Altera provides various tools for development of hardware and software for embedded systems. This handbook complements the primary documentation for these tools by describing how to most effectively use some of these tools. It recommends design styles and practices for developing, debugging, and optimizing embedded systems using Altera-provided tools such as the Software Build Tools for Eclipse and SOPC Builder. The handbook introduces concepts to new users of Altera's embedded solutions, and helps to increase the design efficiency of the experienced user.

This handbook is not a comprehensive reference guide. For general reference and detailed information, refer to the primary documentation cited in this handbook.

This first chapter of the handbook contains information about the Altera® embedded development process and procedures for the first time user. The remaining chapters focus on specific aspects of embedded development for Altera FPGAs.

This handbook does not provide information about the Qsys system integration tool.

First Time Designer's Guide Introduction

This chapter is for first time users of Altera's embedded development tools for hardware and software development. The chapter provides information about the design flow and development tools interaction, and describes the differences between the Nios® II processor flow and a typical discrete microcontroller design flow.

However, this chapter does not replace the basic reference material for the first time designer, such as the *Nios II Processor Reference Handbook*, the *Nios II Software Developer's Handbook*, the *SOPC Builder User Guide*, the *Embedded Peripherals User Guide*, and the *Nios II Flash Programmer User Guide*.

FPGAs and Soft-Core Processors

FPGAs can implement logic that functions as a complete microprocessor while providing many flexibility options.

An important difference between discrete microprocessors and FPGAs is that an FPGA contains no logic when it powers up. Before you run software on a Nios II based system, you must configure the FPGA with a hardware design that contains a Nios II processor. To configure an FPGA is to electronically program the FPGA with a specific logic design. The Nios II processor is a true soft-core processor: it can be placed anywhere on the FPGA, depending on the other requirements of the design. Three different sizes of the processor are available, each with flexible features.

To enable your FPGA-based embedded system to behave as a discrete microprocessor-based system, your system should include the following:

■ A JTAG interface to support FPGA configuration and hardware and software debugging

Subscribe

■ A power-up FPGA configuration mechanism

If your system has these capabilities, you can begin refining your design from a pretested hardware design loaded in the FPGA. Using an FPGA also allows you to modify your design quickly to address problems or to add new functionality. You can test these new hardware designs easily by reconfiguring the FPGA using your system's JTAG interface.

The JTAG interface supports hardware and software development. You can perform the following tasks using the JTAG interface:

■ Configure the FPGA

■ Download and debug software

■ Communicate with the FPGA through a UART-like interface (JTAG UART)

■ Debug hardware (with the SignalTap® II embedded logic analyzer)

■ Program flash memory

After you configure the FPGA with your Nios II processor-based design, the software development flow is similar to the flow for discrete microcontroller designs.

Embedded System Design

Whether you are a hardware designer or a software designer, read the *Nios II Hardware Development Tutorial* to start learning about designing embedded systems on an Altera FPGA. The "Nios II System Development Flow" section is particularly useful in helping you to decide how to approach system design using Altera's embedded hardware and software development tools. Altera recommends that you read this tutorial before starting your first design project. The tutorial teaches you the basic hardware and software flow for developing Nios II processor-based systems.

Designing with FPGAs gives you the flexibility to implement some functionality in discrete system components, some in software, and some in FPGA-based hardware. This flexibility makes the design process more complex. The SOPC Builder system design tool helps to manage this complexity. Even if you decide a soft-core processor does not meet your application's needs, SOPC Builder can still play a vital role in your system by providing mechanisms for peripheral expansion or processor offload.

Figure 1–1 illustrates the overall Nios II system design flow, including both hardware and software development. This illustration is greatly simplified. There are numerous correct ways to use the Altera tools to create a Nios II system.

Figure 1–1. General Nios II System Design Flow
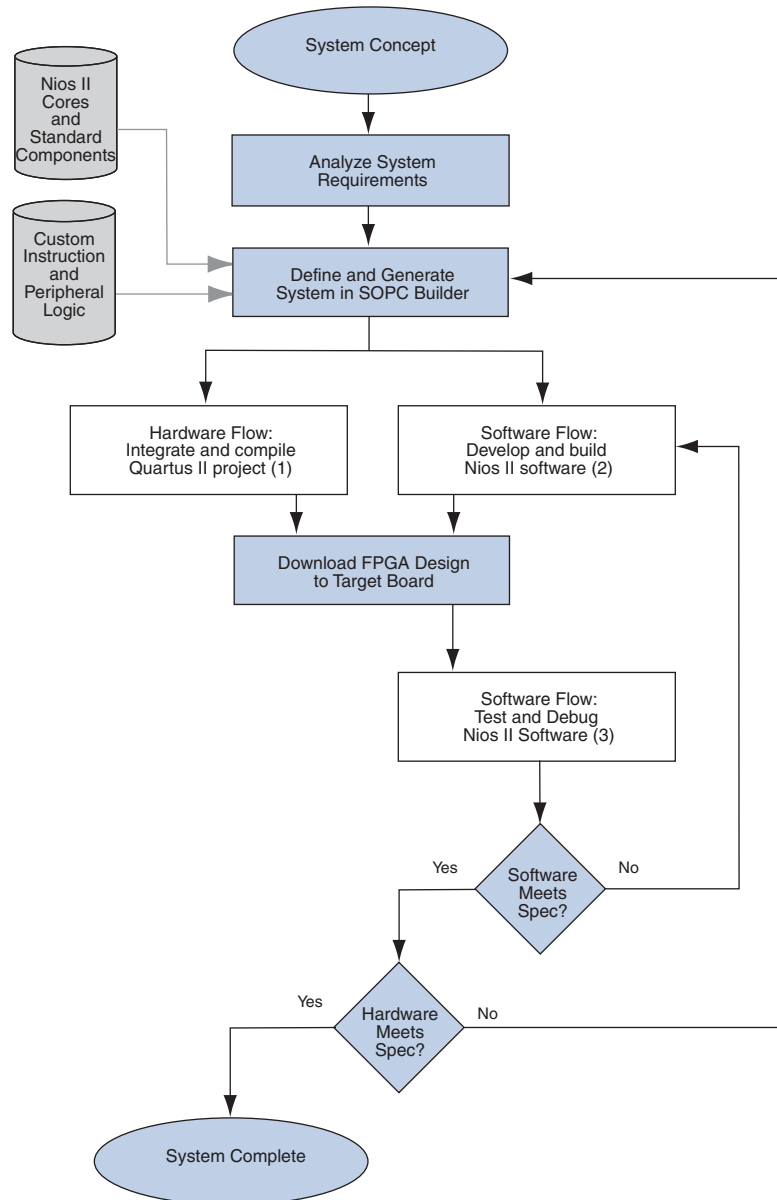
Notes to Figure 1–1:

(1) For details of the hardware flow, refer to Figure 1–2.

