane to the right of the tree. When you select a single setting, the pane shows the setting control, the full setting name, and the setting description.

Enabling and disabling software packages and editing software package settings can have a profound impact on BSP behavior. Refer to the documentation for the specific software package for details.

For the read-only zip file system, refer to the *Read-Only Zip File System* chapter of the *Nios II Software Developer's Handbook*. For the NicheStack TCP/IP Stack - Nios II Edition, refer to the *Ethernet and the NicheStack TCP/IP Stack - Nios II Edition* chapter of the *Nios II Software Developer's Handbook*.

General settings, operating system settings, and driver settings are presented separately, as described in "The Main Tab" and "The Drivers Tab".

The Drivers Tab

The **Drivers** tab allows you to select, enable, and disable drivers for devices in your system, and control driver settings.

At the top of the **Drivers** tab is the driver table, mapping components in the hardware system to drivers. The driver table shows components with driver support. Each component has a module name, module version, module class name, driver name, and driver version, determined by the contents of the hardware system. The table allows you to select the driver by name and version, as well as to enable or disable each driver.

When you select a driver version, all instances of that driver in the BSP are set to the version you select. Only one version of a given driver can be used in an individual BSP.

Many drivers define settings that you can control in your BSP. Available driver settings appear in a tree structure below the driver table, organized into **Common** and **Advanced** settings.

When you select a group of settings, the controls for those settings appear in the pane to the right of the tree. When you select a single setting, the pane shows the setting control, the full setting name, and the setting description.

☞ Enabling and disabling device drivers, changing drivers and driver versions, and editing driver settings, can have a profound impact on BSP behavior. Refer to the relevant component documentation and driver information for details. For Altera components, refer to the *Embedded Peripherals IP User Guide*.

General settings, operating system settings, and software package settings are presented separately, as described in "The Main Tab" and "The Software Packages Tab".

The Linker Script Tab

The **Linker Script** tab allows you to view available memory in your hardware system, and examine and modify the arrangement and usage of linker regions in memory.

When you make a change to the memory configuration, the SBT validates your change. If there is a problem, a message appears in the **Problems** tab in the console area, as described in "The Problems Tab" on page 2–18.

☞ Rearranging linker regions and linker section mappings can have a very significant impact on BSP behavior.

Linker Section Mappings

At the top of the **Linker Script** tab, the **Linker Section Mappings** table shows the mapping from linker sections to linker regions. You can edit the BSP linker section mappings using the following buttons located next to the linker section table:

- **Add**—Adds a linker section mapping to an existing linker region. The **Add** button opens the **Add Section Mapping** dialog box, where you specify a new section name and an existing linker region.
- **Remove**—Removes a mapping from a linker section to a linker region.
- **Restore Defaults**—Restores the section mappings to the default configuration set up at the time of BSP creation.

Linker Regions

At the bottom of the **Linker Script** tab, the **Linker Memory Regions** table shows all defined linker regions. Each row of the table shows one linker region, with its address range, memory device name, size, and offset into the selected memory device.

You reassign a defined linker region to a different memory device by selecting a different device name in the **Memory Device Name** column. The **Size** and **Offset** columns are editable. You can also edit the list of linker regions using the following buttons located next to the linker region table:

- **Add**—Adds a linker region in unused space on any existing device. The **Add** button opens the **Add Memory Region** dialog box, where you specify the memory device, the new memory region name, the region size, and the region's offset from the device base address.
- **Remove**—Removes a linker region definition. Removing a region frees the region's memory space to be used for other regions.
- **Add Memory Device**—Creates a linker region representing a memory device that is outside the hardware system. The button launches the **Add Memory Device** dialog box, where you can specify the device name, memory size and base address. After you add the device, it appears in the linker region table, the **Memory Device Usage Table** dialog box, and the **Memory Map** dialog box.

  This functionality is equivalent to the `add_memory_device` Tcl command.

  Ensure that you specify the correct base address and memory size. If the base address or size of an external memory changes, you must edit the BSP manually to match. The SBT does not automatically detect changes in external memory devices, even if you update the BSP by creating a new settings file.

  For information about `add_memory_device` and other SBT Tcl commands, refer to "Software Build Tools Tcl Commands" in the *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer's Handbook.*

- **Restore Defaults**—restores the memory regions to the default configuration set up at the time of BSP creation.

- **Memory Usage**—Opens the **Memory Device Usage Table**. The **Memory Device Usage Table** allows you to view memory device usage by defined memory region. As memory regions are added, removed, and adjusted, each device's free memory, used memory, and percentage of available memory are updated. The rightmost column is a graphical representation of the device's usage, according to the memory regions assigned to it.
- **Memory Map**—Opens the **Memory Map** dialog box. The memory map allows you to view a map of system memory in the processor address space. The **Device** table is a read-only reference showing memories in the hardware system that are mastered by the selected processor. Devices are listed in memory address order.

  To the right of the **Device** table is a graphical representation of the processor's memory space, showing the locations of devices in the table. Gaps indicate unmapped address space.

  This representation is not to scale.

Enable File Generation Tab

The **Enable File Generation** tab allows you to take ownership of specific BSP files that are normally generated by the SBT. When you take ownership of a BSP file, you can modify it, and prevent the SBT from overwriting your modifications. The **Enable File Generation** tab shows a tree view of all target files to be generated or copied when the BSP is generated. To disable generation of a specific file, expand the software component containing the file, expand any internal directory folders, select the file, and right-click. Each disabled file appears in a list at the bottom of the tab.

This functionality is equivalent to the `set_ignore_file` Tcl command.

If you take ownership of a BSP file, the SBT can no longer update it to reflect future changes in the underlying hardware. If you change the hardware, be sure to update the file manually.

For information about `set_ignore_file` and other SBT Tcl commands, refer to "Software Build Tools Tcl Commands" in the *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer's Handbook.*

Target BSP Directory Tab

The **Target BSP Directory** tab is a read-only reference showing you what output to expect when the BSP is generated. It does not depict the actual file system, but rather the files and directories to be created or copied when the BSP is generated. Each software component, including the operating system, drivers, and software packages, specifies source code to be copied into the BSP target directory. The files are generated in the directory specified on the **Main** tab.