(2) For details of the software development flow, refer to Figure 1–4.

(3) For details of the software debug flow, refer to Figure 1–5.

FPGA Hardware Design

Figure 1–2 illustrates a typical FPGA hardware design process for a Nios II system.

Figure 1–2. Nios II System Hardware Design Flow
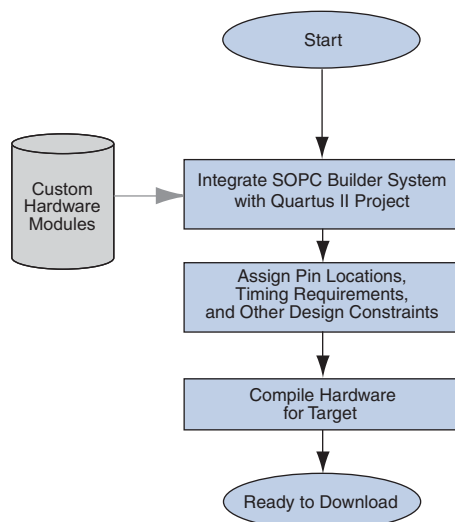
Although you develop your FPGA-based design in SOPC Builder, you must perform the following tasks in other tools:

- Connect signals from your FPGA-based design to your board level design

- Connect signals from your SOPC Builder system to other signals in the FPGA logic

- Constrain your design

Connecting Your FPGA Design to Your Hardware

To connect your FPGA-based design to your board-level design, perform the following two tasks:

1. Identify the top level of your FPGA design.

2. Assign signals in the top level of your FPGA design to pins on your FPGA using any of the methods mentioned at the Altera I/O Management, Board Development Support, and Signal Integrity Analysis Resource Center page of the Altera website.

☞ The top level of your FPGA-based design might be your SOPC Builder system. However, the FPGA can include additional design logic.

Connecting Signals to your SOPC Builder System

You must define the clock and reset pins for your SOPC Builder system. You must also define each I/O signal that is required for proper system operation. Figure 1–3 shows the top-level block diagram of an SOPC Builder system that includes a Nios II processor. The large symbol in this top-level diagram, labeled **std_1s40**, represents the SOPC Builder system. The flag-shaped pin symbols in this diagram represent off-chip (off-FPGA) connections.

> For more information about connecting your FPGA pins, refer to the Altera I/O Management, Board Development Support, and Signal Integrity Analysis Resource Center page of the Altera website.

Figure 1–3. Top-level Block Diagram
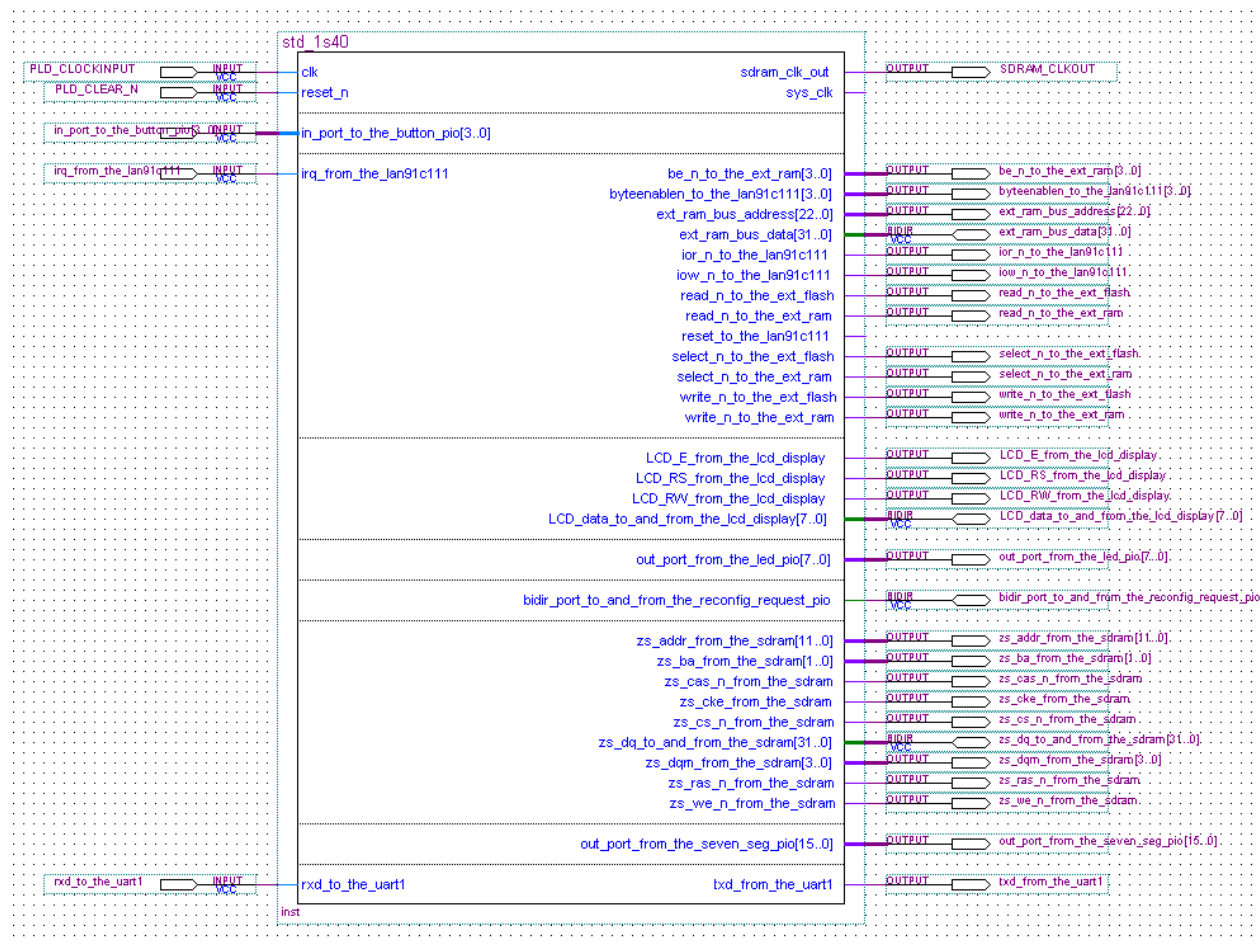
Constraining Your FPGA-Based Design

To ensure your design meets timing and other requirements, you must constrain the design to meet these requirements explicitly using tools provided in the Quartus® II software or by a third party EDA provider. The Quartus II software uses your constraint information during design compilation to achieve Altera's best possible results.

> Altera's third-party EDA partners and the tools they provide are listed on the Third-Party EDA Software page of the Altera website.

Hardware Design with SOPC Builder

SOPC Builder simplifies the task of building complex hardware systems on an FPGA. SOPC Builder allows you to describe the topology of your system using a graphical user interface (GUI) and then generate the hardware description language (HDL) files for that system. The Quartus II software compiles the HDL files to create an SRAM Object File (**.sof**).

For additional information about SOPC Builder, refer to the *SOPC Builder User Guide*.

SOPC Builder allows you to choose the processor core type and the level of cache, debugging, and custom functionality for each Nios II processor. Your design can use on-chip resources such as memory, PLLs, DSP functions, and high-speed transceivers. You can construct the optimal processor for your design using SOPC Builder.

After you construct your system using SOPC Builder, and after you add any required custom logic to complete your top-level design, you must create pin assignments using the Quartus II software. The FPGA's external pins have flexible functionality, and a range of pins is available to connect to clocks, control signals, and I/O signals.

For information about how to create pin assignments, refer to Quartus II Help and to the *I/O Management* chapter in *Volume 2: Design Implementation and Optimization* of the *Quartus II Handbook*.

Altera recommends that you start your design from a small pretested project and build it incrementally. Start with one of the many SOPC Builder example designs available from the Nios II Embedded Processor Design Examples web page of the Altera website, or with an example design from the *Nios II Hardware Development Tutorial*.

SOPC Builder allows you to create your own custom components using the component editor. In the component editor you can import your own source files, assign signals to various interfaces, and set various component and parameter properties.

Before designing a custom component, you should become familiar with the interface and signal types that are available in SOPC Builder.

Native addressing is deprecated. Therefore, you should use dynamic addressing for slave interfaces on all new components. Dynamically addressable slave ports include byte enables to qualify which byte lanes are accessed during read and write cycles. Dynamically addressable slave interfaces have the added benefit of being accessible by masters of any data width without data truncation or side effects.

To learn about the interface and signal types that you can use in SOPC Builder, refer to *Avalon Interface Specifications*. To learn about using the component editor, refer to the *Component Editor* chapter in the *SOPC Builder User Guide*.

As you add each hardware component to the system, test it with software. If you do not know how to develop software to test new hardware components, Altera recommends that you work with a software engineer to test the components.

The Nios II EDS includes several software examples, located in your Nios II EDS installation directory (**nios2eds**), at *<Nios II EDS install dir>***examples****software**. After you run a simple software design—such as the simplest example, Hello World Small—build individual systems based on this design to test the additional interfaces or custom options that your system requires. Altera recommends that you start with a simple system that includes a processor with a JTAG debug module, an on-chip memory component, and a JTAG UART component, and create a new system for each new untested component, rather than adding in new untested components incrementally.

After you verify that each new hardware component functions correctly in its own separate system, you can combine the new components incrementally in a single SOPC Builder system. SOPC Builder supports this design methodology well, by allowing you to add components and regenerate the project easily.

For detailed information about how to implement the recommended incremental design process, refer to the *Verification and Board Bring-Up* chapter of the *Embedded Design Handbook*.

SOPC Builder Design Replication

The recommended design flow requires that you maintain several small SOPC Builder systems, each with its Quartus II project and the software you use to test the new hardware. An SOPC Builder design requires the following files and folders:

- Quartus II Project File (**.qpf**)
- Quartus II Settings File (**.qsf**)

  The **.qsf** file contains all of the device, pin, timing, and compilation settings for the Quartus II project.

- One of the following types of top-level design file:
  - Block Design File (**.bdf**)
  - Verilog Design File (**.v**)
  - VHDL Design File (**.vhd**)

  If SOPC Builder generates your top-level design file, you do not need to preserve a separate top-level file.

  ☞ SOPC Builder generates most of the HDL files for your system, so you do not need to maintain them when preserving a project. You need only preserve the HDL files that you add to the design directly.

  For details about the design file types, refer to the Quartus II Help.

- SOPC Builder Design File (**.sopc**)
- SOPC Information File (**.sopcinfo**)

  This file contains an XML description of your SOPC Builder system. SOPC Builder and downstream tools, including the Nios II Software Build Tools (SBT), derive information about your system from this file.

- Your software application source files

To replicate an entire project (both hardware and software), simply copy the required files to a separate directory. You can create a script to automate the copying process. After the files are copied, you can proceed to modify the new project in the appropriate tools: the Quartus II software, SOPC Builder, the SBT for Eclipse, the SBT in the command shell, or the Nios II Integrated Development Environment (IDE).

For more information about all of these files, refer to the *Archiving SOPC Builder Projects* chapter in the *SOPC Builder User Guide*.