When you generate the BSP, existing BSP files are overwritten, unless you disable generation of the file in the **Enable File Generation** tab.

The Console Area

The console area shows results of settings and commands that you select in the command area. The console area consists of the following tabs:

- The **Information** tab
- The **Problems** tab
- The **Processing** tab

The following sections describe each tab.

The Information Tab

The **Information** tab shows a running list of high-level changes you make to your BSP, such as adding a software package or changing a setting value.

The Problems Tab

The **Problems** tab shows warnings and errors that impact or prohibit BSP creation. For example, if you inadvertently specify an invalid linker section mapping, a message appears in the **Problems** tab.

The Processing Tab

When you generate your BSP, the **Processing** tab shows files and folders created and copied in the BSP target directory.

Exporting a Tcl Script

When you have configured your BSP to your satisfaction, you can export the BSP settings as a Tcl script. This feature allows you to perform the following tasks:

- Regenerate the BSP from the command line
- Recreate the BSP as a starting point for a new BSP
- Recreate the BSP on a different hardware platform
- Examine the Tcl script to improve your understanding of Tcl command usage

The exported Tcl script captures all BSP settings that you have changed since the previous time the BSP settings file was saved. If you export a Tcl script after creating a new BSP, the script captures all nondefault settings in the BSP. If you export a Tcl script after editing a pre-existing BSP, the script captures your changes from the current editing session.

To export a Tcl script, in the Tools menu, click **Export Tcl Script**, and specify a filename and destination path. The file extension is **.tcl**.

You can later run your exported script as a part of creating a new BSP.

To run a Tcl script during BSP creation, refer to "Using a Tcl Script in BSP Creation". For details about default BSP settings, refer to "Tcl Scripts for BSP Settings" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.For information about recreating and regenerating BSPs, refer to "Revising Your BSP" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

Creating a New BSP

To create a BSP in the Nios II BSP Editor, use the **New BSP** command in the File menu to open the **New BSP** dialog box. This dialog box controls the creation of a new BSP settings file. The BSP Editor loads this new BSP after the file is created.

In this dialog box, you specify the following parameters:

- The .**sopcinfo** file defining the hardware platform.
- The CPU name of the targeted processor.
- The BSP type and version.

  You can select the operating system only at the time you create the BSP. To change operating systems, you must create a new BSP.

- The operating system version.
- The name of the BSP settings file. It is created with file extension .**bsp**.
- Absolute or relative path names in the BSP settings file. By default, relative paths are enabled for filenames in the BSP settings file.
- An optional Tcl script that you can run to supply additional settings.

Normally, you specify the path to your .**sopcinfo** file relative to the BSP directory. This enables you to move, copy and archive the hardware and software files together. If you browse to the .**sopcinfo** file, or specify an absolute path, the Nios II BSP Editor offers to convert your path to the relative form.

Using a Tcl Script in BSP Creation

When you create a BSP, the **New BSP Settings File** dialog box allows you to specify the path and filename of a Tcl script. The Nios II BSP Editor runs this script after all other BSP creation steps are done, to modify BSP settings. This feature allows you to perform the following tasks:

- Recreate an existing BSP as a starting point for a new BSP
- Recreate a BSP on a different hardware platform
- Include custom settings common to a group of BSPs

The Tcl script can be created by hand, or exported from another BSP.

"Exporting a Tcl Script" describes how to create a Tcl script from an existing BSP. Refer to "Tcl Scripts for BSP Settings" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

BSP Validation Errors

If you modify a hardware system after basing a BSP on it, some BSP settings might no longer be valid. This is a very common cause of BSP validation errors. Eliminating these errors usually requires correcting a large number of interrelated settings.

If your modifications to the underlying hardware design result in BSP validation errors, the best practice is to update or recreate the BSP. Updating and recreating BSPs is very easy with the BSP Editor.

For complete information about updating and recreating BSPs, refer to "Revising Your BSP" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

If you recreate your BSP, you might find it helpful to capture your old BSP settings by exporting them to a Tcl script. You can edit the Tcl script to remove any settings that are incompatible with the new hardware design.

For details about exporting and using Tcl scripts, refer to "Exporting a Tcl Script" and "Using a Tcl Script in BSP Creation". For a detailed discussion of updating BSPs for modified hardware systems, refer to "Revising Your BSP" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

Configuring Component Search Paths

By default, the SBT discovers system components using the same search algorithm as SOPC Builder or Qsys. You can define additional search paths to be used for locating components.

You define additional search paths through the **Edit Custom Search Paths** dialog box. In the Tools menu, click **Options**, select **BSP Component Search Paths**, and click **Custom Component Search Paths**. You can specify multiple search paths. Each path can be recursive.

Run Configurations in the SBT for Eclipse

Eclipse uses run configurations to control how it runs and debugs programs. Run configurations in the Nios II SBT for Eclipse have several features that help you debug Nios II software running on FPGA platforms.

You can open the run configuration dialog box two ways:

- You can right-click an application, point to **Run As**, and click **Run Configurations**.
- You can right-click an application, point to **Debug As**, and click **Debug Configurations**.

Depending on which way you opened the run configuration dialog box, the title is either **Run Configuration** or **Debug Configuration**. However, both views show the same run configurations.

If your project was created with version 10.1 or earlier of the Nios II SBT, you must re-import it to create the Nios II launch configuration correctly.

Each run configuration is presented on several tabs. This section describes each tab.

The Project Tab

On this tab, you specify the application project to run. The **Advanced** button opens the **Nios II ELF Section Properties** dialog box. In this dialog box, you can control the runtime parameters in the following ways:

- Specify the processor on which to execute the program (if the hardware design provides multiple processors)
- Specify the device to use for standard I/O
- Specify the expected location, timestamp and value of the system ID
- Specify the path to the Quartus II JTAG Debugging Information File (**.jdi**)
- Enable or disable profiling

The Nios II SBT for Eclipse sets these parameters to reasonable defaults. Do not modify them unless you have a clear understanding of their effects.

The Target Connection Tab

This tab allows you to control the connection between the host machine and the target hardware in the following ways:

- Select the cable, if more than one cable is available
- Allow software to run despite a system ID value or timestamp that differs from the hardware
- Reset the processor when the software is downloaded

The **System ID Properties** button allows you to examine the system ID and timestamp in both the .**elf** file and the hardware. This can be helpful when you need to analyze the cause of a system ID or timestamp mismatch.

The Debugger Tab

In this tab, you optionally enable the debugger to halt at a specified entry point.

Nios II Hardware v2 (beta)

Starting with version 13.1, run configurations and debug configurations have a launch type called Nios II Hardware v2 (beta). To create this launch type, in the Run menu select either **Run Configurations** or **Debug Configurations**. In the **Run/Debug Configurations** dialog box, select **Nios II Hardware v2 (beta)** and click the **New** button to create a new launch configuration.

Nios II Hardware v2(beta) has options below.

The Main Tab

This tab allows you to select the following options:

- Specify the application project to run and the ELF File location
- Specify the processor and the JTAG UART connection to use
- Enable or disable system ID and timestamp checks
- Enable or disable processor controls such as download ELF, reset processor or start processor

The Debugger Tab

In this tab, you optionally enable the debugger to halt at a specified entry point.

Multi-Core Launches

If you have multiple run configurations, create an Eclipse launch group. Launch groups are an Eclipse feature that allows multiple run configurations to be started at the same time. You choose which run configurations are added to the group. You can use the launch group in any place where you can use a run configuration.

For details about Eclipse launch groups, refer to the Eclipse help system.

Optimizing Project Build Time

When you build a Nios II project, the project makefile builds any components that are unbuilt or out of date. For this reason, the first time you build a project is normally the slowest. Subsequent builds are fast, only rebuilding sources that have changed.

To further optimize your project build time, disable generation of the objdump linker map.

Nios II software build performance is generally better on Linux platforms than on Windows platforms.

Importing a Command-Line Project

If you have software projects that were created with the Nios II SBT command line, you can import the projects into the Nios II SBT for Eclipse for debugging and further development. This section discusses the import process.

Your command-line C/C++ application, and its associated BSP, is created on the command line. Any Nios II SBT command-line project is ready to import into the Nios II SBT for Eclipse. No additional preparation is necessary.

The Nios II SBT for Eclipse imports the following kinds of Nios II command-line projects:

- Command-line C/C++ application project
- Command-line BSP project
- Command-line user library project

You can edit, build, debug, and manage the settings of an imported project exactly the same way you edit, build, debug, and manage the settings of a project created in Nios II SBT for Eclipse.

The Nios II SBT for Eclipse imports each type of project through the **Import** wizard. The **Import** wizard determines the kind of project you are importing, and configures it appropriately.

You can continue to develop project code in your SBT project after importing the project into Eclipse. You can edit source files and rebuild the project, using the SBT either in Eclipse or on the command line.

For information about creating projects with the command line, refer to the *Getting Started from the Command Line* chapter of the *Nios II Software Developer's Handbook*.

Road Map

Importing and debugging a project typically involves several of the following tasks. You do not need to perform these tasks in this order, and you can repeat or omit some tasks, depending on your needs.

- Import a command-line C/C++ application
- Import a supporting project
- Debug a command-line C/C++ application
- Edit command-line C/C++ application code

When importing a project, the SBT for Eclipse might make some minor changes to your makefile. If the makefile refers to a source file located outside the project directory tree, the SBT for Eclipse treats that file as a linked resource. However, it does not add or remove any source files to or from your makefile.

When you import an application or user library project, the Nios II SBT for Eclipse allows you to choose Eclipse source management or user source management. Unless your project has an unusual directory structure, choose Eclipse source management, to allow the SBT for Eclipse to automatically maintain your list of source files.

You debug and edit an imported project exactly the same way you debug and edit a project created in Eclipse.

Import a Command-Line C/C++ Application

To import a command-line C/C++ application, perform the following steps:

1. Start the Nios II SBT for Eclipse.
2. On the File menu, click **Import**. The **Import** dialog box appears.
3. Expand the **Nios II Software Build Tools Project** folder, and select **Import Nios II Software Build Tools Project**.
4. Click **Next**. The **File Import** wizard appears.
5. Click **Browse** and locate the directory containing the C/C++ application project to import.
6. Click **OK**. The wizard fills in the project path.

7. Specify the project name in the **Project name** box.

   You might see a warning saying “There is already a **.project** file at: <*path*>”. This warning indicates that the directory already contains an Eclipse project. Either it is an Eclipse project, or it is a command-line project that is already imported into Eclipse.

   If the project is already in your workspace, do not re-import it.

8. Click **Finish**. The wizard imports the application project.

After you complete these steps, the Nios II SBT for Eclipse can build, debug, and run the complete program, including the BSP and any libraries. The Nios II SBT for Eclipse builds the project using the SBT makefiles in your imported C/C++ application project. Eclipse displays and steps through application source code exactly as if the project were created in the Nios II SBT for Eclipse. However, Eclipse does not have direct information about where BSP or user library code resides. If you need to view, debug or step through BSP or user library source code, you need to import the BSP or user library. The process of importing supporting projects, such as BSPs and libraries, is described in “Import a Supporting Project”.

Importing a Project with Absolute Source Paths

If your project uses an absolute path to refer to a source file, the SBT for Eclipse imports that source file as a linked resource. In this case, the import wizard provides a page where you can manage how Eclipse refers to the source: as a file, or through a parent directory.

For information about managing linked resources, refer to “Absolute Source Paths and Linked Resources” on page 2–10.