Customizing and Accelerating FPGA Designs

FPGA-based designs provide you with the flexibility to modify your design easily, and to experiment to determine the best balance between hardware and software implementation of your design. In a discrete microcontroller-based design process, you must determine the processor resources—cache size and built-in peripherals, for example—before you reach the final design stages. You may be forced to make these resource decisions before you know your final processor requirements. If you implement some or all of your system's critical design components in an FPGA, you can easily redesign your system as your final product needs become clear. If you use the Nios II processor, you can experiment with the correct balance of processor resources to optimize your system for your needs. SOPC Builder facilitates this flexibility, by allowing you to add and modify system components and regenerate your project easily.

To experiment with performance and resource utilization tradeoffs, the following hardware optimization techniques are available:

■ **Processor Performance**—You can increase the performance of the Nios II processor in the following ways:

- ■ **Computational Efficiency**—Selecting the most computationally efficient Nios II processor core is the quickest way to improve overall application performance. The following Nios II processor cores are available, in decreasing order of performance:

  - ■ Nios II/f—optimized for speed

  - ■ Nios II/s—balances speed against usage of on-chip resources

  - ■ Nios II/e—conserves on-chip resources at the expense of speed

- ■ **Memory Bandwidth**—Using low-latency, high speed memory decreases the amount of time required by the processor to fetch instructions and move data. Additionally, increasing the processor's arbitration share of the memory increases the processor's performance by allowing the Nios II processor to perform more transactions to the memory before another Avalon master port can assume control of the memory.

- ■ **Instruction and Data Caches**—Adding an instruction and data cache is an effective way to decrease the amount of time the Nios II processor spends performing operations, especially in systems that have slow memories, such as SDRAM or double data rate (DDR) SDRAM. In general, the larger the cache size selected for the Nios II processor, the greater the performance improvement.

■ **Clock Frequency**—Increasing the speed of the processor's clock results in more instructions being executed per unit of time. To gain the best performance possible, ensure that the processor's execution memory is in the same clock domain as the processor, to avoid the use of clock-crossing FIFO buffers.

One of the easiest ways to increase the operational clock frequency of the processor and memory peripherals is to use a FIFO bridge IP core to isolate the slower peripherals of the system. With a bridge peripheral, for example, you can connect the processor, memory, and an Ethernet device on one side of the bridge, and connect all of the peripherals that are not performance dependent on the other side of the bridge.

Similarly, if you implement your system in an FPGA, you can experiment with the best balance of hardware and software resource usage. If you find you have a software bottleneck in some part of your application, you can consider accelerating the relevant algorithm by implementing it in hardware instead of software. SOPC Builder facilitates experimenting with the balance of software and hardware implementation. You can even design custom hardware accelerators for specific system tasks.

To help you solve system performance issues, the following acceleration methodologies are available:

■ Custom peripherals

■ Custom instructions

■ The Nios II C-to-Hardware (C2H) Acceleration Compiler

The method of acceleration you choose depends on the operation you wish to accelerate. To accelerate streaming operations on large amounts of data, a custom peripheral may be a good solution. Hardware interfaces (such as implementations of the Ethernet or serial peripheral interface (SPI) protocol) may also be implemented efficiently as custom peripherals. The current floating-point custom instruction is a good example of the type of operations that are typically best accelerated using custom instructions.

Working with a software or systems engineer, use the C2H Compiler to help analyze sophisticated algorithms to determine potential hardware acceleration gains. As in any hardware acceleration methodology, you must make trade-offs between performance and resource consumption. When a C compiler compiles code using a high level of optimization, the resulting executable program typically runs faster, but also often consumes more memory than similar code compiled with a lower level of optimization. Similarly, accelerators built with the C2H Compiler typically run faster than the unaccelerated code, but they consume more FPGA resources.

For information about hardware acceleration, refer to the *Hardware Acceleration and Coprocessing* chapter of the *Embedded Design Handbook*. For information about how to use the C2H Compiler, refer to the *Nios II C2H Compiler User Guide* and to the *Optimizing Nios II C2H Compiler Results* chapter of the *Embedded Design Handbook*. For information about custom instructions, refer to the *Nios II Custom Instruction User Guide*. For information about creating custom peripherals, refer to the *SOPC Builder Component Development Walkthrough* chapter in the *SOPC Builder User Guide*.

Nios II Software Design

This section contains brief descriptions of the software design tools provided by the Nios II EDS, including the Nios II SBT development flow and the Nios II IDE development flow.

Nios II Tools Overview

The Nios II EDS provides the following tools for software development:

■ GNU toolchain: GCC-based compiler with the GNU binary utilities

For an overview of these and other Altera-provided utilities, refer to the *Nios II Command-Line Tools* chapter of the *Embedded Design Handbook*.

■ Nios II processor-specific port of the newlib C library

■ Hardware abstraction layer (HAL)

The HAL provides a simple device driver interface for programs to communicate with the underlying hardware. It provides many useful features such as a POSIX-like application program interface (API) and a virtual-device file system.

For more information about the Altera HAL, refer to *The Hardware Abstraction Layer* section of the *Nios II Software Developer's Handbook*.

■ Nios II SBT

The Nios II SBT development flow is a scriptable development flow. It includes the following user interfaces:

■ The Nios II SBT for Eclipse—a GUI that supports creating, modifying, building, running, and debugging Nios II programs. It is based on the Eclipse open development platform and Eclipse C/C++ development toolkit (CDT) plug-ins.

■ The Nios II SBT command-line interface—From this interface, you can execute SBT command utilities, and use scripts (or other tools) to combine the command utilities in many useful ways.

For more information about the Nios II SBT flow, refer to the *Developing Nios II Software* chapter of the *Embedded Design Handbook*.

■ Nios II IDE—an alternative GUI environment in which you can create, modify, build, run, and debug Nios II programs. The Nios II IDE flow does not use the Nios II SBT. This flow provides limited control over the build process and the project settings, with no support for customized scripting. Nios II IDE projects are not compatible with SBT projects.

☞ In most cases, you should create new projects using the Nios II SBT from the command line or from Eclipse. 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

👣 For further information about the Nios II IDE, refer to *Appendix A. Using the Nios II Integrated Development Environment* in the *Nios II Software Developer's Handbook*.

Figure 1–4 illustrates the available Nios II software development flows.