Import a Supporting Project

While debugging a C/C++ application, you might need to view, debug or step through source code in a supporting project, such as a BSP or user library. To make supporting project source code visible in the Eclipse debug perspective, you need to import the supporting project.

If you do not need BSP or user library source code visible in the debugger, you can skip this task, and proceed to debug your project exactly as if you had created it in Eclipse.

If you have several C/C++ applications based on one BSP or user library, import the BSP or user library once, and then import each application that is based on the BSP or user library. Each application’s makefile contains the information needed to find and build any associated BSP or libraries.

The steps for importing a supporting project are exactly the same as those shown in “Import a Command-Line C/C++ Application”.

User-Managed Source Files

When you import a Nios II application or user library project, the Nios II SBT for Eclipse offers the option of user source management. User source management is helpful if you prefer to update your makefile manually to reflect source files added to or removed from the project.

With user source management, Eclipse never makes any changes to the list of source files in your makefile. However, the SBT for Eclipse manages all other project parameters and settings, just as with any other Nios II software project.

If your makefile refers to a source file with an absolute path, when you import with user source management, the absolute path is untouched, like any other source path. You might use an absolute path to refer to common source files in a fixed location. In this situation, you can move the project to a different directory without disturbing the common source file references.

User source management is not available with BSP projects. BSP makefiles are based on the operating system, BSP settings, selected software packages, and selected drivers. You do not specify BSP source files directly.

For details about how the SBT for Eclipse handles makefiles with user-managed sources, refer to "User Source Management" on page 2–11.

Packaging a Library for Reuse

This section shows how to create and use a library archive file (**.a**) in the Nios II Software Build Tools for Eclipse. This technique enables you to provide a library to another engineer or organization without providing the C source files. This process entails two tasks:

1. Create a Nios II user library
2. Create a Nios II application project based on the user library

Creating the User Library

To create a user library, perform the following steps:

1. In the File menu, point to **New** and click **Nios II Library**.
2. Type a project name, for example `test_lib`.
3. For **Location**, browse to the directory containing your library source files (**.c** and **.h**).
4. Click **Finish**.
5. Build the project to create the **.a** file (in this case **libtest_lib.a**)

Using the Library

To use the library in a Nios II application project, perform the following steps:

1. Create your Nios II application project as described in "Creating a Project" on page 2–2.
2. To set the library path in the application project, right-click the project, and click **Properties**.
3. Expand **Nios II Application Properties**. In **Nios II Application Paths**, next to **Application include directories**, click **Add** and browse to the directory containing your library header files.
4. Next to **Application library directories**, click **Add** and browse to the directory containing your **.a** file.
5. Next to **Library name**, click **Add** and type the library project name you selected in "Creating the User Library".
6. Click **OK**.
7. Build your application.

As this example shows, the **.c** source files are not required to build the application project. To hand off the library to another engineer or organization for reuse, you provide the following files:

- Nios II library archive file (**.a**)
- Software header files (**.h**)

Creating a Software Package

This section shows how you can build a custom library into a BSP as a software package. The software package can be linked to any BSP through the BSP Editor.

This section contains an example illustrating the steps necessary to include any software package into a Nios II BSP.

To create and exercise the example software package, perform the following steps:

1. Locate the **ip** directory in your Altera Complete Design Suite installation. For example, if the Altera Complete Design Suite version 11.0 is installed on the Windows operating system, the directory might be **c:\altera\11.0\ip**. Under the **ip** directory, create a directory for the software package. For simplicity, this section refers to this directory as <*example package*>.
2. In <*example package*>, create a subdirectory named **EXAMPLE_SW_PACKAGE**. In <*example package*>**/EXAMPLE_SW_PACKAGE**, create two subdirectories named **inc** and **lib**.
3. In <*example package*>**/EXAMPLE_SW_PACKAGE/inc**, create a new header file named **example_sw_package.h**. Insert the code shown in Example 2–1.

Example 2–1. Contents of example_sw_package.h

```
/* Example Software Package */

void example_sw_package(void);
```

4. In <*example package*>**/EXAMPLE_SW_PACKAGE/lib**, create a new C source file named **example_sw_package.c**. Insert the code shown in Example 2–2.

Example 2–2. Contents of example_sw_package.c

```
/* Example Software Package  */
#include <stdio.h>
#include "..\inc\example_sw_package.h"

void example_sw_package(void)
{
    printf ("Example Software Package. \n");
}
```

5. In <*example package*>, create a new Tcl script file named **example_sw_package_sw.tcl**. Insert the code shown in Example 2–3.
6. In the SBT for Eclipse, create a Nios II application and BSP project based on the Hello World template. Set the application project name to `hello_example_sw_package`.

Example 2–3. Contents of example_sw_package_sw.tcl

```
#
# example_sw_package_sw.tcl
#

# Create a software package known as "example_sw_package"
create_sw_package example_sw_package

# The version of this software
set_sw_property version 11.0

# Location in generated BSP that sources should be copied into
set_sw_property bsp_subdirectory Example_SW_Package

#
# Source file listings...
#

# C/C++ source files
#add_sw_property c_source EXAMPLE_SW_PACKAGE/src/my_source.c

# Include files
add_sw_property include_source
EXAMPLE_SW_PACKAGE/inc/example_sw_package.h

# Lib files
add_sw_property lib_source
EXAMPLE_SW_PACKAGE/lib/libexample_sw_package_library.a

# Include paths for headers which define the APIs for this package
# to share w/ app & bsp
# Include paths are relative to the location of this software
# package tcl file

add_sw_property include_directory EXAMPLE_SW_PACKAGE/inc

# This driver supports HAL & UCOSII BSP (OS) types
add_sw_property supported_bsp_type HAL
add_sw_property supported_bsp_type UCOSII

# Add example software package system.h setting to the BSP:
add_sw_setting quoted_string system_h_define \
  example_sw_package_system_value EXAMPLE_SW_PACKAGE_SYSTEM_VALUE 1 \
  "Example software package system value"

# End of file
```

7. Create a new C file named **hello_example_sw_package.c** in the new application project. Insert the code shown in Example 2–4.

Example 2–4. Contents of hello_example_sw_package.c

```
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It also
 * tests inclusion of a user software package.
 */

#include <stdio.h>
#include "example_sw_package.h"

int main()
{
  printf("Hello from Nios II!\n");
  example_sw_package();
  return 0;
}
```

8. Delete **hello_world.c** from the hello_example_sw_package application project.
9. In the File menu, point to **New** and click **Nios II Library**
10. Set the project name to `example_sw_package_library`.
11. For **Location**, browse to *<example package>***EXAMPLE_SW_PACKAGE\lib**

    Building the library here is required, because the resulting **.a** is referenced here by **example_sw_package_sw.tcl**.

12. Click **Finish**.
13. Build the example_sw_package_library project to create the **libexample_sw_package_library.a** library archive file.
14. Right-click the BSP project, point to **Nios II**, and click **BSP Editor** to open the BSP Editor.
15. In the **Software Packages** tab, find example_sw_package in the software package table, and enable it.

    If there are any errors in a software package's ***_sw.tcl** file, such as an incorrect path that causes a file to not be found, the software package does not appear in the BSP Editor.
16. Click the **Generate** button to regenerate the BSP. On the File menu, click **Save** to save your changes to **settings.bsp**.
17. In the File menu, click **Exit** to exit the BSP Editor.
18. Build the hello_example_sw_package_bsp BSP project.
19. Build the hello_example_sw_package application project.

hello_example_sw_package.elf is ready to download and execute.

Programming Flash in Altera Embedded Systems

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 SBT for Eclipse provides flash programmer utilities to help you manage and program the contents of flash memory. The flash programmer allows you to program any combination of software, hardware, and binary data into flash memory in one operation.

Starting the Flash Programmer

You start the flash programmer by clicking **Flash Programmer** in the Nios II menu.

When you first open the flash programmer, no controls are available until you open or create a Flash Programmer Settings File (**.flash-settings**).

Creating a Flash Programmer Settings File

The **.flash-settings** file describes how you set up the flash programmer GUI to program flash. This information includes the files to be programmed to flash, a **.sopcinfo** file describing the hardware configuration, and the file programming locations. You must create or open a flash programmer settings file before you can program flash.

You create a flash programmer settings file through the File menu. When you click **New**, the **New Flash Programmer Settings File** dialog box appears.

Specifying the Hardware Configuration

You specify the hardware configuration by opening a **.sopcinfo** file. You can locate the **.sopcinfo** file in either of two ways:

- Browse to a BSP settings file. The flash programmer finds the **.sopcinfo** file associated with the BSP.
- Browse directly to a **.sopcinfo** file.

Once you have identified a hardware configuration, details about the target hardware appear at the top of the Nios II flash programmer screen.

Also at the top of the Nios II flash programmer screen is the **Hardware Connections** button, which opens the **Hardware Connections** dialog box. This dialog box allows you to select a download cable, and control system ID behavior, as described in "The Target Connection Tab" on page 2–21.

The Flash Programmer Screen Layout

The flash programmer screen is divided into two areas. The top area is the command area, and the bottom is the console area. The details of the flash programmer screen areas are described in this section.

Below the console area is the **Start** button. This button is enabled when the flash programmer parameters are valid. It starts the process of programming flash.

The Command Area

In the command area, you specify settings and other parameters defining the flash programmer settings file. The command area contains one or more tabs. Each tab represents a flash memory component available in the target hardware. Each tab allows you to view the parameters of the memory component, and view and edit the list of files to be programmed in the component.

The **Add** and **Remove** buttons allow you to create and edit the list of files to be programmed in the flash memory component.

The **File generation command** box shows the commands used to generate the Motorola S-record Files (**.flash**) used to program flash memory.

The **File programming command** box shows the commands used to program the **.flash** files to flash memory.

The **Properties** button opens the **Properties** dialog box, which allows you to view and modify information about an individual file. In the case of a **.elf**, the **Properties** button provides access to the project reset address, the flash base and end addresses, and the boot loader file (if any).

The flash programmer determines whether a boot loader is required based on the load and run locations of the `.text` section. You can use the **Properties** dialog box to override the default boot loader configuration.

The Console Area

The console area shows results of settings and commands that you select in the command area. The console area consists of the following tabs:

- The **Information** tab
- The **Problems** tab
- The **Processing** tab

This section describes each tab.

The Information Tab

The **Information** tab shows the high-level changes you make to your flash programmer settings file.

The Problems Tab

The **Problems** tab shows warnings and error messages about the process of flash programmer settings file creation.

The Processing Tab

When you program flash, the **Processing** tab shows the individual programming actions as they take place.

Saving a Flash Programmer Settings File

When you have finished configuring the input files, locations, and other settings for programming your project to flash, you can save the settings in a **.flash-settings** file. With a **.flash-settings** file, you can program the project again without reconfiguring the settings. You save a **.flash-settings** file through the File menu.

Flash Programmer Options

Through the Options menu, you can control several global aspects of flash programmer behavior, as described in this section.

For details about these features, refer to the *Nios II Flash Programmer User's Guide*.

Staging Directories

Through the **Staging Directories** dialog box, you control where the flash programmer creates its script and **.flash-settings** files.

Generate Files

If you disable this option, the flash programmer does not generate programming files, but programs files already present in the directory. You might use this feature to reprogram a set of files that you have previously created.

Program Files

If you disable this option, the flash programmer generates the programming files and the script, but does not program flash. You can use the files later to program flash by turning off the **Generate Files** option.

Erase Flash Before Programming

When enabled, this option erases flash memory before programming.

Run From Reset After Programming

When enabled, this option resets and starts the Nios II processor after programming flash.

Creating Memory Initialization Files

Sometimes it is useful to generate memory initialization files. For example, to program your FPGA with a complete, running Nios II system, you must include the memory contents in your **.sof** file. In this configuration, the processor can boot directly from internal memory without downloading.