Figure 1–4. Nios II Software Development Flows: Developing Software
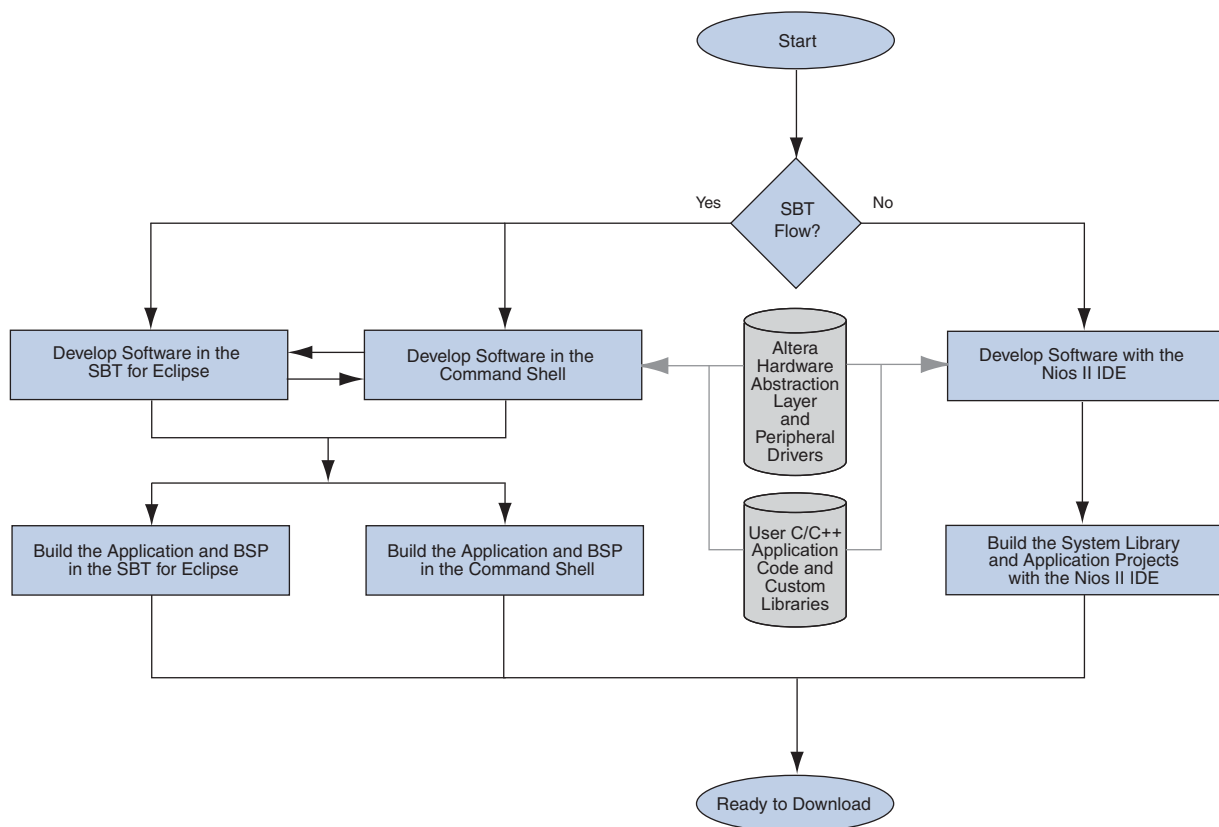
Altera recommends that you view and begin your design with one of the available software examples that are installed with the Nios II EDS. From simple "Hello, World" programs to networking and RTOS-based software, these examples provide good reference points and starting points for your own software development projects. The Hello World Small example program illustrates how to reduce your code size without losing all of the conveniences of the HAL.

☞ Altera recommends that you use an Altera development kit or custom prototype board for software development and debugging. Many peripheral and system-level features are available only when your software runs on an actual board.

For more detailed information, refer to the *Nios II Software Developer's Handbook*.

Nios II Software Build Tools Flow

The Nios II SBT flow uses the Software Build Tools to provide a flexible, portable, and scriptable software build environment. Altera recommends that you use this flow. The SBT includes a command-line environment and fits easily in your preferred software or system development environment. The Nios II SBT is the basis for Altera's future development.

The SBT flow requires that you have a **.sopcinfo** file for your system. The flow includes the following steps to create software for your system:

1. Create a board support package (BSP) for your system. The BSP is a layer of software that interacts with your development system. It is a makefile-based project.

2. Create your application software:

    a. Write your code.

    b. Generate a makefile-based project that contains your code.

3. Iterate through one or both of these steps until your design is complete.

For more information, refer to the software example designs that are shipped with every release of the Nios II EDS. For more information about these examples, refer to one of the following sections:

- "Getting Started" in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*

- "Nios II Example Design Scripts" in the *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer's Handbook*

Nios II IDE Flow

To learn about the Nios II IDE, refer to the Nios II software development tutorial.

This tutorial is contained in the Nios II IDE Help system. To open this Help system, in the Nios II IDE, on the Help menu, click **Welcome**. You can also refer to the *Nios II IDE Help System* in PDF form on the Altera website.

Software Debugging Options

The Nios II EDS provides the following programs to aid in debugging your hardware and software system:

- The Nios II SBT for Eclipse debugger

- The Nios II IDE debugger

- Several distinct interfaces to the GNU Debugger (GDB)

- A Nios II-specific implementation of the First Silicon Solutions, Inc. FS2 console (available only with the Nios II IDE and the command-line SBT on Windows platforms)

- System Console, a system debug console

Figure 1–5 illustrates two commonly-used tool flows for debugging Nios II software.