Creating a Hexadecimal (Intel-Format) File (**.hex**) is a necessary intermediate step in creating such a **.sof** file. The Nios II SBT for Eclipse can create **.hex** files and other memory initialization formats.

To generate correct memory initialization files, the Nios II SBT needs details about the physical memory configuration and the types of files required. Typically, this information is specified when the hardware system is generated.

If your system contains a user-defined memory, you must specify these details manually. For information, see "Memory Initialization Files for User-Defined Memories".

To generate memory initialization files, perform the following steps:

1. Right-click the application project.
2. Point to **Make targets** and click **Build** to open the **Make Targets** dialog box.
3. Select **mem_init_generate**.
4. Click **Build**. The makefile generates a separate file (or files) for each memory device. It also generates a Quartus II IP File (**.qip**). The **.qip** file tells the Quartus II software where to find the initialization files.
5. Add the **.qip** file to your Quartus II project.
6. Recompile your Quartus II project.

If your hardware system was generated with SOPC Builder, you can alternatively use the legacy method to generate memory initialization files. However, this method is not preferred. To generate memory initialization files by the legacy method, perform the following steps:

1. Right-click the application project.
2. Point to **Make targets** and click **Build** to open the **Make Targets** dialog box.
3. Select **mem_init_install**.
4. Click **Build**. The makefile generates a separate file (or files) for each memory device. The makefile inserts the memory initialization files directly in the Quartus II project directory for you.
5. Recompile your Quartus II project.

For more information about creating memory initialization files, refer to "Common BSP Tasks" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

Memory Initialization Files for User-Defined Memories

Generating memory initialization files requires detailed information about the physical memory devices, such as device names and data widths. Normally, the Nios II SBT extracts this information from the **.sopcinfo** file. However, in the case of a user-defined memory, the **.sopcinfo** file does not contain information about the data memory, which is outside the system. Therefore, you must provide this information manually.

You specify memory device information when you add the user-defined memory device to your BSP. The device information persists in the BSP settings file, allowing you to regenerate memory initialization files at any time, exactly as if the memory device were part of the hardware system.

Specify the memory device information in the **Advanced** tab of the **Add Memory Device** dialog box. Settings in this tab control makefile variables in **mem_init.mk**.

On the **Advanced** tab, you can control the following memory characteristics:

- The physical memory width.
- The device's name in the hardware system.
- The memory initialization file parameter name. Every memory device can have an HDL parameter specifying the name of the initialization file. The Nios II ModelSim launch configuration overrides the HDL parameter to specify the memory initialization filename. When available, this method is preferred for setting the memory initialization filename.

  For further information about this parameter, refer to "Embedded Software Assignments" in the *Publishing Component Information to Embedded Software* chapter of the *Nios II Software Developer's Handbook*.

- The **Mem init filename** parameter can be used in Nios II systems as an alternative method of specifying the memory initialization filename. The **Mem init filename** parameter directly overrides any filename specified in the HDL.
- Connectivity to processor master ports. These parameters are used when creating the linker script.
- The memory type: volatile, CFI flash or EPCS flash.
- Byte lanes.

You can also enable and disable generation of the following memory initialization file types:

- **.hex** file
- **.dat** and **.sym** files
- **.flash** file

Running a Nios II System with ModelSim

You can run a Nios II program on Nios II hardware, such as an Altera development board, or you can run it in the Nios II ModelSim® simulation environment.

If your project was created with version 10.1 or earlier of the Nios II SBT, you must re-import it to create the Nios II launch configuration correctly.

Using ModelSim with an SOPC Builder-Generated System

If your hardware system was generated by SOPC Builder, running a software project in ModelSim is very similar to running it on Nios II hardware. Follow the instructions in "Running the Project on Nios II Hardware" on page 2–5, except that when you right-click the application project name, point to **Run As**, and click **Nios II ModelSim**.

Similarly, to debug a software project in ModelSim, right-click the application project name, point to **Debug As**, and click **Nios II ModelSim**.

Using ModelSim with a Qsys-Generated System

To run a Qsys-generated Nios II system with ModelSim, you must first create a simulation model and testbench, and specify memory initialization files. You create your Nios II simulation model and testbench using the steps that apply to any Qsys design.

Refer to "Qsys Design Flow" in the *Creating a System with Qsys* chapter in Volume 1 of the *Quartus II Handbook*.

Creating the software projects is nearly the same as when you run the project on hardware. To prepare your software for ModelSim simulation, perform the following steps:

1. Create your software project, as described in "Creating a Project" on page 2–2.

   Be sure to specify the Quartus II project path, as described in "Creating a Simple BSP" on page 2–8.

   If you need to initialize a user-defined memory, you must take special steps to create memory initialization files correctly. These steps are described in "Memory Initialization Files for User-Defined Memories" on page 2–33.

2. Build your software project, as described in "Building the Project" on page 2–5.
3. Create a ModelSim launch configuration with the following steps:
   a. Right-click the application project name, point to **Run As**, and click **Run Configurations**. In the **Run Configurations** dialog box, select **Nios II ModelSim**, and click the **New** button.
   b. In the **Main** tab, ensure that the correct software project name and **.elf** file are selected.
   c. Click **Apply** to save the launch configuration.
   d. Click **Close** to close the dialog box.

   If you are simulating multiple processors, create a launch configuration for each processor, and create a launch group, as described in "Multi-Core Launches" on page 2–22.

4. Open the run configuration you previously created. Click **Run**. The Nios II SBT for Eclipse performs a `make mem_init_generate` command to create memory initialization files, and launches ModelSim.
5. At the ModelSim command prompt, type `ld`↵.

When you create the launch configuration, you might see the following error message:

SEVERE: The Quartus II project location has not been set in the ELF section. You can manually override this setting in the launch configuration's ELF file 'Advanced' properties page.

To correct this error, perform the following steps:

1. Click the **Advanced** button.
2. In the **Quartus II project directory** box, browse to locate the directory containing your Quartus II project **.spd** file.
3. Click **Close**.

To avoid this error condition, specify the Quartus II project directory when you create your application project, as described in "Creating a Simple BSP" on page 2–8.

Starting with version 13.1, run configurations has the launch type Nios II Hardware v2 (beta). To create this launch type, in the Run menu select **Run Configurations**. In the **Run Configurations** dialog box, select **Nios II Hardware v2 (beta)** and click the **New** button to create a new launch configuration. Nios II Hardware v2 (beta) has the following options:

- Specify the application project to run and the ELF file location
- Specify the SPD file location and Modelsim path
- Specify the SPD file

Nios II GCC Tool chain upgrade from GCC 4.1.2 to GCC 4.7.3

In Nios II EDS version 13.1, the Nios® II GNU tool chain is upgraded from GCC 4.1.2 to GCC 4.7.3. When upgrading to the new tool chain you should note the following changes.

Nios II specific changes:

Use __buildin_custom_* instead of -mcustom-* or #pragma to reliably generate Nios II Floating Point Custom Instructions (FPCI), independent of compiler optimization level and command line flags.

To use -mcustom-* or #pragma for Nios II Floating Point Custom Instructions (FPCI):

the -ffinite-math-only flag must be used to generate fmins and fmax FPCI

the optimization (non -O0 flag) must be used to generate fsqrts FPCI

Users implementing transcendental functions in hardware must use the -funsafe-math-optimizations flag to generate the FPCI for the transcendental functions fsins(), fcoss(), ftans(), fatans(), fexps(), flogs() and corresponding double-precision functions

The Pragma format has changed from eg. #pragma custom_fadds 253 to #pragma GCC target("custom-fadds=253") and function attributes provide an alternative format __attribute__((target("custom-fadds=253"))).

Use the -mel/-meb flags instead of -EL/-EB for endian settings. Software Build Tool for Eclipse (SBTE) users must regenerate the BSP for this setting to take effect.

The -mreverse-bitfields flag and reverse_bitfields pragma are no longer supported.

The -fstack-check flag must be used instead of -mstack-check to enable stack checking.

GCC changes and enhancements:

The -Wa,-relax-all flag in nios2-elf-gcc GCC 4.7.3 supports function calls and programs exceeding the 256MB limit.

When used with optimization, inline assembly code with the asm operator needs to declare values imported from C and exported back to C, using the mechanisms described in "http://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html" \l "Extended-Asm", http://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html#Extended-Asm.

Pre-standard C++ headers are not supported in GCC 4.7.3. Replace pre-standard C++ with standard C++ eg. #include <iostream.h>, cout, endl with #include <iostream>, std::cout and std::endl respectively.

The compile flag -Wl,--defsym foo=bar where bar is an undefined symbol, will generate error at the linker level in GCC 4.7.3. GCC 4.1.2 does not include this check.

GNU also provides a porting guide to GCC4.7 to document common issues at :*http://gcc.gnu.org/gcc-4.7/porting_to.html*

Full GCC release notes are available at *http://gcc.gnu.org/releases.html*.

For general information about the GCC toolchains, refer to “Altera-Provided Development Tools” in the *Nios II Software Build Tools* chapter in 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*. For detailed information about installing the Altera Complete Design Suite, refer to the *Altera Software Installation and Licensing Manual*.

Eclipse Usage Notes

The behavior of certain Eclipse and CDT features is modified by the Nios II SBT for Eclipse. If you attempt to use these features the same way you would with a non-Nios II project, you might have problems configuring or building your project. This section discusses such features.

Configuring Application and Library Properties

To configure project properties specific to Nios II SBT application and library projects, use the **Nios II Application Properties** and **Nios II Library Properties** tabs of the **Properties** dialog box. To open the appropriate properties tab, right-click the application or library project and click **Properties**. Depending on the project type, **Nios II Application Properties** or **Nios II Library Properties** tab appears in the list of tabs. Click the appropriate Properties tab to open it.

The **Nios II Application Properties** and **Nios II Library Properties** tabs are nearly identical. These tabs allow you to control the following project properties:

- The name of the target **.elf** file (application project only)
- The library name (library project only)
- A list of symbols to be defined in the makefile
- A list of symbols to be undefined in the makefile
- A list of assembler flags
- Warning level flags
- A list of user flags
- Generation of debug symbols
- Compiler optimization level
- Generation of object dump file (application project only)
- Source file management
- Path to associated BSP (required for application, optional for library)

Configuring BSP Properties

To configure BSP settings and properties, use the Nios II BSP Editor.

For detailed information about the BSP Editor, refer to "Using the BSP Editor" on page 2–12.

Exclude from Build Not Supported

The **Exclude from Build** command is not supported. You must use the **Remove from Nios II Build** and **Add to Nios II Build** commands instead.

This behavior differs from the behavior of the Nios II SBT for Eclipse in version 9.1.

Selecting the Correct Launch Configuration Type

If you try to debug a Nios II software project as a CDT Local C/C++ Application launch configuration type, you see an error message, and the Nios II Debug perspective fails to open. This is expected CDT behavior in the Eclipse platform. Local C/C++ Application is the launch configuration type for a standard CDT project. To invoke the Nios II plugins, you must use a Nios II launch configuration type.

☞ If your project was created with version 10.1 or earlier of the Nios II SBT, you must re-import it to create the Nios II launch configuration correctly.

Target Connection Options

The Nios II launch configurations offer the following Nios II-specific options in the **Target Connection** tab:

- Disable 'Nios II Console' view
- Ignore mismatched system ID

- Ignore mismatched system timestamp
- Download ELF to selected target system
- Start processor
- Reset the selected target system

Renaming Nios II Projects

To rename a project in the Nios II SBT for Eclipse, perform the following steps:

1. Right-click the project and click **Rename**.
2. Type the new project name.
3. Right-click the project and click **Refresh**.