Figure 1–5. Nios II Software Development Flows: Testing Software
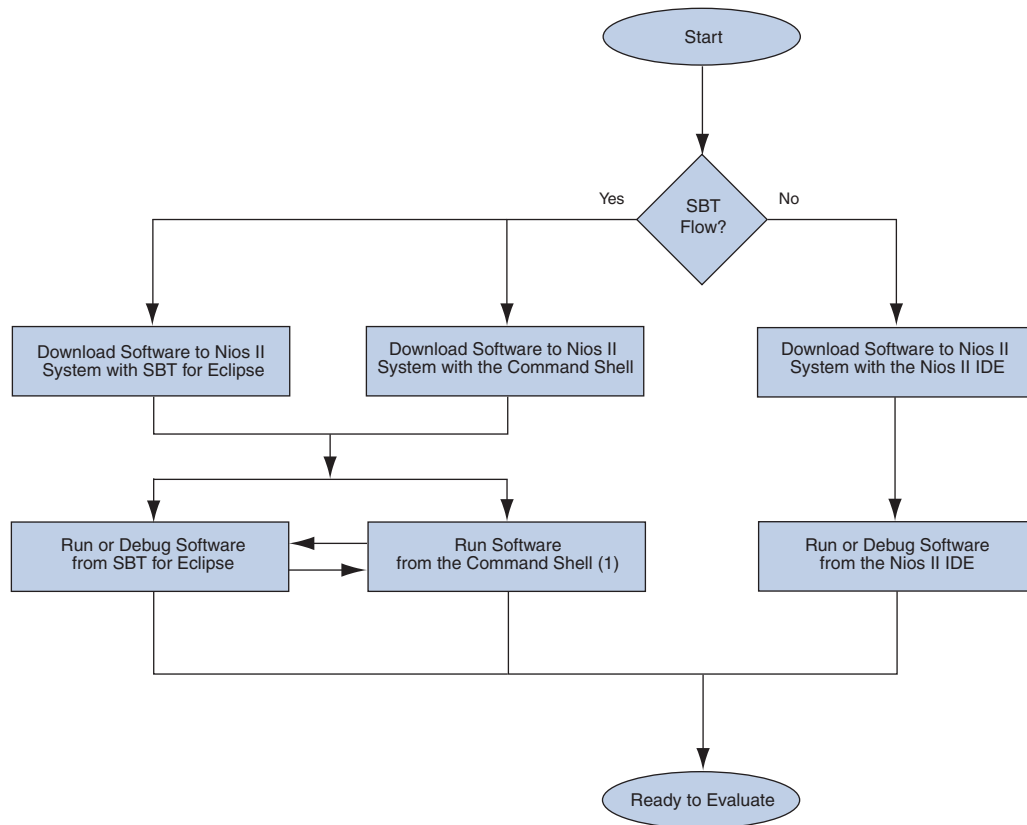
Note to Figure 1–5:

(1)   To debug a command-line project, import it to the SBT for Eclipse.

You can begin debugging software immediately using the built-in Nios II SBT for Eclipse debugger. This debugging environment includes advanced features such as trace, watchpoints, and hardware breakpoints.

The Nios II EDS includes the following three interfaces to the GDB debugger:

■ GDB console (accessible through the Nios II SBT for Eclipse and the Nios II IDE)

■ Standard GDB client (nios2-elf-gdb)

■ Insight GDB interface (Tcl/Tk based GUI)

Additional GDB interfaces such as Data Display Debugger (DDD), and Curses GDB (CGDB) interface also function with the Nios II version of the GDB debugger.

For more information about these interfaces to the GDB debugger, refer to the *Nios II Command-Line Tools* and *Debugging Nios II Designs* chapters of the *Embedded Design Handbook*. For detailed information about the FS2 console, refer to the documentation in the *<Nios II EDS install dir>***bin****fs2****doc** directory and to the *Verification and Board Bring-Up* chapter of the *Embedded Design Handbook*.

The System Console is a system debug console that provides the SOPC Builder designer with a Tcl-based, scriptable command-line interface for performing system or individual component testing.

For detailed information about the System Console, refer to the *Analyzing and Debugging Designs with the System Console* chapter in volume 3 of the *Quartus II Handbook*. On-line training is available at the Altera Training page of the Altera website.

Third party debugging environments are also available from vendors such as Lauterbach Datentechnik GmBH and First Silicon Solutions, Inc.

Rebuilding Software from the Command Line

Rebuilding software after minor source code edits does not require a GUI. You can rebuild the project from a Nios II Command Shell, using your application's makefile. To build or rebuild your software, perform the following steps:

1. Open a Nios II Command Shell by executing one of the following steps, depending on your environment:

    ■ In the Windows operating system, on the Start menu, point to **Programs** > **Altera** > **Nios II EDS**, and click **Nios II Command Shell**.

    ■ In the Linux operating system, in a command shell, type the following sequence of commands:

    ```
    cd <Nios II EDS install path>
    ./nios2_command_shell.sh
    ```

2. Change to the directory in which your makefile is located. If you use the Nios II IDE for development, the correct location is often the **Debug** or **Release** subdirectory of your software project directory.

3. In the Command Shell, type one of the following commands:
    ```
    make ↵
    ```
    or
    ```
    make -s ↵
    ```

Example 1–1 illustrates the output of the make command run on a sample system.

Example 1–1. Sample Output From make -s Command

```
[SOPC Builder]$ make -s
Creating generated_app.mk...
Creating generated_all.mk...
Creating system.h...
Creating alt_sys_init.c...
Creating generated.sh...
Creating generated.gdb...
Creating generated.x...
Compiling src1.c...
Compiling src2.c...
Compiling src3.c...
Compiling src4.c...
Compiling src5.c...
Linking project_name.elf...
```

☞ If you add new files to your project or make significant hardware changes, recreate the project with the original tool (the Nios II SBT or the Nios II IDE). Recreating the project recreates the makefile for the new version of your system after the modifications.

Board Design Considerations

You must choose the method to configure, or program, your FPGA, and the method to boot your Nios II processor.

Configuration

Many FPGA configuration options are available to you. The two most commonly used options configure the FPGA from flash memory. One option uses a CPLD and a CFI flash device to configure the FPGA, and the other uses a serial flash EPCS configuration device. The Nios II development kits use these two configuration options by default.

Choose the first option, which uses a CPLD and a CFI-compliant flash memory, in the following cases:

■ Your FPGA is large

■ You must configure multiple FPGAs

■ You require a large amount of flash memory for software storage

■ Your design requires multiple FPGA hardware images (safe factory images and user images) or multiple software images

EPCS configuration devices are often used to configure small, single-FPGA systems.

☞ The default Nios II boot loader does not support multiple FPGA images in EPCS devices.

👣 For help in configuring your particular device, refer to the device family information on the Altera Devices page of the Altera website.

Figure 1–6 shows the block diagram of the configuration controller used on the Nios II Development Kit, Cyclone® II Edition. This controller design is used on older development kits, and is a good starting point for your design.

Figure 1–6. Configuration Controller for Cyclone II Devices
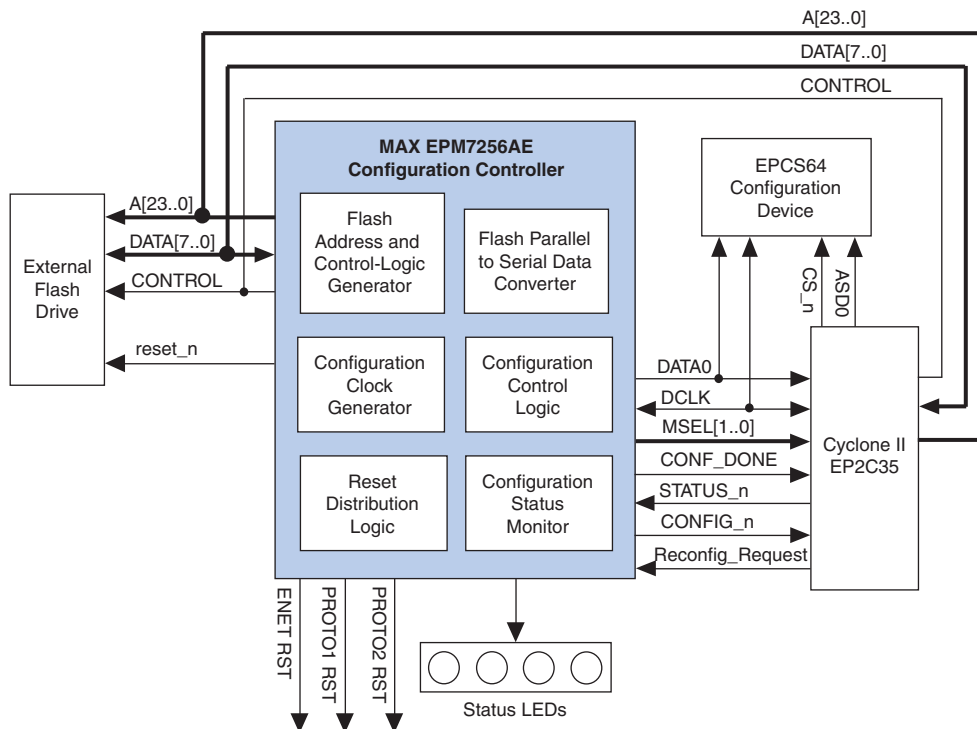
For more information about controller designs, refer to *AN346: Using the Nios II Configuration Controller Reference Designs*.

altremote_update Megafunction-Based Configuration

Newer devices such as the Cyclone III, Stratix® II, and later devices include the built-in ALTREMOTE_UPDATE megafunction to help you configure your FPGA. For these newer devices, no additional Programmable Logic Device (PLD) is necessary for configuration control. However, older devices require a configuration controller device, as shown in Figure 1–6.

For information about the ALTREMOTE_UPDATE megafunction, refer to the *Remote System Upgrade (ALTREMOTE_UPDATE) Megafunction User Guide*. The Application Selector example uses this megafunction in the Nios II Embedded Evaluation Kit (NEEK), Cyclone III Edition.

Booting

Many Nios II booting options are available. The following options are the most commonly used:

■ Boot from CFI Flash

■ Boot from EPCS

■ Boot from on-chip RAM

The default boot loader that is included in the Nios II EDS supports boot from CFI flash memory and from EPCS flash memory. If you use an on-chip RAM that supports initialization, such as the M4K and M9K types of RAM, you can boot from the on-chip RAM without a boot loader.

> For additional information about Nios II boot methodologies, refer to *AN458: Alternative Nios II Boot Methods*.

Additional Embedded Design Considerations

Consider the following topics as you design your system:

- JTAG signal integrity
- Extra memory space for prototyping
- System verification

JTAG Signal Integrity

The JTAG signal integrity on your system is very important. You must debug your hardware and software, and program your FPGA, through the JTAG interface. Poor signal integrity on the JTAG interface can prevent you from debugging over the JTAG connection, or cause inconsistent debugger behavior.

You can use the System Console to verify the JTAG chain.

> JTAG signal integrity problems are extremely difficult to diagnose. To increase the probability of avoiding these problems, and to help you diagnose them should they arise, Altera recommends that you follow the guidelines outlined in *AN428: MAX II CPLD Design Guidelines* and in the *Verification and Board Bring-Up* chapter of the *Embedded Design Handbook* when designing your board.

> For more information about the System Console, refer to the *Analyzing and Debugging Designs with the System Console* chapter in volume 3 of the *Quartus II Handbook*.

Memory Space For System Prototyping

Even if your final product includes no off-chip memory, Altera recommends that your prototype board include a connection to some region of off-chip memory. This component in your system provides additional memory capacity that enables you to focus on refining code functionality without worrying about code size. Later in the design process, you can substitute a smaller memory device to store your software.

Embedded System Verification

> For useful information about design techniques for your embedded system, refer to the *Verification and Board Bring-Up* chapter of the *Embedded Design Handbook*. Altera recommends that you read this chapter before you begin your design.

Embedded Design Resources

This section contains a list of resources to help you find design help. Your resource options include traditional Altera-based support such as online documentation, training, and My Support, as well as web-based forums and Wikis. The best option depends on your inquiry and your current stage in the design cycle.

Altera Embedded Support

Altera recommends that you seek support in the following order:

1. Look for relevant literature on the Literature and Technical Documentation page of the Altera website, especially on the Literature: Nios II Processor page and the Literature: SOPC Builder page.