If you neglect to refresh the project, you might see the following error message when you attempt to build it:

```
Resource <original_project_name> is out of sync with the system
```

Running Shell Scripts from the SBT for Eclipse

Many SBT utilities are implemented as shell scripts. You can use Eclipse external tools configurations to run shell scripts. However, you must ensure that the shell environment is set up correctly.

To run shell scripts from the SBT for Eclipse, execute the following steps:

1. Start the Nios II Command Shell, as described in the *Getting Started from the Command Line* chapter of the *Nios II Software Developer's Handbook*.
2. Start the Nios II SBT for Eclipse by typing the following command:

   ```
   eclipse-nios2↵
   ```

   You must start the SBT for Eclipse from the command line in both the Linux and Windows operating systems, to set up the correct shell environment.
3. From the Eclipse Run menu, point to **External Tools**, and click **External Tools Configurations**.
4. Create a new tools configuration, or open an existing tools configuration.

5. On the **Main** tab, set **Location** and **Argument** as shown in Table 2–3.

Table 2–3. Location and Argument to Run Shell Script from Eclipse

Platform	Location	Argument
Windows	`${env_var:QUARTUS_ROOTDIR}\bin\cygwin\bin\sh.exe`	`-c "`*<script name> <script args>*`"`
Linux	`${env_var:SOPC_KIT_NIOS2}/bin/`*<script name>*	*<script args>*

For example, to run the command `elf2hex --help`, set **Location** and **Argument** as shown in Table 2–4.

Table 2–4. Location and Argument to Run elf2hex --help from Eclipse

Platform	Location	Argument
Windows	`${env_var:QUARTUS_ROOTDIR}\bin\cygwin\bin\sh.exe`	`-c "elf2hex --help"`
Linux	`${env_var:SOPC_KIT_NIOS2}/bin/elf2hex`	`--help`

6. On the **Build** tab, ensure that **Build before launch** and its related options are set appropriately.

   By default, a new tools configuration builds all projects in your workspace before executing the command. This might not be the desired behavior.

7. Click **Run**. The command executes in the Nios II Command Shell, and the command output appears in the Eclipse **Console** tab.

Must Use Nios II Build Configuration

Although Eclipse can support multiple build configurations, you must use the Nios II build configuration for Nios II projects.

If your project was created with version 10.1 or earlier of the Nios II SBT, you must re-import it to create the Nios II launch configuration correctly.

CDT Limitations

The features listed in the left column of Table 2–5 are supported by the Eclipse CDT plugins, but are not supported by Nios II plugins. The right column lists alternative features supported by the Nios II plugins.

Table 2–5. Eclipse CDT Features Not Supported by the Nios II Plugins (Part 1 of 3)

Unsupported CDT Feature	Alternative Nios II Feature
New Project Wizard	
C/C++ ■ C Project ■ C++ Project	To create a new project, use one of the following Nios II wizards: ■ Nios II Application ■ Nios II Application and BSP from Template ■ Nios II Board Support Package ■ Nios II Library

Table 2–5. Eclipse CDT Features Not Supported by the Nios II Plugins (Part 2 of 3)

Unsupported CDT Feature	Alternative Nios II Feature
■ Convert to a C/C++ Project ■ Source Folder	
Build configurations	
■ Right-click project and point to **Build Configurations**	The Nios II plugins only support a single build configuration.
■ **Debugger** tab	
■ **Stop on startup**	This feature is supported only at the top of `main()`.
Exclude from Build (from version 10.0 onwards)	
Right-click source files	Use **Remove from Nios II Build** and **Add to Nios II Build**.
Project Properties	
C/C++ Build ■ Builder Settings	
■ Makefile generation	By default, the Nios II SBT generates makefiles automatically.
■ Build location	The build location is determined with the **Nios II Application Properties** or **Nios II BSP Properties** dialog box.
■ Behavior ■ Build on resource save (Auto build) ■ Build Variables ■ Discovery Options ■ Environment ■ Settings ■ Tool Chain Editor ■ Current builder	
■ Used tools	To change the toolchain, use the **Current tool chain** option
Project Properties, continued	
C/C++ General ■ Enable project specific settings ■ Documentation tool comments ■ Documentation ■ File Types ■ Indexer	
■ Build configuration for the indexer	The Nios II plugins only support a single build configuration.
■ Language Mappings	
■ Paths and Symbols	Use **Nios II Application Properties** and **Nios II Application Paths**

Table 2–5. Eclipse CDT Features Not Supported by the Nios II Plugins (Part 3 of 3)

Unsupported CDT Feature	Alternative Nios II Feature
Window Preferences	
C/C++	
■ Build scope	
■ Build project configurations	The Nios II plugins only support a single build configuration.
■ Build Variables	
■ Environment	
■ File Types	
■ Indexer	
■ Build configuration for the indexer	The Nios II plugins only support a single build configuration.
■ Language Mappings	
■ New CDT project wizard	

Document Revision History

Table 2–6 shows the revision history for this document.

Table 2–6. Document Revision History

Date	Version	Changes
January 2014	13.1.0	■ Added section on Nios II Hardware v2 beta ■ Updated GCC4 toolchain from 4.1.2 to GCC 4.7.3 ■ Removed “Managing Toolchains in Eclipse” section.
May 2011	11.0.0	■ Introduction of Qsys system integration tool impacts ModelSim flow ■ Launch configuration change requires re-importation of existing projects ■ Using variables to link to external resources ■ The GCC 3 toolchain is an optional feature ■ Minor corrections to Table 2–5 on page 2–54
February 2011	10.1.0	■ Do not mix versions of GCC. ■ How to create and use a library archive file (**.a**). ■ How to create a software package. ■ Describe Eclipse launch groups. ■ Removed “Referenced Documents” section.
July 2010	10.0.0	■ Document how to import and use projects with user-managed source files. ■ Document how to import and use projects with linked resources. ■ Document **Remove from Nios II Build** command. ■ Update BSP Editor documentation. ■ Document **Add Memory Device** command. ■ Document **Enable File Generation** tab.
November 2009	9.1.0	Initial release.