2. Contact your local Altera sales office or sales representative, or your field application engineer (FAE).

3. Contact technical support through the myAltera page of the Altera website to get support directly from Altera.

4. Consult one of the following community-owned resources:

   ■ The Nios Forum, available on the Altera Forum website (www.alteraforum.com)

   ■ The Altera Wiki website (www.alterawiki.com)

   ☞ Altera is not responsible for the contents of the Nios Forum and Altera Wiki websites, which are maintained by groups outside of Altera.

Altera Embedded Training

To learn how the tools work together and how to use them in an instructor-led environment, register for training. Several training options are available. For information about general training, refer to the Training & Events page of the Altera website.

For detailed information about available courses and their locations, visit the Embedded SW Designer Curriculum page of the Altera website. This page contains information about both online and instructor-led training.

Altera Embedded Documentation

You can access documentation about the Nios II processor and embedded design from your Nios II EDS installation directory at *<Nios II EDS install dir>***documents****index.htm**. To access this page directly on Windows platforms, on the Start menu, click **All Programs**. On the All Programs menu, on the Altera submenu, on the Nios II EDS *<version>* submenu, click **Nios II *<version>* Documentation.** This web page contains links to the latest Nios II documentation.

The Literature: Nios II Processor page of the Altera website includes a list and links to available documentation. At the bottom of this page, you can find links to various product pages that include Nios II processor online demonstrations and embedded design information.

If you are running the Nios II IDE, information for first-time users appears on the Welcome page. This page appears when you first open the Nios II IDE after installation. You can open it at any time by clicking **Welcome** on the Help menu.

The other chapters in the *Embedded Design Handbook* are a valuable source of information about embedded hardware and software design, verification, and debugging. Each chapter contains links to the relevant overview documentation.

Third Party Intellectual Property

Many third parties have participated in developing solutions for embedded designs with Altera FPGAs through the Altera AMPP^{SM} Program. For up-to-date information about the third-party solutions available for the Nios II processor, visit the Embedded Processing page of the Altera website, and click **Altera Embedded Alliance Partners**.

Several community forums are also available. These forums are not controlled by Altera. The Altera Forum's Marketplace provides third-party hard and soft embedded systems-related IP. The forum also includes an unsupported projects repository of useful example designs. You are welcome to contribute to these forum pages.

Traditional support is available from the Support Center or through your local Field Application Engineer (FAE). You can obtain more informal support by visiting the Nios Forum section of the Altera Forum (www.alteraforum.com) or by browsing the information contained on the Altera Wiki (www.alterawiki.com). Many experienced developers, from Altera and elsewhere, contribute regularly to Wiki content and answer questions on the Nios Forum.

Altera Embedded Glossary

The following definitions explain some of the unique terminology for describing SOPC Builder and Nios II processor-based systems:

■ **Component**—A named module in SOPC Builder that contains the hardware and software necessary to access a corresponding hardware peripheral.

■ **Custom instruction**—Custom hardware processing integrated with the Nios II processor's ALU. The programmable nature of the Nios II processor and SOPC Builder-based design supports this implementation of software algorithms in custom hardware. Custom instructions accelerate common operations. (The Nios II processor floating-point instructions are implemented as custom instructions).

■ **Custom peripheral**—An accelerator implemented in hardware. Unlike custom instructions, custom peripherals are not connected to the CPU's ALU. They are accessed through the system interconnect fabric. (*See* **System interconnect fabric**). Custom peripherals offload data transfer operations from the processor in data streaming applications.

■ **ELF (Executable and Linking Format)**—The executable format used by the Nios II processor. This format is arguably the most common of the available executable formats. It is used in most of today's popular Linux/BSD operating systems.

■ **HAL (Hardware Abstraction Layer)**—A lightweight runtime environment that provides a simple device driver interface for programs to communicate with the underlying hardware. It provides a POSIX-like software layer and wrapper to the newlib C library.

■ **Nios II C-To-Hardware Acceleration (C2H) Compiler**—A push-button ANSI C-to-hardware compiler that allows you to explore algorithm acceleration and design-space options in your embedded system.

■ **Nios II Command Shell**—The command shell you use to access Nios II and SOPC Builder command-line utilities.

  ■ On Windows platforms, a Nios II Command Shell is a Cygwin bash with the environment properly configured to access command-line utilities.

  ■ On Linux platforms, to run a properly configured bash, type
    <Nios II EDS install path>/nios2_command_shell.sh↵

■ **Nios II Embedded Development Suite (EDS)**—The complete software environment required to build and debug software applications for the Nios II processor. The EDS includes the Nios II IDE. (*See* **Nios II IDE**).

■ **Nios II IDE**—An Eclipse-based development environment for Nios II embedded designs with limited control over the software build process.

■ **Nios II Software Build Tools (SBT)**—Software that allows you to create Nios II software projects, with detailed control over the software build process.

■ **Nios II Software Build Tools for Eclipse**—An Eclipse-based development environment for Nios II embedded designs, using the SBT for project creation and detailed control over the software build process. The SBT for Eclipse provides software project management, build, and debugging capabilities.

■ **SOPC Builder**—Software that provides a GUI-based system builder and related build tools for the creation of FPGA-based subsystems, with or without a processor.

■ **System interconnect fabric**—An interface through which the Nios II processor communicates to on- and off-chip peripherals. This fabric provides many convenience and performance-enhancing features.

Conclusion

This chapter is a basic overview of the Altera embedded development process and tools for the first time user. The chapter focuses on using these tools and where to find more information. It references other Altera documents that provide detailed information about the individual tools and procedures. It contains resource and glossary sections to help orient the first time user of Altera's embedded development tools for hardware and software development.

Document Revision History

Table 1–1 shows the revision history for this document.

Table 1–1. Document Revision History

Date	Version	Changes
July 2011	2.3	■ Clarified this handbook does not include information about Qsys. ■ Updated location of hardware design examples. ■ Added hardware optimization information. ■ Updated references.
March 2010	2.2	Updated for the SBT for Eclipse.
January 2009	2.1	Updated Nios Wiki hyperlink.
November 2008	2.0	Added System Console.
March 2008	1.0	Initial release.